Statistics and Computing

Leland Wilkinson

The Grammar of Graphics

Second Edition

Springer

Statistics and Computing

Series Editors:
J. Chambers
D. Hand
W. Härdle

Leland Wilkinson

The Grammar of Graphics

Second Edition

With 410 Illustrations, 319 in Full Color

With contributions by Graham Wills, Dan Rope, Andrew Norton, and Roger Dubbs

Springer

Leland Wilkinson
SPSS Inc.
233 S. Wacker Drive
Chicago, IL 60606-6307
USA
leland@spss.com

Series Editors:

J. Chambers
Bell Labs, Lucent
Technologies
600 Mountain Ave.
Murray Hill, NJ 07974
USA

D. Hand
Department of Mathematics
South Kensington Campus
Imperial College, London
London
SW7 2AZ
United Kingdom

W. Härdle
Institut für Statistik und
Ökonometrie
Humboldt-Universität zu Berlin
Spandauer Str. 1
D-10178 Berlin, Germany

Library of Congress Control Number:

ISBN-10: 0-387-24544-8
ISBN-13: 978-0387-24544-7

Printed on acid-free paper.

© 2005 Springer Science+Business Media, Inc.
All rights reserved. This work may not be translated or copied in whole or in part without the written permission of the publisher (Springer Science+Business Media, Inc., 233 Spring Street, New York, NY 10013, USA), except for brief excerpts in connection with reviews or scholarly analysis. Use in connection with any form of information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed is forbidden.
The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

Printed in Canada.

9 8 7 6 5 4 3 2 1

springeronline.com

To John Hartigan and Amie Wilkinson

Who can hide in secret places so that I cannot see them? Do I not fill heaven and earth?
Jeremiah 23.24

Cleave a piece of wood, I am there; lift up the stone and you will find me there.
Gospel of Thomas 77

God hides in the smallest pieces.
Caspar Barlaeus

God hides in the details.
Aby Warburg

God is in the details.
Ludwig Mies van der Rohe

The devil is in the details.
George Shultz

Bad programmers ignore details. Bad designers get lost in details.
Nate Kirby

Preface

Preface to First Edition

Before writing the graphics for SYSTAT in the 1980's, I began by teaching a seminar in statistical graphics and collecting as many different quantitative graphics as I could find. I was determined to produce a package that could draw every statistical graphic I had ever seen. The structure of the program was a collection of procedures named after the basic graph types they produced. The graphics code was roughly one and a half megabytes in size.

In the early 1990's, I redesigned the SYSTAT graphics package using object-based technology. I intended to produce a more comprehensive and dynamic package. I accomplished this by embedding graphical elements in a tree structure. Rendering graphics was done by walking the tree and editing worked by adding and deleting nodes. The code size fell to under a megabyte.

In the late 1990's, I collaborated with Dan Rope at the Bureau of Labor Statistics and Dan Carr at George Mason University to produce a graphics production library called GPL, this time in Java. Our goal was to develop graphics components. This book was nourished by that project. So far, the GPL code size is under half a megabyte.

I have not yet achieved that elusive zero-byte graphics program, but I do believe that bulk, in programming or in writing, can sometimes be an inverse measure of clarity of thought. Users dislike "bloatware" not only because it is a pig that wastes their computers' resources but also because they know it usually reflects design-by-committee and sloppy thinking.

Notwithstanding my aversion to bulk, this book is longer than I had anticipated. My original intent was to outline a new paradigm for quantitative graphics using examples from publications and from SYSTAT. As the GPL project proceeded and we were able to test concepts in a working program, I began to realize that the details of the system were as important as the outlines. I also found that it was easier to write about the generalities of graphics than about the particulars. As every programmer knows, it is easier to wave one's hands than to put them to the keyboard. And as every programmer knows in the middle of the night, the computer "wonderfully focuses the mind."

The consequence is a book that is not, as some like to say, "an easy read." I do not apologize for this. Statistical graphics is not an easy field. With rare exceptions, theorists have not taken graphics seriously or examined the field deeply. And I am convinced that those who have, like Jacques Bertin, are not often read carefully. It has taken me ten years of programming graphics to understand and appreciate the details in Bertin.

I am not referring to the abstruseness of the mathematics in scientific and technical charts when I say this is not an easy field. It is easier to graph Newton's law of gravitation than to draw a pie chart. And I do not mean that no one has explored aspects of graphics in depth or covered the whole field with illumination. I mean simply that few have viewed quantitative graphics as an area that has peculiar rules and deep grammatical structure. As a result, we have come to expect we can understand graphics by looking at pictures and speaking in generalities. Against that expectation, I designed this book to be read more than once. On second reading, you will discover the significance of the details and that will help you understand the necessity of the framework.

Who should read this book? The simple answer is, of course, anyone who is interested in business or scientific graphics. At the most elementary level are readers who are looking for a graphical catalog or thesaurus. There are not many types of graphics that do not appear somewhere in this book. At the next level are those who want to follow the arguments without the mathematics. One can skip all the mathematics and still learn what the fundamental components of quantitative graphics are and how they interact. At the next level are those who have completed enough college mathematics to follow the notation. I have tried to build the argument, except for the statistical methods in Chapter 7, from elementary definitions. I chose a level comparable to an introductory computer science or discrete math text, and a notation that documents the algorithms in set terminology computer science students will recognize.

I intend to reach several groups. First are college and graduate students in computer science and statistics. This is the only book in print that lays out in detail how to write computer programs for business or scientific graphics. For all the attention computer graphics courses devote to theory, modeling, animation, and realism, the vast majority of commercial applications involve quantitative graphics. As a software developer, I believe the largest business market for graphics will continue to be analysis and reporting, despite the enthusiastic predictions (driven by conventional wisdom) for data mining, visualization, animation, and virtual reality. The reason, I think, is simple. People in business and science have more trouble communicating than discovering.

The second target group for this book comprises mathematicians, statisticians, and computer scientists who are not experts in quantitative graphics. I hope to be able to convey to them the richness of this field and to encourage them to explore it beyond what I have been able to do. Among his many accomplishments in the fields of graphics and statistics, William Cleveland is largely responsible for stimulating psychologists (including me) to take a closer look at graphical perception and cognition. I hope this book will stimulate experts in other fields to examine the language of graphics.

The third target group consists of statistics and computer science specialists in graphics. These are the colleagues most likely to recognize that this book is more the assembly of a large puzzle than the weaving of a whole cloth. I cannot assume every expert will understand this book, however, for reasons similar to why expertise in procedural programming can hinder one from learning object-oriented design. Those who skim through or jump into the middle of this book are most likely to misunderstand. There are many terms in this book — *graph*, *graphic*, *variable*, *frame*, *point*, *line*, *shape*, *scale* — that have unfortunately come to be overloaded in ordinary and technical discourse. I have taken care to define these terms when they first appear and then to refine the definitions in context later in the book. Preconceptions about these terms, however technical, can prevent one from following my argument. And those who have heard me talk about graphics algebra in meetings or colloquia need to keep in mind that algebra is only one of fifteen chapters in this book. Before drawing any conclusions, one should read the book from start to finish and attend to the details.

The popular saying "God is in the details," whose lineage I have traced on the frontispiece, has an ancient heritage. It is usually attributed in English to the architect Ludwig Mies van der Rohe, who probably was quoting the art historian Aby Warburg. Elizabeth Sears, a Warburg scholar, told me that Warburg's saying is "much quoted, its sources variously sought." She cited Kany (1985), who devoted an entire article to the topic. William Heckscher (1958) found a possible source in the 17th century humanist Caspar Barlaeus (see correspondents' notes in Safire, 1997). The idea has much older roots in Western culture, however. It is a corollary of an immanent creator — the opposite of an absconding God. Church fathers and rabbis discussed God's omnipresence along these lines in the first millennium, and I have cited a verse from Jeremiah that gives evidence of biblical roots. In our time, we have altered its meaning by focusing on our attending to details rather than on God being in them. I do not know if George Shultz is the first to have given the saying an ironic twist. He used the expression "the devil is in the details" when referring to the intricacies of the SALT talks in a speech to the Council on Foreign Relations. In retrospect, however, Shultz informed me that he may have been quoting some earlier wit. My favorite recent redaction is by a programmer at SPSS. Nate Kirby's observation that bad programmers ignore details and bad designers get lost in them captures for me the difficulty in creating a complex system.

This book was composed in Times® Roman and Italic with Adobe FrameMaker®. The quantitative graphics were produced with SYSTAT®. Rick Wessler drew Figure 9.57 with Caligari TrueSpace®. Figure 20.1 (also on the cover) was created originally in GPL. The remaining non-statistical diagrams and tables were drawn with tools in FrameMaker and Adobe Photoshop®.

I have many to thank. Dan Rope is one of the few individuals I have met who is both master designer and master coder. He gave me the rare opportunity to test almost every idea in this book in detail and contributed many of his own. The few untested ideas will probably be discovered some years from now when others program them, but they will not be Dan's fault. Dan Carr, my other GPL collaborator, taught me with examples. One graphic from Dan (he has done many) can teach me more about good design than some books.

The group once at Bell Labs and now at its descendent institutions has continued to be a unique source of inspiration. Bill Cleveland has energized and advised me for almost two decades; he wins the citation derby in this book. John Chambers, John Tukey, Paul Tukey, Rick Becker, Deborah

Swayne, Andreas Buja, Allan Wilks, Daryl Pregibon, James Landwehr, Lorraine Denby, and Mark Hansen have listened encouragingly and repeatedly to half-baked ideas.

At Northwestern, Bruce Spencer, Shelby Haberman, Martin Tanner, Keith Burns, and Amie Wilkinson have provided critical advice and support. At the University of Chicago, Stephen Stigler, William Kruskal, Ron Thisted, Benson Farb, and Peter McCullaugh have done the same. My Yale cohorts Jerry Dallal and Bill Eddy have helped me over more years than the work in this book spans.

At SYSTAT, Dan Michaels and Steve Willie introduced me to object-oriented design. Tom Leuthner worked extensively with me on maps. Mark Bjerknes, Greg Mullins, Mike Pechnyo, and Sasha Khalileev helped alert me to graphical issues raised by Mac, DOS, Windows, UNIX, and other environments. Laszlo Engelman shared numerical and graphical tricks that continue to amaze me. Pat Fleury and MaryAnn Hill made sure I continued to think about the graphical needs of real users.

At SPSS, Jack Noonan, Ed Hamburg, Mark Battaglia, and Nancy Dobrozdravic provided me the environment needed to complete this project. Josh Barr, Matt Levine, Andrew Walaszek, Jay Jayaprasad, Matt Rubin, Rajesh Selukar, John Burkey, Scott Adler, Janice Krinsky, Sheri Gilley, ViAnn Beadle, Rick Marcantonio, Joel York, Keith Kroeger, Jing Shyr, Ming-Long Lam, Jim Cortese, Scott Sipiora, Dave Hess, Tex Hull, Kim Peck, John Fry, Bonnie Shapiro, Richard Oliver, and Hiroaki Minato contributed numerous suggestions.

Elsewhere, I have received support and advice from several colleagues. The most significant have been Robert Abelson (my dissertation advisor at Yale), Susanna Epp, and Helmut Epp. Jack Cohen and Louis Guttman, no longer here to guide me, are always in my thoughts.

Some have graciously taken time to read all or part of this manuscript. I thank Amie Wilkinson, Paul Velleman, Howard Wainer, Wendell Garner, Michael Kubovy, Stephen Kosslyn, Wayne Oldford, David Scott, Cynthia Brewer, Alan MacEachren, Dianne Cook, Jürgen Symanzik, Jim Russell, and Stephen Stigler. And I especially thank a series editor at Springer-Verlag who made the effort to tell me what was sense and what was nonsense so I could revise intelligently.

At Springer-Verlag, I thank John Kimmel. John is a gentleman. A gentleman (or gentlewoman) is someone who never needs to sign a contract. I finally signed one for this book, but it was many years after I knew, and I know John knew, that I would not publish this book with anyone but John. I also thank Mary Ann Cottone for gentle persistence in coordinating the production of an extremely complex four-color manuscript.

My wife Ruth VanDemark has patiently lived with this book through many nights and weekends. And, while thanking family, I must confess a special pleasure in joining my son-in-law Benson Farb as a Springer author. Benson tutored me in the rudiments of geometry.

Lastly, the two to whom this book is dedicated — my mentor and my daughter. John Hartigan made me feel at home during my visits to the Yale statistics department in the early 1970's and encouraged my steps. Amie Wilkinson urged me to keep walking and taught me new steps. John and Amie share a remarkable intensity and an ability, rare among mathematicians, to explain to people like me what they do. This book is only a shadow of what they gave.

Chicago, Illinois Leland Wilkinson

Preface to Second Edition

The first edition of this book was a monograph. I intended to present a new, object-oriented way of thinking about statistical graphics. The second edition is a multigraph. I intend to supplement this presentation with a survey of ideas that are useful for understanding the meaning of statistical graphics. I have extended the grammar metaphor by calling the first part *Syntax*, and the second *Semantics*. Consequently, the second part of this book includes more of the work of others. Although it is not a survey, it covers the main themes that I consider essential to understanding how charts and graphs work.

Part 2 takes mathematical, psychological, and applied points of view. Quantitative graphics are meaningless unless they have mathematical foundations. They are solipsistic unless they follow experimentally tested psychological principles. And they are irrelevant unless they reveal patterns in real data so we can understand our world.

The first edition used "I" throughout. The second uses "we" because the visualization team I have been privileged to work with at SPSS has shaped my ideas as much as I have shaped theirs. Dan Rope, as I mentioned in the first edition, has been a master at developing these ideas; he has added management to his other talents without giving up coding. Graham Wills, in a fortunate circumstance, joined us shortly after the first edition of this book appeared; he brought unique energy (if you've met Graham you know what I mean) and extensive knowledge of graphics and statistics to our thinking. Andy Norton has been an old friend; he brought us a deep understanding of object-oriented design, Java, and database technology. And Roger Dubbs joined SPSS from a computer game company; it should be obvious what he brought us, in addition to a keen design sense and extensive knowledge of programming methodology. I should also mention Taylor Stockwell, Valeri McGuire, Will LaForest, and Fred Esch, who worked for Illumitek and eventually for SPSS; their enthusiasm and dedication remind me of the early days at SYSTAT in the 1980's.

Graham Wills contributed to the *Automation* chapter and developed the proofs in the *Algebra* chapter. He and Roger Dubbs wrote ViZml. Dan Rope contributed to the *Control* chapter and wrote the bulk of nViZn. Andy Norton contributed to the *Time* chapter and wrote Dancer and the GPL interpreter. Roger Dubbs contributed to the *How to Make a Pie* chapter and, with Graham, wrote ViZml. I also thank Andy for an especially close reading of the entire book; all the mistakes in this book are Andy's fault, not mine.

As we mention in the text, this book is about more than visualization. It is about communicating information. This perspective unites the era of computing with the eras of mapping, charts, and graphics from previous centuries. We have tried to reference the original sources for ideas whenever possible and we have tried to avoid referencing rediscoveries and reinterpretations that appear in different fields unless they contribute substantial new information. If our references look quaint sometimes, it is because they are original. As much as possible, we have tried to avoid reprising Stigler's law of eponomy ("No scientific discovery is named after its original discoverer").

Finally, I want to thank Jack Noonan for supporting an environment that has demonstrated that discovery and innovation are possible in a publicly–owned company.

Leland Wilkinson
Chicago, 2005

Contents

1

Introduction

Grammar gives language rules. The word stems from the Greek noun for letter or mark (*γράμμα*). And that derives from the Greek verb for writing (*γράφω*), which is the source of our English word **graph**. Grammar means, more generally, rules for art and science, as in the richly illustrated *The Grammar of Ornament* (Jones, 1856), and Karl Pearson's *The Grammar of Science* (Pearson, 1892).

Grammar has a technical meaning in linguistics. In the transformational theory of Chomsky (1956), a grammar is a formal system of rules for generating lawful statements in a language. Chomsky helped explain many surface characteristics of specific natural languages through a deep, universal structure. Chomsky's context-free grammar is the progenitor of modern computer language parsers.

Grammar makes language expressive. A language consisting of words and no grammar (*statement* = *word*) expresses only as many ideas as there are words. By specifying how words are combined in statements, a grammar expands a language's scope.

This book is about grammatical rules for creating perceivable graphs, or what we call **graphics**. The grammar of graphics takes us beyond a limited set of charts (words) to an almost unlimited world of graphical forms (statements). The rules of graphics grammar are sometimes mathematical and sometimes aesthetic. Mathematics provides symbolic tools for representing abstractions. Aesthetics, in the original Greek sense, offers principles for relating sensory attributes (color, shape, sound, etc.) to abstractions. In modern usage, aesthetics can also mean taste. This book is not about good taste, practice, or graphic design, however. There are many fine guides to creating good graphics (*e.g.*, Cleveland, 1985, 1995; Tufte, 1983, 1990, 1997; Kosslyn, 1994). This book focuses instead on rules for constructing graphs mathematically and then representing them as graphics aesthetically.

The title of this book also recalls Bertin's *Semiology of Graphics* (1967), the first and most influential structural theory of statistical graphics. Bertin's work has pervaded our thinking. Semiology deals with signs. Although Bertin put his signs on paper, his work applies as well to virtual space.

Some of the rules and graphics in this book may seem self-evident, especially to those who have never written a computer program. Programming a computer exposes contradictions in commonsense thinking, however. And programming a computer to draw graphs teaches most surely the ancient lesson that God is in the details.

1.1 Graphics Versus Charts

We often call graphics **charts** (from *χάρτης* or Latin *charta*, a leaf of paper or papyrus). There are pie charts, bar charts, line charts, and so on. This book shuns chart typologies. For one thing, charts are usually instances of much more general objects. Once we understand that a pie is a divided bar in polar coordinates, we can construct other polar graphics that are less well known. We will also come to realize why a histogram is not a bar chart and why many other graphics that look similar nevertheless have different grammars.

There is also a practical reason for shunning chart typology. If we endeavor to develop a charting instead of a graphing program, we will accomplish two things. First, we inevitably will offer fewer charts than people want. Second, our package will have no deep structure. Our computer program will be unnecessarily complex, because we will fail to reuse objects or routines that function similarly in different charts. And we will have no way to add new charts to our system without generating complex new code. Elegant design requires us to think about a theory of graphics, not charts.

A chart metaphor is especially popular in user interfaces. The typical interface for a charting program is a catalog of little icons of charts. This is easy to construct from information gathered in focus groups, surveys, competitive analysis, and user testing. Much more difficult is to understand what users intend to do with their data when making a graphic. Instead of taking this risk, most charting packages channel user requests into a rigid array of chart types. To atone for this lack of flexibility, they offer a kit of post-creation editing tools to return the image to what the user originally envisioned. They give the user an impression of having explored data rather than the experience.

If a chart view is restrictive, how do we impose structure on a graphic view? The concept of a graphic is so general that we need organizing principles to create instances of graphics. We may not want to put a pie chart in a catalog, but we need to give users some simple way to produce one. For that, we need methodologies based on object-oriented design.

1.2 Object-Oriented Design

Many of the insights in this book were stimulated by a recent development in applied computer science called **object-oriented analysis and design** (Meyer, 1988; Rumbaugh *et al.*, 1991; Booch, 1994). Object-oriented design (OOD) involves a plethora of techniques and principles that often baffle those trained in classical procedural programming languages and software design. Its methodology resembles a search for the objects that throw shadows on the wall of Plato's cave. Good objects are hard to find.

1.2.1 What is OOD?

OOD uses a variety of strategies for making software flexible and reusable:

- **Objects** are basic components of systems. They represent relatively autonomous agents that go about their business doing things useful for each other and for the general community of objects that comprise the system. The names of some of the most widely used objects in contemporary OOD systems express this utilitarian perspective: **Factory, Decorator, Facade, Proxy, Iterator, Observer, Visitor** (Gamma *et al.*, 1995). These objects do things that are aptly described by their names. A factory builds things. A decorator applies patterns to things. An observer looks for messages. A visitor roams and brings gifts.
- Objects communicate with each other through simple **messages**. These messages are distributed throughout the system. Because they may float freely throughout system, instead of being confined to the rigid protocols of classical programs, they resemble the communications within a living community.
- Objects are relatively stupid. They do a few things well, as do lobsters.
- Intelligence resides in the system, not in objects, because activities in concert have a life of their own that cannot be explained by separate, uncoordinated activities. For an OOD, as for life itself, the whole is more than the sum of its parts.
- Because objects are relatively stupid, they are also relatively simple and useful for a variety of purposes, even in new systems. Objects are often **reusable**, although this aspect of OOD has been oversold by some proponents.
- Because objects respond only to a few messages, and talk to other objects via simple messages of their own, what they do is **encapsulated**. Other objects have no idea how they work. And they don't care. They only need to know what to do with messages.
- Components of object-oriented systems are relatively **modular**. If parts of the system are discarded or malfunction, the rest of the system usually can continue to function.

- Objects can **inherit** attributes and behavior from other objects. This saves time and space in a well-organized system, because we can derive instances of things from more general classes of things.
- Objects are often **polymorphous**. That is, different objects can be induced to respond to the same message in different ways. Their responses may even be unanticipated by their designer, but in an elegant system their responses will not usually be harmful. Polymorphism also implies that objects don't care what type of data they process. They are flexible enough to return reasonable responses to all sorts of data. This includes the simplest response, which is not to respond. This kind of robustness is quite different from classical procedural systems which crash or cause other routines to crash when fed illegal or unanticipated data. Well-designed polymorphous objects are not perverse.
- OOD induces designers to **abstract**. Nate Kirby, an object-oriented programmer and designer at SPSS, has noted that bad programmers *ignore* details and bad designers *get lost in* details. To a designer, whenever a category or class of object seems fitting, it elicits thoughts of a more general category of which it is an instance. Because of this generalizing tendency, object-oriented systems are intrinsically **hierarchical**.

1.2.2 What is not OOD?

1.2.2.1 OOD is not a Language

OOD is not a programming language. Some languages, like Simula, Smalltalk, and Java, make it easy to implement objects and difficult to implement procedures. Others, like *C*++, enable development of both objects and procedures. And others, like *C*, Pascal, Ada, BASIC, Algol, Lisp, APL, and FORTRAN, make it difficult (but not impossible) to develop objects. Using a language that facilitates object specifications is no guarantee that a system will be object-oriented, however. Indeed, some commercial *C*++ graphics and numerical libraries are translations of older FORTRAN and *C* procedures. These older routines are disguised in **wrappers** whose names are designed to make them appear to be objects. By the definitions in this book, a Java library with classes that are called PieChartModel, PieChartViewer, and PieChartController, is no more object-oriented than a FORTRAN program with three subroutines of the same names.

1.2.2.2 OOD is not a GUI

OOD has been associated with the development of modern graphical user interfaces (GUIs) because it is easiest to instantiate the behavior of an icon or graphic control through well-defined objects. OODs can be implemented in scripting or command-based systems, however, and GUIs with behavior indistinguishable from object-driven ones can be programmed through direct calls to an operating system tool kit. It is extremely difficult (though not impossible) to infer the design of a system through its behavior.

1.2.2.3 OOD is not an Interactive System

While a modern desktop system tends to allow a user to interact with its components in a flexible manner, this has nothing to do with OOD. For example, Data Desk, the most interactive commercial statistics package (Velleman, 1998), is not based on OOD. This is not necessarily a drawback. Indeed, it can be an advantage. Data Desk's design has served the package well because it was conceived with a close attention to the capabilities of the operating systems under which it resides. The extent to which an OOD system is interactive depends on how controller classes are implemented. Without user controls, OOD systems may be relatively autonomous.

1.2.3 Why OOD?

OOD has failed to realize some of the more extravagant claims of its proponents in the last decade. In our experience, OOD has not brought increased reliability to the development process. Reliability of a system depends more on clean design and intensive testing early in the development process than on a particular design method. Nor has OOD given us increased portability of programs. Operating systems have evolved more rapidly in the last few years than ever before. Manufacturers' promises that their object **frameworks** (the objects programmers use for routine tasks) would remain immutable, or at least upward-compatible, have not been kept. Nor has OOD given us more rapid and responsive software. It is hard to beat assembly language or *C* programs at execution time. While there are exceptions, it is generally true that the most attractive elements of OOD — encapsulation and polymorphism — usually penalize performance. Nor has OOD given us more rapid development schedules. Indeed, OOD can retard development because objects are often more difficult to conceive properly, and modifying pre-existing objects is more difficult than changing procedures. Despite the marketing hype for OOD, it is hard to beat the development cycles realized in some of the APL and Lisp systems of two decades ago.

Still, an OOD paradigm is the best way to think about graphics. APL is ideally suited to developing small matrix algebra functions because it is a matrix functional language. It is unbeatable for prototyping numerical methods. Lisp is ideal for manipulating lists of words and symbols because it is a list processing language. OOD, by contrast, is a natural framework for thinking about graphics because graphics *are* objects (Hurley and Oldford, 1991). We can see and touch these objects. Having a language that naturally implements these objects is an added benefit. If none of this work appeared on a computer, however, we would still find the effort worthwhile. Defining objects helps organize thoughts.

1.3 An Object-Oriented Graphics System

A **graph** is a set of points. A mathematical graph cannot be seen. It is an abstraction. A **graphic**, however, is a physical representation of a graph. This representation is accomplished by realizing graphs with **aesthetic** attributes such as *size* or *color*.

An object-oriented graphics system requires explicit definitions for these realizations and rules for relating them to data and for organizing their behavior in a computational environment. If we are lucky, this graphics system should have generality, yet will rest on a few simple objects and rules. This book is an attempt to reveal the richness of such a system.

From the OOD perspective, graphics are collections of objects. If the messages between these objects follow a simple grammar, then they will behave consistently and flexibly. To introduce this idea, we will focus on three stages of graphic creation:

1) Specification
2) Assembly
3) Display

After introducing these stages, we will show how they work in an example.

1.3.1 Specification

Specification involves the translation of user actions into a formal language. The user may not be aware of this language, but it is required for an automated system to understand the graphic request. Another way of defining specification is to say that it is the deep grammar of a graphic. A graphic, unlike a picture, has a highly organized and constrained set of rules. A picture, of course, has its own rules, especially real pictures such as photographs and videos (Biederman, 1981). Nevertheless, an artist is privileged to bend the rules to make a point (Bosch, Dali, or Picasso, obviously, but also Rembrandt, Cezanne, or Close). And a special-effects technician may use tricks to make us think that a video or virtual scene is more real than what we observe in our daily lives. Not so with graphics. We have only a few rules and tools. We cannot change the location of a point or the color of an object (assuming these are data-representing attributes) without lying about our data and violating the purpose of the statistical graphic — to represent data accurately and appropriately. Consequently, the core of a graphics system must rest on specification.

Statistical graphic specifications are expressed in six statements:

1) DATA: a set of data operations that create variables from datasets,
2) TRANS: variable transformations (*e.g., rank*),
3) SCALE: scale transformations (*e.g.*, *log*),
4) COORD: a coordinate system (*e.g.*, *polar*),
5) ELEMENT: graphs (*e.g.*, *points*) and their aesthetic attributes (*e.g.*, *color*),
6) GUIDE: one or more guides (*axes*, *legends*, etc.).

In most of the figures in this book, we will add a syntactical specification of the graphic in order to make the definition explicit. An earlier version of this specification language (Wilkinson, 1996) incorporated all aspects in a single algebra. The notation was unwieldy and idiosyncratic, however, so we have separated them into components. These components link data to objects and specify a scene containing those objects.

1.3.2 Assembly

A scene and its description are different. In order to portray a scene, we must coordinate its geometry, layout, and aesthetics in order to render it accurately. A statistical graphics computer program must be able to assemble a graphical scene from a specification in the same manner as a drawing or modeling program puts together a realistic scene from specification components. This book is more about specification than scene assembly, but it is important to think about assembly while learning about specification so that we do not confuse surface features with deep structures. How we build a scene from a specification affects how the result behaves. A scene can be dynamic or static, linked to external data or isolated, modifiable or immutable, depending on how we assemble it.

1.3.3 Display

For us to perceive it, a graph must be rendered using aesthetic attributes and a display system (*e.g.*, paper, video, hologram). Contemporary operating systems provide access to rendering methods, preferably through well-defined, abstract interfaces. Production graphics require little in this area other than basic drawing capabilities. Dynamic graphics and scientific visualization, by contrast, require sophisticated designs to enable **brushing**, **drill-down**, **zooming**, **linking**, and other operations relating data to graphics. Becker and Cleveland (1987), Cleveland and McGill (1988), Cook and Weisberg (1994), and Swayne *et al.* (1998) introduce these issues. More recently, virtual reality displays and immersive environments have expanded the available aesthetics to touch and sound.

1.4 An Example

Figure 1.1 shows a graphic of 1990 death rates against birth rates per 100,000 population for 27 selected countries in a UN databank. The plot contains two graphic elements: a *point* (collection of points) whose labels show country names, and a *contour* of a kernel density estimate (Silverman, 1986) that represents the concentration of the countries. We have also included three *guides* that help us understand the graphics. The first is a general geometric object called a *form* that is, in this instance, a line delineating zero population growth. Countries to the left of this line tend to lose population, and countries to the right tend to gain. The other two are guides that delineate axes for the represented space. Other examples of guides are legends and titles.

The graphic is striking because it reveals clearly the patterns of explosive population growth. The density contours show two concentrations of countries. One, to the left, has relatively lower death rates and small-to-moderate birth rates. The second, in the upper right, has high death rates and extraordinarily high birth rates. The latter countries tend to be developing. We have kept the sample of countries small so that we can read the country labels. Adding other countries from the database does not change the overall picture.

```
ELEMENT: point(position(birth*death), size(0), label(country))
ELEMENT: contour(position(
            smooth.density.kernel.epanechnikov.joint(birth*death)),
            color.hue())
GUIDE: form.line(position((0,0),(30,30)), label("Zero Population Growth"))
GUIDE: axis(dim(1), label("Birth Rate"))
GUIDE: axis(dim(2), label("Death Rate"))
```

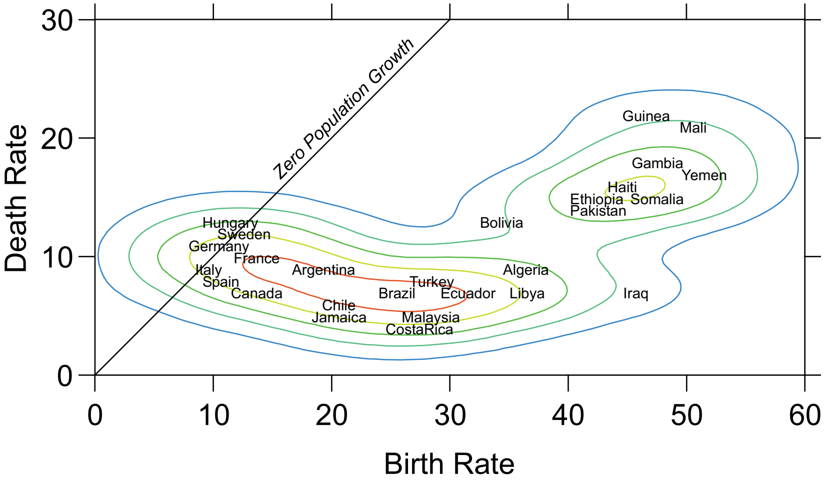


Figure 1.1 *Plot of death rates against birth rates for selected countries*

1.4.1 Specification

The specification above the figure makes use of only ELEMENT, and GUIDE components. The data are assumed to have been organized in a cases-by-variables matrix, there are no transformations, and the coordinates are rectangular, so we can assume default settings. The first two lines show the two graphic elements in the plot: a *point*, and a *contour*. Both graphic elements are positioned by the variables birth and death, which are scaled as percents. The frame in which they are embedded is determined by the algebraic expression birth*death. The *point* graphic actually does not show because its *size* attribute is set to zero. Normally, we would see symbols (perhaps dots) for each country. Instead, we have country labels for each point in the cloud set to the values of the variable country in the data. The *contour* graphic represents the density of countries in different regions of the frame. Where there are more countries near each other, the density contour is higher. These contours are computed by a kernel smoothing algorithm that we will discuss further in Chapter 7. The dot notation *smooth.density.kernel.epanechnikov.joint*() means that Epanechnikov kernel smoothing is a member of a hierarchy of density smoothing methods. Different contours are given a *color.hue* aesthetic based on the kernel density values at the level of each contour.

The guides consist of the line, the axes, and their corresponding scales and labels. The*form* guide is displayed with a line from (0,0) to (30,30) in the metric anchored by both rate measures. This line is labeled with an associated text string ("Zero Population Growth"). In most of the figures, we will omit GUIDE specifications to keep the description simpler.

1.4.2 Assembly

Assembling a scene from a specification requires a variety of structures in order to index and link components with each other. One of the structures we can use is a network or a tree. Figure 1.2 shows a tree for the graphic in Figure 1.1. Each node in the tree, shown with a box, represents a type of object in Figure 1.1. Each branch in the tree, shown with an arrow, represents a type of relation between objects. The triangular-headed arrows represent “is a” relations. The diamond-headed arrows represent “has a” relations.

"Is a" relations provide a way to derive common functionality from one class. The result of such relations is **inheritance**. For example, an Axis is a Guide in the same sense that a piano is a keyboard instrument. Any aspect of a piano that has to do with being a keyboard instrument (having a sound produced by pressing one or more keys, for example) is inherited by other keyboard instruments. Any aspect of a piano that does not have to do with being a keyboard instrument (having hammers, for example) is not necessarily shared by other keyboard instruments (harpsichords pluck, pianos strike). If we derive common functionality from a general class, then a subclass can inherit skills from its parent without having to do any extra work. Tasks related to having keys, for example, can be defined and implemented in one Keyboard class. Tasks relating to guiding, such as relating numeric values to text strings, can be implemented in one Guide class. In a similar manner, the Contour and Point classes are both Graphs. They inherit capabilities that enable them to represent themselves in a frame.

"Has a" relations provide a way to group related attributes and functions under a class. The result of such relations is **aggregation**. For example, an Axis has a Scale, a Rule, and a Label in the same sense that a piano has a keyboard, strings, and pedals. The aggregation of these features and functions is what helps us distinguish a piano from other objects. In a similar manner, a Chart has a Frame, one or more Guides, and one or more Graphs. If we implement aggregation well, our objects will be small and efficient and our computer code will be comprehensible and modular.

.

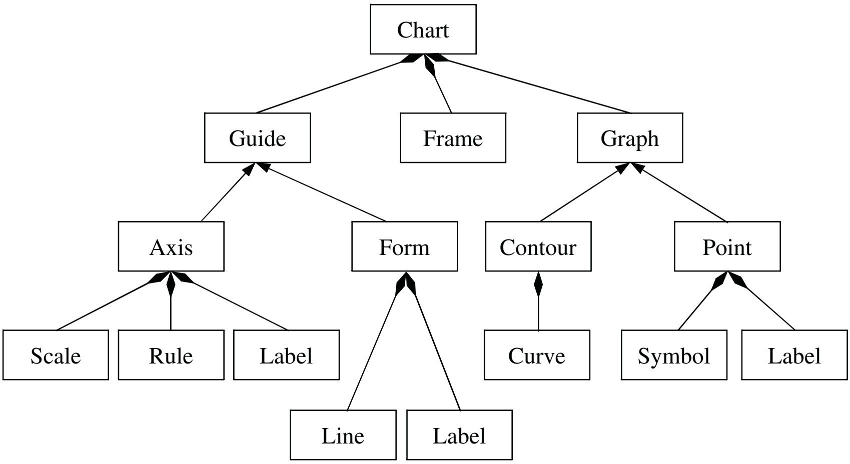


Figure 1.2 *Design tree for chart in Figure 1.1*

1.4.3 Display

The tree in Figure 1.2, together with a set of rendering tools (symbols, polylines, polygons) and layout designer, provides a structured environment in which each object in a graphic can draw itself. There is no single agent needed to figure out all the rules for drawing each object.

A grammar of graphics facilitates coordinated activity in a set of relatively autonomous components. This grammar enables us to develop a system in which adding a graphic to a frame (say, a *surface*) requires no adjustments or changes in definitions other than the simple message “add this graphic.” Similarly, we can remove graphics, transform scales, permute attributes, and make other alterations without redefining the basic structure.

1.4.4 Revision

Revision means, literally, to see again. For graphics, it implies that we want to change, query, and explore without having to go through all the work of specifying and creating a new graphic. By carefully separating the process of graph creation into hierarchical components, we enable a flexible environment that offers new views without recalculating every step in the system. And we can link controllers to any component or property in the system to provide direct manipulation of data, variables, frames, or rendering. If more than one graphic depends on the same sub-component, then they are linked as well.

Figure 1.3 shows an example. Even though the graphic looks different, the positional frame is the same as in Figure 1.1. We have omitted *point* and *form* and we have replaced *contour* with *polygon* to represent a kernel density. The *hue* of each polygon comes from the estimated density of the countries at that location. We have omitted the guides from this and subsequent specifications to save space. These will be discussed in more detail in Chapter 12.

ELEMENT: *polygon*(*position*(
smooth.density.kernel.epanechnikov.joint(birth*death)), *color.hue*())

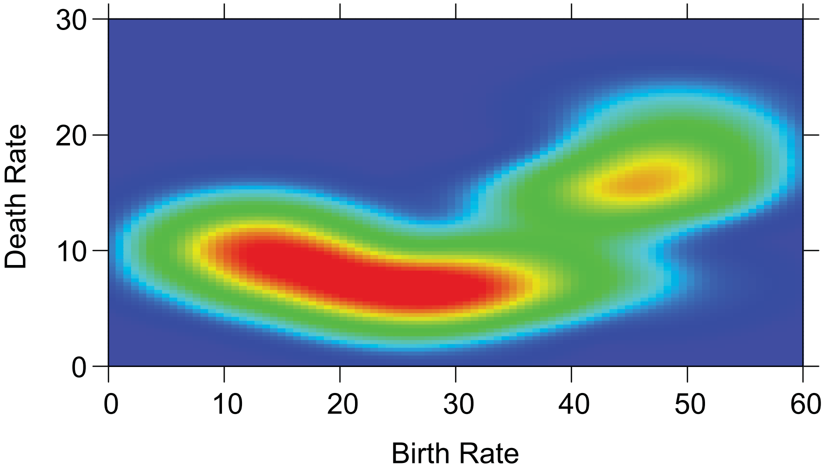


Figure 1.3 *Kernel density of death and birth rates*

Figure 1.4 adds a new variable to the specification of the point graphic in Figure 1.1. This variable, military, is the annual military expenditures per capita normalized as U.S. dollar equivalents. We are using this variable to determine the size of each symbol, so that size of plotting symbol represents military expenditure for each country.

ELEMENT: *contour*(*position*(
smooth.density.kernel.epanechnikov.joint(birth*death), *color.hue*())
ELEMENT: *point*(*position*(birth*death), *size*(military))

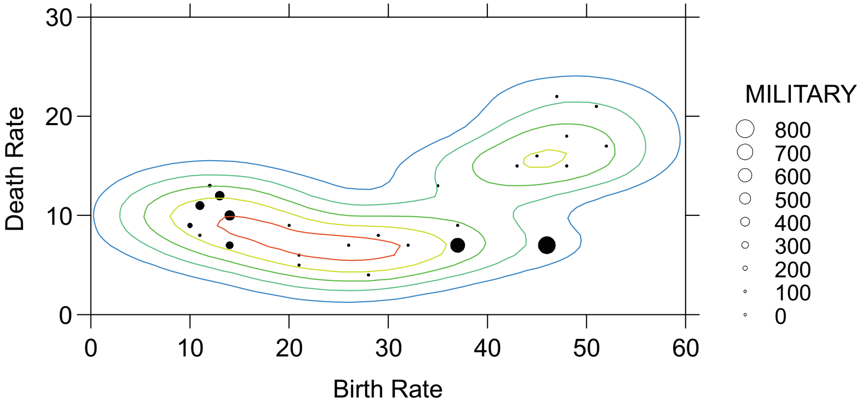


Figure 1.4 *Military expenditures vs. birth and death rates*

Figure 1.4 conveys a rather troubling message. The highest relative military expenditures are often in the most rapidly growing, politically unstable countries. Still, this statistic conceals the *absolute* level of military expenditures. The highest absolute levels of military spending are in advanced nations with larger populations.

Finally, Figure 1.5 shows a map of the difference between death and birth for the selected countries. The goal is to reveal the location of countries that are growing rapidly in population. We have used size of the plotting symbol to represent the magnitude of the difference (the few small negative differences have been set to zero).

There are two sets of positioning variables that define the frame. The first, lat and lon, represent the location of the countries measured. These are used to plot the circles showing death-birth differences. The second, latitude and longitude, are used to denote the locations on the map that anchor the boundaries of the polygons defining the continental borders. The polygons for the map are read from a shape file containing their vertices. The *map* data function handles the translation of polygon IDs and polygon vertices in the file to a splitter variable and longitude and latitude vertices for the *polygon* geometric function. The *point* and *polygon* graphics both use a *position* attribute to control which variables determine their position. The *position* dimensions are transformed with a *mercator* cartographic projection. The axes and grid lines respond to the projection, as well as the graphics in the frame. We will examine in Chapter 9 map projections that are better suited for representing the countries data.

```
DATA: longitude, latitude = map(source("World"))
TRANS: bd = max(birth-death, 0)
COORD: project.mercator()
ELEMENT: point(position(lon*lat), size(bd), color(color.red))
ELEMENT: polygon(position(longitude*latitude))
```

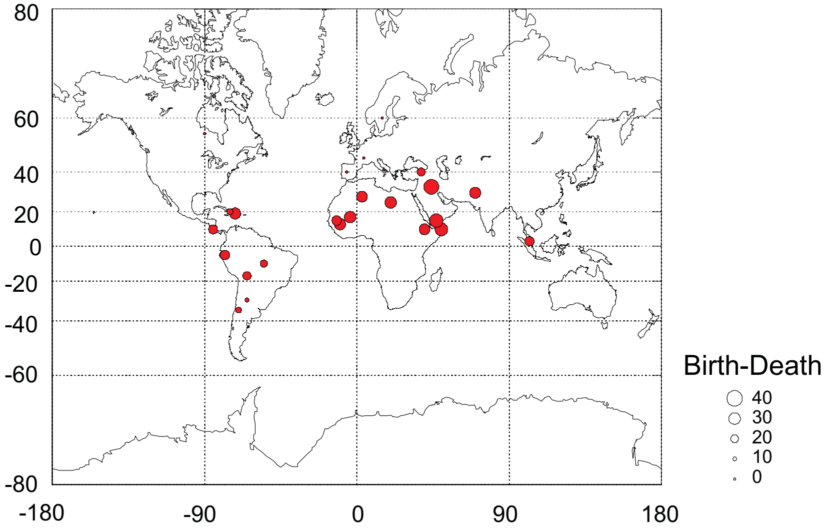


Figure 1.5 *Excess birth (vs. death) rates in selected countries*

1.5 What This Book Is Not

Because this book spans the fields of computer science, geography, statistics, and graphic design, it is likely to be misunderstood by specialists in different areas for different reasons. We cannot anticipate all of these reasons, but here are a likely few.

1.5.1 Not a Command Language

A cursory reading of this book might lead one to conclude that its purpose is to present a new graphics scripting language. Indeed, each figure is accompanied by a specification that resembles a command language. One impetus for this conclusion would be occasional similarities to existing quantitative graphics languages such as those in Mathematica$^{\circledR}$, SYSTAT$^{\circledR}$, S-Plus$^{\circledR}$, and SAS-Graph$^{\circledR}$. These packages can produce a large variety of statistical graphics because they evolved to fulfill the needs of statisticians for sophisticated and flexible technical graphics. They were not developed with a comprehensive theory of graphics in mind, however. Often, their constructs have similar regularities because of the constraints of the graphics world. We owe a debt to all of these systems for being able to produce unusual graphics with them and to discover the common implicit structures. To appreciate the real difference between this and command-based systems, however, see Chapter 17.

Another conclusion one might draw after a brief glance is that this system is designed to be a static specification language instead of a dynamic, exploratory system. On the contrary, by regularizing the rules for graph behavior in graphics frames, it provides a richer environment for dynamic and exploratory graphics. This is especially true for paneled graphics, which are either avoided altogether in most dynamic graphics systems or hard-wired to specific data structures. In fact, the primary focus of our interest is in designing a system that is flexible enough to change state without re-specification. A naive approach to implementing such a system would be to create commands from user gestures, feed those commands to an interpreter, and then display the results. This method, employed in some existing packages, would indeed constrain it to be a static system. There is nothing in the theory presented in this book, however, to suggest that this is the best or even most appropriate implementation method.

1.5.2 Not a Taxonomy

Taxonomies are useful to scientists when they lead to new theory or stimulate insights into a problem that previous theorizing might conceal. Classification for its own sake, however, is as unproductive in design as it is in science. In design, objects are only as useful as the system they support. And the test of a design is its ability to handle scenarios that include surprises, exceptions, and strategic reversals. This book includes a few classifications, but they are in the service of developing objects that are flexible and powerful in a coherent system. Other classifications of the same problem domain are possible, but many of them would not lead to a parsimonious and general system. Some classifications have been attempted based on cluster analyses of ordinary users' visual judgments of similarities among real statistical graphics (*e.g.*, Lohse *et al.*, 1994). This approach may be useful for developing interfaces but contributes nothing to a deeper understanding of graphs. Customary usage and standards can blind us to the diversity of the graphics domain; a formal system can liberate us from conventional restrictions.

1.5.3 Not a Drafting Package

This system was not designed to produce any graphic imaginable. Indeed, the motivation is almost the opposite: to develop a closed system and *then* to examine whether it can produce both popular and esoteric graphics. We have tried to avoid adding functions, graphs, or operators that do not work independently across the system. There are doubtless many statistical graphics the system in this book cannot completely specify. We can add many different graphs, transformations, types of axes, annotations, etc., but there are two limitations we will always face with a formal system.

The first limitation applies to any free-hand drawing. Clearly, we cannot expect to use a formal data-driven system to produce sketches on cocktail napkins. It will always be possible to find creative designs that are not formally linked to data. The province of drafting systems is computer-assisted design (CAD) and desktop publishing (DTP). Those areas have their own rules driven more by the physical appearance of real objects than by the theoretical constructs of functional and data analysis.

The second limitation derives from the syntactical structure of the system itself. The operators in this system are capable, as we shall see, of producing a surprisingly wide variety of graphics, perhaps more than any other formal system or computer graphing program. Nevertheless, one can imagine certain structures that may not be modeled by a language with the operators presented here. It is, after all, a closed system. This graphics system was designed with surveys of statistical graphics usage (*e.g.*, Fienberg, 1979) and existing commercial and scientific graphics software in mind. Nevertheless, one cannot over-estimate the inventiveness and ingenuity of real users when they display their ideas.

1.5.4 *Not a Book of Virtues*

This system is capable of producing some hideous graphics. There is nothing in its design to prevent its misuse. We will occasionally point out some of these instances (*e.g.*, Figure 9.25). That the system *can* produce such graphics is simply a consequence of its basis on the mathematical rules that determine the meaning of graphs, rather than on the *ad hoc* rules we sometimes use to produce graphics. These rules are not based on personal preferences but rather on the mathematics and perceptual dimensions underlying the graphics we draw in practice. These rules are just as capable of producing graphics for *USA Today* as for *Scientific American*.

This system cannot produce a meaningless graphic, however. This is a strong claim, vulnerable to a single counter-example. It is a claim based on the formal rules of the system, however, not on the evaluation of specific graphics it may produce. This is an essential difference between the approach in this book and in other texts on statistical graphics and visualization. We are much less interested in designing or evaluating specific graphics than in understanding the rules that produced them. Unless one specifies those rules explicitly, one cannot begin to claim that a particular graphic is meaningless or not.

We also cannot disagree strongly enough with statements about the dangers of putting powerful tools in the hands of novices. Computer algebra, statistics, and graphics systems provide plenty of rope for novices to hang themselves and may even help to inhibit the learning of essential skills needed by researchers. The obvious problems caused by this situation do not justify blunting our tools, however. They require better education in the imaginative and disciplined use of these tools. And they call for more attention to the way powerful and sophisticated tools are presented to novice users.

1.5.5 Not a Heuristic System

The title of this book is *The Grammar of Graphics*, not *A Grammar of Graphics*. While heuristic strategies are fun, pragmatic, and often remarkably adaptive, there is seldom reason to pursue them unless formal systems are shown to be deficient or elusive. It is sometimes fashionable to apply heuristics to well-defined problems in the name of artificial intelligence. If we take such an approach, it is our burden to prove that a heuristic system can accomplish everything a formal system can plus something more. Until we define the capabilities of a formal system, there is no way to make such a comparison.

Defining a formal system has practical implications in this field. Until recently, graphics were drawn by hand to represent mathematical, statistical, and geometric relations. Computer graphics programs, particularly scientific and mathematical plotting packages, have made this task much easier but they have not altered its *ad hoc* aspect. Nor have statistical and mathematical packages that generate more complex graphics contributed to our understanding of how they are created. Each new graphic in these programs was developed by an engineer who knew many of the rules in this book instinctively and applied them to a specific instance.

Now that **data mining** is popular, we need to be able to construct graphics systematically in order to handle more complex multivariate environments. Unfortunately, the sophistication of data mining algorithms far exceeds the graphical methods used in their displays. Most data mining systems still rely on pie, line, and bar charts of slices of **data cubes** (multi-way aggregations of a subset of a database). These charts fail to reveal the relationships among the entities they represent because they have no deep grammar for generating them. They are simply hard-wired to facets of the data cube. If we **drill through** the cube to view a different slice of the data, we still get a simple pie chart. A similar hard-wiring exists in displays from tree classifiers, neural networks, and other algorithms.

The remarkable consequence of building a closed formal system is that, while it solves more complex applied problems, it can appear more adaptive to the user. Paradoxically, closed systems often behave more “openly” than open systems. We should not confuse heuristics with flexibility. In the end, this book rests on what is perhaps an extreme position, but one we share with Jacques Bertin: designing and producing statistical graphics is not an art.

1.5.6 Not a Geographic Information System

This book includes several maps (*e.g.*, Figure 1.5). That might lead some readers to conclude that we regard it as the framework for a geographical information system (GIS). Indeed, we adopted from geographers some basic parts of this system, such as projections, layering, and aesthetic attributes (graphic variables). We believe that many statisticians interested in graphics have not given enough attention to the work of geographers. This situation has been changing recently, thanks to efforts of statisticians such as Daniel Carr and Linda Pickle, as well as geographers such as Mark Monmonier, Waldo Tobler, and Alan MacEachren.

The system in this book is not a model for a GIS, however, because geography and statistics differ in a crucial respect. Geography is anchored in real space-time and statistics in abstract dimensions. This is a distinction along a continuum rather than a sharp break; after all, there is a whole field called spatial statistics (Cressie, 1991). But this difference in focus clearly means that a system optimized to handle geography will not be graceful when dealing with statistical graphics such as Figure 11.15, and the system in this book would not do well if asked to provide a real-time tour through a geographical scene. There are many other consequences of this difference in focus. Geographers have developed topological algebras for scene analysis (*e.g.*, Egenhofer *et al.*, 1991), whereas we have employed a design algebra to model factorial structures. Geographers are concerned with iconography; we are concerned with relations.

Some geographers might disagree with our real-abstract distinction here. There is no question that the capabilities of a GIS can prove invaluable in visualizing abstract data. As Pinker (1996) has said, statistical graphics are often most effective when they exploit mental models that evolved as humans struggled to survive in a competitive world. But that brings us to our next point.

1.5.7 Not a Visualization System

This book includes some visualizations (*e.g.*, Figure 9.55). Scientific visualization uses realistic solid modeling and rendering techniques to represent real and abstract objects. We have taken advantage of some methods developed in the visualization literature. A visualization data-flow model is used for the backbone of this system, for example. And as with GIS, there is some cross-fertilization with statistics. Statisticians such as Dianne Cook, Jürgen Symanzik, and Edward Wegman (*e.g.*, Symanzik *et al.*, 1997) have employed immersive visualization technology developed by computer scientists such as Carolina Cruz-Neira and Thomas DeFanti (Cruz-Neira *et al.*, 1993) to display data.

We could define scientific visualization broadly to include GIS and statistical graphics. This would, we believe, vitiate its meaning. A better way to understand the differences between visualization and statistical graphics would be to compare visualization programs like PV~Wave® and Data Visualizer® to statistical graphics packages like SYSTAT® or S-Plus®. Going to extremes, we could even use a CAD-CAM engineering or an illustration package to do statistical graphics. Because we could does not mean we should.

1.6 Background

The scope of this book precludes an historical review of the field of statistical graphics. When designing a system, however, it is crucial to keep in mind the diverse and long-standing history of graphics and charts. Collins (1993) classifies historical trends in visualization, reminding us that the display methods used in modern computer visualization systems are centuries old. His illustrations, some dating to the 12th century, are especially informative. Collins shows that the principal contribution of the enormous recent literature on scientific visualization is its application to non-physical data rather than the displays themselves. Funkhouser (1937), Tilling (1975), Beniger and Robyn (1978), Fienberg (1979), Robinson (1982), Stigler (1983), Tufte (1983, 1990, 1997), and Wainer (1997) offer additional material on the history of statistical graphics that supports Collins' argument.

Others have investigated statistical graphics from a theoretical viewpoint, often providing ideas that are used in this book. Bertin (1967, 1977) is the pioneer in the area of modern graphic classification and design; his work underlies almost all of Chapter 10. Mackinlay (1986) extended Bertin's ideas to develop a system that intelligently created graphics from relational data. Cleveland (1985) helped establish a framework for understanding the role of graphical elements in displays. Pinker (1990) proposed an information-processing model of graphical reading. Brodlie (1993) integrated concepts underlying both scientific visualization and statistical graphics. MacEachren (1995) extended Bertin's work by following systematic psychological and design principles. And Roth *et al.* (1995) developed a graphical system for querying and manipulating data. Their work, independent of ours and based on a different foundation, is nevertheless close in spirit to the content of this book.

1.7 Sequel

The remainder of this book consists of two parts. Part 1 lays out the *syntactic* foundations of the grammar of graphics; it covers most of the first edition of this book. Part 2 is devoted to the *semantic* foundations of graphics; it covers concepts from mathematics, psychology, and statistics that are fundamental to understanding what graphics and visualizations mean. Part 2 also has several chapters that summarize research in the area of graphical analysis and production.

Part 1 is summarized in Chapter 2, particularly in Figure 2.2. We begin by defining concepts through the task of constructing a pie graphic from data. The subsequent chapters proceed through the components of Figure 2.2, from left to right. The title of each chapter is a major component. Chapter 3 covers data and datasets, including functions for organizing data. Chapter 4 covers variables and variable sets, the entities by which we define graphs. The next six chapters — *Algebra*, *Scales*, *Statistics*, *Geometry*, *Coordinates*, and *Aesthetics* — form the heart of the system. These are the six components that we assemble in various combinations to construct a huge variety of graphics. *Algebra* comprises the operations that allow us to combine variables and specify dimensions of graphs. *Scales* involves the representation of variables on measured dimensions. *Statistics* covers the functions that allow graphs to change their appearance and representation schemes. *Geometry* covers the creation of geometric graphs from variables. *Coordinates* covers coordinate systems, from polar coordinates to more complex map projections and general transformations. Finally, *Aesthetics* covers the sensory attributes used to represent graphics. The remaining chapters in Part 1 involve additional aspects of graphics grammar. Chapter 11 covers facets, the coordinates that enable us to construct graphics of tables and tables of graphics. Chapter 12 deals with guides, such as axes and legends.

Part 2 begins with three chapters that summarize issues involved in graphing space, time, and uncertainty. The following chapter, titled *Analysis*, covers methods for preparing data for graphing. Some methods presented in this chapter are statistical in nature, but each is peculiarly constrained by the requirements of visualizing the results. The chapter titled *Control* covers methods for creating and exploring graphics interactively. And, by contrast, the chapter titled *Automation* covers methods for doing these things programmatically, without interaction. The chapter titled *Reader* inverts the problems covered elsewhere in this book; given a graphic, how do we read it? Finally, Part 2 concludes with a detailed analysis of two beautiful graphics in order to show how syntax and semantics are linked in the process of constructing and understanding graphics.

The titles of each chapter in Part 1 are (with the exception of the next) a single word. These words designate components that contain objects and behaviors more like each other than like those in other components. Not every one of these components is named in Figure 2.2 because some are subsystems or stages of the ones named. Nevertheless, it is reasonable to regard these chapters as cumulative, so it would not be easy to jump in at randomly selected locations. You may want to look ahead to see where applications fit into the system, but much of the terminology in later chapters depends on the definitions given earlier in the book.

Part I

Syntax

Syntax is derived from the Greek word *σύνταξις*, which means an orderly arrangement (as in a lining up of soldiers). In classical grammar, syntax is the set of rules for combining words and sentences.

The first part of this book is about the syntax needed to create charts. Nearly every chart in this part is accompanied by a grammar-of-graphics specification. These specifications are subsets of the general graphics language that we discuss later in Chapter 18. In Part 2, we will discuss the *semantics* of graphics, or the meanings of the representative symbols and arrangements we use to display information. Part 1 presents a unique system for generating charts through a parsimonious syntax. Part 2 discusses the meanings of these charts through concepts employed in a variety of fields.

2

How To Make a Pie

A pie chart is perhaps the most ubiquitous of modern graphics. It has been reviled by statisticians (unjustifiably) and adored by managers (unjustifiably). It may be the most concrete chart, in the sense that it *is* a pie. A five-year-old can look at a slice and be a fairly good judge of proportion. (To prevent bias, give the child the knife and someone else the first choice of slices.) The pie is so popular nowadays that graphical operating systems include a primitive function for drawing a pie slice.

Figure 2.1 shows a simple **data flow** model of how to make a pie. Data values flow from the data **store** called Source through a Make-a-pie **process** that creates a graphic, which is then sent to an **actor** called Renderer. The details of the Renderer are ignored here. It could render to any one of many graphics formats, or render a text description of the graphic, or even render a sonification.

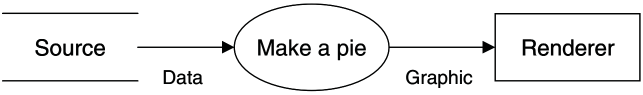


Figure 2.1 *How to make a pie*

Foley *et al.* (1993) discuss graphics pipelines, and Upson *et al.* (1989) discuss how pipeline architecture is used in scientific visualization. This pipeline could be (and has been) written as a single function. Nothing could be simpler. However, simple things usually deserve deeper examination. What is the format of the data being passed in? How are the pie wedges to be colored? What variables should we use to label the pie? Do we want to have a table of pie charts by subgroup? Once we have a pie function that has options to account for these questions, we then have to consider the bar chart, the scatterplot, the Pareto chart, and so on *ad infinitum*. Each chart type has to answer these questions and more.

The grammar of graphics was developed with all these questions in mind in order to produce a flexible system that can create a rich variety of charts as simply as possible, without duplication of methods. It is also extensible, which means that processes can be added easily to create new kinds of charts.

Despite the apparent simplicity of a pie, making one invokes almost every aspect of the grammar. Figure 2.2 shows a data-flow diagram for how a chart is constructed under this system. Figure 2.2 is simply a refinement of Figure 2.1. The internal processes of *make a pie* are shown in more detail. These internal processes constitute the syntax of the grammar of graphics.

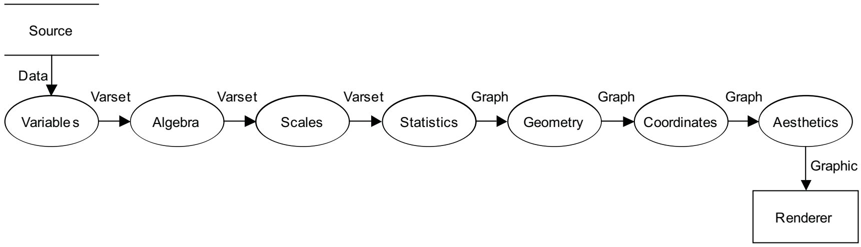


Figure 2.2 *From data to graphic*

Mixing and matching the available processes at each step creates a wide variety of charts with a minimum of effort. Some charts will be as simple as the pie. Others will be more complex, as for instance the map of Napoleon's march to Moscow and associated temperature graphic in Chapter 20. If we learn how to make a pie, we can create almost any statistical graphic. We will first present the general recipe for making a graphic and then we will go step-by-step through the process of making a pie, pausing occasionally for definitions.

Figure 2.2 is only one part of the design of the system. As a data flow diagram, it simply shows what the stages are, how they must be ordered, and what data are required along the way. It says very little about the actual implementation of the system. For example, it could be implemented as:

- A procedural library in which the various processes are procedures that are assembled in a main program loop for each chart.
- A functional program in which each process is a function and the actual graphing function itself is computed and then executed.
- An object-oriented program in which each of the processes is an object with its own data and behavior.
- A path in a graph model through which an application pushes data.
- A path in a graph model through which the renderer pulls data.

It is important to keep in mind that Figure 2.2 is only one slice through the architecture of a system that produces graphics. As Booch (1994) indicates, an object-oriented system can be represented by a series of "orthogonal" object diagrams, each of which provides a different functional view of the system. Figure 2.2 gives us the ingredients and the dependencies among them, but it does not tell us how to assemble the ingredients.

A premise of this book, however, is that we cannot change the ordering of stages in the pipeline. We cannot compute scales after we do statistics, for example. And, obviously, we cannot apply aesthetics before we compute geometric objects that can be colored and textured. In one sense, the data flow model is a truism. Data must be processed before they can be plotted. Many have used some type of data flow to illustrate how visualization systems work. Few have identified the necessary sub-sequences these systems must follow.

In the first edition of this book, this chapter showed how to make one pie. We will make two pies this time. Making two illustrates graphics algebra and other components of the pipeline more completely. Before making our pies, however, we will provide some elementary definitions of terms so that we can assemble ingredients.

2.1 Definitions

The following definitions are fairly standard. Because words like *relation* are used differently in various applications in mathematics, computer science, and statistics, we have tried to give them a common notation and structure wherever possible. This means that our notation may differ from that used in some specialized fields. Choosing a level of abstraction and a level of detail is not easy. We have tried to avoid abstraction when it does not suit our purposes and we have included terms only when they are needed for clarity.

2.1.1 Sets

A **set** is a collection of unique objects, which we denote by a capital letter (*e.g.*, *X*). An object in a set is called an **element** or a **member** of the set. We denote an element by a lower-case letter (*e.g.*, *x*), and state that "*x* is an element of *X*" with the notation $x \in X$. We delimit the elements in a set with braces, e.g., $X = \{a, b, c\}$. We define the **complement** of a set, denoted $\bar{X}$, as the set of all elements not in X.

The **null set** ($\varnothing$) is an **empty set**, or a set with no elements. We denote the set of real numbers by ***R***, the integers by ***Z***, and natural numbers (integers greater than zero) by ***N***. We use the notation $A = \{a : a \in \boldsymbol{R}, a > 0\}$ to denote the statement "*A* is the set of elements *a* such that each *a* is a member of the set of real numbers and each *a* is greater than zero." If every element of a set *A* is also an element of a set *B*, then *A* is a **subset** of *B*, denoted $A \subseteq B$. If *A* is a subset of *B*, but there is at least one element of *B* that is not in *A*, then *A* is a **proper subset** of *B*, denoted $A \subset B$.

If each element of A can be paired with an element of B such that each element of A occurs exactly once, and each element of B occurs exactly once, in the pairings, we say A and B are **equivalent**, denoted $A \sim B$. If a set A is equivalent to a subset of the set of integers, we say the set A is **countable**. If A is null or equivalent to the set $\{1, 2, \ldots, m\}$, where m is a positive integer, we say A is a **finite set**. The **cardinality** of a finite set is m, the count of its elements. An **indexed set** is a set of the form $\{(1, a_1), (2, a_2), \ldots, (m, a_m)\}$.

A **bag** is a set in which duplicate elements are allowed. We delimit the elements in a bag with brackets, e.g., $B = \langle a, b, b, c \rangle$. A **list** is an ordered bag. Another way to define a list is to say it is an indexed set. Another way to define a list is to say it is a completely ordered sequence of zero or more elements.

Two real numbers a and b, with $a < b$, determine an **interval** in $\boldsymbol{R}$. Two types of intervals are

$$(a, b) = \{x : a < x < b\} \quad \text{an open interval, and}$$

$$[a, b] = \{x : a \leq x \leq b\} \quad \text{a closed interval}$$

We may also have intervals closed on the left and open on the right, or open on the left and closed on the right.

The **intersection** of two sets A and B, denoted by $A \cap B$, is the set

$$A \cap B = \{x \colon x \in A \textit{ and } x \in B\}$$

If $A = \{1, 2\}$ and $B = \{2, 3, 4\}$ then $A \cap B = \{2\}$. Two sets are **disjoint** if

$$A \cap B = \varnothing$$

The **union** of two sets A and B, denoted by $A \cup B$, is the set

$$A \cup B = \{x \colon x \in A \textit{ or } x \in B\}$$

For example, if $A = \{1, 2\}$ and $B = \{2, 3, 4\}$, then $A \cup B = \{1, 2, 3, 4\}$.

The **disjoint union** of two sets A and B, denoted by $A \sqcup B$, produces a set whose members are tagged elements. A tagged element is one of the form *x:$,* where $x \in X$ is the element and the symbol *$* is the **tag**. A tag is sometimes called an **identifier** or a **color**; it may be a string, a numerical value, or another piece of information. With a disjoint union, we tag an element with the name of the set containing it. For example, if $A = \{1, 2\}$ and $B = \{2, 3, 4\}$ then $A \sqcup B = \{1{:}A, 2{:}A, 2{:}B, 3{:}B, 4{:}B\}$. Tags are only tags. They do not enter into numerical calculations.

A **partition** of a set A is a collection of nonempty, pairwise disjoint subsets $\{A_1, \ldots, A_n\}$ whose union is A. The subsets are called **blocks**. One partition P_1 is said to **refine** another partition P_2 if every block of P_1 is contained in some block of P_2. Successive refinement (P_1 *refine* P_2 *refine* P_3 ...) is called **recursive partitioning** in the literature on clustering and decision trees (e.g., Breiman *et al.*, 1984).

The (Cartesian) **product** of sets A and B, denoted by $A \times B$, is the set

$$A \times B = \{(a, b) \colon a \in A \ and \ b \in B\}$$

For our example, $A \times B = \{(1,2), (1,3), (1,4), (2,2), (2,3), (2,4)\}$.

We call (a, b) an **ordered pair** or a **tuple**. Although the notation is the same as that for an open interval, the meaning should be clear from context. We call $(a_1, a_2, \ldots, a_n)$ an **n-tuple**. We call the item a_i ($i = 1, \ldots, n$) in an n-tuple an **entry**. The **degree** of an n-tuple is n. We denote the product set of n-tuples of the real numbers by $\boldsymbol{R}^n$.

2.1.2 Relations

Let A and B be sets. A **binary relation** R between A and B is a subset of $A \times B$. Given a tuple (a, b) in $A \times B$, we say that a is related to b by R if $(a, b) \in R$. An example is the "less than or equal to" relation between the set of real numbers $\boldsymbol{R}$ and itself given by $R = \{(x, y) : x \le y, x \in \boldsymbol{R}, y \in \boldsymbol{R}\}$. Another example is the gender-name relation R between the sets

$A = \{$"boy", "girl"$\}$, and

$B = \{$"Mary", "John", "Jean", "Pittsburgh"$\}$, given by

$R = \{$("boy", "John"), ("boy", "Jean"), ("girl", "Mary"), ("girl", "Jean")$\}$

It is possible for some members of A not to be related through R to any member of B and possible for some members of B not to be related through R to any members of A (unless we assume someone might be named Pittsburgh!).

An **n-ary relation** R on $A_1 \times A_2 \times \ldots A_n$ is a subset of $A_1 \times A_2 \times \ldots A_n$. Some authors notate such a relation by tagging, *e.g.*,

$$R = \{(a_{11}\colon A_1, a_{12}\colon A_2, \ldots, a_{1n}\colon A_n), \ldots, (a_{m1}\colon A_1, a_{m2}\colon A_2, \ldots, a_{mn}\colon A_n)\}$$

where $a_{ij} \in A_j$

2.1.3 Functions

Suppose that to each element of a set A we assign a unique element of a set B. The collection of these assignments is a **function**, which is also called a **mapping** from A into B. To indicate that f assigns elements of B to elements of A, we write

$$f\colon A \to B$$

We call A the **domain** of f and B the **co-domain** of f. The element $f(a) \in B$ is the unique element in B that f assigns to $a \in A$. We call the set of these elements $\{f(a)\colon a \in A\}$ the **image** of A under f or the **range** of f. Another way to describe a function (without naming it explicitly) is to use the symbol $\mapsto$. In this usage, "$a \mapsto b$" means "the function that assigns b to a."

An object that acts like a function can be viewed as a black box that receives input and returns output. For every input, there is only one possible output. That output need not be a single number or string. The output is whatever form the elements of the co-domain B take. Many different inputs may produce the same output, but we may not have more than one tagged output for a given input. This black-box definition includes the function $f(x) = x^2$ as well as the function f(*a binary tree*) = *the list of its parent-child relations*.

2.1.4 Graphs

For each function $f\colon A \to B$ there is a subset of $A \times B$,

$$\{(a, f(a))\colon a \in A\}$$

that we call the **graph** of f. The graph of the function $f(x) = x^2$, where x belongs to the set of real numbers, is the set of all tuples (x, x^2), which is a subset of the crossing of the set of real numbers with itself. The graph of the function f(*a binary tree*) = *the list of its parent-child relations* is the set of all tuples defined by (*a binary tree*, *the list of its parent-child relations*), which is a subset of the crossing of the set of all binary trees with the set of all lists of parent-child relations. The graph of a function uniquely determines the function, and *vice versa*. For example, if (2, 4) is the graph of f, then $f(2) = 4$.

2.1.5 Compositions

A **composition** is a function formed from a chain of functions. Let $f\colon X \to Y$ and $g\colon Y' \to Z$ be functions for which the co-domain of f is a subset of the domain of g (*i.e.*, $Y \subseteq Y'$). The function $g \circ f\colon X \to Z$ defined by the rule

$$(g \circ f)(x) = g(f(x)) \quad \text{for all } x \in X$$

is the composition or **composite function** of f and g.

For example, if f and g are string functions and the functional rule for f is <*capitalize leftmost letter*> and the rule for g is <*count number of capitals*>, then the following compositions are defined because all the inputs are members of the set of text strings and any output of f is a legal input for g.

$g(f(\text{"wow"})) = 1$
$g(f(\text{"Wow"})) = 1$
$g(f(\text{"123"})) = 0$
$g(f(\text{""})) = 0$

The range of the composition is the range of g, namely, the set of non-negative integers. Also, the null string is a member of the set of strings. If we implement these functions in a language like *C*, we must be sure to handle nulls properly.

2.1.6 Transformations

A **transformation** is a function $f: A \to A$ mapping a set A to itself. All transformations are functions, but not all functions are transformations. Because a transformation maps a set to itself, a composition of transformations is a transformation.

For example, if f and g are text string transformations and the rule for f is <*capitalize leftmost letter*> and the rule for g is <*append exclamation*>, then the following compositions are all transformations.

$g(f(\text{"wow"})) = \text{"Wow!"}$
$f(f(\text{"wow"})) = \text{"Wow"}$
$g(g(\text{"wow"})) = \text{"wow!!"}$
$f(g(f(\text{"wow"}))) = \text{"Wow!"}$
$g(f(g(\text{"wow"}))) = \text{"Wow!!"}$
$f(g(\text{""})) = \text{"!"}$

2.1.7 Algebras

An **algebra** is a collection of 1) **sets**, 2) **operators** on sets, and 3) **rules** for the combination of these operators. This definition includes algebras more general, limited, or abstract than the classical algebra underlying ordinary arithmetic on real numbers.

An operator generalizes the notion of transformation. An operator on a set X is a function defined on the set $X \times X \times \ldots X$ that returns a value in X. Operators are **unary** or **monadic** (having one argument, *i.e.*, defined on X, and so a transformation), **binary** or **diladic** (having two arguments, *i.e.,* defined on $X \times X$), or ***n*-ary** (having many arguments, *i.e.,* defined on the n-fold product of X with itself). Algebraic rules specify how operators are to be composed. An example is the operator “+” on the set $\boldsymbol{R}$ defined by $(a, b) \mapsto a+b$.

2.1.8 Variables

A **variable** X is a mapping $f: O \to V$, which we consider as a triple:

$X = [O, V, f]$, where

the domain O is a set of objects,

the codomain V is a set of values,

the function f assigns to each element of O an element in V, The image of O under f contains the values of X. We denote a possible value as x, where $x \in V$. We denote a value of an object as $X(o)$, where $o \in O$. A variable is **continuous** if V is an interval. A variable is **categorical** if there exists an equivalence between V and a finite subset of the integers.

Variables may be multidimensional. $\boldsymbol{X}$ is a p-dimensional variable made up of p one-dimensional variables:

$$\begin{aligned} \boldsymbol{X} &= (X_1, \ldots, X_p) \\ &= [O, V_i, f], \quad i = 1, \ldots, p \\ &= [O, \boldsymbol{V}, f] \end{aligned}$$

The element $\boldsymbol{x} = (x_1, \ldots, x_p), \boldsymbol{x} \in \boldsymbol{V}$, is a p-dimensional value of $\boldsymbol{X}$.

2.1.9 Varsets

We call the triple:

$$\mathsf{X} = [V, \tilde{O}, f]$$

a **varset**. The word stands for *variable set.*

A varset inverts the mapping used for variables. That is,

the domain V is a set of values,

the codomain $\tilde{O}$ is a set of all possible bags of objects,

the function f assigns to each element of V an element in $\tilde{O}$,

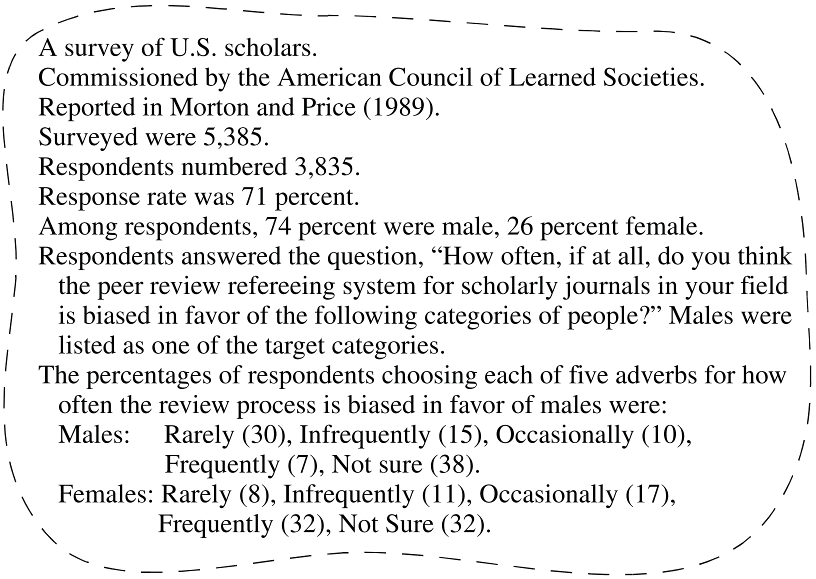
We invert the mapping customarily used for variables in order to simplify the definitions of graphics algebra operations on varsets. In doing so, we also replace the variable's set of objects with the varset's set of bags. We use bags in the codomain because it is possible for a value to be mapped to an object more than once (as in repeated measurements).

2.1.10 Frames

A **frame** is a set of tuples $(x_1, \ldots, x_p)$ ranging over all possible values in the domain of a p-dimensional varset. Frames thus depend on algebra expressions. Frames serve as reference structures for computing aesthetics. Popular writers often call a frame the rectangular bounds demarcated by axes — in analogy to a picture frame. There are several problems with this popular notion. First, axes are guides for coordinate systems; they are not bounds on a space. Second, frames are not only positional; we can have a color frame constructed from an algebraic expression that yields a color space, for example. Third, frames are not rectangles; they are sets of tuples.

2.2 Recipe

The following subsections will implement our recipe from beginning to end. We will use as our example the ACLS survey data summarized in Figure 2.3 (Morton and Price, 1989). We will look at the perceived bias against women in reviewing academic papers, as seen by each gender.



A survey of U.S. scholars.
Commissioned by the American Council of Learned Societies.
Reported in Morton and Price (1989).
Surveyed were 5,385.
Respondents numbered 3,835.
Response rate was 71 percent.
Among respondents, 74 percent were male, 26 percent female.
Respondents answered the question, "How often, if at all, do you think the peer review refereeing system for scholarly journals in your field is biased in favor of the following categories of people?" Males were listed as one of the target categories.
The percentages of respondents choosing each of five adverbs for how often the review process is biased in favor of males were:
Males: Rarely (30), Infrequently (15), Occasionally (10), Frequently (7), Not sure (38).
Females: Rarely (8), Infrequently (11), Occasionally (17), Frequently (32), Not Sure (32).

Figure 2.3 *ACLS data*

2.2.1 Create Variables

Our first step is to extract data into variables. There are many versions of this process. For example, one version could create a variable from a relational database. Another could create a variable by indexing a stream of data. Another could create a variable from a text document by indexing words. Another could generate values from a mathematical function. For our pies, we will simply load the data from a database. We will assume that the data in Figure 2.3 are organized in a database table of the form shown in Table 2.1.

Table 2.1 **ACLS Database Table**

CaseID	Gender	Bias in favor of males	...

We can extract the data we need with the following queries:

```
Response = loadFromSQL("ACLS", "bias_toward_males", "case")
Gender = loadFromSQL("ACLS", "Gender", "case")
```

The function `loadFromSQL(table, variable, case)` constructs a query of the form `(SELECT variable, case FROM ACLS)`, connects to the database and executes the query, storing the result in a variable. For our pie we get variables shown in Table 2.2. For convenience, we represent variables as columns in a table, even though the definition of a variable in Section 2.1.8 does not imply this organization.

Table 2.2 **Subtables From ACLS Database**

CaseID	Response
1	Frequently
2	Not Sure
3	Frequently
...	...
3834	Rarely
3835	Infrequently

CaseID	Gender
1	Male
2	Female
3	Male
...	...
3834	Male
3835	Female

2.2.2 *Apply Algebra*

A general definition of an algebra is given in Section 2.1.7. Chapter 5 defines a graphics algebra consisting of three operators — *cross*, *blend* and *nest* — applied to a set of variables, together with a set of associated rules. The output of the algebra is a varset.

Since our extract-variables process made two separate variables, we need to combine them into one variable so that the rows are associated with each other. We use the algebraic *cross* function to accomplish this. In Chapter 5 we use an operator (*) for the *cross* function in order to construct more complex expressions in an algebraic notation, but we do not need that complexity here. The result appears in Table 2.3.

Table 2.3 ***cross***(`Response`, `Gender`)

CaseID	Response	Gender
1	Frequently	Male
2	Not Sure	Female
3	Frequently	Male
...	...	...
3834	Rarely	Male
3835	Infrequently	Female

2.2.3 Apply Scales

The categorical variables `Response` and `Gender` must have a defined order. A categorical scale transformation associates the values of a categorical variable with the set of integers. It may do this through a variety of possible methods. Any number of automatic categorical scale transformations can be devised, such as natural (alphabetical) order, relative frequency of the answer, or even the length of the string.

We will use a custom ordering that maps Rarely→1, Infrequently→2, Occasionally→3, Frequently→4, Not Sure→5. This is derived from the ordering of the frequency implied by the responses, where the Not Sure case isn't comparable and is arbitrarily placed last. We will use an alphabetical ordering for the `Gender` variable. The result appears in Table 2.4.

Table 2.4 ***cat***(`Response,` ***values***(`"Rarely" ... "Not Sure"`)), ***cat***(`Gender,` ***values***(`"Female", "Male"`))

CaseID	Response	Gender
1	4	2
2	5	1
3	4	2
...	...	...
3834	1	2
3835	2	1

2.2.4 Compute Statistics

Our system will ultimately make one graphical element per row of a varset. A pie chart of the data we have so far would therefore have 3,835 slices in it. This data layout isn't what we usually have in mind when we make a pie. In order to reduce the number of cases, we will need to do some statistics on the data.

A pie chart uses the pie wedges to represent the proportion of the whole for some category. The *summary.proportion* function is an aggregating statistic. It aggregates over columns constituting a whole pie's domain — in this case, each gender is a whole. The aggregated variable is `Response`. Table 2.5 shows the result, using the *summary.proportion*() statistical method.

***Table 2.5 summary.proportion*()**

CaseID	Summary	Response	Gender
{females responding "rarely"}	0.08	1	1
{females responding "infrequently"}	0.11	2	1
{females responding "occasionally"}	0.17	3	1
{females responding "frequently"}	0.32	4	1
{females responding "not sure"}	0.32	5	1
{males responding "rarely"}	0.30	1	2
{males responding "infrequently"}	0.15	2	2
{males responding "occasionally"}	0.10	3	2
{males responding "frequently"}	0.07	4	2
{males responding "not sure"}	0.38	5	2

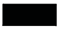
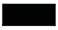
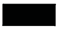
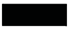
2.2.5 Construct Geometry

The grammar of graphics has a variety of different geometric graphing operations, as described in Chapter 8. In this case, we will use the *interval* graph. The *interval*() function converts an interval into a (usually rectangular) geometric object with a length proportional to the interval and a nonzero (usually constant) width. How does *interval*() get an interval out of the *summary*() statistical function if a summary is only one point statistic? The convention we use is that *interval*() will add a lower bound of zero if there is only one point statistic it receives. This convention is derived from ordinary usage, where bar charts commonly are anchored at zero unless an explicit lower bound is specified.

The *interval.stack*() function cumulates the intervals constructed by the interval geometric object. In other words, *stack* is a modifier method on the *interval* class. Each interval is modified by incrementing its lower and upper bounds by the upper bound of the preceding interval in a sequence.

Table 2.6 shows the result. Note that we maintain case information in our graph (the leftmost column), as we will need it when we compute aesthetics.

Table 2.6 interval.stack()

CaseID	Interval	Response	Gender
{females responding "rarely"}		1	1
{females responding "infrequently"}		2	1
{females responding "occasionally"}		3	1
{females responding "frequently"}		4	1
{females responding "not sure"}		5	1
{males responding "rarely"}		1	2
{males responding "infrequently"}		2	2
{males responding "occasionally"}		3	2
{males responding "frequently"}		4	2
{males responding "not sure"}		5	2

2.2.6 Apply Coordinates

To make pie wedges, we apply a polar transformation to the shapes that were produced from the geometry. We send (*x*, *y*) to (*r*, *theta*) for the shapes. Our program might accomplish this by digitizing the edges and sending each of the points through a simple transformation, or it might recognize that when we transform a rectangle, we actually are creating a partial annulus and represent the shape directly. We can even special-case the pie wedge to take advantage of the *disk-sector* graphical primitive provided by many graphics systems.

Table 2.7 shows the result of the polar coordinate transformation. We note that this transformation applies only to the *position* aesthetic. It is possible in the grammar of graphics to apply coordinate transformations to other aesthetics as well, *e.g.*, COORD: *polar.theta*(*aesthetic*(*color*)). This is how we can employ different color spaces, for example, in a graphic. It is important to remember that a frame is a space and that every aesthetic, not just *position*, has a frame. We are so accustomed to thinking about positional aspects of visualizations that we forget that coordinate spaces apply to all the aesthetics, using similar rules in each ease.

***Table 2.7 polar.theta*()**

CaseID	Slice	Response	Gender
{females responding "rarely"}		1	1
{females responding "infrequently"}		2	1
{females responding "occasionally"}		3	1
{females responding "frequently"}		4	1
{females responding "not sure"}		5	1
{males responding "rarely"}		1	2
{males responding "infrequently"}		2	2
{males responding "occasionally"}		3	2
{males responding "frequently"}		4	2
{males responding "not sure"}		5	2

By postponing the coordinates operation as late in the pipeline as possible, we have made the system more flexible. The same code can produce a divided bar or pie. This is one example that follows one of our favorite maxims of good design, the title of an MIT AI Lab report, “Planning is just a way of avoiding figuring out what to do next” (Brooks, 1987).

2.2.7 Compute Aesthetics

We have created a graph, but a graph is a mathematical abstraction. We cannot sense mathematical abstractions. We must therefore give this abstraction perceivable form. Aesthetic functions translate a graph into a **graphic**. Table 2.8 adds three aesthetic functions — *position*, *color*, and *label*. We have not included the text for each label in the table, but assume the proper text is accessible in an additional column.

The input to each of the aesthetic functions is a set of tuples, constituting the frame for each aesthetic. Let’s try to specify each in a sentence. The tuples input to the *position* aesthetic are in a region defined by the polar transformation of the region created by the *interval.stack*() geometric element operating on the *summary.proportion*() function operating on the varset created by crossing `Response` and `Gender`; the output tuples are used as references to screen or page coordinates. The tuples input to the *color* aesthetic are in a region containing all possible values of the `Response` variable; the output tuples are used as pointers to the entries of a color table. The tuples input to the *label* aesthetic are the same as those input to the *color* aesthetic; the output tuples are used as pointers to the character strings containing the labels.

The output of each aesthetic function is a **graphic**, which is a set of drawing instructions for a renderer. The *position*() aesthetic translates the tuples corresponding to the vertices of the objects to be drawn into a renderer coordinate system (usually pixel-based). The *color*() aesthetic function produces an index to a color table so that the renderer can apply the appropriate color. We have colored the wedges in the table to signify this. The *label*() aesthetic function associates strings with values.

Sometimes attention to detail leads to a level of detail that appears unnecessary. In programming these operations, however, we cannot ignore details. Entrusting a rendering system to implement the details can cause problems in a grammar-based system. For example, the grammar requires us to be able to chain coordinate systems, as we have done here for the pie chart. Some renderers that offer different coordinate systems do not allow chaining them, however, because not all coordinate transformations are invertible. If we use default coordinate systems in these graphics systems, we will not be implementing the grammar.

Table 2.8 ***position***(`Response, Gender`), ***color***(`Response`), ***label***(`Response`)

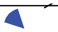
CaseID	Slice	Response	Gender
{females responding "rarely"}		1	1
{females responding "infrequently"}		2	1
{females responding "occasionally"}		3	1
{females responding "frequently"}		4	1
{females responding "not sure"}		5	1
{males responding "rarely"}		1	2
{males responding "infrequently"}		2	2
{males responding "occasionally"}		3	2
{males responding "frequently"}		4	2
{males responding "not sure"}		5	2

Figure 2.4 shows a possible result of a renderer working on the data in Table 2.8, together with the formal specification for the graph. The layout follows certain conventions that we will discuss later in this book. For example, the two pies are arranged horizontally and the labels are placed below the pies. These conventions, of course, are arbitrary. It is important, of course, that they be applied consistently and with good design considerations.

```
DATA: response = Response
DATA: gender = Gender
SCALE: cat(dim(1),
       values("Rarely","Infrequently","Occasionally","Frequently","Not Sure"))
SCALE: cat(dim(2), values("Female","Male"))
COORD: rect(dim(2), polar.theta(dim(1)))
ELEMENT: interval.stack(position(summary.proportion(response*gender)),
         label(response), color(response))
```

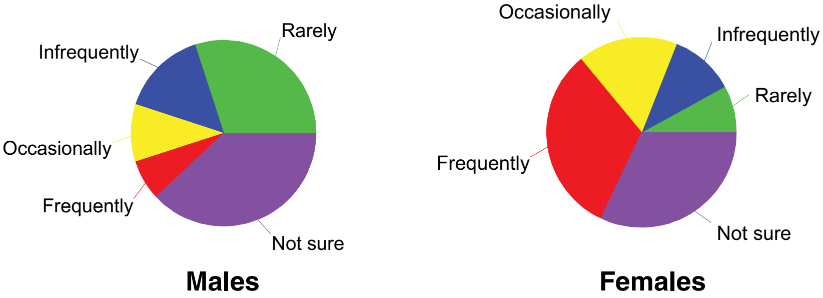


Figure 2.4 *Pair of pies*

2.3 Notation

Each graphic in the first part of this book is accompanied by a symbolic specification. The main purpose of the specification syntax is to allow compact summarization of the components represented in Figure 2.2. When you examine a particular graphic, it is important to spend enough time on the specification to understand the meaning of the graphic. The meaning of a graphic is its specification and associated data.

2.3.1 Specifications

The specifications at the head of each graphic follow these conventions:

1) SMALL-CAPS: statement type
2) `typewriter`: data fields
3) sans-serif: variables
4) *italic*: functions

Not every printed specification is sufficient for producing the entire graph shown in the figure. We have omitted details such as positioning of legends, annotation, and labeling of axes in order to focus on the fundamental structure of the graphics. In addition, we have occasionally used color to highlight graphical elements referenced by the text. This use of color does not always appear in the specification. Finally, we have used text sparingly in some places in order to emphasize the essential features of the graphics being discussed. Some of the graphics in this book lack the labels and other annotation that would be desirable for substantive publications.

For any graphic, the expression, attributes, and graphs will be represented in a multi-line notation that comprises the specification. The first line or lines contain the DATA functions that create variables from data. These are normally omitted if the data are assumed to be organized in a cases-by-variables matrix. The next lines give the TRANS specifications that define the transformation to be applied to the variables. These lines are optional if we assume an identity transformation. The next lines are the SCALE specifications that specify dimensions on which the graphs will orient themselves. The SCALE specifications are optional if an ordinary interval scale is used on all dimensions. The next line or lines is the COORD specification that defines the coordinate system in which the graphs are to be embedded. We usually set the coordinates for the *position* aesthetic, but we could set coordinates for other aesthetics as well in a similar statement. This line is optional if we assume rectangular Cartesian coordinates. The next line or lines is the GUIDE specification for axes and legends and other guide notation. The remaining lines contain the geometric graphing functions for the graphs appearing in the frame. These are denoted with the label ELEMENT.

For example, a two-dimensional scatterplot can be represented by

DATA: x = x
DATA: y = y
TRANS: x = x
TRANS: y = y
SCALE: *linear*(*dim*(1))
SCALE: *linear*(*dim*(2))
COORD: *rect*(*dim*(1, 2))
GUIDE: *axis*(*dim*(1))
GUIDE: *axis*(*dim*(2))
ELEMENT: *point*(*position*(x*y))

or equivalently for this example,

ELEMENT: *point*(*position*(x*y))

The DATA specification assigns a unique index to each element of the data using an index function. Sequential indexing is used in this example, so it can be omitted. The TRANS specification transforms the resulting variables. The identity transformation is used here, so it can be omitted. The SCALE specification sets the scales for each dimension. Linear decimal scales (as opposed to, say, log scales) are used, so it can be omitted. The COORD specification sets the coordinate system. Rectangular coordinates are used, so it can be omitted. The GUIDE specifications determine two axes. These default to the dimensions in the frame, so they can be omitted. And the GRAPH specification includes any functional graphs.

2.3.2 Functions

Functions are notated in italics, *e.g.*,

color()
position()
point.statistic.mean()
log()

Functions may be **overloaded**. This is a method for grouping under one name several functions that perform similar tasks but have different arguments. If we need to assign an index to a value of a variable, for example, we can make several versions of a function and name all of them *index*(). One version accepts real numbers, another accepts strings, and so on. By using the same name for each function and letting the system determine which version to use after examining the type of arguments, we can keep our design simple. For example, the following three functions return the same result (assuming the variable group contains a single value of either *color.red* or 0 and assuming the value of *color.red* is 0):

color(*color.red*)
color(0)
color(group)

Precise definitions of this behavior are required, but overloading is easy to implement in object-oriented languages like Java. It is more difficult (but not impossible) in languages like *C* and FORTRAN.

See Chapter 18 for more information on the Graphics Production Language (GPL) used in this book. It is a well-formed executable computer language.

2.4 Sequel

The rest of this book examines Figure 2.2 in more detail. The next chapter starts at the beginning. It covers data and datasets, the classes of objects that are at the lowest stage of the graph creation hierarchy.

3

Data

The word **data** is the plural of the Latin *datum*, meaning a given, or that which we take for granted and use as the basis of our calculations. This meaning is carried in the French word for statistical data, *données*. We ordinarily think of data as derived from measurements from a machine, survey, census, test, rating, or questionnaire — most frequently numerical. In a more general sense, however, data are symbolic representations of observations or thoughts about the world. As we have seen, we do not need to begin with numerals to create a graphic. Text strings, symbols, shapes, pictures, graphs themselves, can all be graphed.

Older graphing systems required data to be stored in **flat files** containing row-by-column arrays of numerical observations on a set of variables. Spreadsheet graphics systems also arrange data this way. In these systems, the structure of the data determines the types of displays possible: line charts, bar charts, pie charts, or scatterplots. More recent object-oriented graphics systems assume their data are in a **data source**. And the most flexible systems provide an object-oriented interface to the data source which makes no assumptions about the underlying structure of the data. The graphing system itself has a **view** of the data that it uses for its initial mapping. This view may have no simple relation to the actual organization of the data source. In addition, the view may change from moment to moment because the system may be fed by a **streaming** data source. A well-designed graphics system must be able to handle these situations in order to avoid static views that can misrepresent the underlying data.

This chapter outlines three types of data: empirical, abstract, and meta. These types are distinguished by their sources and function. Empirical data are collected from observations of the real world. Abstract data arise from formal models. And metadata are data on data. Treinish (1993) provides further details on these types and elucidates them within the general context of graphics production. Fortner (1995) discusses empirical and abstract scientific data. We will first summarize data functions and then discuss these three types of data. We will conclude with a brief section on the emerging field of data mining and its relationship to the models needed to support a graphical system.

3.1 Data Functions

Table 3.1 lists some functions we will use in this chapter to create variables from datasets. We use the convention that the `typewriter` letters refer to column names (domain names) of a relation. The results of these functions are function variables. As we mentioned in the last chapter, we could apply these functions to the contents of an object-oriented database as long as we devised a relational interface to insure that related sets are indexed properly. In any case, the actual referencing scheme we need is independent of the physical or formal organization of the data themselves.

Table 3.1 **Data Functions**

Empirical Data	***Abstract Data***	***Metadata***

col(*source*(), *name*(), *unit*(), *weight*())
map(source(), *id*())
stream(source(), *id*())
image(source())
sample(x, *n*)
reshape(x_1,...,x_n, "<*index*>")
	iter(*from*,*to*,*step*)
	mesh(*min*, *max*, *n*)
	count(*n*)
	proportion(*n*)
	percent(*n*)
	constant(*c, n*)
	string("<*string*>", *n*)
	rand(*n*)
		meta(*source*(), *name*())

The empirical functions operate on columns of observed data. The *col*() function links a variable to a column in a data source. If this function is not used in a specification, then the names of variables are assumed to be the same as the names of fields in the data source. The *unit*() argument assigns measurement units. The *weight*() argument maps a column to case weights used for statistical calculations. The *map*() function is for maps, the *stream*() function is for streaming data sources, and the *image*() function is for image sources.

The *sample*() function implements sampling. Sampling methods include *sample.srs* (simple random), *sample.jackknife* (Tukey, 1958) and *sample.boot* (Efron & Tibshirani, 1993). The *reshape*() function reshapes a matrix or table of columns (corresponding to an *n*-ary relation) into a single variable. Let $\mathbf{X}_{m \times n}$ be a matrix produced by concatenating the columns (x_1,...,x_n). Let *i* and *j* be row and column indices respectively of the matrix **X** ($i = 1, \ldots , m$ and $j = 1, \ldots , n$). Let *k* be the row index of the variable **x** output by the *reshape*() function. The *reshape*() functions compute the index *k* as follows:

$$\begin{aligned}
&\textit{reshape.rect}(): && k = n \cdot (i-1) + j \\
&\textit{reshape.tri}(): && k = i \cdot (i-1)/2 + j : (i \geq j) \\
&\textit{reshape.low}(): && k = (i-1) \cdot (i-2)/2 + j : (i > j) \\
&\textit{reshape.diag}(): && k = i : (i = j)
\end{aligned}$$

Note the name of the function is what it converts *from*, not *to*. The *reshape.rect*() function unwraps the rows of a rectangular matrix into a single column. The *reshape.tri*() function unravels the lower triangular half (including the diagonal) of a square matrix into a single column. The *reshape.low*() function does the same for the triangular half excluding the diagonal. And the *reshape.diag*() function places the diagonal of a square matrix into a column.

The <*index*> argument to the *reshape*() function has several alternatives:

value:	the value of the entries
rowindex:	the row index
colindex:	the column index
rowname:	the row name
colname:	the column name

The *reshape*() functions leave one with the impression that a matrix algebra package could provide many (but not all) of the functions needed for structuring data at the first stage of a graphics system. Many of the graphics we draw in practice depend on a matrix data model. Notable exceptions, however, are geographic, physical, and mathematical objects that must depend on different data organizations.

The abstract functions create columns. They operate across rows and most take an argument (*n*) for the number of rows; if the argument is omitted and we are processing a real dataset, *n* is the number of rows in the dataset. The *iter*() function is a simple iterator for creating arithmetic series. The series *iter*(1, 10, 1) contains a sequence of the integers 1 through 10, for example. The *mesh*() functions computes a 1D, 2D, or 3D mesh. The *count*(), *proportion*(), and *percent*() functions are iterators as well. The *count*(10) function, for example, is equivalent to *iter*(1, 10, 1). A comparable *proportion*(10) iterator would be equivalent to *iter*(.1, 1.0, .1). Finally, the *constant*() and *string*() iterators supply *n* instances of an item and the *rand*() iterator generates independent random numbers. The *rand*() iterator has methods such as *rand.uniform*() and *rand.normal*(). We will occasionally use <string> as shorthand for the *string*("<*string*>") function, *e.g.*,

DATA: s = *string*("Hello world")
DATA: s = "Hello world"

The *meta*() function associates metadata with rows of datasets. We have included an atomistic example to suggest the association of a single metadata item with a row of a dataset. Each index in the *source*() of the *meta*() function might point to a video image, a Web address in the form of a Universal Resource Locator (URL), or some other item of metadata.

Finally, there is a system variable that may be used in place of user variables. It is an identity variable, notated **1**, which is an identity element for the cross and nest operators in graphics algebra. It is discussed in Section 5.1.3.3.

3.2 Empirical Data

Empirical data arise from our observations of the world. The Greek term *ἐμπειρία*, the source of the English word *empirical*, means experience or acquaintance with some thing. Among the many prevailing views of the role of empirical data in modern science, there are two opposing extremes. On the one hand, the **realist** assumes data are manifestations of latent phenomena. In this view, data are pointers to universal, underlying truths. On the other hand, the **nominalist** assumes data *are* what they describe. From this latter point of view, as the philosopher Ludwig Feuerbach noted, "you are what you eat." We use the historical realist-nominalist terminology to emphasize that these differing perspectives, in one form or another, have origins in Medieval theological controversies and their classical antecedents in Greek philosophy.

Many working scientists adopt some form of a realist position, particularly those in the natural sciences. Even social scientists, despite the myriad particulars of their field, have often endorsed a similar position. Watson (1924) and Hull (1943), for example, believed that behavioral data could be explained best through universal quantitative laws. More recently, Webb *et al.* (1981) promoted "unobtrusive" data collection methods that point to or help us triangulate on basic truths about behavior.

Alternatively, operationalists like Skinner (1969) have argued that behavioral data are best understood by avoiding unobservables and inferences. Skinner even rejected statistical modeling, arguing that smoothing or aggregation obscures details that could help falsify theories. The graphics in Skinner (1961), for example, are the cumulative traces over time by a stylus connected to a lever or button pressed by a pigeon ("untouched by human hands"). Figure 3.1 shows an example of this type of graphic. The horizontal axis marks the linear movement of the stylus over time. The vertical axis marks the increment of the stylus following each bar press. The vertical trace lines are due to resetting of the stylus to keep it on a single page. Skinner argued that these graphics are sufficient representations of responses to schedules of reinforcement. He contended that smoothing, interpolation, aggregation, or model fitting only hide the associations that the scientist needs to see in order to refine theory.

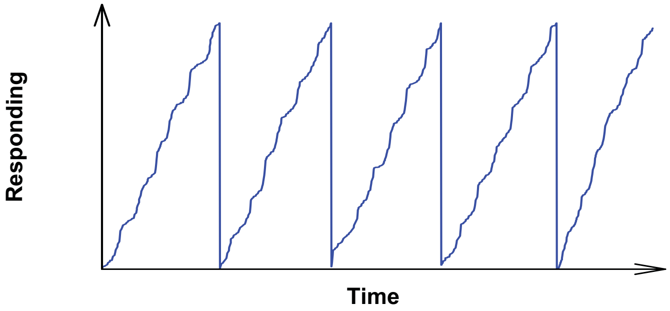


Figure 3.1 *Cumulative record*

Skinner's cumulative-record graphic would appear to short-circuit the diagram presented in Figure 2.2. It would seem that the physical production of such a graphic obviates the need for data functions, variables, indeed, a computer. In fact, however, Skinner's hard-wired device performs our data functions implicitly. Each pen movement involves a sequential indexing of the bar press events and time increments. Skinner's variables are Time and Cumulative Response Count.

Computer data acquisition systems now perform the functions originally designed into mechanical lab equipment like Skinner's. In doing so, they enable alternative data organization and views not always anticipated by the scientist collecting the data. Consider Figure 3.1. It is visually indistinguishable from several of the cumulative records in Ferster and Skinner (1957). However, if we had Skinner's raw data we could use modern statistical time-series methods (unavailable to Skinner) to show that the data underlying Figure 3.1 were not produced by a real organism (human, mouse, *or* pigeon) in an operant conditioning chamber. They were generated by a computer from a stochastic equation. This detective work would involve producing a different graphic from the same data. We can sometimes image-scan older graphics to retrieve data, but there is no substitute for having the original observations and devising our own analyses.

The following examples illustrate some methods for referencing and organizing raw data to produce variables that can be graphed. The first example shows how variables are not always derived from columns in files. The second shows how they may be derived from repeated random samples.

3.2.1 Reshaping Data

Since any matrix is transposable, we can convert row data into column data and graph them when appropriate (*e.g.*, Jobson, 1992, p. 428). Or, as Figure 3.2 shows, we can construct a graphic based on a subset of a matrix.

A correlation matrix is a symmetric matrix of the correlations among a set of variables. If our received data are in a correlation matrix and we wish to graph the correlations, we have to restructure them into one variable using the *reshape.low*() function. This takes the lower-triangular elements of a symmetric matrix and strings them into one variable.

```
DATA: r = reshape.low(pounding, sinking, shaking, nauseous, stiff,
          faint, vomit, bowels, urine, "value")
ELEMENT: point.dodge.asymmetric(position.bin.dot(r))
```

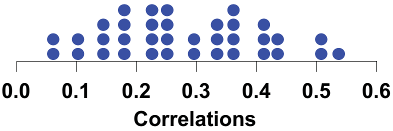


Figure 3.2 *Dot plot of correlations from correlation matrix*

These correlations are based on recalled experiences of symptoms of combat stress (1=yes, 0=no) among soldiers who experience battle in World War II. These data fit a Guttman scale (Stouffer *et al*., 1950). A Guttman scale is an ordinal relation that Guttman called a **simplex**. Presence of a given symptom implies presence of all milder symptoms and absence of a given symptom implies absence of all stronger symptoms. Such a relation implies that the matrix of correlations among symptoms should have all positive elements that follow a banded structure, with the largest correlations near the diagonal and the smallest near the corner opposite the diagonal.

We can display this structure by reshaping the correlation matrix in a different way. This time, we will create three variables — a row index, a column index, and the value of the correlation corresponding to each combination of these indices. Then we will plot the entire correlation matrix using color to represent the value of each correlation. This method works well in this example because all the correlations (as expected for a simplex) are positive.

Figure 3.3 reveals this structure. We have used the *reshape*() data functions to derive indices for plotting the rows and columns of the correlation matrix directly and a *polygon* graph within binning (see Chapter 8) to represent the correlations as colored rectangles. Notice that the colors get warmer as we approach the diagonal.

```
DATA: row = reshape.tri(pounding, sinking, shaking, nauseous,
        stiff, faint, vomit, bowels, urine, "rowname")
DATA: col = reshape.tri(pounding, sinking, shaking, nauseous,
        stiff, faint, vomit, bowels, urine, "colname")
DATA: r = reshape.tri(pounding, sinking, shaking, nauseous, stiff,
        faint, vomit, bowels, urine, "value")
ELEMENT: polygon(position(bin.rect(col*row)), color.hue(r))
```

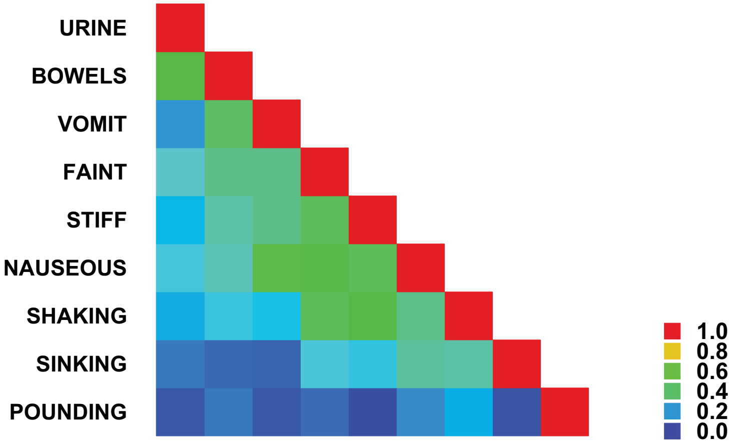


Figure 3.3 *Correlation matrix of combat symptoms*

3.2.2 Bootstrapping

Efron and Tibshirani (1993) discuss a procedure for repeatedly computing a statistic using random samples (with replacement) from a dataset. This procedure, called **bootstrapping**, offers a way to compute confidence intervals when conventional methods are inefficient or unavailable. A bootstrap sample is a sample with replacement from a dataset. Usually, the size of the sample is the same size as the dataset, so that some cases are sampled more than once. Figure 3.4 shows an example of a histogram of a dataset and a histogram of means from 1,000 bootstrap samples of that dataset. The variable mean is indexed to a set of bootstrapped means, each of which was computed from the original dataset. This operation is equivalent to computing the mean of a set of values by making a single pass through a column of a database, repeating this database query 1,000 times, and then assembling all the means into a column of 1,000 values. This computationally intensive operation is not one calculated to endear a user to a database administrator. Although automated boostrapping is not available in database systems, some statistical packages that have data management subsystems do perform bootstrapping.

ELEMENT: *interval*(*position*(*summary.count*(*bin.rect*(military, *dim*(1)))))

DATA: mean = *sample.boot*(military, 1000, "mean")
ELEMENT: *interval*(*position*(*summary.count*(*bin.rect*(mean, *dim*(1)))))

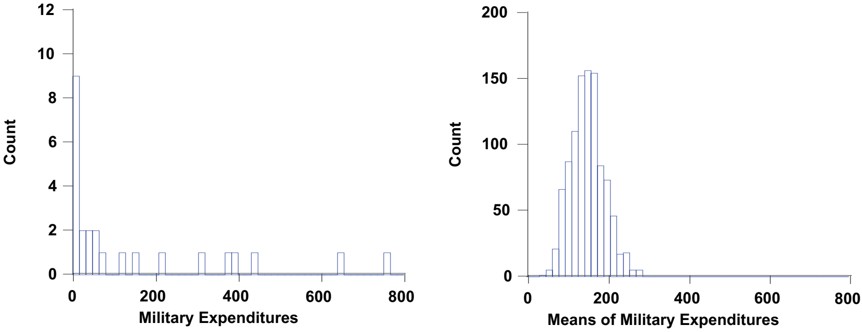


Figure 3.4 *Bootstrapped means*

Two salient contrasts are apparent in Figure 3.4. The first is that the histogram of bootstrapped means is less positively skewed. The second is that the variation in the bootstrapped means is smaller. The standard deviation of the values in the left histogram is 214.1 and the standard deviation of the mean values in the right histogram is 42.3. These differ by a factor of approximately 5, which is the square root of the sample size, 25. The bootstrapping results are consistent with the standard statistical theory for the sampling distribution of means. Nevertheless, the distribution of means is still positively skewed. If the sample size were larger than 25, the means would look more normally distributed.

3.3 *Abstract Data*

Abstract data functions are most often used to create variables consisting of series, lattices, and other indexing schemes that we use to arrange observed data values. There are many generating functions for series that are useful in the data manipulation needed for statistical graphics. Knuth (1968) discusses some of these functions. Sometimes we construct graphics using abstract data that are determined by a mathematical rule or function. We will first present some simple examples of the former and then one example of the latter.

3.3.1 *Time Series*

Time series datasets do not always include a column for time. Even when they do, we sometimes wish to plot against the index of the measurement rather than against time. Figure 3.5 contains 256 instantaneous firing rate measurements of a single cat retinal ganglion cell during a 7 second interval (Levine *et al.*, 1987). The points are connected by a *line* graph. Most time series packages automatically create this index variable for equally spaced time series; some go further and code the series in calendar time. In this example, we have used the *iter* data function to create a time index running from 1 to 256.

We have chosen an aspect ratio for this plot that reveals a low-frequency component of roughly one-and-a-half cycles over the duration of the series. This component is due to the respiration of the cat, according to Michael Levine (personal communication). Firing rate tends to increase with more oxygen in the blood.

DATA: case = *iter*(1, 256, 1)
ELEMENT: *line*(*position*(case*rate))

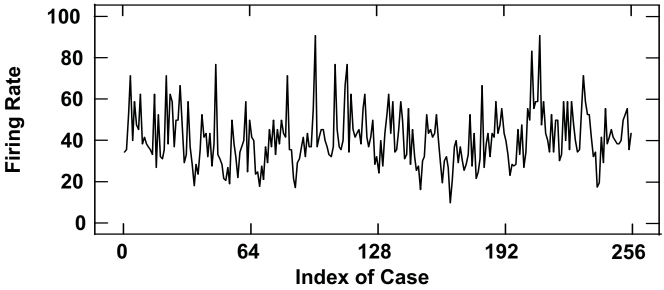


Figure 3.5 *Firing rate of cat retinal cell ganglion*

3.3.2 Counts

Bar charts of counts often need to be constructed when there is no explicit count variable in the data. Figure 3.6 shows a bar chart constructed from the countries data used in Chapter 8. The *constant*() implicit function fills a column with ones. The *summary.count* statistical function (see Chapter 7) tallies the counts in all combinations of **gov** and **urban** values. Notice that we did not use the *bin* statistical function as we did for histograms. The data are already classified into distinct groups by the categorical variables. In some histogram examples and bar charts of counts, we omit the constant variable. In the GPL language, we assume the *summary.count* function constructs it if it is missing.

DATA: z = *constant*(1)
COORD: *rect*(*dim*(1, 2, 3))
ELEMENT: *interval*(*position*(*summary.count*(gov*urban*z)))

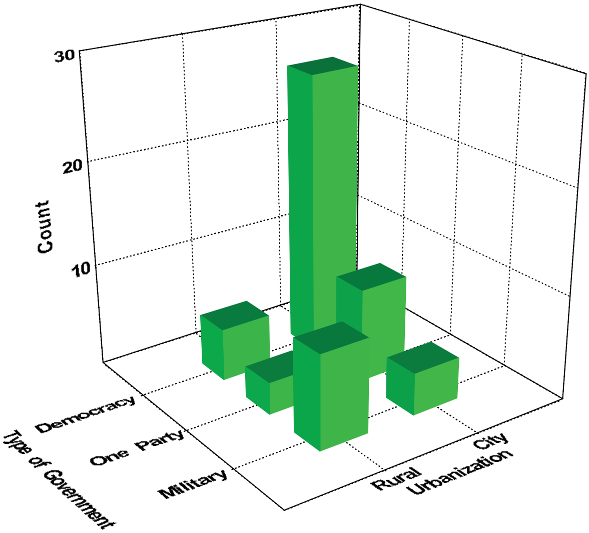


Figure 3.6 *Count bar chart*

3.3.3 Mathematical Functions

Data for a graphing system may exist only as an abstraction. This happened in a system developed for automatically graphing interesting subsets of mathematical functions (Wilkinson, 1993a). The data for this system are expressed by an equation. The system examines the equation to determine bounded, periodic, and asymptotic behavior and then chooses a suitable range and domain for graphing the function. The system then renders the function based on its analysis.

Figure 3.7 shows an example of such a plot using a polar arc-tangent function. The system determined that the behavior outside of [–5 < *x,y* < +5] in the domain could be inferred relatively accurately by looking at the behavior in the visibly graphed domain. It scaled the range to [0 < *f*(*x, y*) < .9] to encompass the interesting behavior there as well.

DATA: x, y = *mesh*(*min*(–5, –5), *max*(5, 5))
TRANS: z = (x^2+y^2)/*sqrt*(x^2+y^2
COORD: *rect*(*dim*(1, 2, 3))
ELEMENT: *surface*(*position*(x*y*z))

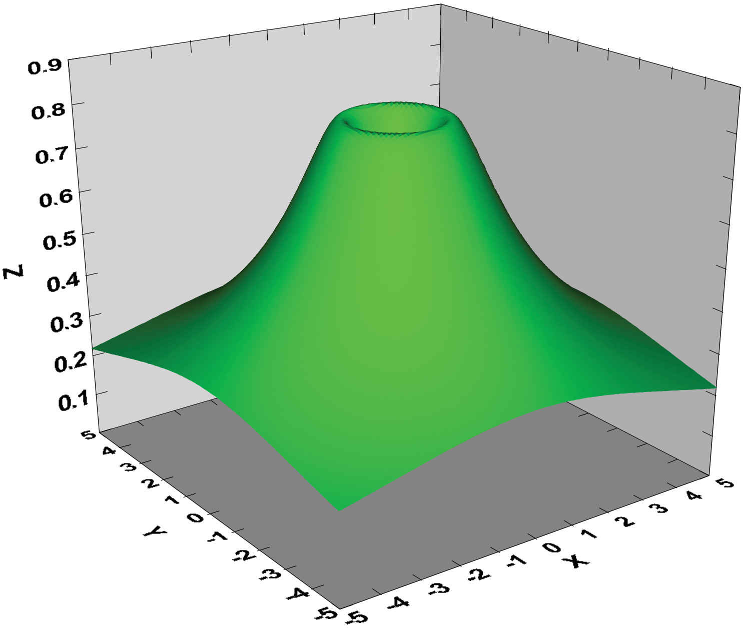


Figure 3.7 *Automated function plot*

It is easy to miss the point in the example because of the existence of symbolic and numeric mathematics systems which plot functions by generating a regular or irregular mesh of data points within a specified range and domain. The searchable dataset in the system described here, by contrast, is theoretically infinite. It varies depending on the function being examined. For some functions, the system spends most of its time examining values densely in a local neighborhood and for others, it ranges far-and-wide looking for global behavior. Thus, abstract data are not static values generated by a theoretical function and residing in memory or a file. They are instead produced by a mathematical system or simulation algorithm that is capable of changing its search space dynamically. Abstract datasets are not massive. They are infinite.

3.4 Metadata

Because our thoughts and observations are structured by the situation in which we make them, data always include **metadata**. These are the facts about the setting and circumstances under which we made our observations. Metadata are data about data. Our countries data, for example, include information about the statistical reporting methods for each country, the reliability and biases in the reports, types of missing data, the collection of these reports through an agency of the United Nations, the encoding into a computer file, the distribution of the file via electronic and printed methods, as well as numerous other details. Metadata affect the way we select data for and interpret data in our graphics. They can influence our display formally through imputation models (Rubin, 1987) or informally through written annotations attached to the graphic.

Several fields of study make use of metadata in different ways. Database metadata define descriptions of fields, columns, domains, and sometimes constraints. A primary advantage of this technique is that applications can discover the structure of information contained in a database at run time. Web applications encode metadata in hidden tags used within the page to provide keywords or other summary information about what a web page contains. This information is specifically hunted by search engines for classifying web pages. In geographic systems, information regarding area, perimeter, topography, or demography is considered metadata and stored separately from the polygons and images. In statistical graphics, the ancillary statistics calculated to produce a graphic can be considered metadata relevant to the graphic. For example, a box plot produces inner and outer outliers, hinge points, fence points and a median. All of this information can be structured in a way that parallels the geometry of the resulting graphic as metadata.

Combining contextual metadata with the graphics grammar can enable us to generate graphics automatically from user queries. Assuming table- and column-level metadata are available to describe logical, metric, and statistical aspects of variables, we can generate grammar rules to produce one or more graphics relevant to the query.

3.5 Data Mining

Recent developments in data warehousing, data mining, and knowledge discovery in databases (KDD) have blurred the distinctions between graphics, statistics, and data management. Indeed, some proponents of KDD have claimed that it will replace graphical and statistical analysis packages in the future. These claims, however extreme, must be examined in terms of models rather than commercial products. It is easy to mimic functionality and wrap old designs in new interfaces in commercial software, so product comparisons are generally not good ways to evaluate such claims.

It is difficult to discuss this field without invoking its acronyms. A popular one is OLAP, or on-line analytical processing. OLAP is intended to describe a computing environment that provides multiple views of data in real time (or, at least, responsive-to-my-needs time). These views are composites of text and graphics that can be manipulated by the user to develop alternative views. There are two main approaches to implementing an OLAP system: Multidimensional OLAP (MOLAP), and Relational OLAP (ROLAP). We will discuss these first and then conclude with a brief introduction to visual query of databases.

3.5.1 MOLAP

MOLAP depends on a basic object called a **data cube**. This object is a multidimensional array. Each dimension of this array is a set of sets representing a content domain such as time or geography. The dimensions are scaled categorically (region of country, state, quarter of year, week of quarter, etc.) so that the whole object defines a multidimensional array of cells. The cells of this array contain aggregated measures of variables such as income or gender. The data cube *is* the data for a MOLAP. In fact, this cube is usually prepared before the user begins to explore patterns in the data, sometimes in overnight processing runs on massive relational databases.

Exploring relations in the cube involves several operations. The popular names for these are **drill-down**, **drill-up,** and **drill-through**. Drill-down involves splitting an aggregation into subsets. For example, data for regions on a geographic dimension may be split into data for states. Drill-up is the reverse of drill-down; the user aggregates subsets on a dimension. Drill-through involves subsets of crossings of sets. The user might investigate statistics within a particular state and time period, for example.

These operations work well with statistics like counts, means, proportions, and standard deviations. Simple aggregations over subclasses can be computed by operating on sums, sums of squares, and other terms that are combined in linear functions to produce basic summary statistics. They do not work properly with statistics like the median, mode, and percentiles because the aggregate of these statistics is not the statistic of their aggregates. The median of medians is not the median of the aggregate, for example. A telling indicator of this fact is the lack of medians and modes from the menus of MOLAP front-ends.

It is unnecessary to provide examples to illustrate the importance of these alternative summary statistics. Every statistics student is introduced to the differences between the mean and median within the first few weeks of the first course in high school or college. One can find few governmental or commercial summaries based on mean income, mean housing prices, or mean cholesterol in the population. The median is used on these variables because of the extreme skewness in their distributions or because of the prevalence of outliers. And for marketing data, the mean cannot substitute for the modal choice over alternatives in a study of preferences.

Few of the graphics in this book and in other important applications can be computed from a data cube. There are several reasons for this situation. First, many of the statistics presented in Chapter 7 require raw data for their computation. These statistics give graphs the ability to represent sampling error within subclassifications. Second, spatial and time series statistics require raw data because the distribution of errors across space and time is not independent of space and time. Third, aggregation often involves weighting of subclassifications by measures other than sample size. This has emerged as a focal problem in the governmental deliberations over census taking versus sampling. Glymour *et al.* (1996) discuss other factors that disqualify the data cube from serious consideration as an informative database exploration model.

A recent exception to this evaluation is a model presented in Sarawagi *et al.* (1998). While the system they describe is not designed for producing general graphics, it does incorporate a statistical model at the cube-building phase that allows later exploration of the cube to be guided by robust statistical procedures. Outlier detection based on these models is used to drive the coloring of surprising cells in the cube so that ordinary users are drawn to investigate anomalies further. Their model could be used as one approach toward adapting cubes to more general graphical displays.

3.5.2 ROLAP

ROLAP places an interactive data-view at the front-end or client-side of a relational database. Some ROLAPs are based on a data cube model and can be disqualified from serious consideration for reasons similar to the ones we have given in the previous section. A more sophisticated ROLAP model has emerged recently, however. It is possible, through several technologies, to give statistical algorithms access to raw data through the relational model in real time. This is an approach we advocate in this book.

One method is to use extended Structured Query Language (SQL) to create relations that can be presented to other language clients (Java, *C*++, etc.) for use as variables. This approach requires either storage of the data on the client side (the preferable arrangement) or slow, case-by-case processing across a communications link (network or Internet). Nevertheless, this method provides access to the raw data and, if an extension to SQL is used, allows the kind of data reshapings that are featured in this and the next chapter.

Another method has become available with the advent of platform-independent languages like Java and the encapsulation offered by software **components**. These components permit remote invocation of methods that ordinarily would be an internal part of an executing program. In this way, statistical modules can be sent to servers or remote sites to process data locally and return summaries and other statistical objects. This approach is more promising than the rigid aggregations offered by structures such as data cubes. In the most elegant form of this architecture, applications can request remote components to provide information about their data-handling methods and take suitable action depending on the returned information. In this form, component architecture can achieve the real promise of distributed computing: design and execution that are independent of site, operating system, or language.

3.5.3 Visual Query of Databases

In a MOLAP or ROLAP, graphical displays are driven by the relational structure of a database. Queries through a language such as SQL yield tables that can be graphed. Researchers are beginning to develop approaches that reverse this dependency: they are designing search methods that are driven by visual query languages rather than relational languages. The original impetus for these methods came from Geographic Information Systems (GIS), where spatial relations rather than variable relations necessarily govern the search structure.

Papantonakis and King (1995) devised a graphical query language called GQL, which functions like SQL but operates on graphical objects rather than relational variables. Derthick, Kolojejchick, and Roth (1997) have extended this model to allow dynamic exploration rather than static queries. In their system, graphics are linked views into a database; manipulating graphical objects immediately updates data objects and updating data objects immediately updates graphical objects. Many of the graphical methods for accomplishing these actions are derived from the work of the original Bell Labs statistics group (see Cleveland and McGill, 1988).

Chi *et al.* (1997) have used a spreadsheet metaphor to organize graphical exploration. Their implementation is a prototype, but it can integrate multiple data sources in a single display. This allows exploration through dynamically linked views. This work resembles the trellis graphic of Becker and Cleveland (1996) but is designed to be a controller as well as a display.

The graphics algebra presented in this book is related to this research but comes from the opposite direction — derived from the structure of graphics rather than the structure of data. We can expect to see convergence in database query methods and graphic displays in coming years. Graphics will evolve beyond passive displays and will begin to play a role in the organization of data. This trend is driven by the need to get beyond static query to real-time interaction.

3.6 Sequel

Now that we have data assembled into a reference system, we need to link data to theoretical constructs. The next chapter covers variables and variable sets (varsets). These are the entities that graphs describe.

4

Variables

The word **variable** is derived from the Latin *variare*, to vary. A variable gives us a method for associating a concept like *income* with a set of data. Variables are used together with operators in the syntactical portion of our specification language (*e.g.*, death*birth).

In older statistical graphics systems, a variable refers to a column in a rectangular cases-by-variables file. Many statistical packages assume rows are cases or observations that comprise instances or samples of a variable represented by the column. There is nothing in our definition of a variable which requires it to represent a row or column, however. The only requirement is that the variable mapping function return a single value in the range for every index. This generality is especially important for graphing geometric and spatial data (Cressie, 1991). But it affects the way we approach other data as well. In a landmark book, now out of print and seldom read by statisticians, Coombs (1964) examined the relationship between structural models and patterns of data. Like Guttman (1971, 1977), Coombs believed that the prevalent practice of modeling based on cases-by-variables data layouts often prevents researchers from considering more parsimonious structural theories and keeps them from noticing meaningful patterns in their data.

In computer languages, a variable is a symbol for a data structure or container. The data contained in the structure are assumed to vary from some state of a program to another. The index function for a variable in a computer language is often called an **address**. Some computer languages type variables (logicals, strings, integers, reals, etc.) by confining their ranges to one of these or some other data types. Typing can prevent undefined or nonsensical results with some operations (adding apples and oranges), but there is nothing in our definition of a variable to require typing based on data classes. Some operations on variables do not require type compatibility. We *do* require that scales (see Chapter 6) map sets of variables to a common range, however. Otherwise, some coordinate transformations covered in Chapter 9 would not be possible.

4.1 Transforms

Transforms are transformations on variables. One purpose of transforms is to make statistical operations on variables appropriate and meaningful. Another is to create new variables, aggregates, or other types of summaries.

Table 4.1 shows several variable transforms. This sample list is intended to cover the examples in this book and to be a template for designing the signature and behavior of new functions. We omit ordinary algebraic operators and standard functions here, but will use them elsewhere in the book.

Table 4.1 **Variable Transforms**

Mathematical	***Statistical***	***Multivariate***
log(x)	*mean*(x)	*sum*(x_1,x_2,...,x_n)
exp(x)	*median*(x)	*diff*(x_1,x_2)
sin(x)	*mode*(x)	*prod*(x_1,x_2)
cos(x)	*residual*(x,y)	*quotient*(x_1,x_2)
tan(x)	*sort*(x)	*influence*(x_1,x_2,...,x_n)
asin(x)	*rank*(x)	*miss*(x_1,x_2,...,x_n,"<*f*>")
acos(x)	*prank*(x)	
atan(x)	*cut*(x,*k*)	
atanh(x)	*zinv*(x)	
sign(x)	*lag*(x)	
pow(x, *p*)	*grpfun*(x,g,"<*f*>")	

The mathematical functions return values that are case-by-case transformations of the values in x, where x is a varset. The *log* function returns natural logarithms. The exponential and trigonometric functions are standard. The *sign*() function returns 1 if a value of x is positive and –1 if it is negative. The *pow*() function transforms x to the *p*th power of itself.

The statistical functions compute basic statistics. The *mean*, *median*, and *mode* fill each value in the column with the corresponding column summary statistic. The *residual* function returns the residuals of a regression of y on x. It has several methods for regression type, with *linear* being the default. The *sort* function orders all variables in a variable set according to the sorted order of x. The *rank* function fills a column with the ranks of x, assigning fractional ranks when there are ties. The *prank* function fills each case in the column with the value (*i*–.5)/*n* for *i* = 1, ... , *n* cases assuming (as if) the column were sorted on x. The *cut* function cuts a sorted column into *k* groups, replacing the value of the argument with an integer from 1 to *k*. The *zinv* function replaces a value with the inverse normal cumulative distribution function value. The *lag* function replaces value x_i with $x_{i\text{-}1}$, setting the first value in the column to missing. It has an optional second argument, a positive or negative integer specifying the amount and direction of shifting. The *grpfn* function is a routine that evaluates the function *f* named between the quotes separately for each of the groups specified by the ***g*** argument in the parameter list.

The multivariate functions use more than one variable to construct a new variable. The *sum* function fills a column with the sum of values of its arguments. The *diff* function does the same for differences. The *prod* and *quotient* functions compute products and ratios. The *influence* function computes an influence function (see Figure 10.34). The *miss* function imputes missing values assuming a function named *f* (see Figure 20.1).

4.2 Examples

The following examples have been constructed to show that transforms are more powerful than simple recodings of variables. Together with data functions, transforms can help us create graphics that employ structures unavailable to ordinary graphics systems.

4.2.1 Sorting

We once met a statistician who worked for the FBI. He had helped uncover the Chicago Machine's voting fraud in the 1970's. We were curious about the methods the Federal team had used to expose the fraud, so we asked him about discriminant analysis, logistic regression, and other techniques the statisticians might have used. He replied, "We sorted the voter tape and looked for duplicate names and addresses." This statistical methodology may have been inspired by the fabled Chicago Machine slogan, "Vote early and often."

Sorting is one of the most elementary and powerful methods of statistical and graphical analyses. A sort is a one-to-one transformation. When we use position to represent the values of a variable, we are implicitly sorting those values. Sorting variables displayed by position not only reveals patterns but also makes it easier to make comparisons and locate subsets. Sorting categorical variables according to the values of associated numerical variables can itself constitute a graphical method.

Figure 4.1 shows an example. The data are numbers of arrests by sex reported to the FBI in 1985 for selected crimes. The graphic displays the differences in proportions of each crime committed by males and females. To create the graphic, we must standardize the data within crime category (dividing by the row totals). The first two TRANS statements accomplish this. The next TRANS statement creates an mf varset consisting of the difference in crime proportions. We sort this varset and then plot it against the crime categories. The pattern of the dots indicates that males predominate in almost every crime category except vice and runaways. The largest biases are in the violent crimes. Rape, not surprisingly, is almost exclusively male. For brevity, we have omitted the GPL for the bipolar arrow annotation at the bottom.

```
TRANS: total = sum(male, female)
TRANS: m = quotient(male, total)
TRANS: f = quotient(female, total)
TRANS: mf = diff(m, f)
TRANS: mf = sort(mf)
ELEMENT: point(position(mf*crime))
```

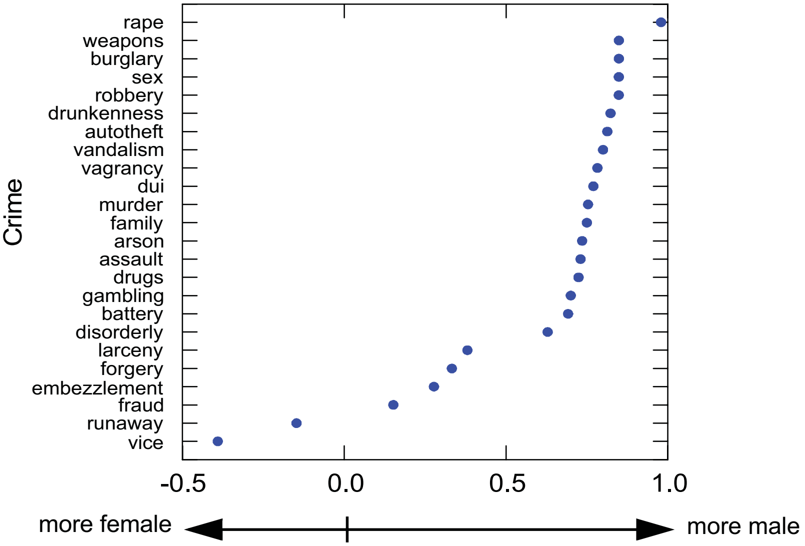


Figure 4.1 *Gender differences in crime patterns*

4.2.2 Probability Plots

Probability plots compare the *n* ordered values of a variable to the *n*-tiles of a specified probability distribution. If we have 100 values, for example, we base a normal probability plot on the pairs (x_1, z_1), (x_2, z_2), ... , (x_{99}, z_{99}), where x_i are the data values ordered smallest to largest and z_i is the lower 100α percentage point of the standard normal probability distribution. To fit all 100 values, we change the computation of α from *i*/100 to (*i*-.5)/100. If a probability plot follows roughly a diagonal straight line, then we infer that the shape of our sample distribution is approximately normal.

Figure 4.2 shows a probability plot of military expenditures from the countries data. Our first transformation is *prank*(), a proportional-rank that computes the value (*i*–.5) / *n*, *i* = 1 , ... *n*, corresponding to the values x_i after an ascending sort. Instead of reordering the data, however, *prank*() returns the values in the original data sequence. Next, the *zinv*() function computes the values of the cumulative standard normal distribution corresponding to the points produced by *prank*(). In other words, we apply the *zinv* function to the *prank*() values to get the theoretical normal variables we plot the data against (students usually have difficulty with this two-fold indirection). The left panel of Figure 4.2 shows the raw military values and the right panel shows the plot on a log_{10} scale. Logging straightens out the plot, which is another way of saying that it transforms the distribution from a positively skewed shape to a more normal one.

A probability plot can also be produced through a probability scale transformation. We will show an example in Figure 6.13. We compute *prank*() and then rescale the α values through a probability scale transformation. The method in this chapter produces a new variable *zinv*(), while the scale method in Chapter 6 changes only the scale on which *prank*() is plotted.

```
TRANS: alpha = prank(military)
TRANS: z = zinv(alpha)
ELEMENT: point(position(military*z))

TRANS: alpha = prank(military)
TRANS: z = zinv(alpha)
SCALE: log(dim(1), base(10))
ELEMENT: point(position(military*z))
```

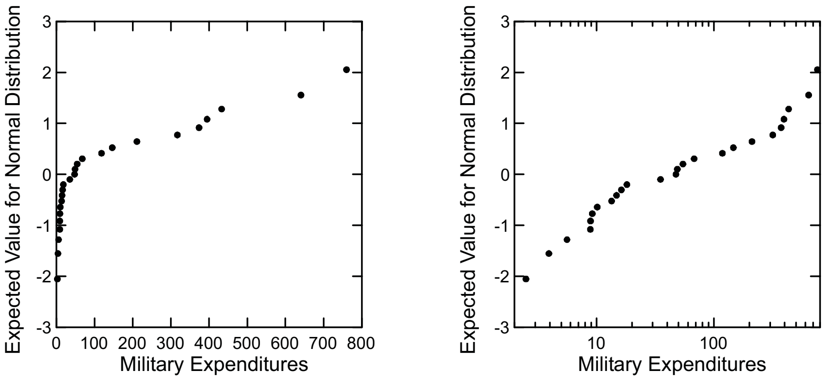


Figure 4.2 *Probability plots of military expenditures*

4.2.3 Aggregating Variables

Figure 4.3 shows a box plot (see Section 8.1.1.6) of the birth rate data. This plot divides a variable into a set of fractiles and then displays the variation in each fractile against the median value of the variable within that fractile. It can be used to determine whether we need a transformation for **heteroscedastic** data, in which the spread increases (or decreases) proportionally to some power of the location. See Section 15.3.1.1 for a similar plot.

The *grpfun*() function computes a function (*median*) on a variable (birth) within each separate group defined by a grouping variable (quartile). Notice that the frame crosses the same variable (birth) with itself, but the *position*() function for the box graph selects the values of birthquart to fix the location of each box.

```
TRANS: quartile = cut(birth, 4)
TRANS: birthquart = grpfun(birth, quartile,"median")
ELEMENT: schema(shape(shape.box), position(birthquart*birth))
```

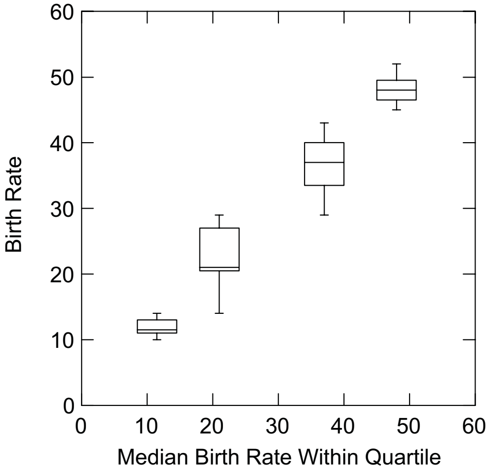


Figure 4.3 *Box plot of a variable against its quartile medians*

4.2.4 Regression Residuals

Figure 4.4 shows a linear regression residuals plot. The independent variable in the regression is the birth rate variable from the countries data and the dependent variable is death rate. The residuals are calculated from this regression, standardized, and then used as the dependent variable in the frame model for Figure 4.4. The U-shaped pattern of residuals suggests that the linear model is not a good representation of the relationship between death rates and birth rates for the countries.

One might ask whether statistical procedures have a useful role within a graphical system. Two alternatives are traditional graphics packages, which receive pre-calculated data from spreadsheets and statistics packages, and traditional statistics packages, which send their calculations to basic graphics sub-systems. Both of these are compromises, however. Obviously, a graphics system should not incorporate the entire range of procedures within a statistical package; this would reduce its focus. However, there are analyses that can be performed only when graphics and statistics are intimately tied.

Exploratory statistical packages such as Data Desk implement such a model. Embedding statistical procedures such as regression and correlation in a graphics system provides functions that are unavailable in other software. In the end, the grammar of graphics has much to do with the grammar of statistics. We will discuss this further at the end of Chapter 7.

TRANS: residual = *residual.linear.student*(birth, death)
ELEMENT: *point*(*position*(birth*residual))

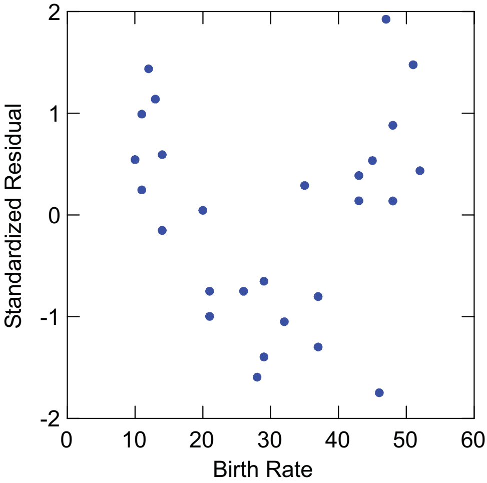


Figure 4.4 *Studentized residual plot*

4.3 Sequel

Now that we have variables to describe relationships through links to data, we need a system for expressing those relationships. The next chapter covers the formal definitions for varset algebra.

5

Algebra

The word **algebra** is derived from the Arabic *al-jebr*, which means the restoring or reunion of broken parts. Its use in the West dates from the publication in the 9th century of Muhammad ibn Musa al-Khwarizmi's *Book of Restoring and Balancing*. Khwarizimi's name gave rise to the word **algorithm**. A classic discussion of the origins of algebra is given in Jourdain (1919).

This chapter deals with restoring and balancing sets of variables in order to create the specification for the frames in which graphs are embedded. The first part of a specification contains the algebraic expression relating sets of variables. We will review the rules for syntactical expressions and then present examples of typical expressions.

5.1 Syntax

5.1.1 Symbols

A symbol is used to represent an entity operated on by an algebra. The symbols in varset algebra are varsets. We will use capital italic letters for these names in this chapter. This notation emphasizes that we are dealing with sets when we do these operations. In examples involving algebraic specifications on variables based on real data, we will use lowercase sans-serif names of variables. We will also use a special variable (**1**), which represents the unity variable. Its range is one unity value. When we make a scale for this variable, no tick marks or scale values appear, but unity is located at the middle of the scale.

5.1.2 Operators

An **operator** is a method for relating symbols in an algebra. There are three operators in the graphical system, called *cross*, *nest,* and *blend*. We will introduce them with simple tabular examples and then show the resulting mappings for typical varsets A and B and their value domains V_{A} and V_{B}.

5.1.2.1 Cross (∗)

Cross joins the left varset with the right to produce a set of tuples:

$$\begin{array}{|c|}\hline x\\\hline y\\\hline z\\\hline\end{array} \; * \; \begin{array}{|c|}\hline a\\\hline a\\\hline b\\\hline\end{array} \; = \; \begin{array}{|c|c|}\hline x & a\\\hline y & a\\\hline z & b\\\hline\end{array}$$

The resulting set of tuples is a subset of the product of the domains of the two varsets. The domain of a varset produced by a cross is the product of the separate domains. One may think of a cross as a horizontal concatenation of the tabular representation of two varsets, assuming the rows of each varset are equivalent and in the same order. The definition of cross is:

$$\mathsf{A} * \mathsf{B} : (s, t) \rightarrow \mathsf{A}(s) \cap \mathsf{B}(t)$$

$$V_{\mathsf{A} * \mathsf{B}} = \{(s, t), \forall s \in V_{\mathsf{A}}, \forall t \in V_{\mathsf{B}}\}$$

The following example shows the mappings and domains for a crossing of two varsets using simple integer keys for the objects:

$$\begin{array}{l} \mathsf{A}:\; \textit{red} \rightarrow \langle 1, 4\rangle \\ \quad\;\; \textit{blue} \rightarrow \langle 2, 3\rangle \\ V_{\mathsf{A}} = \{\textit{red}, \textit{blue}\} \end{array}$$

$$\begin{array}{l} \mathsf{B}:\; -10 \rightarrow \langle 1\rangle \\ \qquad\quad 5 \rightarrow \langle 2, 3\rangle \\ \qquad\; 10 \rightarrow \langle 4\rangle \\ V_{\mathsf{B}} = [-10, 10] \end{array}$$

$$\begin{array}{l} \mathsf{A{*}B}:\; (\textit{red}, \text{-}10) \rightarrow \langle 1\rangle \\ \qquad\quad (\textit{blue}, 5) \rightarrow \langle 2, 3\rangle \\ \qquad\quad (\textit{red}, 10) \rightarrow \langle 4\rangle \\ V_{\mathsf{A{*}B}} = \{\textit{red}, \textit{blue}\} \times [-10, 10] \end{array}$$

5.1.2.2 Nest (/)

Nest produces a varset that looks like the result of a cross:

$$\begin{array}{|c|}\hline x\\\hline y\\\hline z\\\hline\end{array} \; / \; \begin{array}{|c|}\hline a\\\hline a\\\hline b\\\hline\end{array} \; = \; \begin{array}{|c|c|}\hline x & a\\\hline y & a\\\hline z & b\\\hline\end{array}$$

The domain of a nest is *not* a subset of the domain of a cross, however. Even though they look the same, the tuples of a nest are colored or tagged and the tuples of a cross are not. The definition of nest is

$$\begin{aligned}\mathsf{A/B} &: (s, t) \rightarrow \mathsf{A}(s) \cap \mathsf{B}(t) \\ V_{\mathsf{A/B}} &= \{(s, t), \forall s \in V_{\mathsf{A}}, \forall t \in V_{\mathsf{B}} : \mathsf{A}(s) \cap \mathsf{B}(t) \neq \varnothing\}\end{aligned}$$

The nesting variable tags the nested variable such that (s, t) implies $(s : t)$. We will not use this notation below, but we will assume an algebra system can identify the nesting elements through a tagging index. The domain for a nest must be pre-defined. To construct a nested domain, we have three options:

1) Data values — identify the minimal domain containing the data by enumerating unique data tuples.
2) Metadata — define the domain using external rules contained in a metadata resource or from known principles.
3) Data organization — identify nested domains using the predefined structure of a hierarchical database or OLAP cube.

The following example shows a nesting of two categorical variables:

$$\begin{aligned}\mathsf{A}: \quad & \textit{ant} \rightarrow \langle 1 \rangle \\ & \textit{fly} \rightarrow \langle 2, 3 \rangle \\ & \textit{bee} \rightarrow \langle 4 \rangle \\ V_{\mathsf{A}} &= \{\textit{ant}, \textit{fly}, \textit{bee}\}\end{aligned}$$

$$\begin{aligned}\mathsf{B}: \quad & \textit{noun} \rightarrow \langle 1, 2, 4 \rangle \\ & \textit{verb} \rightarrow \langle 3 \rangle \\ V_{\mathsf{B}} &= \{\textit{noun}, \textit{verb}\}\end{aligned}$$

$$\begin{aligned}\mathsf{A/B}: \quad & (\textit{ant}, \textit{noun}) \rightarrow \langle 1 \rangle \\ & (\textit{fly}, \textit{noun}) \rightarrow \langle 2 \rangle \\ & (\textit{fly}, \textit{verb}) \rightarrow \langle 3 \rangle \\ & (\textit{bee}, \textit{noun}) \rightarrow \langle 4 \rangle \\ V_{\mathsf{A/B}} &= \{(\textit{ant}, \textit{noun}), (\textit{fly}, \textit{noun}), (\textit{fly}, \textit{verb}), (\textit{bee}, \textit{noun})\}\end{aligned}$$

If we look at the first entry in each tuple resulting from a nesting, then we can interpret nesting as an operation that defines meaning conditionally. In this example, the meaning of *fly* is ambiguous unless we know whether it is a noun or a verb. Furthermore, there is no verb for *ant* or *bee* in the English language, so the domain of A/B does not include this combination. In a sense, there is a *dependency* in the entries of the tuple defined by a nest that does not exist in the entries defined by a cross. And this dependency is described by interpreting the first entry of a tuple conditioned on the value of the second.

If A is a continuous variable, then we have something like the following:

$$
\begin{aligned}
\mathsf{A}\colon\ 0 &\rightarrow \langle 1 \rangle \\
8 &\rightarrow \langle 2 \rangle \\
1.4 &\rightarrow \langle 3 \rangle \\
3 &\rightarrow \langle 4 \rangle \\
10 &\rightarrow \langle 5, 6 \rangle
\end{aligned}
$$

$$V_{\mathsf{A}} = [0, 10]$$

$$
\begin{aligned}
\mathsf{B}\colon\ 1 &\rightarrow \langle 1, 2, 3 \rangle \\
2 &\rightarrow \langle 4, 5, 6 \rangle
\end{aligned}
$$

$$V_{\mathsf{B}} = \{1, 2\}$$

$$
\begin{aligned}
\mathsf{A/B}\colon\ (0, 1) &\rightarrow \langle 1 \rangle \\
(8, 1) &\rightarrow \langle 2 \rangle \\
(1.4, 1) &\rightarrow \langle 3 \rangle \\
(3, 2) &\rightarrow \langle 4 \rangle \\
(10, 2) &\rightarrow \langle 5, 6 \rangle
\end{aligned}
$$

$$V_{\mathsf{A/B}} = \{[0, 8] \times \{1\}, [3, 10] \times \{2\}\}$$

In this example, the elements of the nesting A/B result in intervals conditioned on the values of B. A represents 6 ratings (ranging from 0 to 10) of the behavior of patients by two psychiatrists. B represents the identity of the psychiatrist making each rating. The intervals [0, 8] and [3, 10] imply that psychiatrist 1 will not use a rating greater than 8 and psychiatrist 2 will not use a rating less than 3. Nesting in this case is based on the (realistic) assumption that the two psychiatrists assign numbers to their perceptions in a different manner. A rating of 2 by one psychiatrist cannot be compared to the same rating by the other, because of possible differences in location, scale, and even local nonlinearities. Much of psychometrics is concerned with the problem of equating ratings in this type of example so that nesting would not be needed, although it is not always possible to do so plausibly.

The name *nest* comes from design-of-experiments usage (*e.g.*, Neter, Wasserman, and Kutner, 1990). We use the word *within* to describe its effect. For example, the phrase "teachers *within* schools" means that teachers are nested within schools. If each teacher works at only one school, two teachers with the same name at different schools are different people.

Those familiar with experimental design may recognize that the expression A/B is equivalent to the notation A(B) in a design specification. Both expressions mean "A is nested within B." Statisticians' customary use of parentheses to denote nesting conceals the fact that nesting involves an operator, however. Because nesting is distributive over blending, we have made this operator explicit and retained the conventional mathematical use of parentheses in an algebra.

5.1.2.3 Blend (+)

Blend produces a union of varsets:

$$\begin{array}{|c|}\hline x\\\hline y\\\hline z\\\hline\end{array} + \begin{array}{|c|}\hline a\\\hline a\\\hline b\\\hline\end{array} = \begin{array}{|c|}\hline x\\\hline y\\\hline z\\\hline a\\\hline a\\\hline b\\\hline\end{array}$$

Notice that we lose information about which value came from which column. We should restrict blend to varsets with composable domains, even though we do not need this restriction for the operation to be defined. It would make little sense to blend Age and Weight, much less Name and Height. The definition of blend is

$$\mathsf{A} + \mathsf{B} : s \rightarrow \mathsf{A}(s) \cup \mathsf{B}(s)$$
$$V_{\mathsf{A}+\mathsf{B}} = V_{\mathsf{A}} \cup V_{\mathsf{B}}$$

In vernacular, we often use the conjunction *and* to signify that two sets are blended into one (although the word *or* would be more appropriate technically). For example, if we measure diastolic and systolic blood pressure among patients in various treatment conditions and we want to see blood pressure plotted on a common axis, we can plot diastolic *and* systolic against treatment. The following example shows a blending of two varsets:

$$\begin{aligned}\mathsf{A}:\quad 0 &\rightarrow \langle 1\rangle\\ 120 &\rightarrow \langle 2\rangle\\ 90 &\rightarrow \langle 3, 4\rangle\end{aligned}$$
$$V_{\mathsf{A}} = [0, 120]$$

$$\begin{aligned}\mathsf{B}:\quad 10 &\rightarrow \langle 1\rangle\\ 200 &\rightarrow \langle 2, 3\rangle\\ 90 &\rightarrow \langle 4\rangle\end{aligned}$$
$$V_{\mathsf{B}} = \{10, 200\}$$

$$\begin{aligned}\mathsf{A+B}:\quad 0 &\rightarrow \langle 1\rangle\\ 10 &\rightarrow \langle 1\rangle\\ 120 &\rightarrow \langle 2\rangle\\ 90 &\rightarrow \langle 3, 4, 4\rangle\\ 200 &\rightarrow \langle 2, 3\rangle\end{aligned}$$
$$V_{\mathsf{A+B}} = [0, 200]$$

5.1.2.4 Shorthand

For convenience, we occasionally make use of two operator aliases that reduce the complexity of graphics algebra expressions. The first is "exponentiation"

$$X\hat{}2 = X*X$$

$$X\hat{}3 = X*X*X$$

The second is a "dot-cross" operation (assuming the arguments conform)

$$(X + Y) \bullet (A + B) = X*A + Y*B$$

Of course, this operation is practical only if we have our hands on X, Y, A, and B. Once they are blended, we cannot disentangle them.

Neither of these additional operators is necessary, but they are convenient. Inventing new operations leads to a question: Can we get along with only three operators or is that too many? It is not easy to prove necessity or sufficiency for the graphical applications of the three operators in this chapter because the domain of applications is ill-defined. Nevertheless, we claim that only three operators are necessary and sufficient for doing statistical graphics. The bases for this claim are several assertions.

First, these operators have reproduced published graphics via an automated system. They have been tested against a set of graphics we have collected over a fifteen-year period. Second, there were four operators in this system before we discovered that one could be replaced with a simple combination of two others. We have not found any straightforward way to collapse the current three into two and have not found a published graphic that requires four.

In evaluating these assertions, one must not interpret these three operators as data transformations or as means for organizing data before plotting — as if we were rearranging cells in a spreadsheet prior to producing a graphic. We have seen those types of data operations in Chapter 3. They occur before algebraic operations on variables. If we replaced operators with *ad hoc* combinations of rows and columns of data, then we would lose the connection between variables and graphics. An automated system would not be able to respond to queries concerning variables. We will discuss some of these issues further at the end of this chapter when we compare graphics algebra to other algebras.

5.1.3 Rules

We separate each of the following proofs into two parts. First, we show mappings are the same for the left and right varset expressions in each equality colored red; that is, we show $L(s) = R(s)$, where L and R are the varsets on the left and right of each red equality and s is a value in both domains. Second, we show the domains are the same for the left and right varset expressions; that is, we show $V_L = V_R$.

5.1.3.1 Associativity

Cross

$$(\mathsf{X} * \mathsf{Y}) * \mathsf{Z} = \mathsf{X} * (\mathsf{Y} * \mathsf{Z})$$

$$\begin{aligned}((\mathsf{X} * \mathsf{Y}) * \mathsf{Z})(r, s, t) &= (\mathsf{X} * \mathsf{Y})(r, s) \cap \mathsf{Z}(t)\\ &= \mathsf{X}(r) \cap (\mathsf{Y} * \mathsf{Z})(s, t)\\ &= (\mathsf{X} * (\mathsf{Y} * \mathsf{Z}))(r, s, t)\end{aligned}$$

$$\begin{aligned}V_{(\mathsf{X} * \mathsf{Y}) * \mathsf{Z}} &= \{(r, s, t), \forall (r, s) \in V_{\mathsf{X} * \mathsf{Y}}, \forall t \in V_{\mathsf{Z}}\}\\ &= \{(r, s, t), \forall r \in V_{\mathsf{X}}, \forall (s, t) \in V_{\mathsf{Y} * \mathsf{Z}}\}\\ &= V_{\mathsf{X} * (\mathsf{Y} * \mathsf{Z})}\end{aligned}$$

Nest

$$(\mathsf{X}/\mathsf{Y})/\mathsf{Z} = \mathsf{X}/(\mathsf{Y}/\mathsf{Z})$$

$$\begin{aligned}((\mathsf{X}/\mathsf{Y})/\mathsf{Z})(r, s, t) &= (\mathsf{X}/\mathsf{Y})(r, s) \cap \mathsf{Z}(t)\\ &= \mathsf{X}(r) \cap (\mathsf{Y}/\mathsf{Z})(s, t)\\ &= (\mathsf{X}/(\mathsf{Y}/\mathsf{Z}))(r, s, t)\end{aligned}$$

$$\begin{aligned}V_{(\mathsf{X}/\mathsf{Y})/\mathsf{Z}} &= \{((r, s, t), \forall (r, s) \in V_{\mathsf{X}/\mathsf{Y}}, \forall t \in V_{\mathsf{Z}}) : (\mathsf{X}/\mathsf{Y})(r, s) \cap \mathsf{Z}(t) \neq \varnothing\}\\ &= \{((r, s, t), \forall r \in V_{\mathsf{X}}, \forall (s, t) \in V_{\mathsf{Y}/\mathsf{Z}}) : \mathsf{X}(r) \cap (\mathsf{Y}/\mathsf{Z})(s, t) \neq \varnothing\}\\ &= V_{\mathsf{X}/(\mathsf{Y}/\mathsf{Z})}\end{aligned}$$

Blend

$$(\mathsf{X} + \mathsf{Y}) + \mathsf{Z} = \mathsf{X} + (\mathsf{Y} + \mathsf{Z})$$

$$\begin{aligned}((\mathsf{X} + \mathsf{Y}) + \mathsf{Z})(s) &= (\mathsf{X} + \mathsf{Y})(s) \cup \mathsf{Z}(s)\\ &= \mathsf{X}(s) \cup (\mathsf{Y} + \mathsf{Z})(s)\\ &= (\mathsf{X} + (\mathsf{Y} + \mathsf{Z}))(s)\end{aligned}$$

$$\begin{aligned}V_{(\mathsf{X} + \mathsf{Y}) + \mathsf{Z}} &= \{(V_{\mathsf{X}} \cup V_{\mathsf{Y}}) \cup V_{\mathsf{Z}}\}\\ &= \{V_{\mathsf{X}} \cup (V_{\mathsf{Y}} \cup V_{\mathsf{Z}})\}\\ &= V_{\mathsf{X} + (\mathsf{Y} + \mathsf{Z})}\end{aligned}$$

5.1.3.2 Distributivity

Cross

$$\mathsf{X} * (\mathsf{Y} + \mathsf{Z}) = \mathsf{X} * \mathsf{Y} + \mathsf{X} * \mathsf{Z}$$

$$\begin{aligned}(\mathsf{X} * (\mathsf{Y} + \mathsf{Z}))(r, s) &= \mathsf{X}(r) \cap (\mathsf{Y} + \mathsf{Z})(s)\\ &= \mathsf{X}(r) \cap (\mathsf{Y}(s) \cup \mathsf{Z}(s))\\ &= \mathsf{X}(r) \cap \mathsf{Y}(s) \cup \mathsf{X}(r) \cap \mathsf{Z}(s)\\ &= (\mathsf{X} * \mathsf{Y} + \mathsf{X} * \mathsf{Z})(r, s)\end{aligned}$$

$$\begin{aligned}V_{\mathsf{X} * (\mathsf{Y} + \mathsf{Z})} &= \{(r, s), \forall r \in V_{\mathsf{X}}, \forall s \in V_{\mathsf{Y}} \cup V_{\mathsf{Z}}\}\\ &= \{(r, s), \forall (r, s) \in V_{\mathsf{X}} \cap V_{\mathsf{Y}} \cup V_{\mathsf{X}} \cap V_{\mathsf{Z}}\}\\ &= \{(r, s), \forall (r, s) \in V_{\mathsf{X} * \mathsf{Y}} \cup V_{\mathsf{X} * \mathsf{Z}}\}\\ &= V_{\mathsf{X} * \mathsf{Y} + \mathsf{X} * \mathsf{Z}}\end{aligned}$$

$$(\mathsf{X} + \mathsf{Y}) * \mathsf{Z} = \mathsf{X} * \mathsf{Z} + \mathsf{Y} * \mathsf{Z}$$

Similar to above.

Nest

$$\mathsf{X}/(\mathsf{Y} + \mathsf{Z}) = \mathsf{X}/\mathsf{Y} + \mathsf{X}/\mathsf{Z}$$

$$\begin{aligned}
(\mathsf{X}/(\mathsf{Y} + \mathsf{Z}))(r, s) &= \mathsf{X}(r) \cap (\mathsf{Y} + \mathsf{Z})(s) \\
&= \mathsf{X}(r) \cap (\mathsf{Y}(s) \cup \mathsf{Z}(s)) \\
&= \mathsf{X}(r) \cap \mathsf{Y}(s) \cup \mathsf{X}(r) \cap \mathsf{Z}(s) \\
&= (\mathsf{X}/\mathsf{Y} + \mathsf{X}/\mathsf{Z})(r, s)
\end{aligned}$$

$$\begin{aligned}
V_{\mathsf{X}/(\mathsf{Y}+\mathsf{Z})} &= \{(r, s), \forall r \in V_{\mathsf{X}}, \forall s \in V_{\mathsf{Y}+\mathsf{Z}} : \mathsf{X}(r) \cap (\mathsf{Y} + \mathsf{Z})(s) \neq \varnothing\} \\
&= \{(r, s), \forall r \in V_{\mathsf{X}}, \forall s \in V_{\mathsf{Y}} \cup V_{\mathsf{Z}} : \mathsf{X}(r) \cap (\mathsf{Y}(s) \cup \mathsf{Z}(s)) \neq \varnothing\} \\
&= \{(r, s), \forall (r, s) \in V_{\mathsf{X}} \cap V_{\mathsf{Y}} \cup V_{\mathsf{X}} \cap V_{\mathsf{Z}} : \mathsf{X}(r) \cap \mathsf{Y}(s) \cup \mathsf{X}(r) \cap \mathsf{Z}(s) \neq \varnothing\} \\
&= \{(r, s), \forall (r, s) \in V_{\mathsf{X}/\mathsf{Y}} \cup V_{\mathsf{X}/\mathsf{Z}} : \mathsf{X}(r) \cap \mathsf{Y}(s) \cup \mathsf{X}(r) \cap \mathsf{Z}(s) \neq \varnothing\} \\
&= V_{\mathsf{X}/\mathsf{Y}+\mathsf{X}/\mathsf{Z}}
\end{aligned}$$

$$(\mathsf{X} + \mathsf{Y})/\mathsf{Z} = \mathsf{X}/\mathsf{Z} + \mathsf{Y}/\mathsf{Z}$$

Similar to above.

5.1.3.3 Identity

The identity element for blend is a null set. The varset

$$\mathbf{1} : unity \rightarrow \Omega$$

$$V_{\mathbf{1}} = unity$$

where Ω covers all caseIDs, is called the **unity** varset. Its domain is the unity value. We annotate the unity value in GPL as **1**. When we make a scale for unity, no tick marks or scale values appear, but unity is located at the middle of the scale. This varset is a pseudo-identity for cross and nest. The expressions p1980 and p1980*$\mathbf{1}$, for example, produce identical appearing graphics (see Figure 5.1).

5.1.3.4 Commutativity

The first edition asserted commutativity for the blend operator. This is easy to show for setwise operations in the above notation (the proof is left to the reader). Nevertheless, there are certain geometric elements (*path*, for instance), that are order-dependent with respect to blend. Rather than place restrictions on blends, we prefer to make graphics algebra noncommutative. Our parser made no use of commutativity, so nothing changes.

5.1.4 Expressions

An **expression** is an ordered sequence of one or more symbols with operators between each pair of adjacent symbols in the sequence. A **term** is an expression with no + operator (*e.g.*, A or A*B or A∗B/C . A **factor** is a term with no * operator (*e.g.*, A or A/B). The expression A*B*C*D has one term and four factors. The expression (A + B)/C has two terms; after expanding to A/C + B/C , we can see one factor in each term. A **monomial** is an expression with one term. A **polynomial** is an expression with more than one term. The expressions A and A/B are monomials, while the expressions A + B and A + B/C are polynomials.

5.1.4.1 Algebraic Form

An **algebraic form** is a monomial or a polynomial whose terms all have the same number of factors. The **order** of an algebraic form is the number of factors in one of its terms. The expression $A * B + C * D$ is an algebraic form of order 2, but $A * B + C$ is not an algebraic form because the first term has two factors and the second has one.

If we want to identify how many dimensions there are in a graphic constructed from a general algebraic expression, and to find which variables are assigned to each dimension, we can implement a symbolic algebra machine that can convert general algebraic expressions into algebraic forms. The first stage in normalizing an expression to algebraic form is to expand the expression into a collection of monomials. Then we determine the largest order among the monomials, say *k*. Finally, we augment any monomial less than order *k* by crossing on the right with the unity element enough times to make it order *k*. For example, we expand the expression

$$G + (A + B) * C / D$$

to

$$G + A * C / D + B * C / D$$

and note that $k = 2$. Then we convert the expression to

$$G * \mathbf{1} + A * C / D + B * C / D$$

5.1.4.2 Operator Precedence

Nest takes precedence over cross and blend. Cross takes precedence over blend. This hierarchical order may be altered through the use of parentheses.

5.1.5 SQL Equivalents

We will discuss the relation of graphics algebra to relational algebras at the end of this chapter. In this section, we summarize Structured Query Language (SQL) statements that can be used to implement the algebra in a database.

5.1.5.1 Cross

Cross can be accomplished with an **inner join**:

```
select a.*, b.*
from X a, Y b;
where a.key = b.key;
```

5.1.5.2 Nest

Nest can be accomplished through a SQL `nest` operator. This operator requires that the database allow tables as primitives, either as relation-valued attributes (Date and Darwen, 1992) or as nested tables (Abiteboul *et al.*, 1989). If the database does not support the `nest` operator, we can accumulate the subset of tuples in a nest operation with an inner join:

```
select a.*, b.*
from X a, Y b;
```

We then must remember to treat the entries from the right-hand table as tags for the entries on the left.

5.1.5.3 Blend

Blend is performed through `union all`:

```
select * from X
union all
select * from Y;
```

If `union all` is not available, we can concatenate extra key columns to be sure that all rows appear in the result set.

5.1.5.4 Composition and Optimization

SQL statements can be composed by using the grammar for chart algebra. Compound statements can then be submitted for optimization and execution by a database compiler. Preferably, pre-optimization can be performed on the chart algebra parse-tree and the result used to generate SQL. Secondary optimization is then performed by the database compiler.

5.2 *Examples*

We will begin with the simplest syntax for a one-dimensional graph. Figure 5.1 is a scatterplot of the 1980 population of selected world cities. Because there is only one dimension, the *point* cloud designates points on the number line at the values of the data.

ELEMENT: *point*(*position*(pop1980))

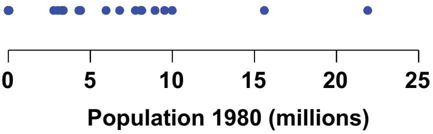


Figure 5.1 *One-dimensional scatterplot*

5.2.1 Cross

Figure 5.2 shows a two-dimensional scatterplot. The 1980 and 2000 city populations are plotted against each other. There is one frame with two dimensions and data values represented by position of points.

ELEMENT: *point*(*position*(pop1980*pop2000))

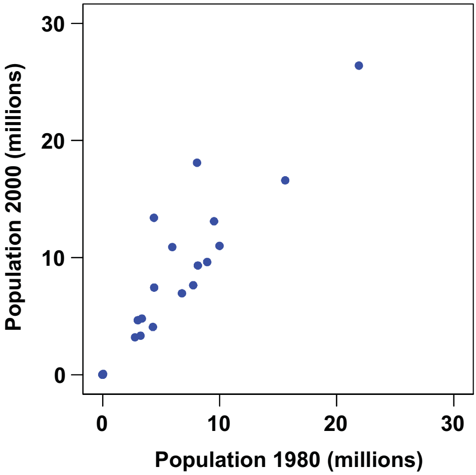


Figure 5.2 *Two-dimensional scatterplot*

If one of the crossed variables is categorical, then it *splits* any graphs in the frame into as many categories as there are in the crossing. Figure 5.3 shows an example of this splitting. We plot 2000 population against the names of the cities. There are 17 sets of points in this frame.

ELEMENT: *point*(*position*(city*pop2000))

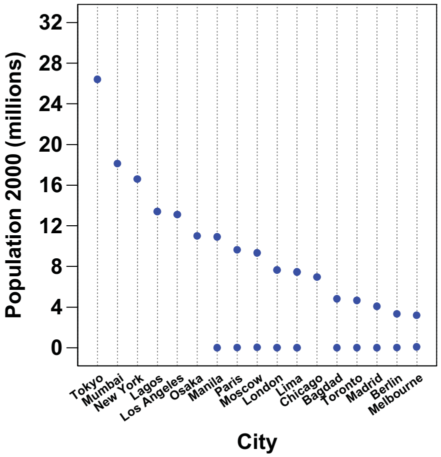


Figure 5.3 *Two-dimensional dot plot*

We notice there are two points for some of the cities. During various periods in US history, it was fashionable to name towns and cities after their European and Asian counterparts. Sometimes this naming was driven by immigration, particularly in the colonial era (*New Amsterdam, New York, New London*). At other times, exotic names reflected a fascination with foreign travel and culture, particularly in the Midwest (*Paris, Madrid*).

Figure 5.4 separates out the two groups of cities by using a three-way crossing. The crossing is based on the three-dimensional algebraic expression city * pop2000 * group, where group contains the values *USA* for every city in the USA and *World* for all other cities. This expression produces a varset with three columns. The first column is assigned to the horizontal axis, the second to the vertical, and the third to the horizontal axis again, which has the effect of splitting the frame into two frames. This general pattern of alternating horizontal and vertical roles for the columns of a varset provides a simple layout scheme for complex algebraic expressions.

Chicago stands out as an anomaly in Figure 5.4 because of its relatively large population. We might want to sort the cities in a different order for the left panel or eliminate cities not found in the US, but the algebraic expression won't let us do that. Because group is crossed with the other variables, there is only one domain of cities shared by both country groups. If we want to have different domains for the two panels, we need our next operator, *nest*.

ELEMENT: *point*(*position*(city*pop2000*group))

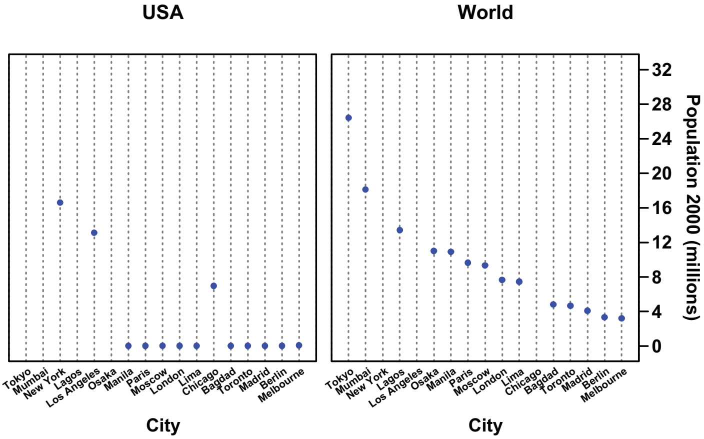
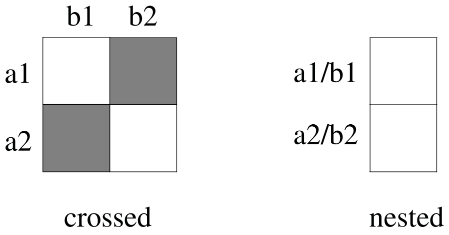


Figure 5.4 *Three-dimensional (grouped) dot plot*

5.2.2 Nest

Nesting looks like crossing in certain respects. That is not a coincidence. In experimental design, nesting is like crossing with certain cells deleted:



By deleting cells, we can rearrange the nesting layout to save space. We nest the expression on the left of the slash under the expression to the right. The layout thus appears to be one-dimensional, but it actually contains a hierarchy, In the figure above, the levels of a are nested within the levels of b.

Figure 5.5 shows a nested dot plot based on the nested algebraic expression city/group*pop2000. Because of the hierarchy of operators, this expression is equivalent to (city/group)*pop2000. The horizontal axis in each panel now shows a different set of cities: one for the USA and one for the rest of the world. This graphic differs from the one in Figure 5.4 not only because the axes *look* different, but also because the meanings of the cities in each panels *are* different. For example, the city named *Paris* appears twice in both figures.

In Figure 5.4, on the one hand, we assume the name *Paris* in the left panel is comparable to the name *Paris* in the right. That is, it refers to a common name occurring in two different contexts. In Figure 5.5, on the other hand, we assume the name *Paris* references two different cities. They happen to have the same name, but are not equivalent. Such distinctions are critical, but often subtle.

ELEMENT: *point*(*position*(city/group*pop2000))

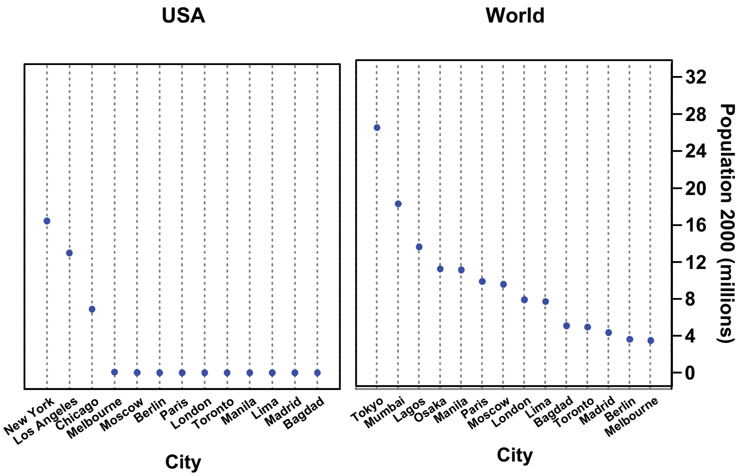


Figure 5.5 *Nested dot plot*

5.2.3 Blend

Blending increases the number of cases in a graphic. Figure 5.6 shows a simple example of this arrangement. We have doubled the number of cases in the graphic by using two variables on the vertical dimension and two on the horizontal. This amounts to two scatterplots overlaid on a common range and domain:

$$\mathsf{A} * \mathsf{C} + \mathsf{B} * \mathsf{D}$$

If we had blended before crossing, we would have produced *four* times the number of cases:

$$(\mathsf{A} + \mathsf{B}) * (\mathsf{C} + \mathsf{D}) \ = \ \mathsf{A} * \mathsf{C} + \mathsf{B} * \mathsf{C} + \mathsf{A} * \mathsf{D} + \mathsf{B} * \mathsf{D}$$

The graphic in Figure 5.6 is based on the expression city*(pop1980+pop2000). The horizontal axis represents the cities and the vertical axis represents the two repeated population measures.

DATA: p1980 = "1980"
DATA: p2000 = "2000"
ELEMENT: *point*(*position*(city*(pop1980+pop2000)), *color*(p1980 + p2000))

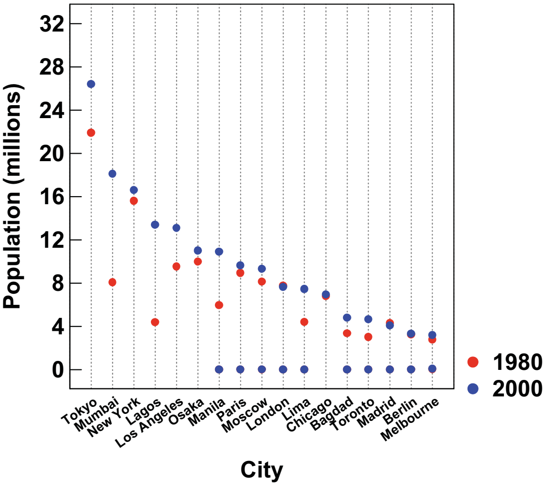


Figure 5.6 *Blended scatterplot*

We use color to distinguish the two sets of cases that the blend operator has created, red for 1980 and blue for 2000. Since a variable that carries this information does not exist in the dataset, we have to create a variable to split the cases and make the legend. The variable p1980 contains the string "1980" for all cases. Similarly, p2000 contains the string "2000" for every case. We use the blend of these two variables to color the cases. Notice that a blend is required to specify the colors. Without a blend in the color specification, we wouldn't have a color for every case in the graph. We will see in Chapter 10 how color aesthetics are used with the algebra.

Blending categorical variables combines common categories. Only the distinct categories from the blended variable sets appear on the common scale. Figure 5.7 shows an example. The data on which this dot plot is based consist of the social status of the first two speakers in Act 1, Scene 1 of each of Shakespeare's plays. The status of the first speaker is encoded in a variable called first, and the second in a variable called second. Each of these variables has been coded into one of six possible social status categories, shown on the scale at the bottom of the plot. The Royalty category includes kings, queens, and emperors. The Nobility category includes dukes, earls, counts, countesses, princes, and marquesses. The Gentry category includes justices, tribunes, archbishops, bishops, and governors. The Citizens category includes citizens, merchants, tradesmen, and ship's masters. The Yeomanry category includes servants, boatswains, messengers, soldiers, hostesses, and porters. Finally, the Beggars category includes beggars.

ELEMENT: *point.stack*(*position*(first+second))

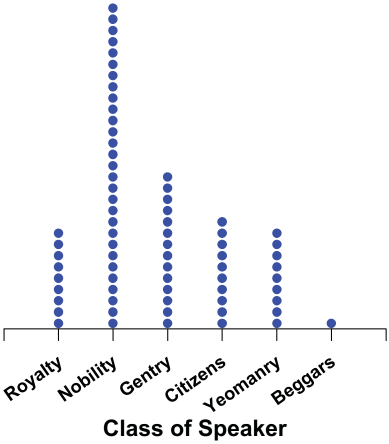


Figure 5.7 *First two speakers in Shakespeare's plays*

Blending and nesting are often combined. In Figure 5.6, for example, it is difficult to distinguish US and world cities. Figure 5.8 makes the distinction clear by splitting the horizontal axis into two nested subgroups.

ELEMENT: *point*(*position*((city/group)*(pop1980+pop2000))

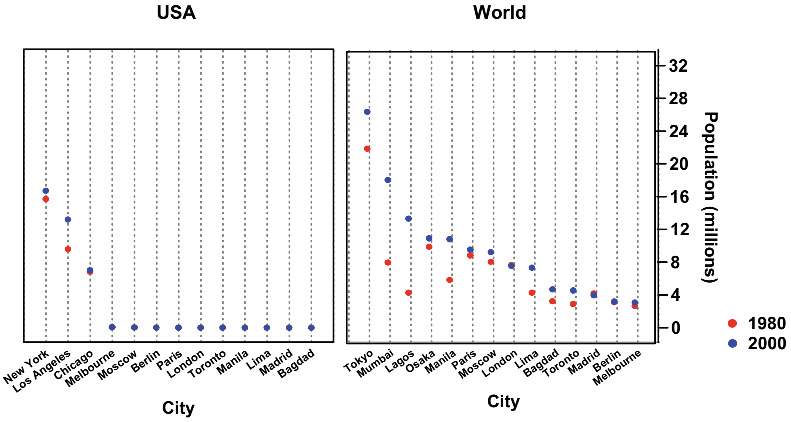


Figure 5.8 *Blended and nested scatterplot*

The graphic in Figure 5.8 is derived from the nested algebraic expression (city/group)*(pop1980+pop2000). The vertical axis represents the two repeated population measures blended on a single dimension. We see most of the cities gaining population between 1980 and 2000. Unfortunately, the vertical scales in Figure 5.8 still make it difficult to discern population growth in the US cities. The blue (2000) symbols cover the red (1980) ones for cities whose populations are less than 3 million. One remedy for this situation is to log the vertical scale. See Figure 6.8 for the result.

Sometimes we nest under a variable that has only one value. This device is used to force separate scales when variables are blended. Figure 5.9 shows an example. The variable p1980 contains the string "Pop 1980 (millions)" for all cases. Similarly, p2000 contains the string "Pop 2000 (millions)" for every case. When we blend pop1980 and pop2000, their values would ordinarily be pooled on a common scale. To prevent this, we blend pop1980/p1980 and pop2000/p2000 so that the population values are different. We have forced, in effect, two panels onto our graphic. Notice that the two vertical scales differ. To facilitate comparisons between panels, it might be better to adjust the aspect ratio of the upper panel to be larger so that the physical units of the scales match. In Chapter 11, we will discuss joint nesting and blending further.

DATA: p1980 = "Pop 1980 (millions)"
DATA: p2000 = "Pop 2000 (millions)"
ELEMENT: *point*(*position*(city*(pop1980/p1980+pop2000/p2000)))

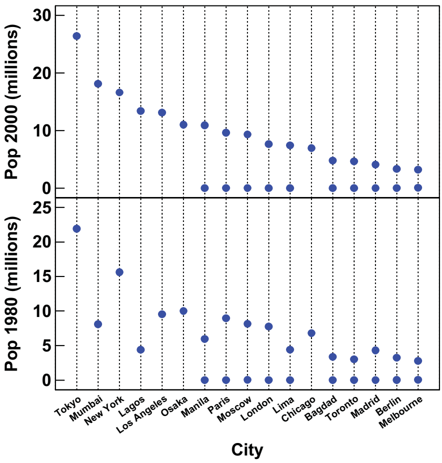


Figure 5.9 *Blending nests to create separate scales*

5.3 Other Algebras

Several algebras developed for computer applications are related to the one we have presented in this chapter. In some cases, we need to show where we have drawn from the ideas behind these algebras, and in others, We need to show where we have not.

5.3.1 Design Algebra

An experimental design is a factorial structure that contains sets of categories embedded within other sets of categories. A design algebra operates on designs and is an application of rules derived from lattice theory and other fields of discrete mathematics. Nelder (1965) introduced a notation for implementing a computer algebra that specifies experimental designs. Wilkinson and Rogers (1973) and others have extended this notation. See Heiberger (1989) for a review.

Nelder's and more recent design algebras are intended for producing matrices of binary indicator variables that represent presence or absence of experimental treatment categories. These matrices are used in the estimation of treatment effects on outcomes for designed experiments. Nelder's work has influenced almost every subsequent statistical implementation of the general (and generalized) linear model, including GENSTAT (Alvey *et al.*, 1977), GLIM (Nelder and Wedderburn 1972; Baker and Nelder, 1978), SAS® GLM (SAS Institute, 1976), and SYSTAT® MGLH (Wilkinson, 1983a).

The cross and nest operators presented in this chapter are related to, but not the same as, the operators of the same name in Nelder's system. Where Nelder's syntax is intended to produce a correct design matrix for a particular design specification, we have been concerned to have the algebra produce a graphic or table that correctly summarizes the design. This does not mean that the two notations are syntactically compatible, however. The blend operator does not exist in Nelder's system, and application of the rules presented in Section 5.1.3 can lead to structures that are not employed in experimental design. Such structures appear in applications such as market research tables and graphical layouts not intended to be analyzed by a single statistical model.

5.3.2 Relational Algebra

The algebra presented in this chapter is related to, but not the same as, a **relational algebra** (Codd, 1970). For an introduction to relational algebras, see Date (1995). A relation *R* is defined in Section 2.1.2. A varset, because it is a function, is a special case of a relation. Consider the relation

$$\{(a_{11}\text{: } D_1, a_{12}\text{:}D_2, \ldots, a_{1p}\text{:}D_p), \ldots, (a_{n1}\text{: } D_1, a_{n2}\text{:}D_2, \ldots, a_{np}\text{:}D_p)\}$$

where a_{ij} are n p-tuples of attribute values measured on p domains D_j, with n representing the cardinality of the set (number of tuples), and p the degree of the relation (number of attributes in each tuple). This type of relation underlies the organization of a **relational database system**. In the language of variables, the a_{ij} are *values* and the D_j are *tags* based on the names of the variables and their ranges. The tagged attribute values $a_{ij}{:}D_j$ share an **equivalence relation** ($a_{ij}{:}D_j \sim a_{kj}{:}D_j$) by being members of a common class. A relational database system uses the rules induced by these equivalence relations to store and retrieve subsets of data. A relational algebra is a set of operators together with a set of relations and rules for the operators. For our purposes, the relational operators of interest are *union*, *join*, and *nest*.

The *union* relational operator (Codd, 1970) produces a union of sets (relations). Relations on identical domains are blended into a single relation and duplicate tuples are eliminated. If we create a unique tagged value for the key in each set and assign these values to a single domain, then *union* resembles *blend* in the graphics algebra.

The *join* relational operator (Codd, 1970) is an element-wise crossing of indexed sets such that tuples with a common index value are merged into new tuples. If we restrict the relational join to a common index (key) that is unique for each tuple, then a *join* of two relations is like *cross* in the graphics algebra. Such a *join* produces a relation of degree 2 (ignoring the index attribute).

The *nest* relational operator (Roth *et al.*, 1988) involves a hierarchy that was not defined in Codd's original algebra. It produces a nesting of relations under other relations and resembles the definition of *nest* in the graphics algebra.

The similarities in algebras lead one to ask whether a graphics system with the capabilities enabled by a graphics algebra could be implemented by attaching a simple viewer to a relational database. An extended Structured Query Language (SQL) that implements the relational model could be used in this effort (see Section 5.1.5). There are several reasons why this approach would not be the best way to implement graphics algebra, however.

First, although one language can be used to imitate another, there are complexities and inefficiencies introduced when trying to adapt a system designed for a different purpose. One problem is commutativity. The graphics algebra operators are non-commutative (because graphs are functional). The corresponding *join* and *nest* relational operators are commutative. Restrictions can be built into a specific implementation, but they would be messy side conditions.

Second, locating the algebraic system in a database query removes knowledge in the graphics system needed for identifying and manipulating graphic objects. This would amount to a client-server model in which a graphics system is simply a viewer into a database. This system would resemble the datacube data-mining system we criticized in Chapter 3.

Third, relational database systems are not designed to interface well with interactive controllers and other tools needed to explore data dynamically. When put to such uses, relational databases are cumbersome and slow because of the high transaction demands placed on them by exploratory graphics clients.

All this is *not* to say that a SQL client-server link could not be valuable for implementing the graphics system outlined in this book. On the contrary, such a link is an essential part of implementing the algebra for business and other systems that require centralized data security and limited user access. Choosing the locus of the algebra, however, profoundly affects the behavior of the system.

Despite these problems, the relational model *can* play a role in a general graphics system. The graphics algebra, like relational algebra, is built on a set-theoretic approach to relations and functions that is easily packaged in objects. Embedding this algebra in an object data-model gives graphs the opportunity to view data without worrying about its structure. Graphs do not understand algebra. Algebra simply creates dimensions that govern their behavior. Consequently, a well-designed graphical system should include abstract interfaces that allow us to perform algebraic computations using functions in a relational database when they are available and appropriate. If these interfaces are designed to allow introspection or reflection (objects providing information about themselves during run-time), then we have a system that adapts to client-server and distributed environments. For an example of an implementation of our graphics algebra within an OLAP environment, see Stolte *et al.* (2000).

5.3.3 Functional Algebra

Functional programming languages (Bird and De Moor, 1996) are based on universal algebras on abstract data classes. Functional algebras have been applied to a variety of computer geometry and graphics problems. For example, Egenhofer, Herring, Smith, and Park (1991), and Rugg, Egenhofer, and Kuhn (1995) have investigated the use of functional algebras for specifying geographic maps and navigating through spatial databases.

As with relational algebras, there are similarities between the graphics algebra operators presented in this chapter and operators in functional algebra systems. The *cross* operator produces a *product* set. The *nest* operator produces a subset of a product set called a *dependent product*. And the *blend* operator produces a *union*.

Functional algebra systems are more general than the system proposed in this chapter. While functional programming languages based on these algebras could be used for creating graph specifications, their generality exceeds the scope needed for a statistical graphics system.

5.3.4 Table Algebra

The US Bureau of Labor Statistics pioneered a language for laying out tables (Mendelssohn 1974). While not a formal algebra, this Table Producing Language (TPL) contained many of the elements needed to assemble complex tables. Gyssens *et al.* (1996) outlined an algebra for displaying relational data; this algebra closely followed TPL, although the latter is not referenced. Wilkinson (1996) presented an algebra for structuring tables and graphics.

5.3.5 Query Algebra

Pedersen *et al*. (2002) described an algebra for querying OLAP cubes. The result sets from their algebraic expressions could be used for graphic displays. Agrawal *et al*. (1997) used a similar algebra for statistical modeling of data contained in a cube.

5.3.6 Display Algebra

Mackinlay (1986) developed an algebra for querying relational databases and generating charts. His general goal was to develop an intelligent system that could offer graphical responses to verbal or structural queries. Roth *et al.* (1994) followed a similar strategy in developing graphical representations of relational data. They extended Mackinlay's and others' ideas by using concepts from computational geometry.

5.4 Sequel

We now have an algebraic system for relating variables. The next chapter will cover scale transformations. These transformations are used primarily to meet statistical assumptions or to scale graphs on meaningful units such as time. Scales must be computed before statistical operations because, among other things, the sum of logs does not equal the log of that sum.

6

Scales

The word **scale** derives from the Latin *scala*, or ladder. The Latin meaning is particularly apt for graphics. The visual representation of a scale — an axis with ticks — looks like a ladder. Scales are the types of functions we use to map varsets to dimensions. At first glance, it would seem that constructing a scale is simply a matter of selecting a range for our numbers and intervals to mark ticks. There is more involved, however. Scales measure the contents of a frame. They determine how we perceive the size, shape, and location of graphics. Choosing a scale (even a default decimal interval scale) requires us to think about what we are measuring and the meaning of our measurements. Ultimately, that choice determines how we interpret a graphic.

6.1 Scaling Theory

Scaling is a field with a long history in physics and psychometrics. We will briefly cover this area. Then we will discuss a variety of measurement scales and how to construct the axes that display them. A useful introduction to these ideas is Luce and Suppes (1991).

6.1.1 Axiomatic Measurement

In a widely cited paper "On the theory of scales of measurement" (1946), the psychophysicist S.S. Stevens presented a theory of measurement based on the invariance of the meaning of scales under different classes of transformations. Stevens showed that measurement scales that preserve meaning under a wide variety of transformations in some sense convey less information than those whose meaning is preserved by only a restricted class of transformations.

Axiomatic scale theory defines scale types through measurement axioms. Stevens postulated four basic scale types: *nominal*, *ordinal*, *interval*, and *ratio*. To define a nominal scale, we assume there exists at least one equivalence class together with a binary equivalence relation ($\sim$) that can be applied to objects in the domain $D^{(X)}$ (e.g., the class of this object is the *same* as the class of that object). We then say that a scale is nominal if, for its values $X(d)$,

$$d_i \sim d_j \Leftrightarrow X(d_i) = X(d_j) \forall d_i, d_j \in D^{(X)}$$

To define an ordinal scale, we assume there exists a binary total order relation ($\succ$) that can be applied to objects in the domain (e.g., this stone is *heavier than* that stone). We then say that a scale is ordinal if

$$d_i \succ d_j \Leftrightarrow X(d_i) > X(d_j) \forall d_i, d_j \in D^{(X)}$$

To define an interval scale, we assume there exists a symmetric concatenation operation ($\oplus$) that can be applied to objects in the domain (e.g., the length of this stick *appended to* the length of that stick). We then say that a scale is interval if

$$d_i \oplus d_j \Leftrightarrow X(d_i) + X(d_j) \forall d_i, d_j \in D^{(X)}$$

To define a ratio scale, we assume there exists a magnitude comparison operation ($\oslash$) that can be applied to objects in the domain (e.g., the *ratio* of the brightness of this patch to the brightness of that patch). We then say that a scale is ratio if

$$d_i \oslash d_j \Leftrightarrow X(d_i) / X(d_j) \forall d_i, d_j \in D^{(X)}$$

Axiomatic scale theory is often used in data mining and graphics, but it is not sufficient for determining scales on statistical graphics produced by chart algebra. The blend operation, for example, allows us to union values on different variables. We can require that blended variables share the same measurement level (e.g., diastolic and systolic blood pressure), but this will not always produce a meaningful scale. For example, we will have a meaningless composite scale if we attempt to blend height and weight, even assuming both are ratio variables. We need a different level of detail so that we can restrict the blend operation in a manner similar to the flagging of *domain check errors* for the *union* operation in a relational database (Date 1990).

6.1.2 Unit Measurement

An alternative scale classification is based on units of measurement. Unit scales permit standardization and conversion of metrics. In particular, the International System of Units (SI), summarized in Taylor (1997), unifies measurement under transformation rules encapsulated in a set of base classes. These classes are *length, mass, time, electric current, temperature, amount of substance*, and *luminous intensity*. Within the base classes, there are default metrics (meter, kilogram, second, etc.) and methods for converting from one metric to another. From these base classes, a set of derived classes yields composite measurements such as pressure, energy, capacitance, volume, power, and illuminance.

Table 6.1 shows a set of **elemental measurement**, or **primary unit**, classes. The first seven constitute the SI base classes. Default SI base units are bolded in the table. We have added three more classes: angular measurement units, counting units, and a base class for currencies. Angular measurement is discussed in the SI standard but is not in the core. Counting units provide number bases. The currency class implements exchange rates. Currencies are time dependent, since daily exchange rates determine conversion rules and an inflation adjustment method varies with time. These adjustments can be imprecise, but they are usually sufficient for reasonable historical comparisons. The SI base classes, under certain assumptions, behave like ratio scales in Stevens' system. Each unit in a cell is related proportionally to the base unit, so conversions require only a single multiplier (e.g., 1 inch = 2.54 cm). And zero is presumably an absolute value (zero length, zero mass).

Table 6.1 **Elemental (Primary) Unit Measurements**

Length	***Mass***	***Time***	***Current***	***Temperature***
meter *point* *pica* *inch* *foot* *yard* *mile* *furlong* *fathom*	***kilogram*** *gram* *grain* *slug* *carat*	***second*** *minute* *hour* *day* *week* *month* *quarter* *year* *century*	***amp***	***kelvin*** *rankine* *celsius* *fahrenheit*
Substance	***Luminous Intensity***	***Angle***	***Count***	***Currency***
mole	***candela***	*radian* *degree* *minute* *second*	*unit* *dozen* *gross*	*dollar* *euro* *pound* *yen*

Table 6.2 shows SI and other **derived measurement**, or **secondary unit**, classes. These units can be derived from the base units by simple algebraic formulas (e.g., 1 acre = 4046.838 meter^2). Several of these derivations are not obvious. Force, for example, is derived from mass relative to gravitational setting. Geocentric latitude and longitude are derived from angle relative to the earth's equatorial plane and standard meridian. Dates are derived from time, using an arbitrary origin and a scale determined by orbits referenced to the moon or sun. Rates are an

exception to the other derived units in Table 6.2. We have printed the selected rates in gray because they are not convertible within the class. To convert one rate to another rate, the classes of the numerators must match and the classes of the denominators must match.

Table 6.2 **Derived (Secondary) Unit Measurements**

Area	***Volume***	***Pressure***	***Density***	***Frequency***
square meter *square mile* *hectare* *acre*	*liter* *teaspoon* *tablespoon* *cup* *pint* *quart* *gallon* *barrel*	*pascal* *bar* *atmosphere*	*gm per liter* *oz per quart*	*hertz*
Inductance	***Voltage***	***Capacitance***	***Charge***	***Resistance***
henry	*volt*	*farad*	*coulomb*	*ohm*
Energy	***Power***	***Force***	***Torque***	***Speed***
joule *erg* *calorie* *therm* *btu*	*watt* *horsepower*	*ounce* *pound* *stone* *ton*	*newton-meter* *foot-pound*	*kph* *mph*
Rate	***Income***	***Latitude***	***Longitude***	***Date***
cost per unit *birth rate* *death rate*	*annual dollar* *weekly euro*	*degree* *minute* *second*	*degree* *minute* *second*	*gregorian* *julian* *chinese* *mayan*

There are some widely used scales that are not classified within the SI system and do not involve measurement units. These so-called **dimensionless measurement** classes involve *categories* (states, countries, colors, names, species), *orders* (letter grades, tennis ladders) and *measures* (correlations, percentages). Table 6.3 contains a sample of scales in these three classes. The entries in the cells of this table are not necessarily convertible through simple algebraic formulas, as are the entries in Table 6.1 and Table 6.2. The *category* class includes names and classifications. The *order* class includes ordinal measures. Ranks vary from a value of 1 (best, highest, ...) to n (worst, lowest, ...). Indices vary from 1 (lowest, ...) to n

(highest, ...). Partial orders are represented by tied ranks or indices. Finally, the *measure* class includes real-valued statistical indices and derived measures,

For our purposes, unit measurement gives us the level of detail needed to construct a numerical or categorical scale. We consider unit measurement a form of strong typing that enables reasonable default behavior. Because of the class structure and conversion methods, we can handle labels and relations for derived quantities such as miles-per-gallon, gallons-per-mile, and liters-per-kilometer. Furthermore, automatic unit conversion within base and derived classes allows meaningful blends. As with domain check overrides in a database, we allow explicit type overrides for the blend operation.

Table 6.3 **Dimensionless Measurements**

Category	***Order***	***Measure***
town *city* *county* *province* *state* *country* *continent* *name* *gender* *class* *species*	*rank* *index*	*correlation* *probability* *variance* *proportion* *percent* *similarity* *dissimilarity* *distance* *ratio*

6.1.3 Applied Scaling

Since Stevens' paper, many methodologists have used his axiomatic system to prescribe, and occasionally proscribe, statistical and graphical methods. The most extreme of Steven's methodological heirs have attempted to scare researchers into using only nonparametric statistical methods. The problem with this approach, as Velleman and Wilkinson (1994) argued, is that data offer scant help in choosing a scale and the wrong choice can hinder discovery. See Hand (1996) for a response and rejoinders to our paper.

Velleman and Wilkinson pointed out that scientists must rely on extrinsic information (metadata) to decide what level of measurement might be appropriate for their data. Since we are not God, the best we can do is understand the context in which the data were collected and use our intuition and substantive scientific knowledge to choose scales. Often, we need to consider several scales for the same data. As Sir Ronald Fisher (1935) noted in reaction to statisticians who were promoting routine use of nonparametric statistics,

> Experimenters should remember that they and their colleagues usually know more about the kind of material they are dealing with than do the authors of text-books written without such personal experience, and that a more complex, or less intelligible, test is not likely to serve their purpose better, in any sense, than those of proved value in their own subject.

6.1.3.1 The Naive Approach

The naive approach to scaling is to decide the measurement level and then assign a variable to a scale corresponding to this measurement level. Figure 6.1 shows the countries data from Chapter 1 mapped to Stevens' scale types. They are represented visually in the figure by a set of axes.

The nominal scale is shown by ordering the country names alphabetically. Any ordering will do. All that matters for a nominal scale is that the mapping be one-to-one: that each country have a unique identifier. The ordinal scale is represented by a rank ordering of the countries on military expenditures. The spacing of the countries on this scale is arbitrary. All that matters is that the ordering preserve the relative ranks of the countries. The interval scale is represented by the military expenditures themselves. The spacing of the countries matters, but the origin is arbitrary. That is, the difference in spending between Iraq and Libya should be comparable to that between Canada and Italy because both pairs are spaced approximately the same distance apart on the scale. (We will discuss how to deal with the compression of the labels at the bottom of the scale later in the chapter.) Finally, the ratio scale is represented by the count of the countries within three major types of government. On this scale, the zero tick is significant; counts are referenced by this point. We can say by looking at the graphic that the count of countries in the sample with a Military form of government is a third larger than the count of countries with One Party government.

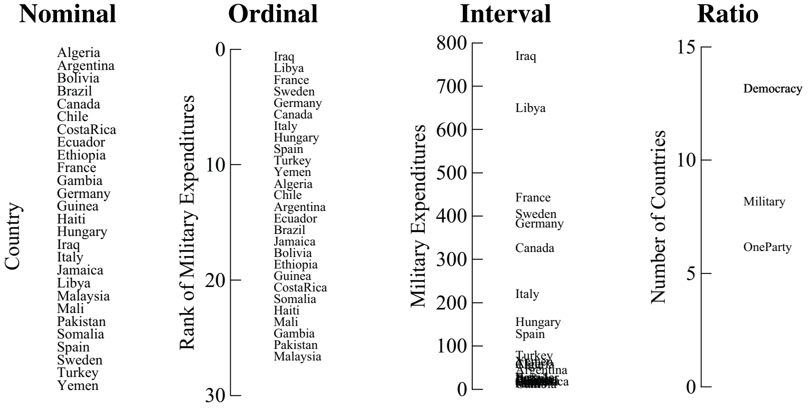


Figure 6.1 *Graphical displays of Stevens' scale types*

6.1.3.2 Ambiguities

If we examine the data more closely, however, the neatness of these assignments breaks down. For example, we could say that normalized per-capita military expenditures map more appropriately to a ratio scale. Zero dollars seems like an absolute reference point. Spending 200 dollars per-capita seems like twice spending 100.

Money is not a physical or fundamental quantity, however. It is a measure of utility in the exchange of goods. Research by Kahneman and Tversky (1979) has shown that zero (no loss, no gain) is not an absolute anchor for monetary measurement. Individual and group indifference points can drift depending on the framing of a transaction or expenditure. This is why we classified the military expenditures as an interval rather than ratio scale.

A similar argument can be made against our classification of the country counts as a ratio scale. It would appear to make more sense to call this an **absolute scale**. While not an original Stevens scale type, it follows from his reasoning that an absolute scale has no permissible transformation other than the identity. This makes sense for counts. Ten means a count of ten — until one asks what is being counted. Counts depend on categorizations, which are not as concrete as Aristotle might have wished. The assignment of One Party, Democracy, and Military as forms of government is non-overlapping in our example. It is arguable, however, that some countries might have more than one of these types. Similar arguments can be made about most of the variables we choose to represent in graphics. While the axioms of scaling theory are undisputable, there is no way to determine surely which one applies to a set of data.

6.1.3.3 Scales as Roles

Sometimes it is more useful to think of scales as roles for dimensions that help reveal patterns in an analysis. Figure 6.2 shows that we can represent the same data on two different scales in the same graphic. It is a quantile plot of per-capita military expenditures. The horizontal scale is perhaps interval, based on the numeric values themselves. What is the vertical scale? It is based on the rank order of the data values, so it might be called ordinal. The equal spacings are important, however, because we are using it to anchor the shape of the curve traced by the points. Thus, it is being employed as an interval scale for the purposes of analyzing shape. If we keep the aspect ratio 1 (a square frame), then the shape is invariant under any interval scale transformation. The positive skewness is readily apparent in the convex shape of the curve. A unimodal symmetric distribution would have plotted in an *S* shape. Finally, we could call it an absolute scale because we are using it to display the **empirical cumulative distribution function** of per-capita military spending. If we are attending to the scale values rather than the shape, then we are processing numerical information on an absolute scale between zero and one.

TRANS: rmil = *prank*(military)
ELEMENT: *point*(*position*(military*rmil))

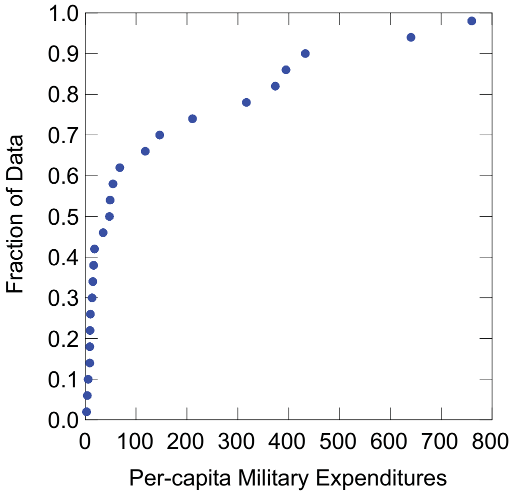


Figure 6.2 *Quantile plot of military expenditure*

6.1.4 Graphics and Scales

Graphics do not care about the scales on which they are drawn. Bars, for example, do not know whether they measure counts, proportions, or other quantities. What, then, can we say about the common prescription (*e.g.*, Schmid and Schmid, 1979; Cleveland, 1985) that bars require a zero base to be meaningful? The answer is, we think, that these are statements about scales rather than graphics. Bars are multi-valued graphics. They represent two values on their range — a lower and upper point. When referenced against a ratio scale, it is most appropriate for the bottom of a bar to be at zero so that ratio comparisons can be made. Even when all the bases are at zero, however, both ends are marking an interval. There is nothing in the definition of a bar itself that requires it to have a base at the value zero.

6.2 Scale Transformations

In Chapter 4 we saw how to transform variables by doing such operations as logging or square-rooting. The purpose of variable transformations in a graphics system is to make statistical operations on variables appropriate and meaningful. Scales, however, operate on *sets* of variables (dimensions). The purpose of scale transformations is similar — to make statistical objects displayed on dimensions appropriate and meaningful. The next chapter will cover a third class: coordinate transformations. The purpose of those transformations is to manipulate the geometry of graphics to help us perceive relationships and find meaningful structures for representing variation. In some cases, we use scale and coordinate transformations together. And in some rare cases, we might apply transformations three times — once to a variable, once to a dimension, and once to a coordinate system.

The reason for separating transformations according to the sets on which they operate (variables, dimensions, and coordinates) is to keep clear the distinction between statistical and geometric operations. Statistical methods (*e.g.*, smoothing and aggregation) often require assumptions about the statistical distribution of the variables on which they operate. Thus, variable and scale transformations must be done *before* these statistical methods do their work. However, coordinate transformations change the appearance of graphics (*e.g.*, bars become pie slices) but do not alter their statistical properties. Thus, coordinate transformations must be done *after* statistical methods do their work.

The *log*() transformation may help us to see this distinction. If we log a variable, the numbers displayed on an axis will be logs (*e.g.*, 1, 2, 3) and the title of the axis should be something like "Log of Income." If we log a dimension, however, the numbers displayed will be on a log scale (*e.g.*, 10, 100, 1000) and the title of the axis should be something like "Income." Finally, if we use a log coordinate transformation, we should expect to see portions of graphics near the high end of the scale compressed more than those near the low end. The numbers on the axis should be unchanged, but their locations (and tick marks) should be compressed logarithmically.

Table 6.4 shows several scale transformation functions. This sample list is intended to cover the examples in this book and to be a template for designing the signature and behavior of new functions. All the functions have a standard form that includes a dimension name followed by other optional parameters. Examples are:

interval(*dim*(1), *min*(0), *max*(1000))
time(*dim*(2), *format*("mmm dd, yyyy"))

The number of optional parameters are peculiar to each function, but their order is irrelevant.

Table 6.4 **Scales**

Categorical	***Interval***	***Time***	***One-bend***	***Two-bend***	***Probability***
cat()	*linear*()	*time*()	*log*() *pow*()	*asn*() *logit*() *probit*() *atanh*()	*prob*()

The categorical scale demarcates *k* points for locating *k* categories in a serial ordering. The interval scale has the property that any two equal-length intervals on the scale have the same measure. For example, if an interval from 0 to 2 on an axis measures two inches, then an interval from 4 to 6 must measure two inches. We use the term linear to describe this scale because it is the most common usage. The time scale measures time. The one-bend functions are increasing or decreasing concave or convex. The *log* scale returns logs. The *pow*() transformation computes $f: x^p$. However, *pow*(**x**,0)=*log*(**x**). The two-bend functions are ogive shaped. The *asn*(), *logit*(), *probit*(), and *atanh*() scales are arcsine, logit, probit, and Fisher's *z* statistical transformations. The probability scales implement various probability distributions.

6.2.1 Categorical Scales

The syntax for *cat*() is

cat(*dim*(), *values*("*string1*","*string2*", ...)) or

cat(*dim*(), *values*(*val1*, *val2*, ...))

The extra parameters after *dim*() are optional. Categorical scales index categories. Because scales must order categories by design, there is no visible difference between a nominal and an ordinal scale in a graphic. String variables are by definition categorical, so we need not use a *cat*() scale to ensure that they are mapped to categories. The optional parameters for a categorical scale determine the choice and ordering of the categories and their format, *e.g.*, *cat*(*dim*(1), *values*("Jane","Jean","June")). The system searches all values available in the data, does a string match to determine the unique assignments, and then uses the strings in the *cat*() list to label the scale.

Picking numbers for numerical categorical scales is easy. Simply choose the natural numbers, since categories are best displayed evenly spaced. Figure 6.3 shows three examples of categorical scales. Notice that the endpoints of the scale are not mapped to categories. This makes it easy to distinguish a categorical scale with numerals from a numerical scale with the same numerals. It also keeps graphics (such as bars) from colliding with the edges of the frame.

We have chosen three categorical representations of hue in Figure 6.3: nanometer wavelength, color name, and a color index. Provided the values are mapped properly to the scales, all three representations would produce the same graphic for a particular graph.

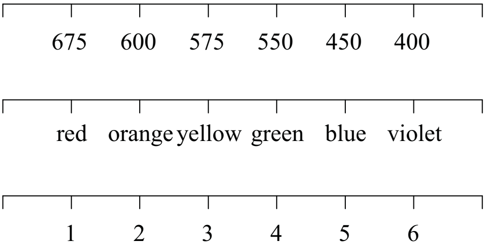


Figure 6.3 *Categorical scales*

6.2.2 Linear Scales

The syntax for *linear*() is

linear(*dim*(), *min*(), *max*(), *base*(), *ticks*(), *delta*(), *cycle*())

The extra parameters after *dim*() are optional. The *min*() and *max*() functions set minimum and maximum scale values. The *base*() function (default is 10) sets the number base. The *ticks*() function sets the number of tick marks (default is determined by program). The *delta*() function sets the interval between ticks (default is determined by program). The *cycle*() function has an integer argument specifying how many cycles there are to be between *min*() and *max*(). The default is 1. This is useful for specifying more than one revolution on polar plots, but it can be used on rectangular coordinates to overlay subsections of a time series.

Constructing axes for default linear scales requires us to choose minimum and maximum values, the number of tick marks, and numbers to go with the tick marks. Most people prefer to see nice numbers on these scales so that they can use them like rulers to measure graphics. There are several approaches to this problem. We will begin with numbers in base 10.

6.2.2.1 Nice Numbers

Nice numbers include the numbers we preferred as children when we learned arithmetic. These numbers persist in our habits and preferences when we label and view decimal scales on graphics. Although there might be some disagreement (perhaps cultural) over definition, nice numbers are members of an infinite subset of the real numbers, *e.g.*, $R_N = \{... , .1, .2, .5, 1, 2, 5, ...\}$. A **nice scale** is an interval scale marked with an ordered sequence of numbers whose first differences are nice numbers. The following sequences all have this property for the set R_N given above.

..., 1, 2, 3, 4, 5, ...
..., 2, 4, 6, 8, 10, ...
..., 0, 50, 100, 150, 200, ...
..., .001, .003, .005, .007, .009, ...

Really nice scales have an additional property: they include zero. The first three series above have this property, but the last does not. Nice scales can also be defined for some nonlinear scales. For scales like log and power, nice scales can be chosen in the transformed metric and inverse transformed to the original. For example, if we choose the nice scale [–1, 0, 1, 2, 3] in logarithms, then our log scale will be [.1, 1, 10, 100, 1000].

Nelder (1976), Stirling (1981), and Heckbert(1990) discuss nice numbers and provide algorithms for producing them when we are given a range of data values. These simple algorithms work well for a variety of instances. When they fail, we end up with too few or too many tick marks or a scale that does not fit closely the range of the data.

A more effective method is to compute scales using a multi-parameter search algorithm. First of all, we need to expand our set of nice numbers a bit in order to improve our chances of an attractive solution. We begin with the finite, ordered set $Q = \{1, 5, 2, 25, 3\}$. We may reorder, expand, or contract this set to suit our tastes, although we have found this set works well in practice. Then our set of nice numbers is $R_N = \{q \times 10^z : q \in Q, z \in \mathbf{Z}\}$, where $\mathbf{Z}$ is the set of integers.

Let n be the cardinality of the set Q (n=5 in this case). Let $i = 1, ... , n$ be the index of an element of Q. Let k be the number of ticks on a scale. Let r_d be the range of the data and r_s be the range of the scale (we assume $r_s \geq r_d$). Let S be a finite, ordered solution set of k numbers for our nice scale. (Note that usually $S \not\subset R_N$.) Let $v = 1$ if S contains zero, otherwise, let $v = 0$. Finally, let m be an ideal number of tick marks for a scale. This may depend on the physical size of a graphic and the size of fonts used for scale numbers, although the choice $m = 5$ usually suffices. Now we can construct a scale goodness index from the following components, each of which varies between 0 and 1:

simplicity: $s = 1 - i / n + v / n$

granularity: $g = \begin{cases} 1 - |k - m| / m \ , & 0 < k < 2m \\ 0 \ , & \text{otherwise} \end{cases}$

coverage: $c = r_d / r_s$

For a given scale S, the simplicity value is based mainly on the index i of the element chosen from Q that determines the set R_N that includes the first differences among the ordered elements of S. This simplicity value is incremented by $1/n$ if the set S contains zero. The granularity value reflects closeness to an ideal number of tick marks. Too few tick marks hamper scale lookup and too many clutter the display. The coverage measure rewards solutions that leave less white space around the data. Cleveland, Diaconis, and McGill (1982) have shown, for example, that this white space can bias judgments of correlation in scatterplots. We prefer to set a floor on coverage, say, .75, so that we will never automatically select a data range that fills less than 3/4 of a scale.

We claim that $w = (s + g + c) / 3$ is an indicator ($0 \le w \le 1$) of the goodness of a scale. There might be other functions of s, g, and c that do better for this purpose, but in our experience, this simple composite works well. Optimizing w can be done by direct search through the whole parameter space or, if we want to consider more values, by using algorithms such as O'Neill (1971). An advantage of the optimization approach is that any parameter can be constrained. This is especially important when users choose to fix the number of ticks, the minimum or maximum scale value, data coverage, or some combination of these and other parameters.

We tend to prefer restricting the search so that extreme ticks and numbers coincide with the ends of the scale, as opposed to allowing the tick marks to float elsewhere along the scale. The examples in this book generally have justified scales like this. When data cover intervals like [0, 1] or [0, 100], we may want to relax this restriction by indenting the tick marks to allow some empty space at both ends.

Nice numbers are not particularly interesting; they're just nice. If we are willing to relax the requirement that tick marks be evenly spaced on a scale, we can pick other number sequences that might be useful for emphasizing features of our data. For example, it may be useful to have more tick marks in the center of a scale than at the edges, depending on the distribution of data ruled by the scale. Knuth (1969) and Conway and Guy (1996) discuss a variety of interesting number sequences and algorithms for generating them. Tufte (1983) discusses scales that have ticks at significant data landmarks.

6.2.2.2 Number Bases

Different number bases affect the choice of nice numbers but not the treatment of tick marks. The methods we outlined in Section 6.2.2.1 can be modified easily for other number bases. We modify R_N to be $R_N = \{q \times b^z \colon q \in Q, z \in \mathbf{Z}\}$, where b is the number base (2, 8, 10, 16, etc.). We also need to modify Q to contain nice numbers in the chosen base.

6.2.3 Time Scales

The syntax for *time*() is

time(*dim*(), *min*(), *max*(), *origin*(), *cycle*())

The extra parameters after *dim*() are optional. The *min*() and *max*() functions set minimum and maximum scale values in formatted time. The *origin*() function (default is January 1, 1900) sets the time origin for the numerical scale. The *cycle*() function is an integer specifying how many cycles there are to be between *min*() and *max*(). The default is 1.

As Einstein famously established, time is the fourth dimension of our physical world. In Western and most other cultures, time is measured in lunar and solar astronomical cycles: days, months, and years. Other units have obscure etiologies. Seconds, minutes, and hours probably derive from the Sumerian practice of sexagesimal (base 60) arithmetic (Conway and Guy, 1996). Weeks, however, may have their origin in the Babylonian sacred number 7 (Ronan, 1991).

The annual circuit of the earth around the sun presents an arithmetic problem. It occurs in approximately 365.242 days. In 46 *BCE*, the Julian calendar added an extra day every four years to adjust for the roughly quarter-day-per-year gain. The difference between .242 and .250 is enough to have caused significant accumulation over centuries, however, so the Gregorian calendar introduced a new calculation in 1582. This calendar, now prevalent, specifies a leap year every four years except for centuries not divisible exactly by 400. Thus 2000 is a leap year, but 1900 is not.

Setting an origin for time requires relative precision (apologies to Einstein). If we want to measure and record century time to the resolution of mean solar seconds (based on a mean solar day of 86,400 seconds), we need a long number (more than 10 decimal digits) to span several centuries in daily units. With that precision, we can place our origin at 1582 if we are historically inclined or at 1900 if we are more financially motivated. The latter scale is used by modern spreadsheets and accounting packages, with positive units denoting day of the century. Negative numbers record time before 1900. Thus, the first author's birthday (November 5, 1944) is the 16,380th day of the 20th century. Apparently the developers of the most widely used spreadsheets did not consult an encyclopedia to learn that 1900 is not a leap year, so their dates are off by one day after February 28, 1900. This error has consequences when data from spreadsheets are imported into other database, statistical, and accounting packages that compute time correctly.

The Year 2000 or Y2K problem, as it is popularly called, arose in software for a variety of reasons. The most common problem is that some computer programs encode years as two-character strings or two-digit integers. This causes problems when they compute time spans or make comparisons. Truncating years is not the only way to make a program vulnerable to the advent of a millennium, however. If software employs a real-valued variable for storing time, careless programmers may fail to use sufficient numerical precision to store day-of-the-century. Finally, as we have noted, some programmers may not know how to compute leap years correctly. Having failed to note that 1900 was not a leap year, they may not realize that 2000 is.

Contemporary popular discussions concerning what an operating system is supposed to be and do often ignore the most important basics like dealing with time correctly. International date formats, time zones, and correct calculation of leap years is a critical function that an operating system should provide so that programmers are freed from having to think about the numerous details involved in calculations. Java performs these tasks correctly and conveniently because it has an object DateFormat that uses a long signed integer for recording absolute time in milliseconds, with an origin at Jan 1, 1970. With wisdom beyond its years, Java also seems to understand leap years backwards and forwards for many centuries.

In a nice time scale, tick numbers are separated by intervals contained in the set $T_N = \{\dots, \mathit{second}, \mathit{minute}, \mathit{hour}, \mathit{day}, \mathit{week}, \mathit{month}, \mathit{year}, \dots\}$, where the words we have used in the braces denote numerical time values anchored at some origin such as the first second of January 1, 1900. While niceness for numbers depends on numerals, niceness for time depends on calendar notation. Thus, an algorithm for nice time scales requires an understanding of Julian or other calendars used to display continuous time. Calendars differ across cultures, so the task can be quite complex when we intend to develop international software.

Figure 6.4 shows examples of several different time scales. Daily and weekly scales have evenly spaced ticks, but monthly and yearly scales do not. Notice that the gap between February and March is less than that between any other two months because February has the fewest days. The same irregularity can occur for years. The yearly scale in this example would have slightly irregular ticks if it included a leap year.

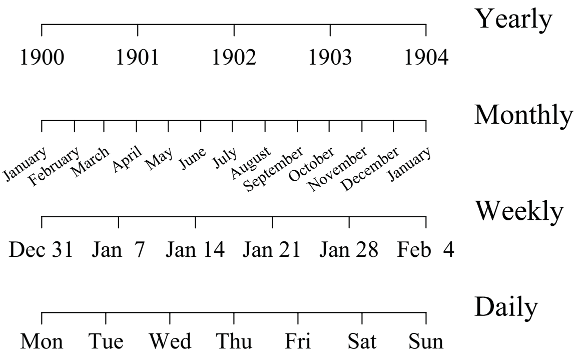


Figure 6.4 *Time scales*

Using time scales allows us to display graphics at the points on the scale that correspond to real dates. Figure 7.5 and Figure 7.13 illustrate this principle. Figure 6.5 shows a selection from the SPSS stock price series that shows specific dates formatted in slash notation. The major tick marks are set to Sundays, so the weekday trades occur between the weekend gaps. We are accustomed to seeing equally-spaced bars in printed charts and computer software. Time series data show us how restrictive this limitation can be.

SCALE: *time*(*dim*(1))
GUIDE: *axis*(*dim*(1), *format*("mm/dd/yy"))
ELEMENT: *interval*(*position*(*region.spread.range*(date*(high+low))))

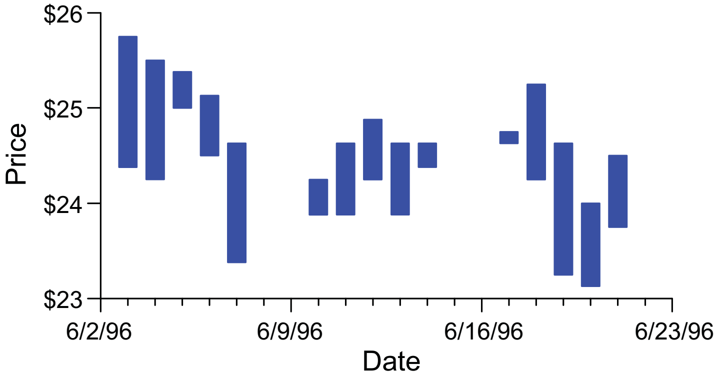


Figure 6.5 *Stock price ranges*

6.2.4 One-bend Scales

One-bend scales are convex (concave) upward or concave (convex) downward in their mapping functions. They include a variety of logarithmic, power, and probability scales.

6.2.4.1 Logarithmic Scales

The syntax for *log*() is

log(*dim*(), *base*(), *min*(), *max*(), *ticks*(), *cycle*())

The extra parameters after *dim*() are optional. The *base*() function (default argument is *e*, or natural logarithm) sets the log base. The *min*() and *max*() functions set minimum and maximum scale values. The *ticks*() function determines the number of ticks. The *cycle*() function takes an integer specifying how many cycles there are to be between *min*() and *max*(). The default is 1.

The logarithmic scale is usually computed on base 10, although other bases are possible and many are useful. The log base has no effect on the visual appearance of the plot; only the labeling of the tick marks on axes is affected. The same is true for most statistical tests of significance on logged data. This equivalence follows from $log_b(x) = ln(x)/ln(b)$, where $ln()$ is the natural logarithm and $b > 1$.

Figure 6.6 shows examples of log scales to several bases. Base 2 is useful when it makes sense to represent doublings. Base 3 is for triplings. Base 7 is for biblical time. (Logging time is an eschatological transformation, or perhaps a sign of old age!) The upper limit of the log septenary scale in Figure 6.6 comes just before Pentecost or Jubilee, depending on whether we count days or years. Base 10, of course, is customary in a decimal society.

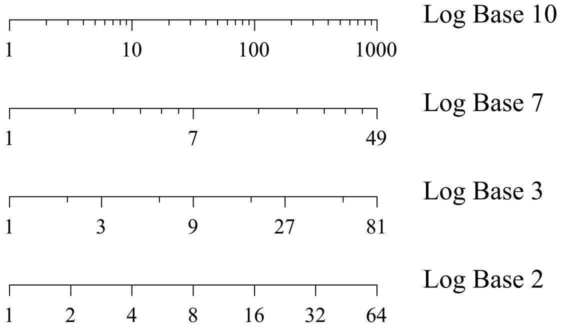


Figure 6.6 *Log base scales*

Programmers often overlook a detail when they implement log scales. If a user requests, say, a minimum value of 3 and a maximum of 105 on a decimal log scale, a well-designed scale algorithm ought to position the minor tick marks at exactly those values and display two major tick marks at 10 and 100. This requires more computation but is worth the effort for saving white space inside the borders of the frame. See Figure 10.37 for an example.

Figure 6.7 shows a log scale using data from Allison and Cicchetti (1976), a study of the relation between sleep habits of animals and their chance of being eaten by predators. The variable brainweight is the weight in grams of the brains of the animals surveyed. The variable exposure is a rating of the degree an animal is exposed to predators while sleeping (1=sheltered, 5=exposed). We have logged the data in order to make them more normally distributed. This makes it more reasonable to use the standard deviation as a symmetric measure of spread. The bars are symmetric in the log metric and asymmetric in the raw metric. We can relate both metrics through viewing the vertical scale. Log and other nonlinear transformations have interesting consequences for statistical inference and interpretation. See Section 9.1.8.1 for further discussion of this topic.

```
SCALE: cat(dim(1))
SCALE: log(dim(2), base(10))
ELEMENT: point(position(summary.mean(exposure*brainweight)))
ELEMENT: interval(position(spread.sd(exposure*brainweight),
         color(color.red))
```

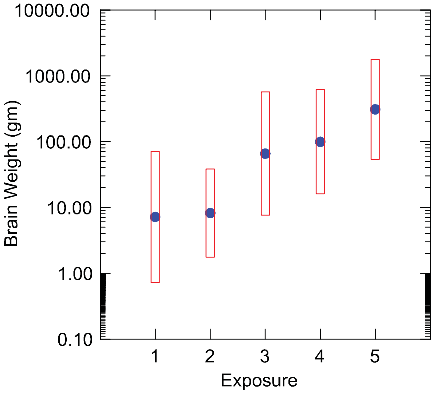


Figure 6.7 *Error bars on logarithmic scale*

Sometimes we log simply to separate overlapping values when the data are positively skewed. Figure 6.8 shows a blended and nested plot on a log scale. It incorporates the world cities data used in Chapter 5. Compare this display to Figure 5.8 to see the difference logging makes. By logging, we are better able to see small and large population changes in the same plot.

```
SCALE: cat(dim(1))
SCALE: log(dim(2), base(10))
ELEMENT: point(position((city/group)*(pop1980+pop2000)))
```

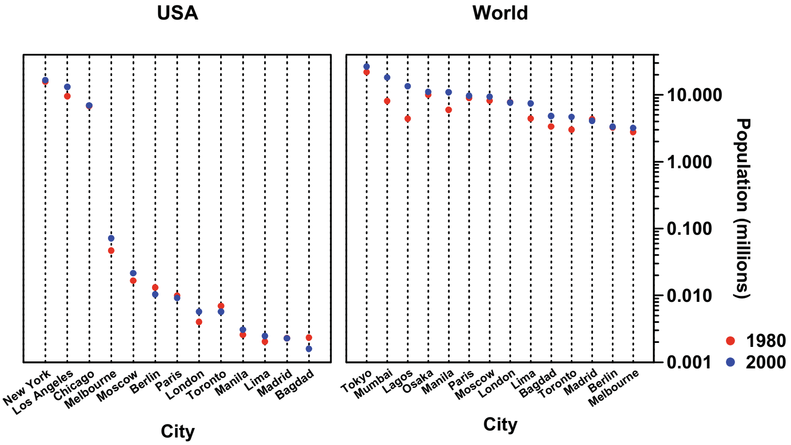


Figure 6.8 *Blended and nested scatterplot on log scale*

6.2.4.2 Power Scales

The syntax for *pow*() is

pow(*dim*(), *exponent*(), *min*(), *max*(), *ticks*(), *cycle*())

The extra parameters after *dim*() are optional. The *exponent*() function (default argument is .5, or square-root) sets the exponent. The *min*() and *max*() functions set minimum and maximum scale values. The *ticks*() function determines the number of ticks. The *cycle*() function takes an integer specifying how many cycles there are to be between *min*() and *max*(). The default is 1.

Figure 6.9 shows examples of power scales to several powers. For the inverse transformation ($p = -1$), we have reflected the scale to keep the polarity consistent. Reflecting the scale for negative powers makes it easier to compare the effects of different power transformations on graphics.

Tukey (1957) introduced and discussed the role of the simple algebraic transformation *pow*: x^p in data analysis. He elucidated how this transformation could be used to handle both negative and positive skewness in data and noted that it formed a continuously varying family of transformations (except, obviously, in the neighborhood of zero). He also showed that *pow* effectively subsumes the *log* transformation when p approaches zero.

If we reëxpress Tukey's transformation as *pow*: $(x^p - 1)/p$, we can see that *pow* approximates the *log* transformation when p approaches zero, since

$$\frac{d}{dp}x^p = x^p \log x$$

Box and Cox (1964) used this modified form of Tukey's power transformation and derived maximum-likelihood estimates for p under the normal regression model. This version of Tukey's *pow*() is called the Box–Cox transformation. Tukey was one of the discussants accompanying Box and Cox's paper.

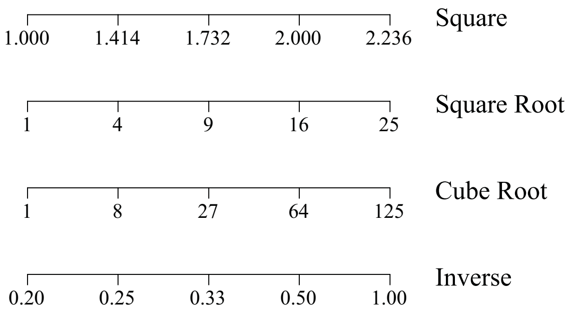


Figure 6.9 *Power scales*

It is difficult to find equally spaced nice numbers for power scales, except for reciprocals of positive integers (*e.g.*, square or cube roots). Consequently, it is easier to move the tick marks than to search for nice numbers on power scales. There is another advantage to this method. If we COORD transform the scale with the reciprocal of the exponent (see Chapter 9), the ticks and scale values will return to their original locations. To gain the same duality with logs, we implement the natural log scale by moving ticks and retaining the original scale values. Then the exponential coordinate transformation *exp*() will return the ticks to their original positions (see Figure 9.32).

Figure 6.10 illustrates three examples of tick-moving for power scales on the brain weight data for p = 1 (bottom), p = .5 (middle) and p = .01 (top). Although the scale values are different, the top plot is visually indistinguishable from a dot plot of the log-transformed values. Notice that the ticks and scale values move to the right as the exponent of the power transformation decreases. On a dynamic display system, this movement provides an additional cue to the change in shape of the distribution.

```
SCALE: pow(dim(1), exponent(.01))
ELEMENT: point.dodge(position(bin.dot(brainweight)))

SCALE: pow(dim(1), exponent(.05))
ELEMENT: point.dodge(position(bin.dot(brainweight)))

SCALE: pow(dim(1), exponent(1.0))
ELEMENT: point.dodge(position(bin.dot(brainweight)))
```

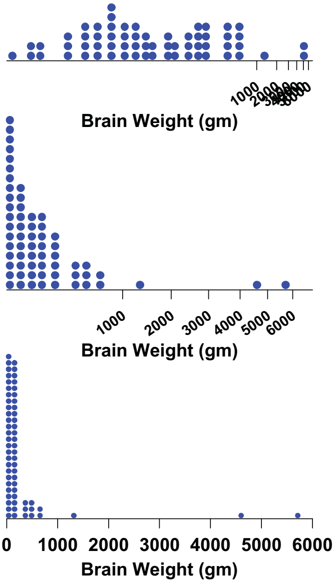


Figure 6.10 *Power transformations*

6.2.5 Two-bend Scales

The syntax for the two-bend scale transformations is, in general

function(*dim*(), ...)

The extra parameters after *dim*() are optional. These are usually probability distribution functions. Two-bend scales are ogival (S shaped) in their mapping functions. They include several statistical scales used to normalize data.

6.2.5.1 Arcsine Scales

Proportions fall in the closed interval [0,1]. The binomial proportion $p = x / n$ (*e.g.*, x heads in n coin tosses) has mean π and variance $\pi(1-\pi)/n$, where π is the population proportion estimated by p. This statistic thus has its variance dependent on its mean. The left panel of Figure 6.11 shows this behavior for 1,000 replications of 25 tosses of a coin. On the lower level is a dot plot of the sample distribution of the proportion of heads in 25 tosses when the coin is biased to come up heads only 10 percent of the time. On the middle level is the dot plot for a fair coin. And the top level shows the dot plot for a coin biased to yield heads 90 percent of the time.

We can derive a transformation of π such that the variance of the transformed variate is constant across its range. Following the inverse weighting approach for stabilizing variance in Section 9.1.4, and assuming that n is fixed, we can integrate the reciprocal function of this variance to get the standardized arcsine transformation

$$asn\colon \pi \rightarrow 2\,asin\sqrt{\pi}$$

In other words, since the derivative of this transformation,

$$1/\sqrt{\pi(1-\pi)}$$

is proportional to the reciprocal of the standard deviation, we end up holding the standard deviation relatively constant for different location values of the transformed variate (Rao, 1973).

The right panel of Figure 6.11 illustrates this transformation on the coin toss data. Notice that the values near the bounds of the [0,1] interval are stretched compared to those in the middle. The only difference in the specifications between the left and right panels is the addition of the *asn*() function to the TRANS specification. As with power scales, it is easier to move tick marks than to search for round numbers in the inverse domain.

ELEMENT: *point.dodge*(*position*(*bin.dot*(p|)) ELEMENT: *point.dodge*(*position*(*bin.dot*(p|)),
SCALE: *asn*(*dim*(1))

ELEMENT: *point.dodge*(*position*(*bin.dot*(p|)) ELEMENT: *point.dodge*(*position*(*bin.dot*(p|)),
SCALE: *asn*(*dim*(1))

ELEMENT: *point.dodge*(*position*(*bin.dot*(p|)) ELEMENT: *point.dodge*(*position*(*bin.dot*(p|)),
SCALE: *asn*(*dim*(1))

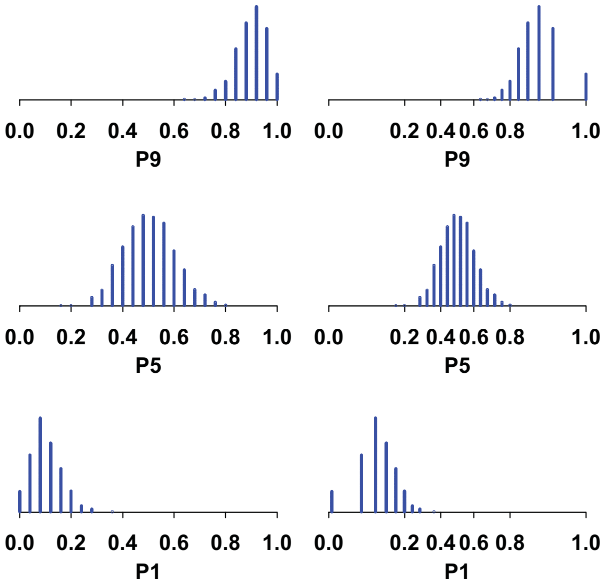


Figure 6.11 *Proportion of heads in 25 tosses for π =.1, .5, .9 (1,000 samples)*

6.2.5.2 Logit and Probit Scales

The *logit* and *probit* transformations are based on probability frames. When *p* is assumed to be a probability measure on the interval [0,1], we can use the inverse probability function to transform *p* into a distribution. When probability density functions are symmetric and unimodal, these transformations have an ogive shape similar to the arcsine transformation. The lower panel of Figure 6.12 shows two of these functions, the *probit*, based on the standard normal distribution:

$$p = \Phi(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}z^2} dz$$

and the *logit*, based on the logistic distribution:

$$p = L(x) = \int_{-\infty}^{x} \frac{e^{-z}}{(1 + e^{-z})^2} dz = \frac{e^x}{1 + e^x}$$

```
DATA: x = iter(0., 1., 0.01)
TRANS: y = asn(x)
ELEMENT: line(position(x*y))
```

```
DATA: x = iter(0., 1., 0.01)
TRANS: y = probit(x)
ELEMENT: line(position(x*y))
```

```
DATA: x = iter(–1., 1., 0.01)
TRANS: y = ath(x)
ELEMENT: line(position(x*y))
```

```
DATA: x = iter(0., 1., 0.01)
TRANS: y = logit(x)
ELEMENT: line(position(x*y))
```

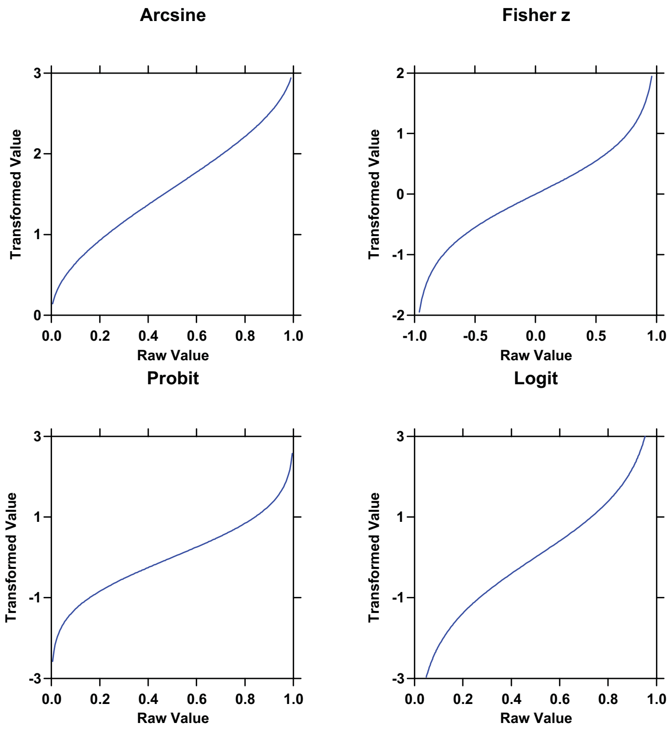


Figure 6.12 *Probit, logit, arcsine, and Fisher's z transformations*

We have drawn the functions in the lower two panels of Figure 6.12 as $\Phi^{-1}(p)$ and $L^{-1}(p)$, respectively, in order to match them to the orientation of the other panels in the figure. Inverse probability functions are usually ancillary to formal statistical modeling using these distributions. Sometimes we need to use them directly, however. One example is probability plotting (*e.g.*, Chambers *et al.*, 1983), in which we plot ordered data values against the inverse probability values of their fractiles, assuming a particular distribution. We also use these transformations occasionally in statistical modeling where the data are probabilities, such as in meta-analysis (Hedges and Olkin, 1985).

6.2.5.3 Fisher's z Scale

Fisher (1915; see also 1925) developed a transformation to stabilize the variance and skewness of the Pearson correlation coefficient:

$$z = \frac{1}{2}\log\left(\frac{1+r}{1-r}\right) = \operatorname{atanh} r$$

Fisher's transformation can be understood with the same logic that we used for the arcsine transformation. In this case, the derivative of the transformation is $1/(1-r^2)$, which is proportional to the reciprocal of the standard deviation of r. Figure 6.12 shows this transformation in the upper right panel. The range of the correlation is (–1, 1) but the shape of the transformation closely resembles the others for proportions and probabilities.

6.2.6 Probability Scales

The syntax for *prob*() is

prob(*dim*(), *distribution*(<*parameters*>), ...)

The extra parameters vary by type of distribution. The default is the normal probability distribution, which yields the same result as *probit*().

We have already seen the *logit* and *probit* scale types under Section 6.2.5. Other probability distribution functions can be used for a variety of purposes. Figure 6.13 shows a gamma probability plot of the firing rate variable in the cat dataset used in Figure 3.5. We have square-rooted the firing rate scale and applied a gamma probability scale with parameter 40 to the fractiles. This joint rescaling straightens out the cumulative distribution function shown in the right panel of the figure. The compression of the tick marks at the center of the vertical scale is the same phenomenon we see with the transformations in Figure 6.12. Since the gamma probability density is not symmetric, however, the tick marks do not compress the same amount at each end.

We might want to follow the enumeration strategy used for log scales in order to enhance the detail at the ends of the probability scale. That is, we have room to add tick marks on the lower end at .01, .001, .0001, and so on. Similarly, we could add marks at .99, .999, .9999 at the higher end. Grid lines at these major tick marks could help in decoding information.

Compare this result with Figure 4.2. What is the difference between using a variable transformation and a scale transformation to produce a probability plot? Does this difference affect real-time events such as brushing, linking, and metadata access? The linearized probability plot is based on the idea that it is easier to perceive straight lines than to evaluate curvilinear cumulative distribution functions in the mind's eye. Are there other graphical applications where a probability scale would be useful? If so, is there any 3D application?

```
TRANS: p = prank(rate)
SCALE: pow(dim(1), exponent(.5))
SCALE: prob(dim(2), gamma(40))
ELEMENT: point(position(rate*p))

TRANS: p = prank(rate)
ELEMENT: point(position(rate*p))
```

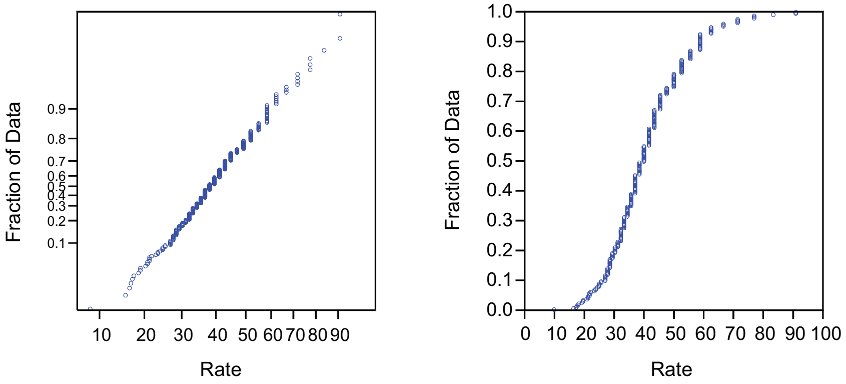


Figure 6.13 *Gamma probability scale*

6.3 Sequel

We now have the tools for making a wide range of basic graphics. More complex graphics require statistical functions. The next chapter covers the statistical functions that graphing functions use to create statistical graphs.

7

Statistics

Statistics state the status of the state. All these *s* words derive from the Greek $\sigma\tau\acute{\alpha}\sigma\iota\varsigma$ and Latin *status*, or standing. Standing (for humans) is a state of being, a condition that represents literally or figuratively the active status of an individual, group, or state. Modern statistics as a discipline arose in the early 18th century, when collection of data about the state was recognized as essential to serving the needs of its constituents. This Enlightenment perspective gave rise not only to the modern social sciences, but also to mathematical methods for analyzing data measured with error (Stigler, 1983).

In a graphical system, **statistics** are methods that alter the position of geometric graphs. We are accustomed to think of a chart as a display of a statistic or a statistical function (*e.g.*, a bar chart of budget expenditures). As such, it would seem that we should begin by aggregating data, computing statistics, and drawing a chart. This would be wrong, however. By putting statistics under control of graphing functions, rather than whole charts under the control of statistics, we accomplish several things. First, we can represent more than one statistic in a frame. One graphic can represent a mean and another a median, in the same frame. Second, making statistics into graphing methods forces them to be views or summaries of the raw data rather than data themselves. In other words, the casewise data and a graphic are inextricably bound because we never break the connection between the variables and the graphics that represent them. This allows us to drill-down, brush, and investigate values with other dynamic tools. This functions would be lost if we pre-aggregated the data. Finally, by putting statistics under the control of graphing functions, we can modularize and localize computations in a distributed system. Adding graphics to a frame is easy when we do not have to worry about the structure of the data and how aggregations were computed. We will return to this issue in Section 7.3 at the end of this chapter.

The simplest graphing method is the one students first learn for plotting algebraic functions: for every x, compute $f(x)$ so that one may draw a graph based on the tuples of the form $(x, f(x))$ that comprise the graph. Students learn to construct a list of these tuples (a finite subset of the graph of the function) in order to plot selected points in Cartesian coordinates. In the functional notation of this book, students usually draw graphs of algebraic functions using the graphing function *line*(*position*(*f*()).

While students learn graphing methods for polynomial and other simple algebraic functions, most charts are based on statistical functions of observed values of one or more variables. In our notation, examples of statistical graphs are produced by the functions

point(position(*summary.mean*())) and
line(position(*smooth.linear*())),

which implement the statistical graphing functions *summary.mean*() and *smooth.linear*(), respectively. Statistical functions can be complicated, but their output looks the same to their geometric clients as the output of algebraic functions. A *line* does not care who produced the points it needs to plot itself.

Statistics are static (unchanging) methods that are available to all geometric graph types. For example, we can use

interval(*position*(*summary.mean*())) or
point(*position*(*summary.mean*())) or
line(*position*(*summary.mean*()))

to produce a geometric graph of a mean. Some of the combinations of graphs and statistical methods may be degenerate or bizarre, but there is no moral reason to restrict them.

Statistics have the potential to alter the appearance of graphics, in some cases as radically as do coordinate transformations. It is sometimes difficult to recognize a graphic after its geometry has been altered by a statistic and, conversely, it often can be difficult to infer a statistical function from the geometry of a graphic. For example,

line(*position*(*smooth.linear*()))

creates a single regression line, while

line(*position*(*region.confi.smooth.linear*()))

creates a pair of lines that delineate a confidence interval on a smoother. They are easy to distinguish. However, both the linear regression

line(*position*(*smooth.linear*()))

and the mean smoother

line(*position*(*smooth.mean*()))

can create straight lines. Sometimes we can distinguish them only if there are differences in the data on which they are based.

This chapter invokes some methods whose particulars are beyond the scope of this book. We have used these in examples to illustrate the diversity of a statistical graphing system and to alert statisticians to the design issues involved in putting statistical procedures at the service of graphics. Although the topic is graphics, the organization of this chapter, summarized in Table 7.1, gives some indication of the way we would design an object-oriented statistical package as well. If the design principles of **orthogonality** (as much as possible, everything should work everywhere and in every combination) and **hierarchy** (complicated tasks should be done by enlisting the support of simple helpers) are applied to statistical methods, a comprehensive package requires a minimum of code.

One can consult a statistics text for further information, although there are many technical issues that are not immediately apparent when looking at statistical formulas. The best sources for further information about computing and statistics are Chambers (1977), Kennedy and Gentle (1980), Maindonald (1984), and Thisted (1988).

7.1 Methods

Table 7.1 lists the most important statistical methods available to graphs. They fall under five super-classes: *bin* (partitions and meshes for tiling and histograms), *summary* (basic statistics), *region* (interval and region bounds), *smooth* (regression, smoothing, interpolation, and density estimation), and *link* (methods for computing edges of graphs based on a set of nodes or points). The table is not exhaustive, of course. We have tried to include enough examples to make clear where new methods can be added to this system and also to cover the examples in this book.

Table 7.1 **Statistical Methods**

Bin	***Summary***	***Region***	***Smooth***	***Link***
rect	*count*	*spread*	*linear*	*join*
tri	*proportion*	*sd*	*quadratic*	*sequence*
hex	*sum*	*se*	*cubic*	*mst*
quantile	*mean*	*range*	*log*	*delaunay*
boundary	*median*	*confi*	*mean*	*hull*
voronoi	*mode*	*mean*	*median*	*tsp*
dot	*sd*	*sd*	*mode*	*complete*
stem	*se*	*smooth*	*spline*	*neighbor*
	range		*density*	
	leaf		*normal*	
			kernel	

Many statistical methods rely on a scalar quantity called a **loss**, which we attempt to minimize when calculating statistics. Others rely on a scalar quantity called a **likelihood** that we attempt to maximize. We may qualify a method by appending the name of its loss or likelihood function to the method name. Ordinary least squares, called *ols*, involves a quadratic loss function of the residuals (differences between fitted and observed values). Robust estimation, by contrast, involves a variety of loss functions that are designed to downweight large residuals (Huber, 1972; Tukey, 1977). These weighted loss functions have names like *biweight*. We may distinguish different methods by adding the proper suffix. For example, we can specify *smooth.linear.ols* or *smooth.linear.biweight*. These examples follow the syntax *smooth.model.loss*. We will now summarize the methods in each of the super-classes.

7.1.1 Bin

The *bin* methods partition a space. Binning must happen before other statistics are calculated. Binning produces two results. First, binning methods use a mesh or tiling that affects the position and shape of geometric objects embedded in the space partitioned by a mesh. For example, rectangular binning of a 1D or 2D space turns a bar chart into a histogram, with the width of the bars determined by the spacing of the mesh. Second, binning tags cases with an identifier of the bin in which they fall. In this way, statistics executed after binning are calculated separately within each bin. In short, binning takes scattered points and collects them into a (usually much smaller) set of regions within which statistics may be calculated by other statistical functions.

The *rect* binning method partitions a space with equally spaced cutpoints on one or more dimensions. In two dimensions, the regions partitioned are rectangles. The *tri* method partitions the plane with equilateral triangles. The *hex* method partitions the plane with hexagons. The *quantile* methods partition by computing sample quantiles (such as quartiles, deciles, or percentiles). An example is *bin.quantile.quartile*. Quantiles from fractions i / n, (i = 1, ..., n) can be used to construct a **sample cumulative distribution function**, on which the sample density is based. Letter values Tukey (1977), used for the box plot are produced by *bin.quantile.letter* and for the stem-and-leaf plot by *bin.stem*.

The *boundary* method partitions the plane with irregular polygons that are usually derived from geographic boundary or shape files. In geographic maps, these polygons represent entities such as counties, states, or countries. Most Geographic Information Systems (GIS) include a key for relating statistical and polygon data. This is efficient but not necessary. We can instead compute membership by testing if a point is inside a polygon.

The remaining methods surround each point in a space with a polygon. The *voronoi* method partitions the plane with Voronoi polygons. By definition, there is only one point contained in a polygon (assuming no ties). We may merge Voronoi polygons, however, to produce a clustering of points. This method is used to create a **gap histogram** (Figure 7.15). The *dot* method partitions a space with an irregular mesh designed to cluster unidimensional data. Some of the resulting polygons overlap, so that *point* elements can be used to produce symmetric or asymmetric dot plots (Wilkinson, 1999).

7.1.2 Summary

The *summary* class includes statistical algorithms for producing a single value that comprises a statistical summary. The *count, proportion,* and *sum* methods return simple or weighted counts, proportions, or sums. The *mean*, *median*, and *mode* methods return measures of location. The *sd*, *se*, and *range* methods return measures of dispersion. Some might regard these methods as disparate from a computational point of view. From a graphical point of view, however, they are all common objects; all produce a result that is graphable as a geometric point. A dot plot of standard deviations within groups, for example, would require the graphing function *position*(*statistic.sd*()) to make a point for each standard deviation. We could also plot standard deviations as intervals using the function *position*(*region.spread.sd*()), but if our interest is only in dispersion and not location, it is easier to examine standard deviations directly than to view them as differences between the bounds of intervals. Finally, the *leaf* function computes leaves for a stem-and-leaf plot using a modulo function on the binning used on the stems.

7.1.3 Region

The *region* class includes statistical algorithms that produce two values bounding an interval in one dimension, or a set of vertices bounding a region in higher dimensions. There are two ways to do this.

The first method is to use a measure of spread (*sd*, *range*, etc.). Examples are *region.spread.sd* and *region.spread.range*. The *sd* method is the default.

The second method for computing a *region* is to use a theoretical distribution to compute a confidence interval, not necessarily symmetric, on some statistic. This is what the *confi* method does. Unlike *spread*, which bounds only points, the *confi* method can construct regions that bound a variety of statistical objects. For example, *region.confi.mean* is a method for producing a confidence interval on a mean using a probability distribution (usually Student's *t* distribution). The method *region.confi.sd* produces a confidence interval on a standard deviation. The method *region.confi.smooth.linear* produces a confidence interval on an ordinary regression line. As this last example indicates, we can append to the *confi* method any statistical methods for which we know how to produce a valid (under appropriate assumptions) confidence interval. This can result in a rather long method function, but one that makes sense hierarchically (from left to right) and also reveals the order in which we do the computations (from right to left).

7.1.4 Smooth

The *smooth* class includes a variety of methods for computing smoothed values. The term derives from the smoothness of a parametric function, but it covers a more general class of functions of variables that produce connected sets of values. These include ordinary linear regression, interpolators, and step functions that pass through data values. The word *smooth* should not be taken to imply that the result has many derivatives or is even continuous.

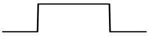
The *linear* smoothing method is perhaps the most widely used in practice. It computes the function $f(\mathbf{x}) = \mathbf{xb}$ for any real-valued input vector $\mathbf{x}$. The default *linear.ols* smoother computes the vector $\mathbf{b}$ via the method of least squares on the set of pairs $\{(\mathbf{x}_i, y_i): i = 1, ..., n\}$ taken from a set of data values. If our frame is $\boldsymbol{x}*\boldsymbol{y}$, then $\mathbf{x} = \boldsymbol{x}$ and $y = \boldsymbol{y}$. The graphic of this 2D smooth is a straight line whose intercept and slope are b_0 and b_1, respectively. If our frame is $\boldsymbol{x}*\boldsymbol{y}*\boldsymbol{z}$, then $\mathbf{x} = (\boldsymbol{x}, \boldsymbol{y})$ and $y = \boldsymbol{z}$. The graphic of this 3D smooth is a plane whose intercept and slopes are b_0, b_1, and b_2, respectively.

The *quadratic* smoothing method fits a quadratic polynomial using a linear model $f(\mathbf{x}) = \mathbf{xb}$. If our frame is $\boldsymbol{x}*\boldsymbol{y}$, then $\mathbf{x} = (\boldsymbol{x}, \boldsymbol{x}^2)$ and $y = \boldsymbol{y}$. The graphic of this 2D smooth is a parabola. If our frame is $\boldsymbol{x}*\boldsymbol{y}*\boldsymbol{z}$, then $\mathbf{x} = (\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{x}^2, \boldsymbol{y}^2, \boldsymbol{xy})$ and $y = \boldsymbol{z}$. The graphic of this 3D smooth is a parabolic surface or a saddle. The *cubic* method and higher-degree polynomial smoothers are defined similarly.

The *log* smoothing method fits the function $f(x) = log(a) + b\,log(x)$ to compute smoothed values. If our data follow the model $y = ax^b\varepsilon$, with ε representing identically distributed independent errors, then we can log transform y and x to fit this model with ordinary least squares. Estimating the coefficients of other nonlinear models usually requires iterative methods that must be carefully designed to avoid local minima and loss of accuracy (Dennis and Schnabel, 1983). Robust nonlinear methods can be implemented through the same terminology used for linear methods (*e.g.*, *smooth.log.biweight*).

The *mean*, *median*, and *mode* smoothing methods produce a constant smoother. A graphic of *line*(*position.smooth.mean*()) in the frame **x*y** is a horizontal line whose height corresponds to the mean of $\boldsymbol{y}$. In the frame **x*y*z**, a graphic of *line*(*position.smooth.mean*()) is a horizontal plane positioned at the mean of **z**. These constant smoothers are useful as reference objects in graphics (to mark a mean level, for example). The *smooth.median* method is useful for computing reference levels when the values to be smoothed are skewed or contain outliers. The *smooth.mode* method is useful for computing reference levels when the values to be smoothed are categorical.

So far, we have given global definitions for parametric smoothers. There is also a large literature in statistics that covers what is usually called **nonparametric smoothing**, **locally parametric smoothing**, or **kernel smoothing** (Härdle, 1990; Hastie and Tibshirani, 1990; Scott, 1992; Green and Silverman, 1994; Fan and Gijbels, 1996; Simonoff, 1996). These local smoothing methods are flexible, follow the data closely, and are especially useful for exploratory data analysis. They work by computing a weighted fit of a smoothing model inside a window containing an ordered subset of the data. The set of weights used to define the smoothing window is derived from a probability **kernel** function, which is a function on the real numbers with integral $1/n$, where n is the number of cases in the set of data to be smoothed. In 2D, the most useful window functions are the following:

 *uniform*: $f(x) = a : (-w \le x \le w)$, else 0

epanechnikov: $f(x) = a(1 - (x/w))^2 : (-w \le x \le w)$, else 0

biweight: $f(x) = a(1 - (x/w)^2)^2 : (-w \le x \le w)$, else 0

triweight: $f(x) = a(1 - (x/w)^2)^3 : (-w \le x \le w)$, else 0

tricube: $f(x) = a(1 - |x/w|^3)^3 : (-w \le x \le w)$, else 0

gaussian: $f(x) = ae^{-(x/w)^2}$

cauchy: $f(x) = a/(b + (x/w)^2)$

The constant a that scales these formulas as probability kernel functions may be set to 1 for smoothers because it cancels out in the algorithms if they are designed properly. The constant w determines the width of the window or, for kernels like *gaussian* and *cauchy* that have nonzero tails, the spread of the kernel. We can adapt these functions to 3D smoothing (assuming x and y are independent and identically distributed) by transforming them into polar coordinates. Non-circular 3D window functions are slightly more complex. See Figure 14.5 for a graphical example of how kernel smoothing works.

There are two simple ways, among others, that we can set the window width w for computing a smoothed value at a location x_i (not necessarily a data point). The first is to set w to be *fixed* at a chosen **bandwidth** value for all x_i. The second is to set w to be the distance from x_i to the kth nearest neighbor in the data. This method, called *knn*, generally means that the width of a window will vary at different points x_i. The *fixed* window guarantees that smoothed values will be based on a common range and the *knn* window guarantees that they will be based on the same number of data points, namely k.

As a consequence of their evolution, nonparametric smoothers have acquired many different names, but there is a simple scheme that makes it easy to implement almost every one of them in an object-oriented system. Each of the global methods we have described (*linear*, *quadratic*, *cubic*, *log*, *mean*, *median*, and *mode*) can be turned into a local method by adding the name of the type of kernel weighting function used to do the smoothing. An example is *smooth.linear.epanechnikov*. For local smoothers, the default bandwidth is *fixed* and the default loss function is *ols*. If we wish to specify *knn* or other loss functions, we can use the extended syntax *smooth.model.kernel.window.loss*. An example is *smooth.mean.uniform.knn.biweight*.

This scheme yields many of the kernel and polynomial smoothers described in the literature under other names. For example, a **running-means** or **moving-averages** smoother (Makridakis and Wheelwright, 1989) is *smooth.mean.uniform*. The Nadaraya–Watson kernel smoother (Nadaraya, 1964; Watson, 1964) is *smooth.mean.epanechnikov*. Shepard's smoother

(Shepard, 1965), sometimes called an **inverse-distance** smoother (McLain, 1974), is *smooth.mean.cauchy*. The **distance-weighted least squares** (DWLS) smoother (McLain, 1974) is *smooth.quadratic.gaussian*. The **step** smoother (Cleveland, 1995) is *smooth.mean.uniform.knn* with $k = 1$ (the number of nearest neighbors is set to 1). For convenience, we will name this *smooth.step* in the examples. The image-processing digital filter called a **discrete gaussian convolution** that is used to smooth black-and-white images (Gonzalez and Wintz, 1977) is *smooth.mean.gaussian.knn*. It is a *knn* method because the pixels on which it operates are evenly spaced, so it does weighted averages over a fixed number of pixels. Finally, Cleveland's *loess* smoother (Cleveland and Devlin, 1988) is *smooth.linear.tricube.knn.biweight*. Notice that Cleveland's loss function involves robust biweighting instead of ordinary least squares, which makes his smoother resistant to outliers. For convenience, we have denoted it as *smooth.loess* in the examples. The quadratic version of *loess* is *smooth.quadratic.tricube.knn.biweight*.

The local version of the *median* smoother is discussed in Tukey (1977). Tukey's smoother is equivalent to *smooth.median.uniform.knn*. The local *mode* method is based on a kernel estimate of the conditional mode, discussed in Scott (1992). The *spline* method implements cubic splines, which are piecewise cubic polynomials. These are used most frequently for smooth interpolation of a set of data points (Lancaster and Salkauskas, 1986) but also have applications in nonparametric regression (Wahba, 1990). Finally, note that all the *smooth* methods provide the possibility of computing confidence intervals when they are defined. This is done by using the *region.confi* method prefix on the smoother, as in *region.confi.smooth.linear*.

Finally, the *density* methods estimate a density using a variety of smoothers. The *normal* method estimates the parameters of the normal distribution from sample data and returns the density function. Other parametric densities can be computed similarly. The *kernel* methods perform kernel density estimation using the kernels discussed under smoothing methods above. An example is *density.kernel.epanechnikov*. Silverman (1986) and Scott (1992) discuss kernel density estimation.

7.1.5 Link

Finally, the *link* methods input tuples representing nodes in a graph, and return edges according to various algorithms. Given a list of n nodes (n assumed even) represented by the indices $\langle 1, 2, \ldots n \rangle$, the *join* method returns a list of edges represented by the indices $\langle (1, 1 + n/2), (2, 2 + n/2), \ldots \rangle$. This method is useful for blending two varsets of nodes because it pairs the first node in the first column with the first node in the second, and so on. We use this method for constructing trees from parent-child lists. The *sequence* method links adjacent nodes in a sequence $\langle 1, 2, \ldots n \rangle$, producing the list of links $\langle (1, 2), (2, 3), \ldots \rangle$.

The *mst* method returns the minimum spanning tree, which is the shortest set of edges (in terms of total length) connecting every node. The *delaunay* method returns the Delaunay triangulation of the nodes, which is the triangulation of a set of points that is as close as possible to being isosceles, thus minimizing narrow triangles. The *hull* method returns the convex hull of a set of points, which is the set of edges connecting the outermost nodes in a planar graph. In three dimensions, the convex hull is made up of planar facets instead of lines. The *tsp* method computes the traveling salesman problem, or at least an approximation. This computation involves finding the shortest path through a set of points such that every point is visited once. The lonely salesman is usually allowed to return home. The *complete* method returns the set of edges in a complete graph for a set of nodes. The *neighbor* method returns the edges in a nearest-neighbor graph.

Algorithms for many of these methods, including the Voronoi tessellation used for *bin*, have been developed by computational geometers. The standard reference in this field is Preparata and Shamos (1985). A more recent reference is O'Rourke (1998).

7.1.6 Conditional and Joint Methods

There are two subclasses of statistical functions: **conditional** and **joint**. The input to the conditional form of the functions is a finite set $\{(\mathbf{x}_i, y_i)\}$, where $\mathbf{x}$ is a d-dimensional vector variable and $i = 1, ..., n$. This set is defined by the d factors of the common term $\boldsymbol{x}_1 * \boldsymbol{x}_2 * ... * \boldsymbol{x}_d * \boldsymbol{y}$ found after expanding the frame specification to algebraic form using the distributivity axioms presented in Chapter 5. The input to the joint form of the functions is the finite set of data $\{\mathbf{x}_i\}$, where $\mathbf{x}$ is a d-dimensional vector variable and $i = 1, ..., n$. This set is defined by the d factors of the common term $\boldsymbol{x}_1 * \boldsymbol{x}_2 * ... * \boldsymbol{x}_d$.

The conditional methods compute $f(\mathbf{x})$ from y conditioned on values of $\mathbf{x}$. These include methods like linear and nonlinear regression. The joint methods compute $f(\mathbf{x})$ from $\mathbf{x}$ alone. These include methods like principal components, orthogonal regression, minimum spanning trees, and network algorithms. In most joint methods, $f(\mathbf{x})$ is a multi-function. In the minimum spanning tree method, for example, we input a set of tuples $\{\mathbf{x}_1 ... \mathbf{x}_n\}$ and output a set of nodes and edges comprising a minimum spanning tree.

We designate conditional or joint methods by adding the suffix *conditional* or *joint* to statistical method functions. For example, if we have a frame consisting of **x*****y**, we can specify the conditional means of **x** given **y** through the graphing function *statistic.mean.conditional*() and we can specify the centroid of **x** and **y** as *statistic.mean.joint*(). The *conditional* methods are useful for displaying prediction models and the *joint* methods are useful for displaying multivariate distributions. See Table 7.2 for examples.

7.1.7 Form and Function

The consequence of distinguishing statistical methods from the graphics displaying them is to separate form from function. That is, the same statistic can be represented by different types of graphics, and the same type of graphic can be used to display two different statistics. Figure 7.1 illustrates the former and Figure 7.2 the latter. This separability of statistical and geometric objects is what gives a system a wide range of representational opportunities. Objects such as error bars, regression lines, and confidence intervals have evolved into customary forms that employ bars, lines, or other geometric objects. This makes them recognizable by viewers trained in certain disciplines, but we must understand that these representations do not necessarily reflect the deeper structure of the graphs. For example, stockbrokers are accustomed to high-low-close plots and scientists are familiar with error bars. Both are using the same graphs but different statistical methods. However, we expect to see binned data displayed using bars in a histogram, but sometimes points or lines are more useful for displaying bins.

No less important, the separation of form and function allows us to conserve computer code. The groupings in Table 7.1 are based not only on taxonomies drawn from the statistical literature, but also on shared algorithms needed to execute these functions. This saves both time and space.

Notice in Figure 7.1 that *region.confi.smooth.linear*() returns intervals ranging between upper and lower confidence bounds on a line, so that each graphic adjusts itself to an interval: *point* marks the upper and lower bound, *area* fills in the area between the bounds, *line* delineates the upper and lower bounds, and *interval* marks the bounds with a set of bars.

In Figure 7.2, *line* responds differently to functions (*statistic*, *smooth*) and multifunctions (*spread*, *confi*). For multifunctions, *line* is split into two or more segments. This is shown in the bottom two panels. The line splits in the lower left panel wherever there is more than one value on female for a value of birth, spanning the range of the data at that point.

```
ELEMENT: point(position(region.confi.smooth.linear(female*birth)))

ELEMENT: line(position(region.confi.smooth.linear(female*birth)))

ELEMENT: area(position(region.confi.smooth.linear(female*birth)))

ELEMENT: interval(position(region.confi.smooth.linear(female*birth)))
```

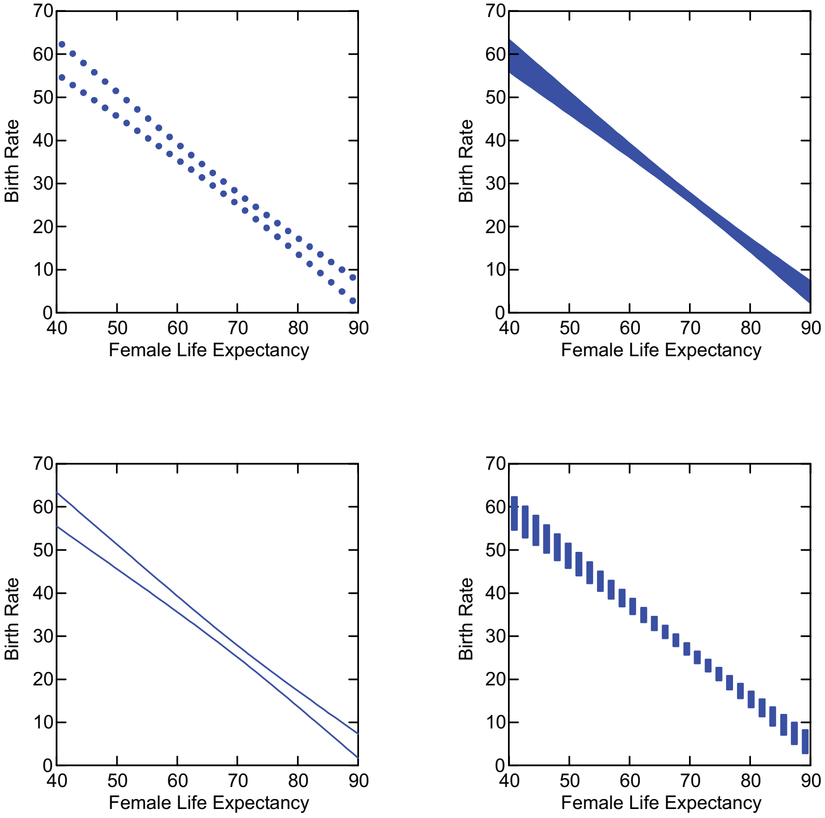


Figure 7.1 *Different graph types, same statistical method*

ELEMENT: *line*(*position*(*summary.mean*(female*birth)))

ELEMENT: *line*(*position*(*region.spread.range*(female*birth)))

ELEMENT: *line*(*position*(*smooth.quadratic*(female*birth)))

ELEMENT: *line*(*position*(*region.confi.smooth.linear*(female*birth)))

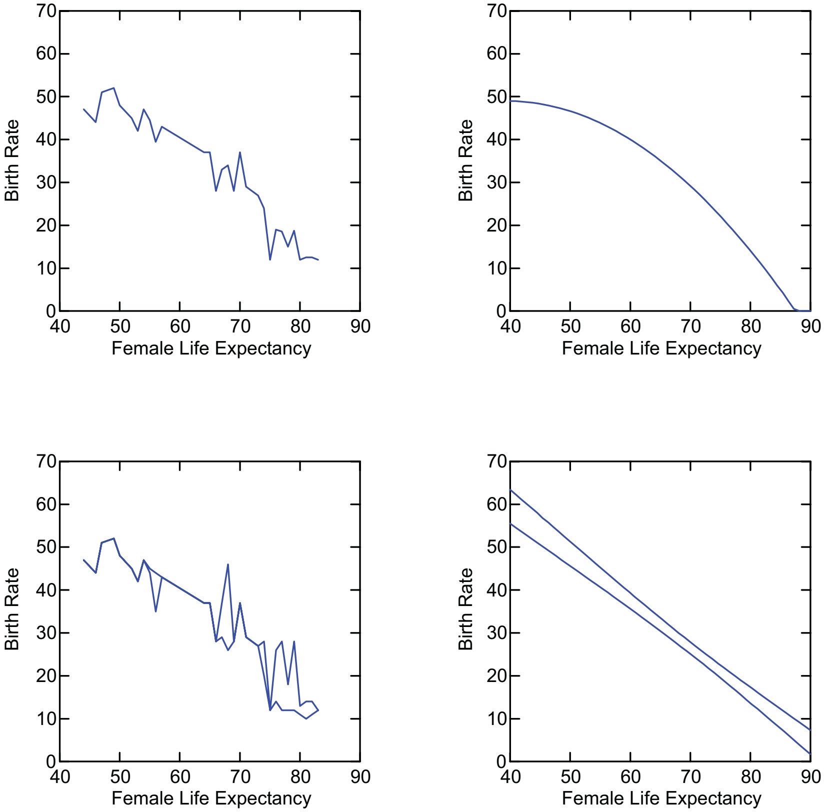


Figure 7.2 *Different statistical methods, same graph type*

7.2 Examples

Now we will proceed to explore the effects of these functional methods on each type of geometric element. The remainder of this section is organized by graph type.

7.2.1 Point

Graphs have a default statistical function. Their position is usually determined by one or more values on the domain and range. Graphs like *schema* employ a multivalued function (multifunction) that outputs values needed to determine the position of features like a midrange box, whiskers, and the outer points (outliers). For other graphs like *point*, *line*, *interval*, or *area*, position may be determined by a single-valued function (such as a mean) or a multifunction (such as a confidence interval). The type of the function determines how many shapes will be drawn for a given value in the domain.

7.2.1.1 Means in 2D

Figure 7.3 illustrates a *point* graphic on the countries data used in Chapter 1. The gov variable represents a classification of the type of government for each country. The female variable represents female life expectancy. Later graphics will also use male, the corresponding variable for male life expectancy. While Figure 1.1 used a subset of the countries, the full dataset of 57 countries is used in this chapter.

ELEMENT: *point*(*position*(*summary.mean*(gov*birth)))

ELEMENT: *point*(*position*(*summary.mean*(female*birth)))

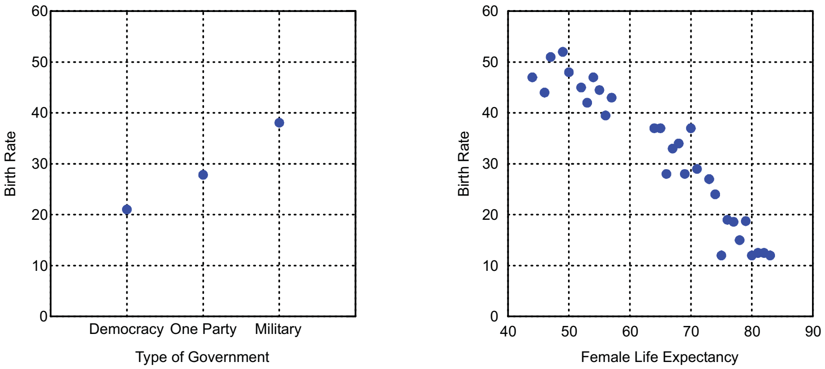


Figure 7.3 *Points on categorical (left) and continuous (right) domains*

The *point* graph can use a summary function to determine location in a frame when there are duplicate cases in a subclass. For example, if we use the *summary.mean*() method, then each point represents the mean of the birth values within each category of gov or value of female. Every dot represents a mean, even for continuous domains. The right graphic in Figure 7.3 looks like the cloud in the right panel of Figure 8.1 only because most of the data points are singletons (tuples with a unique first value). The mean of a singleton is the singleton's value.

7.2.1.2 Means in 3D

Figure 7.4 shows an example of *point* graphics embedded in a 3D frame using the *summary.mean*() function. We have used a three-dimensional marker. Because of scale decoding and perspective illusion problems in 3D graphics (Cleveland, 1985; Kosslyn, 1994), it is usually better to represent summary graphics in 2D by facets (see Chapter 11). With point summary graphics, we want to be able to compare means across different groups. We need to discern the values of the means as well as their differences. The 3D environment makes this especially difficult, even when there are grid lines in the background. Our task in interpreting 3D scatterplots and surfaces, however, is quite different. With those objects, we need to discern the shape of the entire cloud or surface and make a holistic judgment on the relations among the variables assigned to the axes. The 3D display can facilitate these tasks.

```
COORD: rect(dim(1, 2, 3))
ELEMENT: point(position(summary.mean(urban*gov*birth)),
         shape(shape.cube))
```

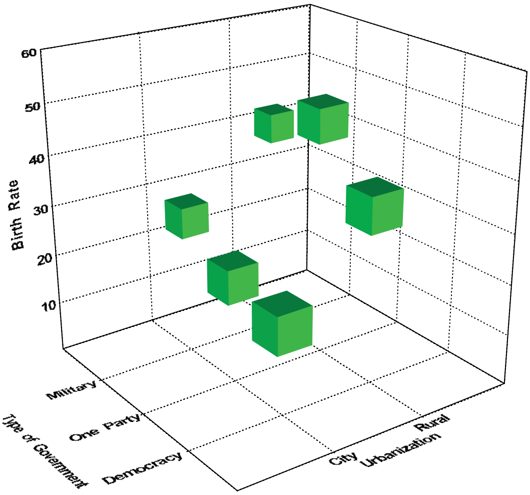


Figure 7.4 *3D point plot on categorical domain*

7.2.2 *Line/Surface*

Lines and surfaces are the same geometric graph type. The differences we associate with them are related to dimensionality rather than geometry. This subsection illustrates several variations in these graphs produced by different statistical methods. Some are simple functions and others are multifunctions.

7.2.2.1 Range Lines

Figure 7.5 shows a time series of the daily price of the stock of SPSS. The same series is used in Figure 7.13. Because we used the graphing multifunction *position.region.spread.range*() to position the *line* graph, we get two lines, one connecting the low and the other connecting the high values. This creates an envelope for the daily stock price.

Graphs like *point* and *line* divide into two or more instances when driven by multifunctions. Other graphs like *interval* and *area* do not divide for two-valued multifunctions because they are intrinsically interval valued. Other examples of this behavior are shown in Figure 7.1.

We plotted this series on a time scale using a monthly format. Other formats can be specified with the GUIDE statement (see Chapter 12). Notice that the tick marks bracketing the month of February are closer together than those bracketing other months. This means that the time scale on the horizontal axis is an interval (rather than categorical) scale. See Chapter 6 for a more detailed discussion.

SCALE: *time*(*dim*(1))
GUIDE: *axis*(*dim*(1), *format*("mmm"))
ELEMENT: *line*(*position*(*region.spread.range*(date*(high+low))))

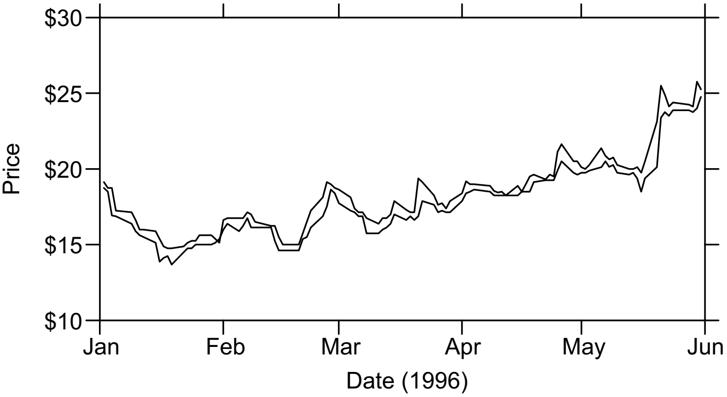


Figure 7.5 *Line plot of high-low-close*

7.2.2.2 The Loess Smoother

The preceding graphing functions compute their summaries using data only for values in the range (y) for a given value in the domain (x_i). The *smooth* function uses other (x, y) values to compute an estimate at x_i. Parametric global smoothers do this by means of a function whose parameters are estimated from all the (x, y) values. Nonparametric smoothers do this with local functions estimated from (usually overlapping) values in the neighborhood of x_i.

Figure 7.6 shows a scatterplot of the birth rates against female life expectancy. It employs the *loess* smoother (Cleveland and Devlin, 1988), which is a locally weighted robust regression method. The smoother is specified by the *smooth.loess* function. The advantage of *loess* over other smoothers is that it employs a robust fitting method that makes it resistant to outliers in the data.

Loess works by fitting polynomial (usually linear) regressions to fixed-size subsets of the data using a weighting function based on the distance of a data point from x_i, the location where we wish to fit $f(x)$. This fit is performed iteratively, using a *biweight* robust weighting function. The robust iterations tend to downweight large residuals, so that the final fit at x_i is more representative of the mass of the data near x_i.

The *loess* smoother indicates that the variables in Figure 7.6 are fairly linearly related to each other. We could explore a linear model to test this possibility. Local smoothers can help us in the choice of models and save us from erroneous conclusions, as the next example will show.

ELEMENT: *line*(*position*(*smooth.loess*(female*birth)))
ELEMENT: *point*(*position*(female*birth))

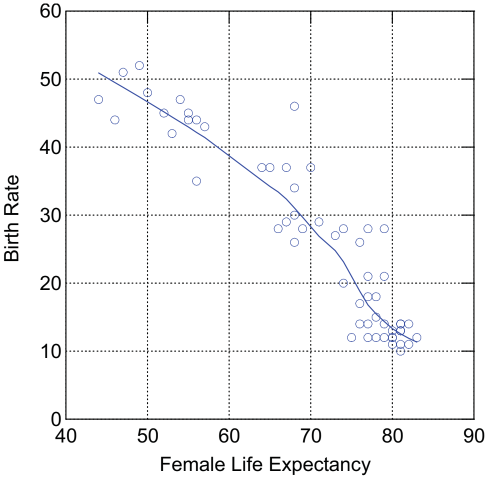


Figure 7.6 *Loess smooth on scatterplot*

7.2.2.3 A Conditional Mode Kernel Smoother

Nonparametric smoothers can be especially useful in detecting nonlinearities before deciding to fit linear models. Figure 7.7 shows an example of an analysis whose conclusions might have been substantially different had non-parametric smoothers been used.

The data underlying the figure were digitized from a graphic in Gonnelli *et al.* (1996). A linear regression (the green line) was computed by the authors separately for males and females to support their conclusion that bone alkaline phosphatase increased linearly for both groups with age. Figure 7.7 shows the plot for females. The curvilinear smoother (red curves) is a modal kernel regression procedure (Scott, 1992) that fits an estimate of the conditional mode at points on the domain. The advantage of a modal smoother is that discontinuities are revealed when they exist in the data. When the data support it, we get two or more smoothers instead of one. Since the smoother can accommodate multimodality, it can even fit multiple estimates at given values in the domain. In other words, we can get two or more smoothers at different altitudes in the same scatterplot. Conventional regression smoothers do not have this property. The modal smoother shows that there is evidence of a discontinuity in BAP levels for females at menopause. There is little evidence of a trend within the separate age levels, however. Kernel smoothers are known for regressing (going flat) at their extremes. For these data, however, other smoothers support the same conclusion (see Figure 15.25).

```
ELEMENT: line(position(smooth.mode.epanechnikov(age*bone)),
         color(color.red))
ELEMENT: line(position(smooth.linear(age*bone)), color(color.green))
ELEMENT: point(position(age*bone))
```

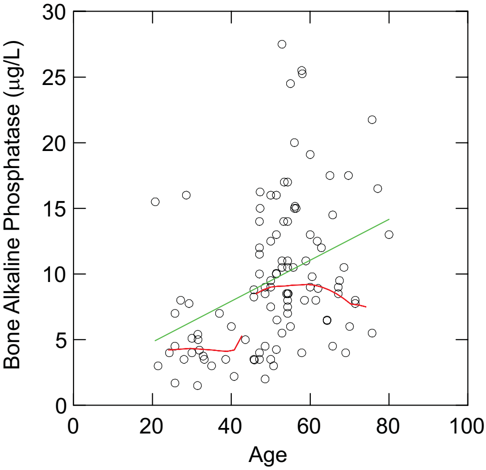


Figure 7.7 *Linear vs. modal smooth*

7.2.2.4 Joint Smoothing

Conditional smoothers minimize the sum of a function of the *vertical* distances (measured along y) between data values and fitted curve for all values measured on **x**. Joint smoothers generally minimize the sum of a function of the *shortest* distances between data values and fitted curve for all values measured on **x**. The joint method can be useful when there is no reason to distinguish a dependent variable from an independent variable. Hastie and Stuetzle (1989) generalize this idea to **principal curves**, which are curvilinear segments that pass through a set of points in a similar manner to the **medial axis** or **skeleton** of set of points (Preparata and Shamos, 1985).

Figure 7.8 shows a simple example of a joint linear fit to Quantitative and Verbal Graduate Record Examination scores of students in a psychology department. The conditional regression line (green) has a shallower slope than the joint regression line. The loss function for conditional linear regression is

$$l_c = \sum_i (y_i - (b_0 + b_1 x_{1_i}))^2$$

and the loss for the orthogonal regression is

$$l_o = \sum_i \frac{(x_{1_i} - (b_1 + b_2 x_{2_i}))^2}{(1 + b_2^2)}$$

ELEMENT: *line*(*position*(*smooth.linear.conditional*(greq*grev)),
color(*color.green*))
ELEMENT: *line*(*position*(*smooth.linear.joint*(greq*grev)), *color*(*color.red*))
ELEMENT: *point*(*position*(greq*grev))

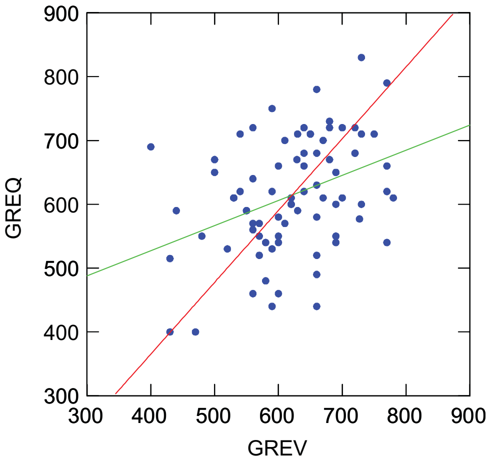


Figure 7.8 *Conditional (green) and joint (red) linear smoothers*

7.2.2.5 2D Step Interpolation

Step interpolation fits points with a step function. Figure 7.9 shows a statistical plot called the Kaplan–Meier survival function. This plot displays the probability of survival from some onset time until a given time, estimated from a sample of cancer patients. The same graphic can be used on other populations subject to finite survival times, such as light bulbs, rumors, and marriages. Step interpolation is also useful for displaying empirical cumulative distribution functions (see Figure 6.2).

ELEMENT: *line*(*position*(*smooth.step*(days*survive)))

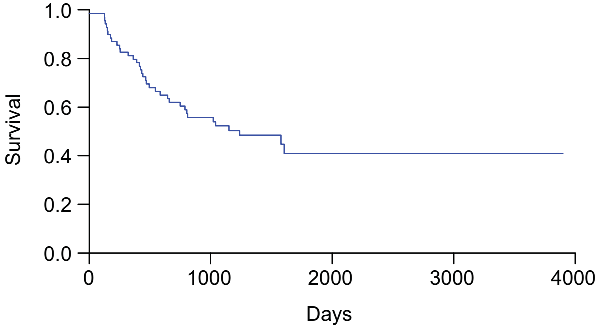


Figure 7.9 *Kaplan–Meier survival curve*

7.2.2.6 3D Step Interpolation

A 3D step smoother resembles a Voronoi tessellation (see Section 7.2.5.3). Figure 7.10 shows a stepped surface fitted to the car data used in Figure 8.2.

COORD: *position*(*rect*(*dim*(1, 2, 3)))
ELEMENT: *surface*(*position*(*smooth.step*(weight*hp*quarter)),
color(*color.green*))
ELEMENT: *point*(*position*(weight*hp*quarter), *color*(*color.blue*))

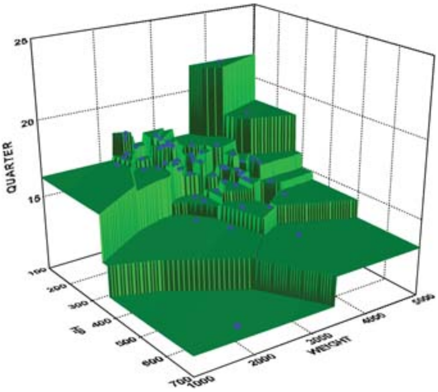


Figure 7.10 *3D stepped surface*

7.2.2.7 3D Inverse-Distance Smoothing

Figure 7.11 shows a 3D surface smoother on the difference between US summer and winter temperatures. The specification includes a stereographic projected frame for the first two dimensions. The map is drawn within this projection via the *polygon* graphic whose *position*(longitude*latitude) function places it on the bottom facet of the 3D frame. The temperature variation for each state is represented in the range by a *contour* graphic. We assume the variables lon, lat, summer, and winter are variables in an associated data file.

The smoother for the *contour* graph is computed by the *smooth.mean.cauchy* function. The contours are colored with *color.hue*. Like the map itself, the contours are plotted on the bottom facet because only two variables are included in its *position*() aesthetic. Finally, the third graphic adds the surface representation of the smoother in 3D. This is accomplished by including all three variables of the specification implicitly in the *position*() aesthetic. The map reveals that midwesterners are hardy folk. They are forced to tolerate fluctuations in the weather. And because of stiff airline fares (see Figure 9.39), they have difficulty fleeing to places where the weather is fair.

```
DATA: longitude, latitude = map(source("US States"))
TRANS: sw = diff(summer, winter)
COORD: rect(dim(1, 2, 3), project.stereo(dim(1, 2)))
ELEMENT: polygon(position(longitude*latitude), color.hue(sw))
ELEMENT: contour(position(smooth.mean.cauchy(lon*lat*sw)), color.hue())
ELEMENT: surface(position(smooth.mean.cauchy(lon*lat*sw)))
```

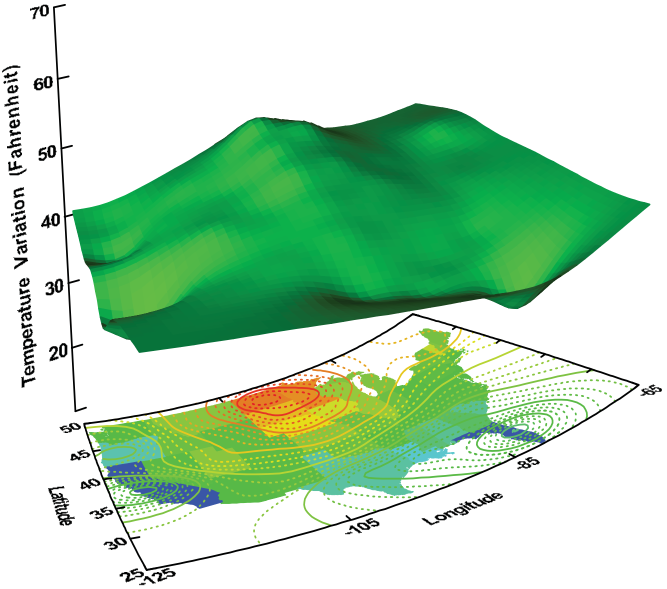


Figure 7.11 *Smooth of temperature variation in continental US*

7.2.3 Interval

Bar graphs represent interval-valued functions, or relations in which the range can have two values in the function set for a given tuple in the domain. Ordinarily, one end of an interval is fixed at the value of zero to make a bar chart. The following examples illustrate applications in which both ends are used.

7.2.3.1 Error bars

Sometimes we need to represent a range, standard deviation, confidence interval, or some other spread measure. The most common example is the ordinary error bar used in scientific graphics. Figure 7.12 shows an example for the birth rate data within different types of government.

ELEMENT: *interval*(*position*(*region.spread.se*(gov*birth)), *shape*(*shape.tick*))
ELEMENT: *point*(*position*(*statistic.mean*(gov*birth)))

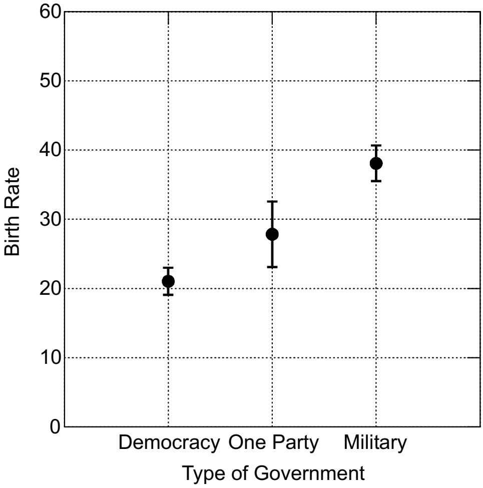


Figure 7.12 *Error bars*

The error bars in this figure are based on one standard-error in both directions from the mean estimates represented by the dots in the graphic. We have used the *region.spread.se*() function to compute an interval containing the mean, plus or minus one standard error. The shape of the bars is arbitrary. The one chosen for the figure, *shape*(*shape.tick*), is the most popular, but solid bars and other graphics for representing an interval would do as well. We could use several different graphics to represent several different spread statistics at the same time (standard deviation and range, for example).

7.2.3.2 Range Bars

The high-low-close graphic used for financial trading series is a combination of an *interval* and a *point* graphic. Figure 7.13 shows a high-low-close graphic for the SPSS stock series. The shape chosen for the graphic emphasizes the closing price with a horizontal tick. Other graphic shapes are possible.

The *region.spread.range*() graphing function creates the vertical lines by setting shape with the *shape*(*shape.line*) aesthetic attribute function (see Figure 6.5 for range bars on this stock series using regular bars). The *range*() statistical method returns two values that define the ends of each bar. The closing price is positioned by the *summary.median*() graphing function. Because there are only three *y* (high+low+close) values for each *x* (date) value, the *median*() method always returns the close value.

The time scale is noteworthy. See Chapter 6 for details. Notice that the time scale spaces dates unevenly according to the lengths of different months. The interval between February and March is smaller than the other intervals because there are fewer days in February. Weekends properly appear as blank spaces because of this time scale. This scaling of time variables is especially critical for graphics based on interrupted or unevenly spaced time series data, such as financial series or clinical trials.

```
SCALE: time(dim(1))
GUIDE: axis(dim(1), format("mmm"))
ELEMENT: interval(position(region.spread.range(date*(high+low+close))),
         shape(shape.line))
ELEMENT: point(position(statistic.median(date*(high+low+close))),
         shape(shape.hyphen))
```

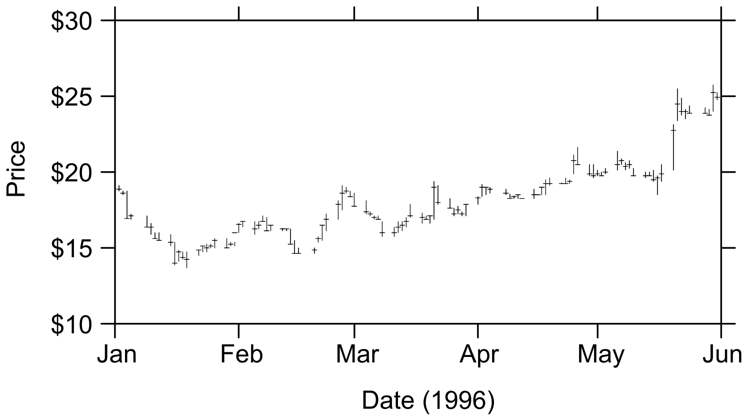


Figure 7.13 *High-low-close plot*

7.2.4 Bins

Densities measure the relative concentration of a sample at different values of a variable. These statistical measures range from ordinary histograms to dot plots to kernel density estimates.

7.2.4.1 Ordinary Histogram

The *interval* graph with a binning statistical function creates histograms. The ordinary histogram is constructed by binning data on a uniform grid. Although this is probably the most widely used statistical graphic, it is one of the more difficult ones to compute. Several problems arise, including choosing the number of bins (bars) and deciding where to place the cutpoints between bars.

The choice of bin width h (or its dual, k, the number of bins) has been a topic of research in the statistical community for more than 70 years. Sturges (1926) proposed that k be proportional to $log_2(n)$, where n is the sample size. He reasoned this from the approximation of the binomial by the normal distribution. For the binomial,

$$n = \sum_{m=0}^{k} \binom{k}{m} = 2^k \text{ , so}$$

$$k = log_2(n)$$

Doane (1976) showed that more bins are needed, particularly for skewed data. He included a measure of skewness into the calculation of k. If we can't read the data to calculate skewness before doing the histogram, the approximation

$$k = 3 + log_{10}(n)log_2(n)$$

increases k enough to cover most of the examples Doane describes.

Working with h instead of k frames the problem in terms of density estimation. Freedman and Diaconis (1981) computed bin width from asymptotic theoretical results. For a normal distribution, Scott (1979) derived a bin width

$$h = 3.5sn^{-1/3}$$

where s is the sample standard deviation.

An additional problem arises when data are granular. The above estimates should not be used when data consist of only a few distinct values. If our data consist of the integers between 0 and 10, for example, we should pick 9 bins regardless of the sample size. If the data consist only of the integers 3 and 7, we should pick the 5 bins {3, 4, 5, 6, 7}. A similar argument applies to real numbers.

It is worth a preliminary pass through the data to determine if granularity exists. A simple test is to sort the data and take differences between adjacent values. If these differences are all an integer multiple of the smallest nonzero difference, then granularity exists. In this case the bin width should be the smallest nonzero difference or the width calculated with Scott's formula, whichever is larger.

Deciding where to place the cutpoints between bars is not simple either. Generally, we want one cutpoint to be at zero, but this should not be a hard rule. Scott (1992) showed that the choice of cutpoints between bars can affect substantially the shape of the histogram. To counteract this tendency, he constructed a density estimator called the Averaged Shifted Histogram (ASH) by shifting the cutpoints while maintaining bin width and then averaging all the bins. There is a trade-off between choosing cutpoints at nice values (see Chapter 6) and choosing them to create a relatively smooth histogram.

Statistics textbooks usually illustrate histograms with bell-shaped datasets. Let's turn things upside down. Figure 7.14 shows traffic fatalities by age for the US in 2001. The statistics, unintentional motor vehicle occupant death rates per 100,000 population, are from the US Centers for Disease Control and Prevention (CDC). We have superimposed a red theoretical curve based on the arcsine distribution.

Notice that we used a weighting variable with the *summary.sum*() statistical function; we did this to illustrate how to construct histograms on weighted variables. If a case-weighting variable is taken from a field in the data source, then the case-by-case values will be used instead of the default value of 1. In most cases, the weighting variable appears nowhere else in the specification; it is always used in statistical calculations, so there is usually no point in adding a redundant reference in a model. With histograms, however, the vertical axis consists of counts (or another statistic such as proportions). Therefore, we use age*weight to determine the 2D frame for the graph.

The arcsine is a special case of the beta distribution (see Section 11.3.2.4) and is useful for representing trade-offs in bounded random variables. Its density function is proportional to $1/\sqrt{x(1-x)}$. Notice the trade-off in this formula. Within the [0, 1] interval, density increases as x deviates from a half. Our simple model specifies that one's chance of a traffic fatality depends on how far one is from 50 years old (we picked a round number). We cannot draw firm conclusions about the relationship of age to driver performance based on these data, however. First of all, these data include both drivers and passengers. Second, they are confounded with alcohol use. Third, the data are not standardized by miles driven. The risk of having an accident is mostly due to the time spent traveling in a car.

DATA: weight = *col*(*source*("traffic.txt"), *name*("percent"), *weight*())
ELEMENT: *interval*(*position*(*summary.sum*(*bin.rect*(age*weight), *dim*(1))))
ELEMENT: *line*(*position*(age*arcsin))

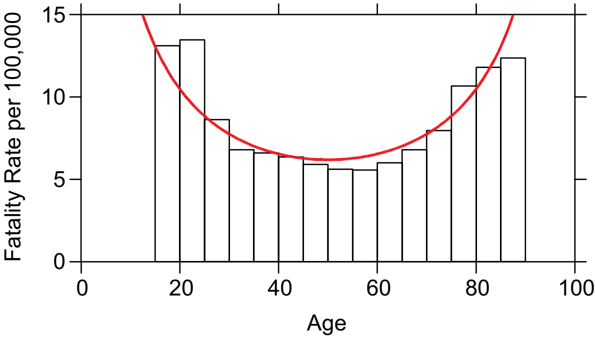


Figure 7.14 *Histogram of traffic fatality rates by age*

7.2.4.2 Gap Histogram

Figure 7.15 shows an ordinary histogram of birth rate in the left panel and a gap histogram in the right. The gap binning is a partial Voronoi tessellation of the data in one dimension (see Section 7.2.5.3 for a 2D example). The edge of each bin represents the Voronoi boundary midway between two points. Not all boundaries are computed, however; some bins are left to contain more than one data point. The area of a bar in either graphic is determined by the number of cases in the bar times its width. The gap histogram is more useful for identifying gaps in the data than for representing the density itself.

ELEMENT: *interval*(*position*(*summary.count*(*bin.rect*(birth))))

ELEMENT: *interval*(*position*(*summary.count*(*bin.voronoi*(birth))))

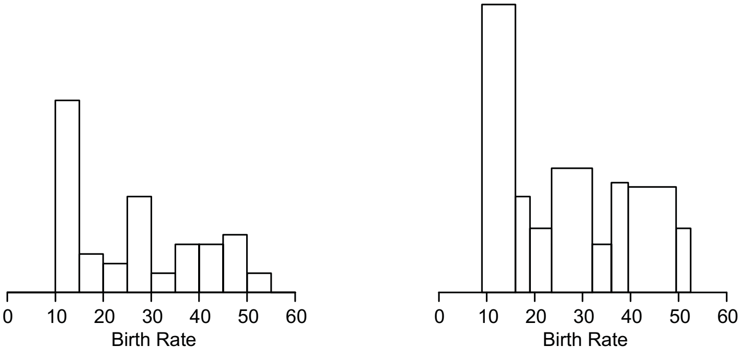


Figure 7.15 *Histogram and gap histogram of birth rates*

Figure 7.16 shows in more detail how a gap histogram works. We have superimposed a dot plot on the gap histogram to illustrate where the cuts are made. Not every adjacent pair of dots is separated by bins. The algorithm stops when it finds local clumps that are better left together than split. Notice, however, how the 1D Voronoi cuts are midway between adjacent towers of dots.

The vertical scales of the two graphics do not coincide. This is because the dot plot has no vertical scale; it is a tally. If we wished to make a dot plot with a scale, we would have to construct a count variable to control the heights of the dot-piles.

ELEMENT: *interval*(*position*(*summary.count*(*bin.voronoi*(birth))))
ELEMENT: *point.dodge*(*position*(*bin.dot*(birth))))

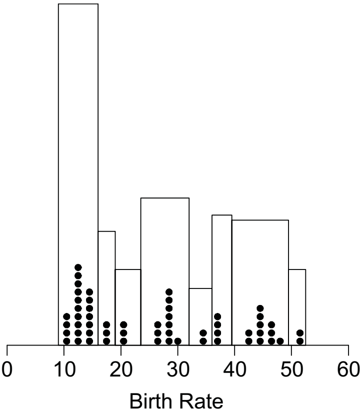


Figure 7.16 *Gap histogram with superimposed dot plot*

7.2.4.3 Dot Plots

Asymmetrical dot plots (which are like dot histograms) and their close cousins, symmetrical dot plots appear in a variety of statistical graphics packages. Most of these displays are computed incorrectly. These packages simply bin the data as if they were doing a regular histogram and then use dots instead of bars to represent the count in each bar. Hand-drawn dot plots, used for well over a century in medicine, economics, and other fields, place the dots where the data values actually occur so that they avoid misleadingly granularizing the data. The stacking that we see in dot plots should happen only when two or more scaled data values differ by less than some proportion of the diameter of a dot. One may think of the dots as a set of poker chips that must be arranged as closely as possible to their coordinate locations without overlapping. The larger the chips, the more stacking there is at common levels. The size of the dot is, in certain respects, a smoothing parameter. Wilkinson (1999) discusses the details of the problem.

Figure 7.17 illustrates a dot plot on the traffic accident data. We have made a weighting variable and multiplied it by 10 so that each dot represents a tenth of a percent (the resolution of the original data source). We did not need to do this with the histogram example because the height of the intervals is measured on a continuous dimension. With dot plots, we don't want to use fractional dots.

The plot tells us the data are granular in the data source, something we could not ascertain with the histogram. There is an important lesson here. Statistics texts and statistical packages that recommend the histogram as the graphical starting point for a data analysis are giving bad advice. The same goes for kernel density estimates. These are appropriate *second* stages for graphical data analysis. The best starting point for getting a sense of the distribution of a variable is a tally, stem-and-leaf, or a dot plot. A dot plot is a special case of a tally (perhaps best thought of as a *delta-neighborhood* tally). Once we see that the data are not granular, we may move on to a histogram or kernel density, which smooths the data more than a dot plot.

The SYSTAT histogram algorithm does take granularity into account (by picking the number of bars to coincide with the number of distinct values even for massive datasets), but this does not save us from investigating the data first with a tally. It simply addresses the problem of misapplying standard binning size estimates to granular data when constructing histograms. Because bars in a histogram insist on staying glued together, we do not get to see the granularity. If we know in advance that our data are granular, of course, then there is no problem.

Incidentally, some might argue that making dot plots a default first step for analysis is not a good idea for massive datasets containing numerous distinct values. Wilkinson (1999) addresses that point; it is not a problem.

DATA: wt = *col*(*source*("traffic.txt"), *name*("percent"), *weight*())
TRANS: wt = *prod*(y, 10)
ELEMENT: *point.dodge*(*position*(*bin.dot*(age)))

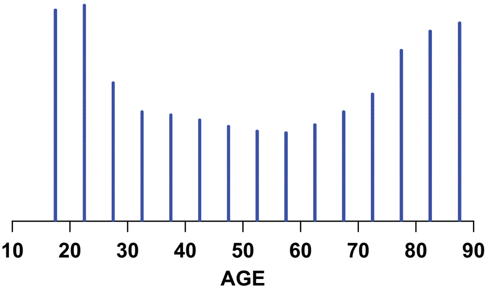


Figure 7.17 *Dot plot on binned data*

7.2.4.4 Stem-and-leaf Diagrams

Tukey (1977) discusses a variety of tallies, from the simple crossed-out groups of five marks to the **stem-and-leaf plot**. This latter display adds to the simple tally a set of significant digits to convey essential numerical information. The most significant digit or digits are chosen to be the stem on the left of the tally. The next digit to the right of the stem in each data value is chosen for the “leaf” on the right. This digit is not rounded. The leaves are stacked in order to the right so that the entire tally resembles a histogram. Figure 7.18 shows an example for the birth rate data. The plot is transposed to take the form of Tukey’s original. In other words, a stem-and-leaf diagram is a transposed, stacked dot plot in which the dots are represented by numerals. The binning method is a bit different from that used by dot plots, however. Can you see why?

```
ELEMENT: point.dodge(position(bin.stem(birth)),
         shape(summary.leaf(birth)))
COORD: transpose(dim(1, 2))
```

```
1  01112222222333444444
1  5788
2  011
2  6678888899
3  04
3  57777
4  23444
4  556778
5  12
```

Figure 7.18 *Stem-and-leaf diagram*

7.2.4.5 Kernel Densities

There are many methods for estimating probability densities from sample data (Silverman, 1986; Scott, 1992). Figure 7.19 shows two popular density estimates superimposed on histograms of brainweight on a log scale. The *normal* method uses a parametric Gaussian distribution to produce the curve from sample estimates of its two parameters. The *kernel* method uses nonparametric kernel density estimation with an Epanechnikov kernel as the default method (Silverman, 1986). Both methods are types of smoothers, so we calculate them in the *smooth*() class. Many of the same basic windowing methods are used for both kernel smoothers and kernel density estimates.

The vertical axis is not shown, but this dimension consists of the density represented by the smooths. The densities match the height of the histogram bars because we used a *proportion*() summary function for the vertical measure of the histogram bars. This requires an area calculation, for both the histograms and the densities, that sums to 1.

```
SCALE: log(dim(1), base(10))
ELEMENT: interval(position(summary.proportion(bin.rect(brainweight))),
           color(color.blue))
ELEMENT: line(position(smooth.density.normal(brainweight)),
           color(color.red))

SCALE: log(dim(1), base(10))
ELEMENT: interval(position(summary.proportion(bin.rect(brainweight))),
           color(color.blue))
ELEMENT: line(position(smooth.density.kernel(brainweight)),
           color(color.red))
```

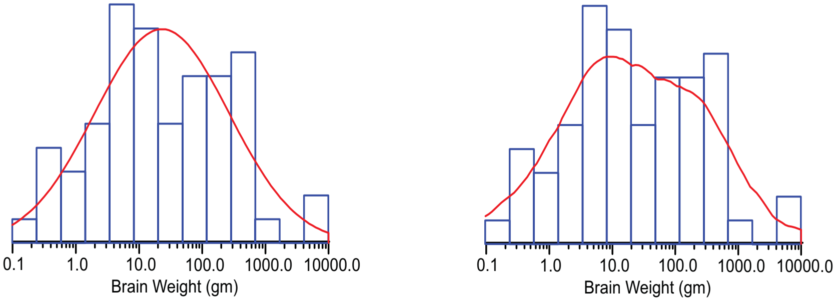


Figure 7.19 *Normal and kernel densities superimposed on histogram*

7.2.4.6 Bivariate Densities

Scott (1992) and Silverman (1986) discuss 2D and higher-dimensional kernel density estimates. Figure 7.20 shows joint bivariate densities for the sleep data. The left panel shows contours for a normal bivariate density estimate, and the right panel shows contours for a kernel density estimate. The normal density is computed from estimates of the parameters of the bivariate normal distribution and the kernel density is computed from an Epanechnikov kernel. This kernel looks like a quadratic function (parabolic solid) in two dimensions that is moved around the plane of the points and is used to compute a weighted sum of the counts of the observations within the bounds of the kernel.

As with the 2D kernel density, the 3D kernel density requires scaling the density to sum to 1. We do this by numerically integrating the density (by Simpson's method) and then choosing level curves that correspond to nice numbers. We start at the mode of the density and move down, so that after rescaling, the numbers on the level curves (assuming we display them) correspond to empirical probabilities. If there are two or more modes, we insure that all of them are included in the scaling calculations. A horizontal scan-plane algorithm helps us with this problem. The net result of all this work is that 95 percent (or another selected value) normals and kernels can be superimposed in the same plot.

```
SCALE: log(dim(1), base(10))
SCALE: log(dim(2), base(10))
ELEMENT: point(position(bodyweight*brainweight))
ELEMENT: contour(position(smooth.density.normal.joint(bodyweight*brainweight)),
         color.hue())

SCALE: position(log(dim(1), base(10)))
SCALE: position(log(dim(2), base(10)))
ELEMENT: point(position(bodyweight*brainweight))
ELEMENT: contour(position(smooth.density.kernel.epanechnikov.joint(
         bodyweight*brainweight)), color.hue())
```

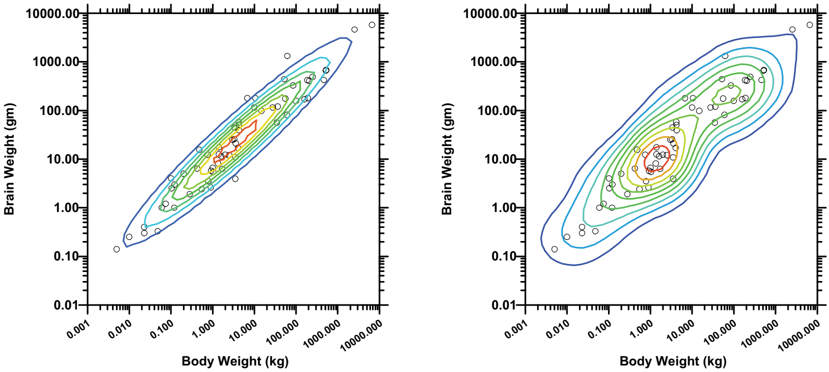


Figure 7.20 *Normal and kernel bivariate densities*

7.2.5 Polygon

We have seen how functions can be used to create different tilings using the same *polygon* element. This section shows a variety of examples.

7.2.5.1 Binning

Figure 7.21 shows a simple scatterplot. The data for the graphic were adapted from Daly *et al.* (1994), who developed an analytic method called Parameter-elevation Regressions on Independent Slopes Model (PRISM) that uses point data and a digital elevation model to generate 2.5 minute square gridded estimates of monthly and annual US climate parameters. Carr *et al.* (1999) used those gridded summaries for the time period 1961–1990 to develop graphics characterizing the spatial variation of climatic parameters within ecoregions. They associated each grid cell with an Omernik level II ecoregion (Omernik 1987, 1995) using a point-in-polygon matching procedure. The horizontal axis of Figure 7.21 represents the average yearly precipitation in millimeters over the three decades. The vertical axis represents average annual growing degree days, a measure of the number of degrees in daily average temperature above 50 degrees summed over all days with a daily average temperature above 50. There are 78,766 data points in Figure 7.21.

ELEMENT: *point*(*position*(rainfall*degdays))

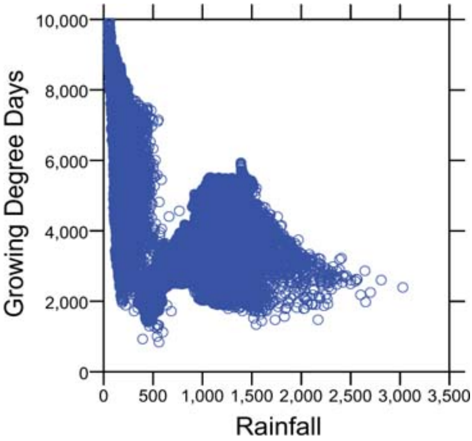


Figure 7.21 *Simple scatterplot*

Scatterplots on large datasets tend to look like inkblots, even when we try to use non-occluding symbols such as hollow circles. Binning helps to reveal more detail. Figure 7.22 shows a 30 by 30 rectangular grid of the same data.

ELEMENT: *polygon*(*position*(*bin.rect*(rainfall*degdays)), *color.hue*(*summary.count*()))

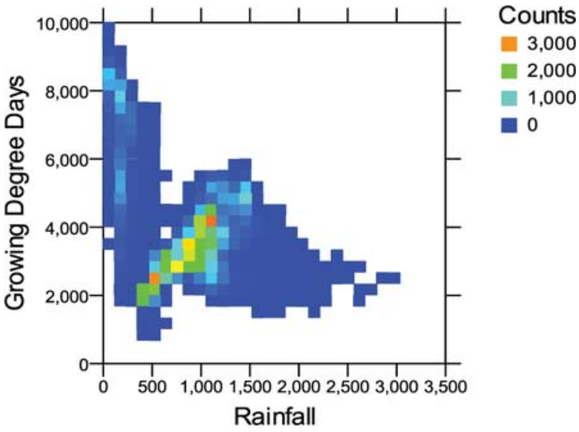


Figure 7.22 *Rectangular binning with color representing counts*

Carr suggested a modification that uses binning but results in a plot that looks more like a scatterplot. We bin as usual but use a symbol to represent the centroid of the points in each bin. The symbols are sized according to the frequency in each bin. Figure 7.23 shows the result for the Omernik data.

ELEMENT: *point*(*position*(*bin.rect.centroid*(rainfall*degdays)), *size*(*summary.count*()))

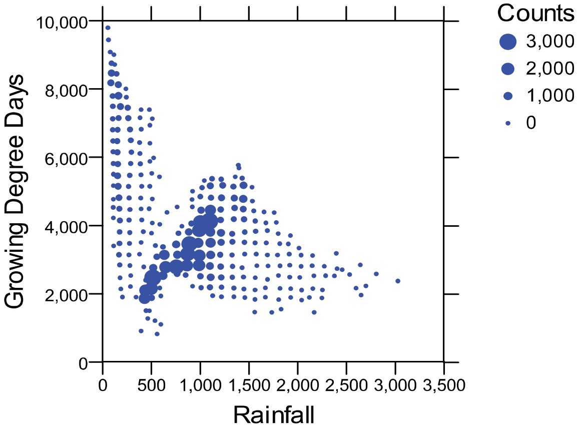


Figure 7.23 *Rectangular binning with symbols positioned at bin centroids and size of symbols representing counts*

Rectangular bins lead the eye to align bin centers and to see regularity where there is none. Such patterning is evident in Figure 7.23 despite moving the points to the bin centroids. Carr *et al*. (1987) devised **hexagon binning** to mitigate this visual artifact. The hexagon tiling staggers the locations of alternate rows of bins. Figure 7.24 shows an example of hexagon binning on the Omernik data. We could also use *bin.hex.centroid* with *point* as in Figure 7.23.

ELEMENT: *polygon*(*position*(*bin.hex*(rainfall*degdays)), *color.hue*(*summary.count*()))

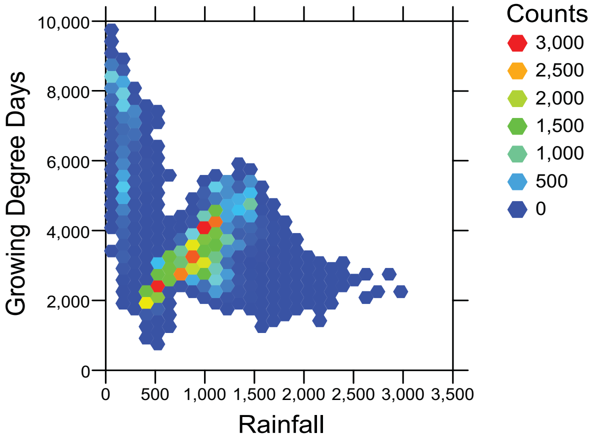


Figure 7.24 *Hexagon binning*

7.2.5.2 Heatmaps

A **heatmap** is a tiled 2D graphic with the tiles colored by a third variable. Figure 7.25 shows a microarray plot (Alon *et al.*, 1999). We have assumed that the binning in this case produces only one bin per value (although we haven't shown a parameter to control the binning in the example), so the data are full-resolution. See Section 16.5 for further discussion of matrix permutation as an analytic procedure.

DATA: x = *reshape.rect*(x(1..62), "colname")
DATA: y = *reshape.rect*(x(1..62), "rowname")
DATA: d = *reshape.rect*(x(1..62), "value")
ELEMENT: *polygon*(*position*(*bin.rect*(x*y)), *color.hue*(d))

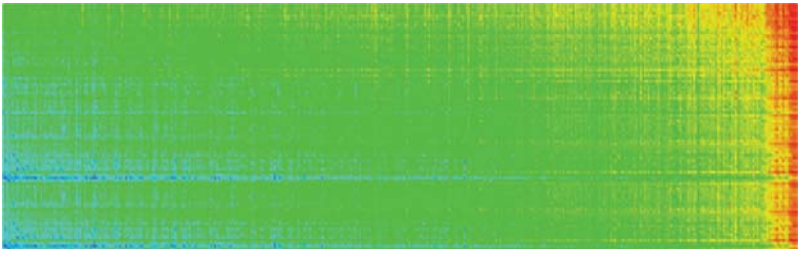


Figure 7.25 *Microarray plot*

7.2.5.3 Voronoi Tessellations

The Voronoi tessellation is one of the most prevalent tilings to appear in scientific graphics. Each pair of points is separated by a boundary based on the perpendicular bisector of the line segment joining both points. Preparata and Shamos (1985) survey algorithms for computing this tiling. Stoyan *et al.* (1987), Cressie (1991), and Okabe *et al.* (1992) cover statistical and probability measures related to the Voronoi tessellation.

Figure 7.26 shows a graphic of the spatial location of fiddler crab holes in an 80 centimeter square section of the Pamet river tidal marsh in Truro, Massachusetts. We collected these data, in bare feet, at substantial personal risk. The right panel contains a sample of uniform random points distributed in the same square. Both panels contain a Voronoi tessellation. Notice the irregular structure of the polygons on the right. The contrast in tiling between the two figures suggests the territorial constraints of the real holes *vs.* the random grouping of the artificial.

```
ELEMENT: polygon(position(bin.voronoi(crabx*craby)))
ELEMENT: point(position(crabx*craby))

DATA: uranx=rand.uniform(23)
DATA: urany=rand.uniform(23)
ELEMENT: polygon(position(bin.voronoi(uranx*urany)))
ELEMENT: point(position(uranx*urany))
```

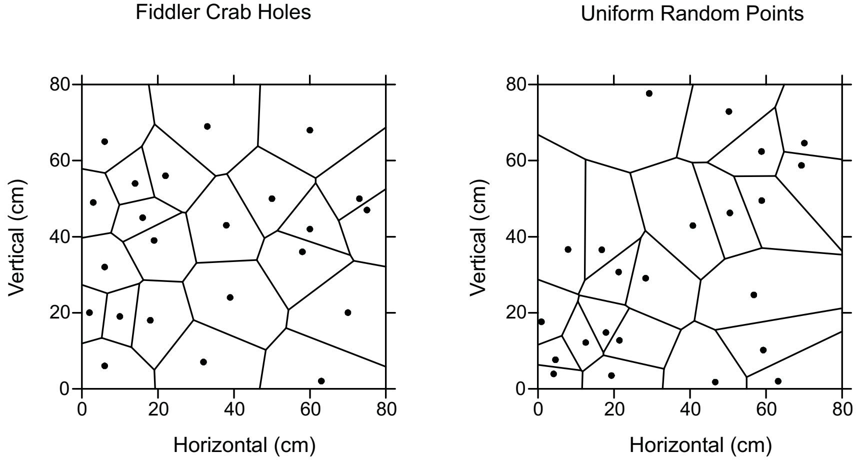


Figure 7.26 *Voronoi tessellation of fiddler crab holes*

7.2.6 Path

The *path*() graphing function computes a path according to the order of a set of points in a list. The order may come from the original dataset or we can sort the data. The following method computes an ordering using an algorithm.

7.2.6.1 Traveling Salesman Problem

Figure 7.27 shows a path that rejoins itself to make a complete circuit. The example here comprises a short path through the continental US that covers every state just once and returns to the beginning. This graphic is an approximate solution to the **traveling salesman problem** (Preparata and Shamos, 1985). Several caveats are in order, however. First, the path was computed in Euclidean 2D space after transforming with the stereographic projection. This method does not produce the shortest path on the surface of the globe. Second, the locus of the point in each state is arbitrary and was not chosen to make the path shorter. Finally, the solution itself was computed by **simulated annealing** (Press *et al.*, 1986), which cannot guarantee the shortest path. The advantage of this sub-optimal algorithm, however, is that Figure 7.27 can be computed in a few seconds on a desktop computer. The *link.tsp*() function computes the path and ensures that it returns to its start in a closed loop or circuit.

DATA: longitude*latitude = *map*(*source*("US states"))
COORD: *project.stereo*()
ELEMENT: *polygon*(*position*(longitude*latitude))
ELEMENT: *path*(*position*(*link.tsp*(lon*lat)))

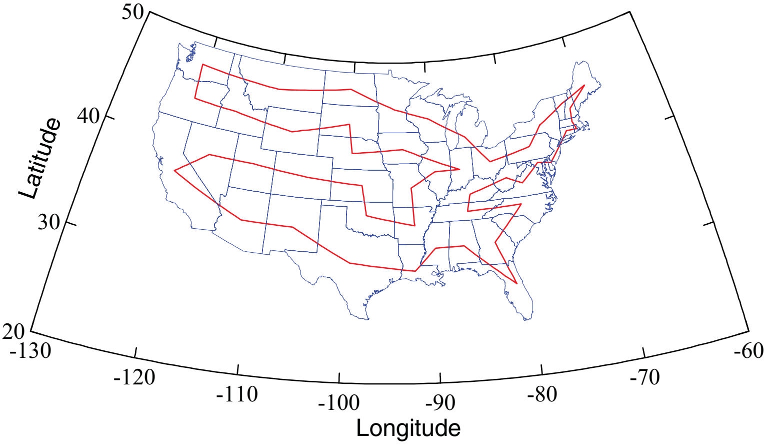


Figure 7.27 *Short bicycle ride through US*

7.2.7 Edge

The *edge*() graphing function computes a vertex-edge graph element. The statistical *link*() functions offer several different methods for computing the links between vertices.

7.2.7.1 Minimum Spanning Tree

A minimum spanning tree (MST) connects points represented in a space using line segments that have minimum total length and that join all points without creating any circuits (cycles). The result is that any two nodes in an MST are connected by exactly one path. This tree has the shortest total length of all possible spanning trees connecting the points in the plotted space. Preparata and Shamos (1985) survey algorithms for computing this tree efficiently and Hartigan (1975a) discusses its application to cluster analysis. Deleting the longest link in a minimum spanning tree results in two clusters whose total edge length is minimum among all possible two-cluster trees. Recursively deleting links follows the same computational steps (in reverse order) as the widely used **single linkage**, or **nearest neighbor** cluster analysis algorithm. The algorithm is conventionally applied to a Euclidean minimum spanning tree, but it can be adapted to other coordinates.

Figure 7.28 shows a Euclidean minimum spanning tree on the fiddler crab data. For our fiddler crabs, the solution in Figure 7.28 shows the smallest amount of wire needed to install telephones in their holes so they can communicate with each other without clicking their claws.

ELEMENT: *point*(*position*(crabx*craby))
ELEMENT: *edge*(*position*(*link.mst*(crabx*craby)))

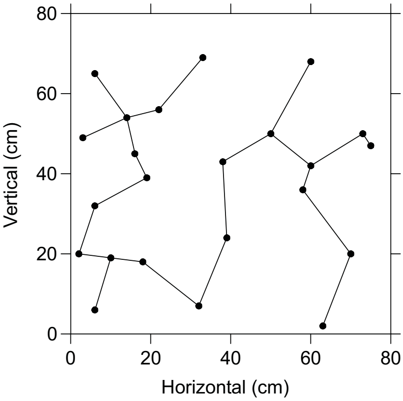


Figure 7.28 *Minimum spanning tree*

7.2.7.2 Hull

Figure 7.29 shows an example of a convex hull. As we shall see in Section 7.2.7.3, the convex hull is computable from the outermost edges of the Delaunay triangulation. The peeled hull (see Figure 8.13) is not a subgraph of the same triangulation, however. It makes more sense to compute hulls with routines tailored to the problem (Preparata and Shamos, 1985).

Our fiddler crabs have installed a convex hull to establish a gated retirement community. We have used an *edge* element to draw the hull, but a *polygon* element would be a reasonable choice as well.

ELEMENT: *point*(*position*(crabx*craby))
ELEMENT: *edge*(*position*(*link.hull*(crabx*craby)))

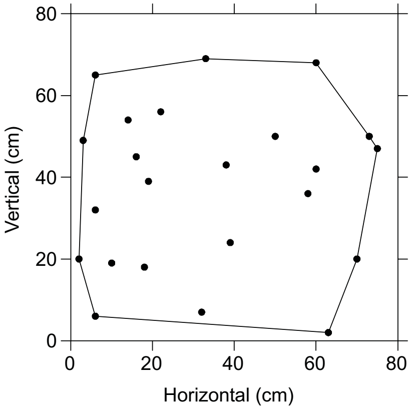


Figure 7.29 *Convex hull around crab holes*

7.2.7.3 Triangulation

A triangulation joins points with segments such that all the bounded regions are triangles. Figure 7.30 shows a Delaunay triangulation on the crab data.

ELEMENT: *point*(*position*(crabx*craby))
ELEMENT: *edge*(*position*(*link.delaunay*(crabx*craby)))

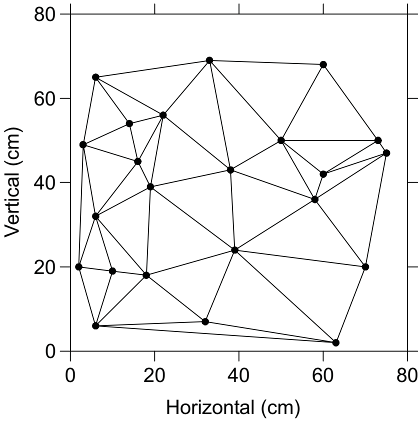


Figure 7.30 *Delaunay triangulation of crab holes*

Figure 7.31 superimposes the last three figures plus the Voronoi diagram.

ELEMENT: *point*(*position*(crabx*craby))
ELEMENT: *edge*(*position*(*bin.voronoi*(crabx*craby)), *color*(*color.red*))
ELEMENT: *edge*(*position*(*link.mst*(crabx*craby)), *color*(*color.blue*))
ELEMENT: *edge*(*position*(*link.hull*(crabx*craby)), *color*(*color.violet*))
ELEMENT: *edge*(*position*(*link.delaunay*(crabx*craby)), *color*(*color.green*))

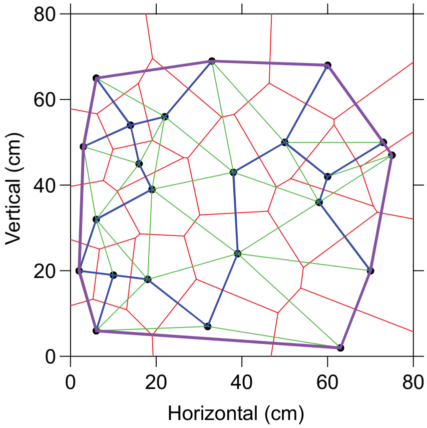


***Figure 7.31** Convex hull (purple), Voronoi tessellation (red), Delaunay triangulation (green), and minimum spanning tree (blue)*

The Delaunay triangulation is a dual of the Voronoi tessellation (Preparata and Shamos, 1985). The computation of one implies the computation of the other. The Voronoi edges are perpendicular to the sides of the Delaunay traingles. Each Delaunay segment bisects a Voronoi edge. The convex hull is the outermost collection of line segments from the Delaunay triangulation. Many algorithms for computing the Voronoi–Delaunay problem yield the convex hull as a by-product. Finally, the minimum spanning tree is a subgraph of the Delaunay triangulation.

7.2.7.4 Building Bridges with Join

The *link.join*() function joins sets of points from a blend. If two variables are blended, *link.join*() joins pairs, if three are blended, *link.join*() joins triples, and so on. A bridge is a *join* that links points from two or more sets of variables in the same graphic. It is designed to display repeated measures, migrations, flows, biplots, correspondence diagrams, and other multiple relations over time or space. Figure 7.32 illustrates an application of this graphic to **Procrustes rotation** (see Borg and Groenen, 1997).

```
ELEMENT: point(position(caragility*carsize+dogagility*dogsize),
         label(car+dog))
ELEMENT: edge(position(link.join(caragility*carsize+dogagility*dogsize)))
```

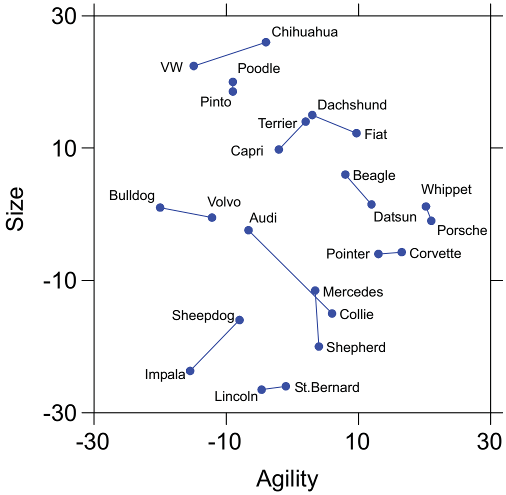


Figure 7.32 *Bridge plot*

Procrustes rotation matches two or more spatial configurations to each other using a loss function based on Euclidean distance or some other discrepancy measure between corresponding points. The data are from Wilkinson (1975). Cars and dogs were rated for similarity by dedicated (obsessed) car and dog club members. The author stepped around cars in front yards and stepped over dogs in living rooms in order to collect these data.

Each subject rated a set of ordinary objects known (and loved) intensely and another set known only superficially. (There were no subjects with enough time in a day to be fascinated with both.) On the basis of external measures, correspondences were inferred between pairs of cars and dogs. These links were used to establish the ordering of coordinates for the Procrustes rotation. Figure 7.32 shows the rotation of the results of two multidimensional scalings of the car and dog similarity ratings. The graphics offer a visual summary of the goodness of fit. The shorter the lines, the better the fit. The results of this analysis have absolutely no commercial potential.

7.2.7.5 Drawing Vectors with Join

Figure 7.33 illustrates how *join* can be used to plot vectors from an origin. The variables factor1 and factor2 are the first two columns of the matrix $\mathbf{V}$ in the singular value decomposition $\mathbf{X} = \mathbf{UDV}^T$. The matrix $\mathbf{X}$ consists of scaled binary data on grounds for divorce among 50 US states in 1971. The original source is Long (1971), suggested to us by Gerard E. Dallal, and analyzed in

Wilkinson *et al.* (1996). By constructing a new variable called zero, we join all the points to paired values at (0,0). In addition, we choose arrows to shape the vectors.

DATA: zero = *constant*(0)
ELEMENT: *edge*(*position*(*link.join*(zero*zero + factor1*factor2)),
shape(*shape.arrow*), *label*(ground))

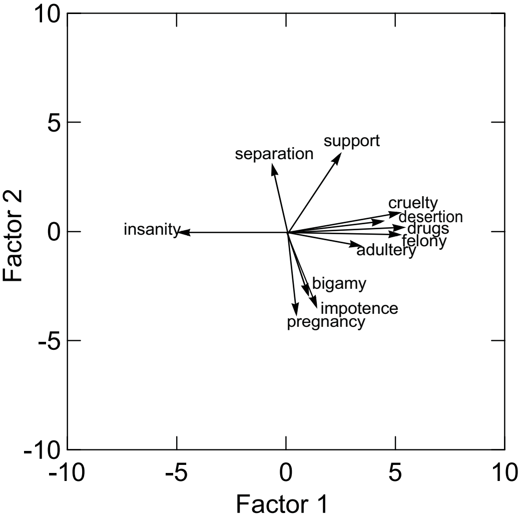


Figure 7.33 *Vector plot*

7.2.7.6 Drawing Vectors to a Line or Surface with Join

Figure 7.34 illustrates the use of *link.join*() for drawing vectors to a line using the countries data. This method can be used to draw vectors to a curve or surface as well. The data are based on the residuals from a linear regression of birth on female for our countries dataset. In this figure, vectors are drawn from residual points to points at a zero value.

We construct this column of zero values by employing the data function *constant*() in the expression zero = *constant*(0). Because the frame model contains female*(residual+zero), the *link.join*() function connects points in the tuple female*residual with points in female*zero, producing the vertical lines. We don't want two sets of *point* clouds, however. The illusory contour of the spikes is sufficient to demarcate the values at zero (Levine, 2000).

DATA: zero = *constant*(0)
ELEMENT: *edge*(*position*(*link.join*(female*(residual+zero)))
ELEMENT: *point*(*position*(female*residual))

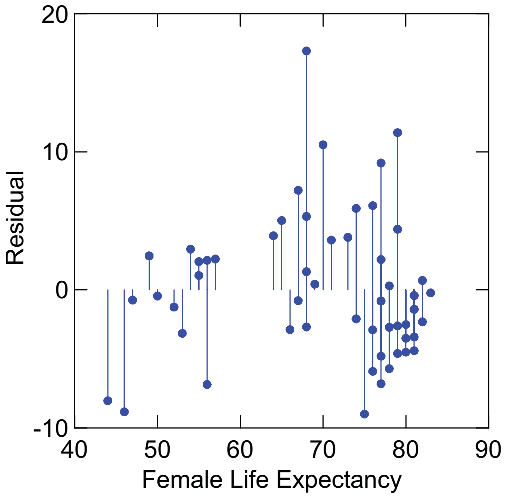


Figure 7.34 *Spikes for residuals*

7.2.7.7 Drawing Trees with Join

Among the most widespread uses of graphical trees in statistics and data analysis are displays of the results of hierarchical cluster analysis (Hartigan, 1975a) and recursive partitioning schemes (Breiman *et al.*, 1984). These structures also appear in the directed graphs used to represent trees in computer science algorithms (Knuth, 1969).

A tree is an acyclic graph that may be represented by a list of parents and their children. If we compute coordinates for parents and children with a layout method (see Section 16.3) then the *join*() statistic makes it easy to draw the tree.

Figure 7.35 shows a tree depicting the results of a single-linkage clustering of sociometric ratings of U.S. cities (Boyer and Savageau, 1996). The distance scale measures the closest distance between pairs of points in the two clusters joined at each node.

ELEMENT: *edge*(*position*(*link.join*(xparent*yparent+xchild*ychild))), *label*(rating))

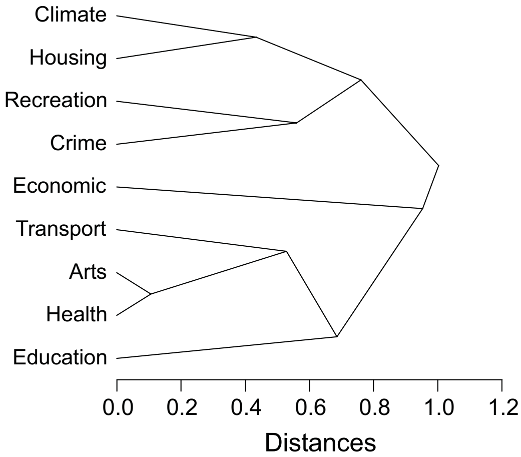


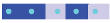
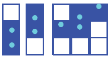
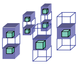
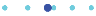
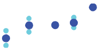
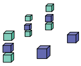
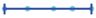
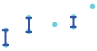
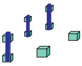
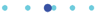
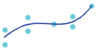
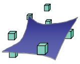
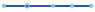
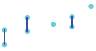
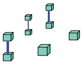
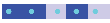
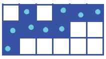
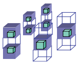
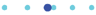
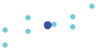
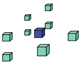
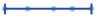
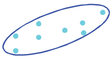
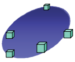
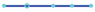
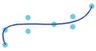
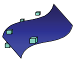
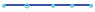
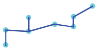
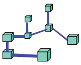
Figure 7.35 *Cluster tree*

7.3 Summary

Table 7.2 summarizes the statistical methods discussed in this chapter. Each cell of the table shows an exemplar for the type of method in 1D, 2D, or 3D. We have included points to represent data values in the graphical exemplars; these are colored light blue. The graphics that display the results of statistical methods are colored dark blue. The conditional methods are in the upper half of the table and the joint methods are in the lower.

The conditional methods compute a unique value or unique set of values on a selected variable for every distinct value of *x*. For 1D, *x* is a constant. For 2D, *x* is a variable. For 3D, *x* is a two-dimensional vector-valued variable. If *x* is categorical, then the computed estimates are spaced on a lattice. If *x* is continuous, then these estimates are distributed in a real space. The *summary* methods involve a single point. The *region* methods involve intervals or types of convex regions. The *smooth* methods involve connected sets of unique points for every value of *x*. The *density* methods involve binning or parametric smoothing for computing a density in a local region. Finally, the *link* methods involve computations on points that are nodes in directed or general graphs.

Table 7.2 **Statistical Methods by Dimensionality**

	1D	2D	3D
Conditional			
Bin			
Summary			
Region			
Smooth			
Link			
Joint			
Bin			
Summary			
Region			
Smooth			
Link			

The joint methods compute a unique value or unique set of values on a selected set of variables. For 1D, these methods are identical to conditional methods because conditioning on a constant is equivalent to not conditioning at all. Thus, the first column of Table 7.2 is identical for conditional and joint methods. The 2D and 3D joint methods, however, differ from their conditional counterparts. The differences are most apparent when considering the geometrical features of the graphics in Table 7.2.

For example, the function *smooth.conditional*() is single-valued for *y* on each different value of *x*. The function *smooth.joint*(), by contrast, can produce a curve that can loop back on itself in 2D or a surface that can do the same in 3D. This is because *smooth.conditional*() is based on a loss defined only on *y*, while *smooth.joint*() is based on a loss defined by *x* and *y* jointly (or **x** and *y* in 3D).

Similarly, the function *region.conditional*() is conditionally one-dimensional in 2D and 3D, while *region.joint*() is usually **convex** in 2D and 3D. (Any line connecting two points in a convex region lies entirely in that region.) Actually, *region.joint*() produces estimates that fall within a superset of the convex class, called **star convex**. (Any line connecting one central point with another point in a star convex region lie entirely in that region.) Thus, while *region.conditional*() produces an interval on y for any given x, *region.joint*() produces a region comprised of intervals constructed in polar coordinates. To compute *region.joint*(), we look in all directions away from a centroid point and we compute bounding intervals in each of those directions. In statistical procedures based on the multivariate normal distribution, this region is calculated from a covariance matrix. The off-diagonal elements of this matrix determine the rotational angle of a joint elliptical region and the diagonal elements determine its major and minor axes. Under other distributions, however, *region.joint*() can produce a region shaped like a starfish or a daisy.

While statistical methods are associated with geometric regions in a space, we must remember that these regions can be represented by various geometric graphs. That is the message of Figure 7.1 at the beginning of this chapter. Some of the results may be aesthetically bizarre, but they are not ill-defined. Indeed, the power of a system that distinguishes geometric graphs and their statistical methods lies in offering a complete choice of representation methods rather than dictating a subset on the basis of custom.

7.4 Sequel

The next chapter presents the different geometry classes and how these functions produce geometric objects that can be represented by aesthetics as perceivable shapes.

8

Geometry

The word **geometry** comes from the Greek *γεωμετρία*, which means land measurement. A geometer measures magnitudes in space. This chapter is about geometric functions produced by a Grapher object. The Grapher object contains functions to create graphs that can be represented by magnitudes in a space. Grapher cannot make every graph in the set of all possible graphs. Grapher produces only certain graphs that can be expressed as geometric objects. We will call these **geometric graphs**.

The geometric graphs in this chapter are subsets of product sets of real numbers $\boldsymbol{R}^m$ or natural numbers $\boldsymbol{N}^m$. We will be concerned with geometric objects for which $1 \leq m \leq 3$. These objects will be embedded in a space $\boldsymbol{R}^n$ in which $m \leq n \leq 3$. Geometric graphs are built from **bounded regions**. Bounded regions are produced by the Cartesian product of bounded intervals. The set B^m is bounded if

$$B^m \subset [a_1, b_1] \times [a_2, b_2] \times \ldots [a_m, b_m]$$

These intervals define the edges of a bounding box (like the bounds of a frame) in *m*-dimensional space. There are two reasons we need bounded regions. First, in order to define certain useful geometric graphs, we need concepts like the *end* of a line or the *edge* of a rectangle. Second, we want to save ink and electricity. We don't want to take forever to compute and draw a line. More precisely, we need to embed geometric graphs in a frame, which is itself a bounded region. We want the image of the function that produces a geometric graph to be bounded (although a few of the transformations in Chapter 9 can produce images unbounded in $\boldsymbol{R}^n$ that we will have to clip in a renderer).

Geometric graphs are produced by graphing functions F: $B^m \rightarrow \boldsymbol{R}^n$ that have geometric names like *line*() or *polygon*(). A geometric graph is the image of *F*. And a graphic, as used in the title of this book, is the image of a graph under one or more aesthetic functions. Geometric graphs are not visible. As Bertin (1967, 1977) points out, visible elements have features not present in their geometric counterparts. Chapter 10 will cover methods for making graphs perceivable. Meanwhile, we will use aesthetic functions in this chapter to illustrate different types of graphs. These graphs will be displayed using position, color, size, and shape, but the same graphs could be realized using sound or even, theoretically, odors. Although this chapter is about geometry, we will discuss briefly issues in rendering these geometric objects in order to reinforce this distinction.

To maintain the distinction between a graph and its physical representation, we will call the output of a *line*() graphing function a *line* graph and we will call the output of the composite of a *line*() graphing function and its aesthetic functions (*position*(), *color*(), *size*(), ...) a *line* graphic. To distinguish geometry from function, we will use Roman type to refer to a geometric line, and Italic type to refer to the specialized *line* produced by *line*().

There are several ways to classify graphics. First, we could organize them by their appearance under standard aesthetic functions: for example, *interval* as opposed to *line*. This would consolidate drawing methods and thus conserve display code. Organizing by surface appearance makes it more difficult to collect similar geometric methods in single classes, however. Although they appear similar, *line* and *path* are fundamentally different geometric objects. A second approach would be to classify them by their geometric dimensionality (the m parameter in B^m). This would consolidate rendering methods. Organizing by dimensionality makes it difficult to consolidate data methods, however. A third approach would be to organize them by their data methods, regardless of appearance. Any method which involved computing a location estimate (mean, median, mode, etc.) could be grouped together. This approach, of course, would disperse drawing methods.

We have chosen to organize graphs by their data *and* geometry. Because this system is about statistical graphics, the most fruitful classifying scheme, we believe, is based on how graphs function in representing statistical data geometrically. Graphs that behave similarly in a variety of contexts are grouped together. This scheme results in three major categories of graphs: **functions**, **partitions**, and **networks**. Functions map values in a domain to values in a range using graphs that enable one to locate a value or collection of values in the range for any selected value in the domain. Partitions separate a set of points into two or more subsets. Networks connect two or more points with line segments. Table 8.1 summarizes these classifications.

The remarkable feature of this table is its parsimony. An enormous number of graphical elements can be grouped into a relatively small number of graphical element types. Undoubtedly there are other graph types needed to reproduce some graphics not found in this book, but adding them to this system should not require altering its architecture. New graphing functions, if defined properly, should be self-contained and housebroken enough to avoid doing violence to the rest of the system. In addition, many graphics that appear radically different from the ones found in this chapter are either transformations of the geometry or functions of the data that underlie the graphs in Table 8.1. Before you wonder why a popular graphic is missing in this chapter, see Chapter 7 or Chapter 9 for how it may be derived from the base graph classes.

Table 8.1 **Geometric Graphs**

Functions	Partitions	Networks
point *line* *area* *interval* *path* *schema*	*polygon* *contour*	*edge*

8.0.1 Collision Modifiers

Ties sometimes cause graphical elements to overlap in frames. Also, aggregation statistics sometimes cause geometric objects to superimpose. For example, a scatterplot may have overlapping points or a bar chart may have multiple bars at one tick point on a categorical axis (each colored by a different value of a splitter). Because geometric objects know their locations and can ask other geometric objects for their locations, they can avoid collisions in a variety of systematic ways. Table 8.2 lists various collision modifiers.

Table 8.2 **Collision Modifiers**

Modifier

stack
  symmetric
  asymmetric
dodge
  symmetric
  asymmetric
jitter
  uniform
  normal

The *stack* method cumulates elements in order of the values on a splitter. For example, we can make a stacked bar chart by having superimposed bars *stack* on their second categorical dimension. The standard stacking option is *asymmetric*; that is, the bottoms of stacks are anchored on a common position. The other option is *symmetric*; the centers of stacks are anchored on a common position. The *dodge* method does not cumulate. It simply moves objects around locally so they do not collide. And the *jitter* method moves objects randomly in their local neighborhood. Sometimes *stack* and *dodge* can produce a similar-appearing graphic. Stack cumulates on a scale (*e.g.*, a stacked bar chart) while dodge piles things in open space (*e.g.*, a tally or dot plot).

8.1 Examples

The following examples show the basic geometric objects embedded in 2D and 3D spaces. While rendering methods in different dimensions are quite different in some cases, the classification we use simplifies data processing in most cases.

8.1.1 Functions

Functional graphs associate one or more values in a domain with a value or collection of values in a range. This is the largest super-class of graphs and one that contains most of the representation objects seen in popular charts.

8.1.1.1 Point

The *point*() graphing function produces a geometric point, which is an *n*-tuple. This function can also produce a finite set of points, called a **multipoint** or a **point cloud**. The set of points produced by *point*() is called a *point* graph.

In graphics software, there is a choice that object-oriented designers have argued over for years. On the one hand, we may call a point cloud a single object. On the other hand, we may call each point a single object. When object-oriented language compilers were slow to create objects, this choice was forced by circumstance. Modern compilers can construct millions of objects in a second, however, so the choice is less obvious. We prefer to regard a point cloud as a collection of points in order to encapsulate functions that are most appropriately handled at the point level. As an object, each point can encapsulate metadata, calculate derived statistics, and do other tasks for clients in the system. If efficiency is still a concern, the designer can incorporate the tradeoff in the software itself. Beyond about 5,000 points, for example, there is not much point in point-based functions. Aggregations and kernels do better.

Rendering a *point* is relatively straightforward. To visualize a *point* graph as a *point* graphic, we need a *shape* attribute of a circle, a diamond, a face, or some other image. And we need a *size* attribute that makes it large enough to be discernible. We also need a *hue* attribute that makes its color different from the background color of the frame graphic in which it is displayed. If points overlap, we can use transparency to prevent occlusion.

Figure 8.1 shows examples of a *point* cloud on a categorical and continuous domain. The left panel contains three *point* graphics (one for each category) and the right panel, one. The *point* graphic can have numerous variations, produced mainly by varying the *shape* attribute of their symbols and by the aggregation function used, as we shall see in Chapter 7, where we will show examples of multivariate *icon* clouds. Otherwise, this most common of graphics is one of the simplest to visualize.

ELEMENT: *point*(*position*(gov*birth))

ELEMENT: *point*(*position*(female*birth))

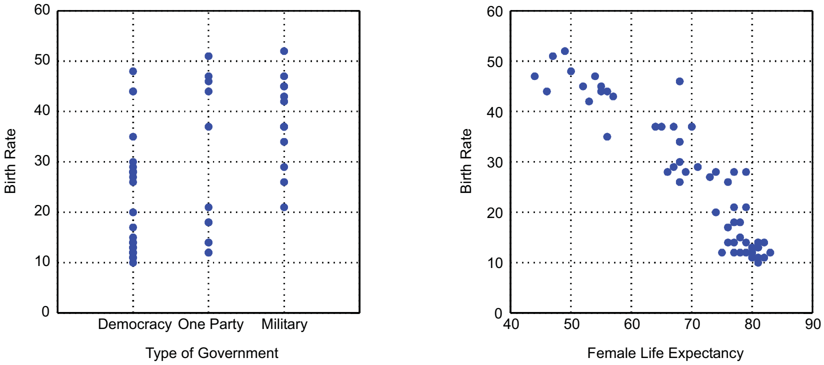


Figure 8.1 *A point cloud on categorical (left) and continuous (right) domains*

Figure 8.2 shows an example of *point* graphics in 3D. For the left panel, we have introduced another categorical variable, urban, which is a measure of urbanization for each country. For the right panel, the data are measurements of automobile performance in various 1996 issues of the magazine *Road & Track*. The three variables are seconds to cover a quarter mile from a standing start (quarter), horsepower (hp), and weight in pounds (weight).

COORD: *rect*(*dim*(1, 2, 3))
ELEMENT: *point*(*position*(urban*gov*birth))

COORD: *rect*(*dim*(1, 2, 3))
ELEMENT: *point*(*position*(weight*hp*quarter))

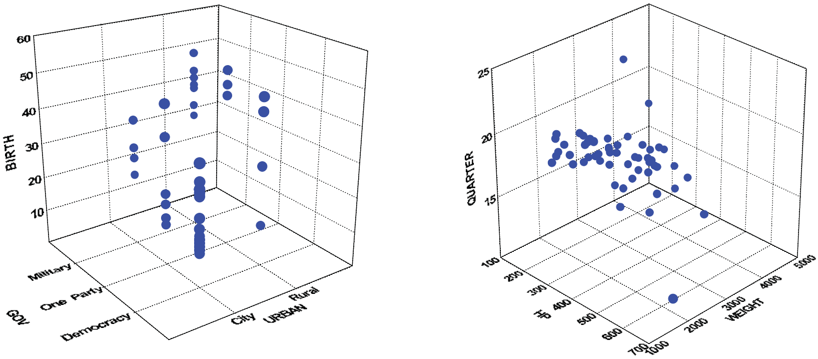


Figure 8.2 *3D points on categorical (left) and continuous (right) domains*

8.1.1.2 Line

Let B^m be a bounded region in $\boldsymbol{R}^m$. Consider the function F: $B^m \rightarrow \boldsymbol{R}^n$, where $n = m+1$, with the following additional properties:

1) the image of F is bounded,

2) $F(x) = (\boldsymbol{v}, f(\boldsymbol{v}))$, where f: $B^m \rightarrow \boldsymbol{R}$ and $\boldsymbol{v} = (x_1, \ldots , x_m) \in B^m$

If $m = 1$, this function maps an interval to a functional curve on a bounded plane. And if $m = 2$, it maps a bounded region to a functional surface in a bounded 3D space. The *line*() graphing function produces these graphs. Like *point*(), *line*() can produce a finite set of lines. A set of *lines* is called a **multiline**. We need this capability for representing multimodal smoothers, confidence intervals on regression lines, and other multifunctional lines. An example is shown in the lower left panel of Figure 7.2.

Rendering a *line* is not simple. The *line*() graphing function returns a value on a *line* for any tuple $(x_1, \ldots , x_m)$. We cannot compute every possible tuple in B^m in order to draw a line as a set of points, however (although some programs like MacSpin (Donoho *et al*., 1988) and Data Desk (Velleman, 1998) that were designed to animate point clouds do represent a line or surface by a fine mesh of points that are rendered in pixels). Instead, we must choose a few **knots** that define the ends of line segments and then interpolate between these knots. For straight lines, this is straightforward: we linearly interpolate between two endpoints. For curved lines, we may either construct many knots and use linear interpolation or make fewer knots and use a spline function to interpolate curvilinearly (Lancaster and Salkauskas, 1986; Dierckx, 1993). The advantage of splines is that some operating systems include them among their primitives, so that time and memory can be saved. For 3D surfaces, knots define a rectangular or triangular mesh that yields polygons for rendering. As with 2D, some operating systems include 3D spline functions, so we should design a renderer to take advantage of them when available.

There are many other problems we face in rendering a *line*. We must be able to handle conditions where the slope approaches infinity. If we have missing values, we need rules to determine how a gap (or a hole in 3D) should be treated. And if we want to achieve the full range of aesthetic representations shown in Chapter 10, we have to treat a *line* as a collection of polygons or sometimes even symbols so that we can give it *size* (thickness), *shape* (symbols), and *texture* (dashing patterns). Few operating systems give us the primitives needed to draw a curved, dashed line made up of dots, for example.

We will further explore the aesthetics of lines in Chapter 10. At this point, however, it is useful to keep in mind that the definition of a *line* as a set of points is probably the best way to approach the rendering problem. What distinguishes a *line* from a cloud of *points* is that the points comprising it are ordered. Thus, we can render a line by stamping copies of symbols in a given ordering or by connecting a set of polygons to make segments.

Figure 8.3 shows *line* graphics of average birth rates on categorical (left) and continuous (right) domains.

ELEMENT: *line*(*position*(*summary.mean*(gov*birth)))

ELEMENT: *line*(*position*(*summary.mean*(female*birth)))

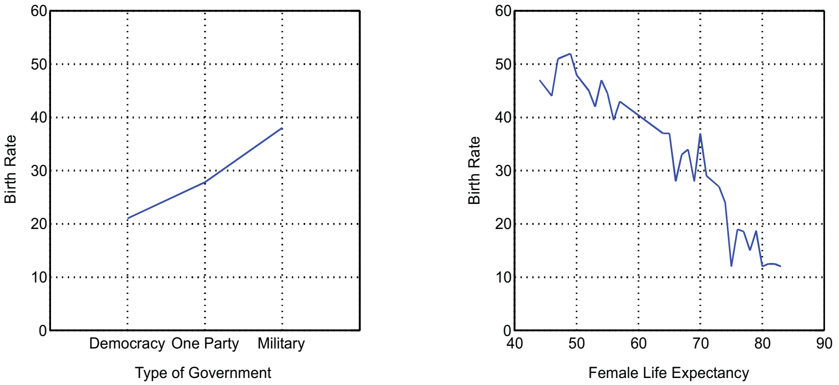


Figure 8.3 *Line on categorical (left) and continuous (right) domains*

Figure 8.4 shows 3D *line* graphics. The left panel is based on a categorical domain (gov*birth), so it contains a collection of *line* graphics. We have given them thickness to make them look like ribbons, a popular representation. We use the alias *surface* for the graphic on the right, but it is nevertheless a *line* class.

COORD: *rect*(*dim*(1, 2, 3))
ELEMENT: *line*(*position*(*summary.mean*(urban*gov*birth)))

COORD: *rect*(*dim*(1, 2, 3))
ELEMENT: *surface*(*position*(x*y*z))

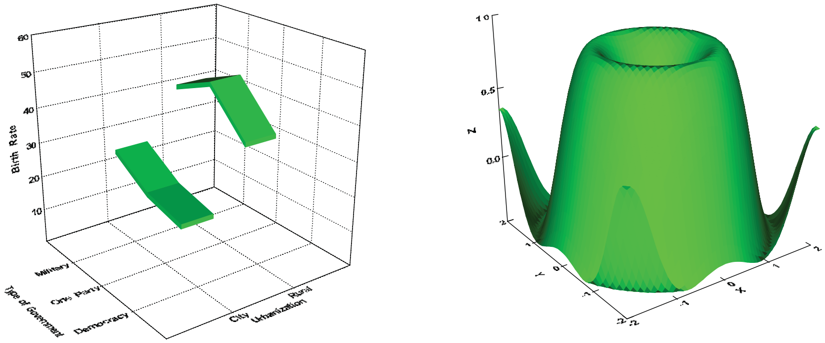


Figure 8.4 *Surface on categorical (left) and continuous (right) domain*

8.1.1.3 Area

The *area*() graphing function produces a graph containing all points within the region under the *line* graph. The *area*() function is the integral of *line*(). Rendering an *area* involves the same caveats as for *line*. Figure 8.5 shows *area* graphics of average birth rates. The *area* graphic looks like a *line* graphic with the area between it and the abscissa filled.

ELEMENT: *area*(*position*(*summary.mean*(gov*birth)))

ELEMENT: *area*(*position*(*summary.mean*(female*birth)))

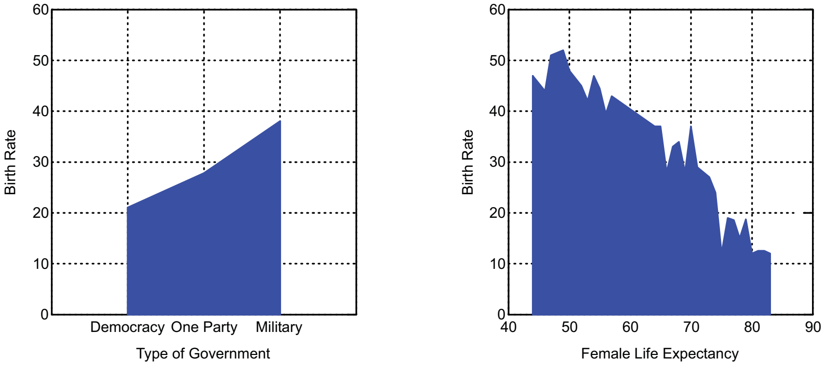


Figure 8.5 *Area on categorical (left) and continuous (right) domains*

The *area* graphic is a *volume* in 3D. Figure 8.6 shows an example.

COORD: *rect*(*dim*(1, 2, 3))
ELEMENT: *area*(*position*(*summary.mean*(urban*gov*birth)))

COORD: *rect*(*dim*(1, 2, 3))
ELEMENT: *area*(*position*(x*y*z))

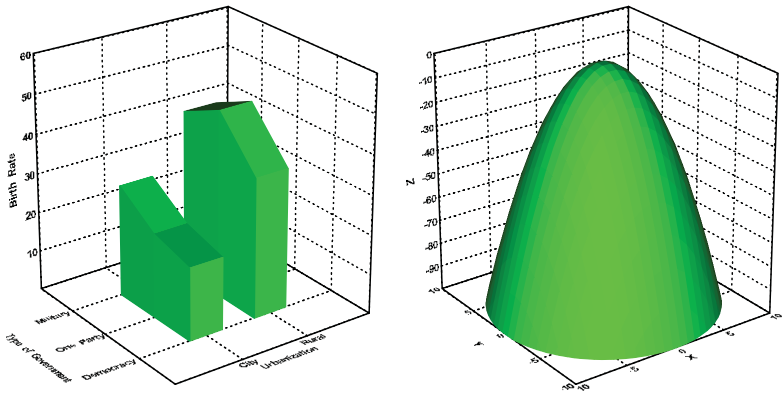


Figure 8.6 *Volume on categorical (left) and continuous (right) domain*

8.1.1.4 Interval

The *interval*() graphing function produces a set of closed intervals. An interval has two ends. Ordinarily, however, bars are used to denote a single value through the location of one end. The other end is anchored at a common reference point (usually zero). Figure 8.7 shows a 2D *interval* graphic.

ELEMENT: *interval*(*position*(*summary.mean*(gov*birth)))

ELEMENT: *interval*(*position*(*summary.mean*(female*birth)))

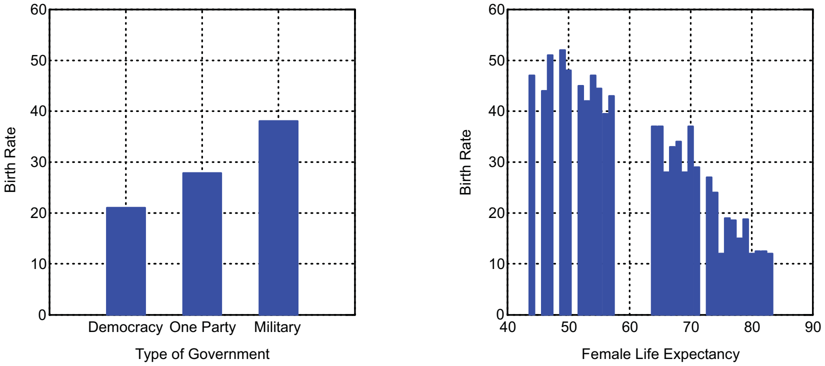


Figure 8.7 *Interval graphic on categorical and continuous domains*

Figure 8.8 shows a 3D *interval* graphic for the car data used in Figure 8.2.

COORD: *rect*(*dim*(1, 2, 3))
ELEMENT: *interval*(*position*(*summary.mean*(urban*gov*birth)))

COORD: *rect*(*dim*(1, 2, 3))
ELEMENT: *interval*(*position*(*summary.mean*(weight*hp*quarter)))

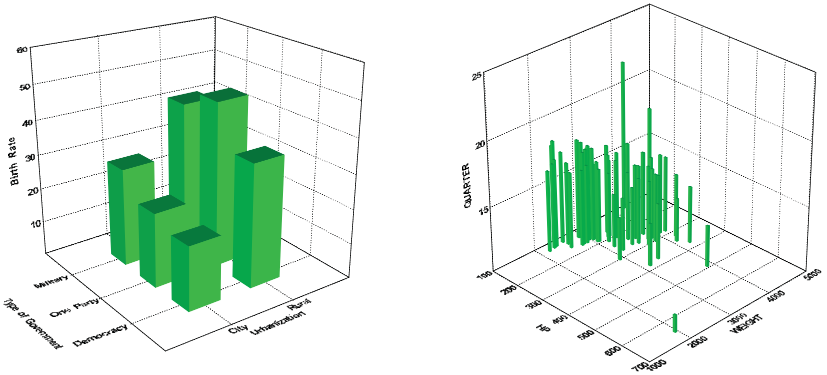


Figure 8.8 *3D interval graphic on categorical and continuous domains*

Despite appearances, there is only one *interval* graphic in each of the right panels of these two figures. The bars in the left panels, by contrast, are split by the categorical variables. This is consistent with the behavior of *point*() (see Figure 8.1, where there are three clouds on the left and one on the right). The only way to sense this behavior would be to query the objects in a dynamic system by brushing or editing. In such an environment, the left panel bars would respond singly and the right ones would respond in unison.

The *interval* graphic uses a mean aggregation function for its position in this example. This function returns the mean of all values in the range for a given value or tuple of values in the domain. The top of each bar in the left panel of Figure 8.7 represents the average birth rate in each of the three gov categories and the top of each bar in the left panel of Figure 8.8 represents the average birth rate in each of the six combinations of gov and urban.

The *interval*() graphing function can be used to produce a histogram. Ordinarily, a bar's width is an arbitrary constant. The width of a histogram bar, by contrast, is determined by the *bin* statistical function. Because *bin* is a partitioning, the result produces space-filling intervals on the dimension of binning. Figure 8.9 shows examples of 2D and 3D histograms.

ELEMENT: *interval*(*position*(*summary.count*(*bin.rect*(birth))))

COORD: *rect*(*dim*(1, 2, 3))
ELEMENT: *interval*(*position*(*summary.count*(*bin.rect*(female*male))))

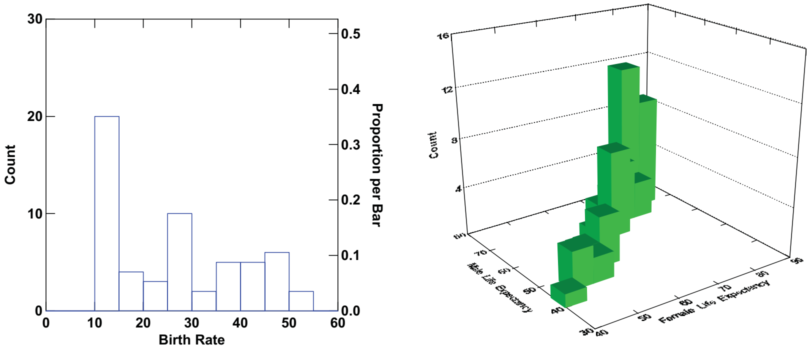


Figure 8.9 *2D and 3D histograms*

The histogram graphic reveals some subtle characteristics when it is compared to similar-appearing graphics. Unlike the ordinary bar graphic, which represents intervals (usually anchored at zero), the histogram graphic represents area. Unlike the *area* graph, which represents a single area, the histogram graphic is a collection of areas, one for each bar. Although most published examples show ordinary histograms with bars representing counts on equal intervals, the bars in the histogram graphic need not be of equal width. Some may even have zero area. Unlike the *interval* graphic applied to statistical functions other than *bin*, however, the bars in a histogram must be connected. There cannot be gaps between bars unless these are due to bars with zero cases in a bin.

8.1.1.5 Path

The *path*() graphing function produces a *path*. A *path* visits all points and connects all points such that each point touches no more than two line segments. Thus, a *path* visits every point in a collection of points only once. If a *path* is closed (every point touches two line segments), we call it a *circuit*.

A *path* sometimes looks like a *line*. There is an important difference between the two, however. A *line* is functional on *x*; there can be only one point on a *line* for any value in the domain. In contrast, a *path* may loop, zigzag, and even cross itself inside a frame. We can use functional *vs*. parametric coordinates to represent this distinction. A 2D line is comprised of the tuples $(x, f(x))$. A 3D line is comprised of the tuples $(x, y, f(x,y))$. If t is a variable representing a sequence, then a 2D path is comprised of the tuples $(x(t), y(t))$ and a 3D path is comprised of the tuples $(x(t), y(t), z(t))$. From this perspective, it is clear that a path can be defined for a continuous function, so we include path as a member of the functions superclass rather than the networks superclass.

There is another important distinction between a *path* and a *line*. The aesthetics of a *path* may vary along different segments of the *path*, while a *line* has a single aesthetic value across its entire extent. We make use of this distinction in the Napoleon graphic in Chapter 20.

Figure 8.10 shows a price–consumption curve for cigarettes sold in the US between 1964 and 1986 (Harris, 1987). The path order, based on time, is labeled on the curve. Notice that the segmented structure of a *path* allows us to label each point along its way. We could not do this with a *line*.

Prior to 1981, the demand for cigarettes was often inelastic. From 1975 to 1981, however, lowering prices were accompanied by reduced demand. After that time, consumption declined further and prices increased. This trend was most likely due to the growth of anti-smoking legislation and negative publicity.

For other examples of paths, see theoretical Hertzsprung–Russell diagrams used by astronomers (Mihalas and Binney, 1981), phase-plane plots (Figure 14.12), or Figure 20.1.

ELEMENT: *path*(*position*(price*consumption), *label*(year))

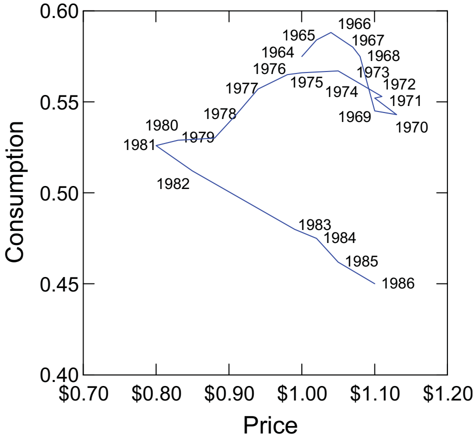


Figure 8.10 *Consumption-price curve for cigarettes*

8.1.1.6 Schema

A **schema** is a collection of one or more points and intervals used to represent a data density. Because it is a collection of relations, the computer code to build a schema is derived from the *point*() and *interval*() functions. The name is due to Tukey (1977), who invented the **schematic plot**, which has come to be known as the **box plot** because of its physical appearance. Schema graphics can take many shapes and can be based on different statistics. Tukey's box plot is the default. This plot is based on statistics called **letter values**. The central vertical line in the box is the **median**, computed by sorting a list of values and taking the middle sorted value. If the number of cases is even, then the two middle values are averaged. The edges of the box are the **hinges**, computed from the medians of the two batches produced when the sorted values are split at the overall median. The ends of the whiskers in the box plot extend to the most extreme values inside the **inner fences**. These fences are defined as follows:

$$lower\ fence = lower\ hinge - 1.5 Hspread$$
$$upper\ fence = upper\ hinge + 1.5 Hspread, \quad \text{where}$$

Hspread is the spread of the hinges, namely, the *upper hinge* minus the *lower hinge*. Finally, the **outer fences** are computed using 3*Hspread* in the same formulas. Values outside the outer fences (far outside values) are plotted with a small circle and remaining values outside the inner fences are plotted with asterisks. This rather complicated set of definitions produces a remarkably parsimonious plot in which outlying values are immediately recognizable and the distribution of the remaining values is schematically represented by a box and whiskers. The general information is conveyed in the box and whiskers, and the particular information is conveyed in the outliers.

Figure 8.11 shows a box plot of the distribution of horsepower among the cars in the *Road & Track* dataset. Two cars are highlighted as extreme. These are the Lamborghini Diablo (asterisk) and the Ferrari 333 race car (circle).

ELEMENT: *schema*(*position*(*bin.quantile.letter*(hp)))

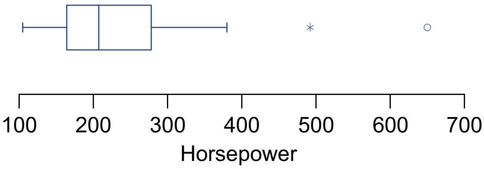


Figure 8.11 *Schematic (box) plot*

When plotted against continuous variables, boxes are usually unevenly spaced. We compute a box where there is more than one *y* value for a given *x*. Elsewhere, the median horizontal line marks the unique *y* value. Figure 8.12 shows an example using data on gas and electricity consumption from Tukey (1977). The electricity values were grouped into batches before plotting.

ELEMENT: *schema*(*position*(*bin.quantile.letter*(electric*gas)))

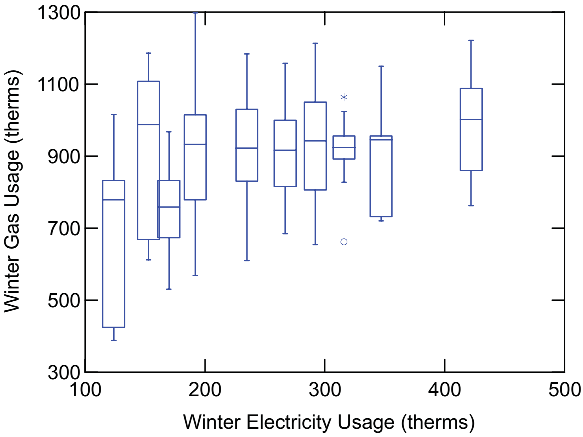


Figure 8.12 *Box plots on continuous domain*

The behavior of the 2D joint box plot is different from the usual conditional box plot. Figure 8.13 shows an example of the joint plot. Notice that the box plot is composed of peeled convex hulls. Tukey (1974; also see Huber, 1972) suggested this method for generalizing the box plot to more than one dimension. Each hull in this plot contains a different percentage of the total number of countries. The outermost hull contains all the countries, and each successive hull contains fewer by about 25 percent. See Rousseeuw, Ruts, and Tukey (1999) for a coded implementation and a beautifully rendered display.

ELEMENT: *schema*(*position*(*bin.quantile.letter.joint*(female*birth)))
ELEMENT: *point*(*position*(*summary.median.joint*(female*birth)))

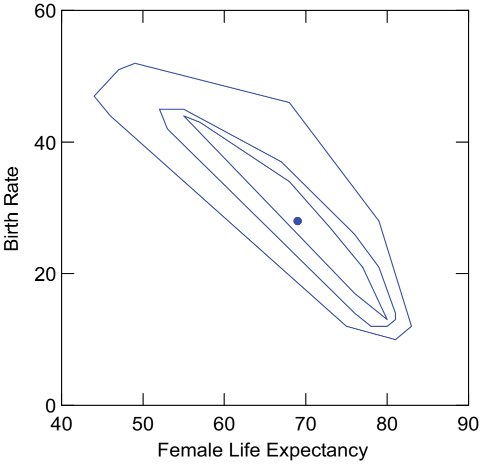


Figure 8.13 *Bivariate box plot*

8.1.2 Partitions

Partitions divide datasets into subsets. One type of element (*polygon*) separates a space into two mutually exclusive regions (areas or volumes). Multiple polygons may overlap each other or not. Mutually exclusive and exhaustive polygons can be used to tile the plane or a 3D space. The second class of partitions (*contour*) separates points into two or more regions, possibly nested. Polygons are closed, but contours need not be. Level curves for saddle functions are open, for example.

8.1.2.1 Polygon

A *polygon*() graphing function can tile a surface or space, filling the space with mutually exclusive polygons. The Latinate *tessellation* (for tiling) is often used to describe the appearance of the result. Figure 8.14 shows a tiling based on the mathematical function

$$z = xy(\sin x^2 - \cos y^2)^2$$

There are 10,000 tiles in the figure (a 100 by 100 rectangular grid).

There is a detail concerning the arguments that we should note. As Chapter 18 describes, GPL functions take functions, primitives, or lists of primitives as arguments. Thus, *min*(–5, –5), *max*(5, 5) is equivalent to the function list *min1*(–5), *max1*(5), *min2*(–5), *max2*(5) in another language. Notice, also, that the DATA statement can take a list of identifiers, x, y and the TRANS statement has an algebraic parser.

```
DATA: x, y = mesh(min(–5, –5), max(5, 5), n(100, 100))
TRANS: z = x*y*(sin(x^2) - cos(y^2))^2
ELEMENT: polygon(position(bin.rect(x*y)), color.hue(z))
```

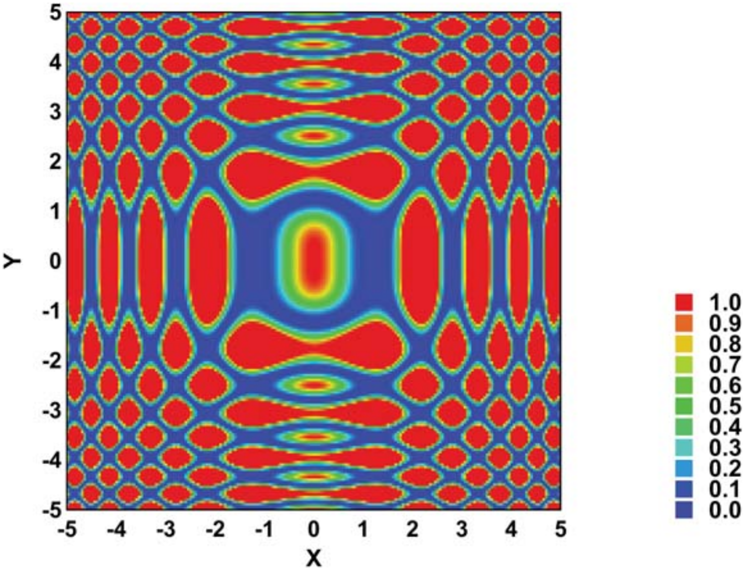


Figure 8.14 *Tiled equation*

The *polygon* graphic is also used in geographic mapping. Figure 8.15 shows a map of Europe using a *robinson* projection. The *map* data function sets the source for the map polygons. We have set it to refer to a shape file called "Europe" that contains country boundaries. We could have set the reference to a different shape file containing other boundaries such as cantons or provinces.

There is no statistical data file in this example, so the position of the polygons (the position frame for the graphic) is determined by the information in the boundary file. Most of the other map examples include statistical data. To save space, we omit the DATA statements for defining the statistical variables in many of these examples (as in the next example), but we assume that they would be included in a GPL program to plot the examples.

Maps and tilings are not the same. The *polygon* graphic is useful for mapping political boundaries, islands, lakes, archipelagos, and any other geographic objects that tile the plane or sphere. In addition, we can use other graphics such as *point* (for towns or cities), *path* and *link* (for roads, railroads, etc.) and even images (for terrain) to create more detailed maps. Comprehensive GIS programs require more graphic representation objects in order to deal with topological data, but a statistical graphics environment can produce a substantial subset of the geographic representations needed for analysis of spatial data.

DATA: longitude, latitude = *map*(*source*("Europe"))
COORD: *project.robinson*(*dim*(1, 2))
ELEMENT: *polygon*(*position*(longitude*latitude))

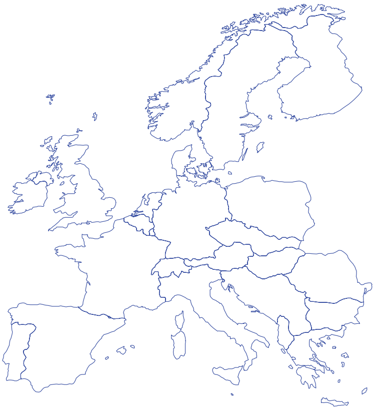


Figure 8.15 *Map of Europe*

8.1.2.2 Contour

A *contour*() graphing function produces contours, or level curves. A *contour* graphic is used frequently in weather and topographic maps. Contours can be used to delineate any continuous surface. Figure 8.16 shows contours for the inverse distance smoother relating average winter temperatures to latitude and longitude. We have superimposed the smoother contours on a map of the US We discussed this inverse distance smoothing method in Chapter 7.

DATA: longitude, latitude = *map*(*source*("US states"), *id*("state"))
COORD: *project.stereo*(*dim*(1, 2))
ELEMENT: *polygon*(*position*(longitude*latitude))
ELEMENT:*contour*(*position*(*smooth.mean.cauchy*(lon*lat*winter)),
color.hue())

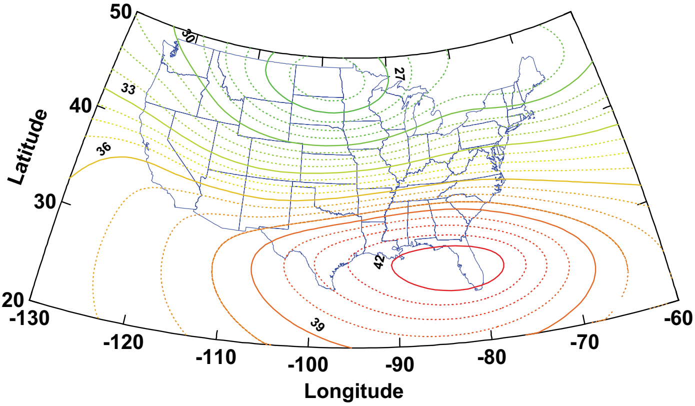


Figure 8.16 *Contour plot of smoothed winter temperatures*

8.1.3 *Networks*

Network graphs join points with lines. Networks represent vertex–edge theoretic graphs (Harary, 1969). Although networks join points, a *point* graph is not needed in a frame in order for a network graph to be visible.

8.1.3.1 Edge

The *edge*() graphing function produces a collection of edges. The various graphs in this class are subsets of a complete network connecting every pair of points. Figure 8.17 shows a graph of all possible links among points at the vertices of an octagon.

ELEMENT: *edge*(*position*(*link.complete*(x*y)))

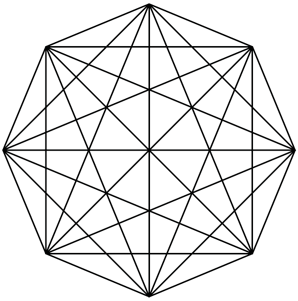


Figure 8.17 *A complete network for eight points at the vertices of an octagon*

Figure 8.18 shows a 3D minimum spanning tree on the car data used in the right panel of Figure 8.2. Adding a tree like this to a 3D scatterplot can sometimes help enhance depth, especially when rotating the plot in real time.

COORD: *rect*(*dim*(1, 2, 3))
ELEMENT: *point*(*position*(weight*hp*quarter))
ELEMENT: *edge*(*position*(*link.complete*(weight*hp*quarter)))

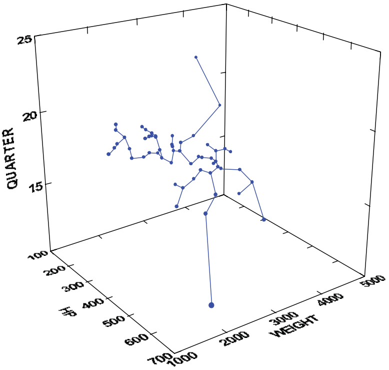


Figure 8.18 *Minimum spanning tree in three dimensions*

8.1.4 Collision Modifier Examples

Collision modifiers generate a variety of graphics whose structure is not immediately obvious. We will provide several examples in this section.

8.1.4.1 Stack

Stacking cumulates elements. Divided bar charts, for example, consist of collections of interval-valued bars. The lower value of each bar is the previous case's value in the dataset (the first case defaults to zero) and the upper value is the current case. Figure 8.19 shows an example on the ACLS data.

ELEMENT: *interval.stack*(*position*(*summary.proportion*(gender*response),
color(response)))

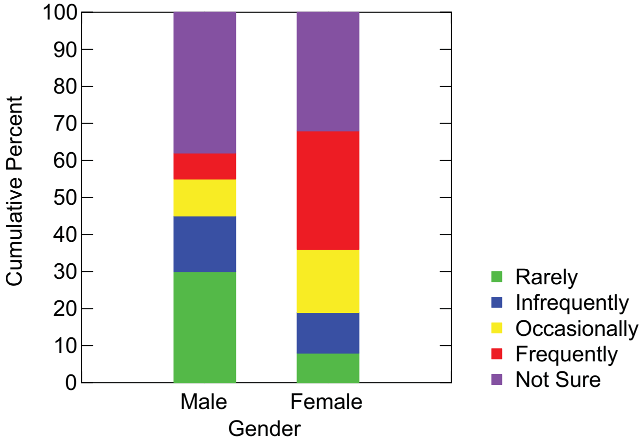


Figure 8.19 *Divided bar graphic*

Figure 8.20 shows a stacked *area* chart. The data are taken from tests on our Java grammar-of-graphics platform. We use this graphic to monitor performance over development cycles. When the total stacked area becomes especially large, we pause for several days and work on optimizing our code. When a particular test performs poorly, we concentrate on the code for that section.

This graphic was generated as part of our automated production process. When programmers change even one line of code, the build system runs unit and acceptance tests to assure that the system is not broken by the changes. Performance tests are run periodically to assure that changes do not adversely affect overall performance of the system.

```
DATA: s1 = string("scatter70K")
DATA: s2 = string("colorSc70K")
...
DATA: s9 = string("pie15x15x8")
ELEMENT: area.stack(position(date*(g1+ ... +g9)), color(s1+ ... +s9))
```

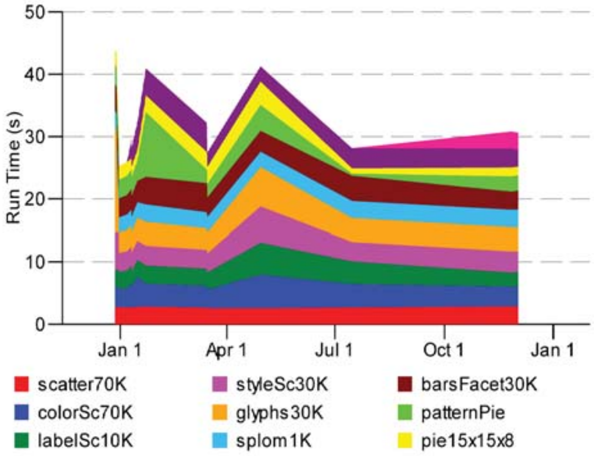


Figure 8.20 *Stacked area chart*

8.1.4.2 Dodge

Figure 8.21 shows examples of dot plots on the countries dataset. The dot plot on the left looks like a tally or a histogram of dots. The dot plot on the right is used by medical researchers and other scientists to graph small batches of data. It is simply a symmetrized form of the asymmetrical dot plot.

```
ELEMENT:point.dodge.asymmetric(position(bin.dot(birth)))

ELEMENT: point.dodge.symmetric(position(bin.dot(birth)))
```

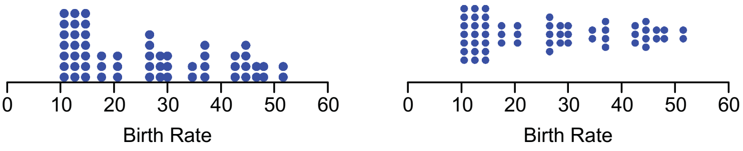


Figure 8.21 *Asymmetric dot plot (left) and symmetric dot plot (right)*

The *dodge*() collision modifier is useful for creating clustered bar charts and other grouped graphics. Figure 8.22 shows an example using the countries data. The default dodging is symmetric about each tick mark on the horizontal axis. The categorical color splitter is responsible for producing two sets of bars, but the *dodge*() function is responsible for separating them.

ELEMENT: *interval.dodge*(*position*(*summary.mean*(gov*birth)),
color(urban))

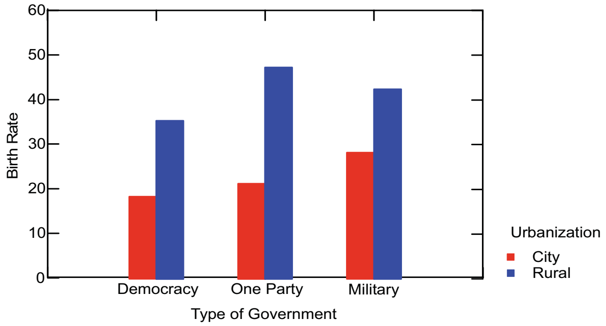


Figure 8.22 *Clustered bar chart.*

There is an important but subtle consequence of using *dodge*() for clustered bar charts. If a subcategory is missing for a class (*e.g.*, rural democracies), it will not appear in the cluster for that class. To allocate space for empty subclasses, we must use a full crossing: *interval*(*position*(urban*birth*birth*gov)).

Figure 8.23 shows an example of a clustered box plot of the same data. The dodging works the same way.

ELEMENT: *schema.dodge*(*position*(*bin.quantile.letter*(gov*birth)),
color(urban))

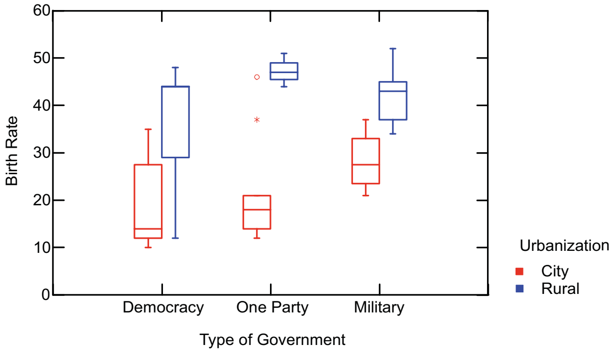


Figure 8.23 *Clustered box plot*

8.1.4.3 Jitter

Figure 8.24 illustrates how to jitter a 1D plot. The data and frame are the same as for Figure 5.1.

ELEMENT: *point.jitter*(*position*(sepallength))

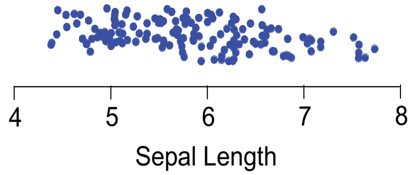


Figure 8.24 *One-dimensional jittered scatterplot*

Figure 8.25 illustrates jittering of a two-way table using normally distributed random stripes. The data are from 1,606 respondents to the General Social Survey (GSS) reported in Davis *et al*. (1993). US residents were asked, "How many sex partners have you had in the last 12 months?" Those reporting more than 4 partners (some reported up to 100) were consolidated into the last category. There were 1,466 responses in the resulting 6 categories. The variable represented on the vertical axis of the graph was measured by the response to the question, "Which of these statements comes closest to describing your feelings about the Bible?" The responses coded are 1 (Word of God), 2 (Inspired Word), and 3 (Book of Fables).

In Figure 8.25, the stripes are jittered as random normals, so that approximately 68 percent of the stripes are within one standard deviation of the mean. Stripe jittering is effective at handling sparse counts but less effective in distinguishing the cells with large counts.

ELEMENT: *point.jitter.normal*(*position*(sex*bible))

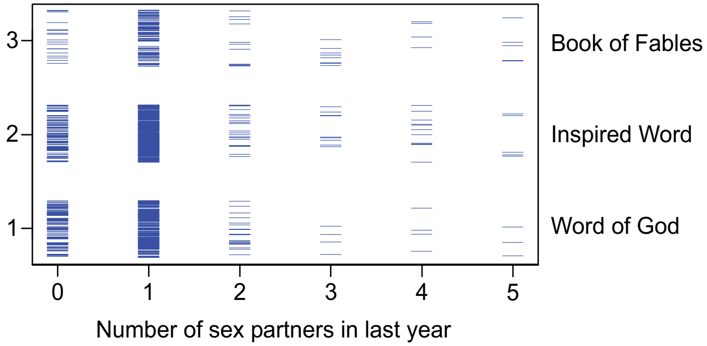


Figure 8.25 *Jittered counts*

8.1.5 Splitting vs. Shading

Categorical variables *split* graphs. Figure 8.26 shows an example of this splitting. The data consist of centimeter measurements from the Anderson Iris flower dataset analyzed by Fisher (Anderson, 1935). Anderson measured sepal length and width and petal length and width in centimeters for 50 flowers in each of three Iris species — *Setosa, Versicolor, and Virginica.* Since there are three categories for species, the *point* graph is split into three objects.

Because species is set to be categorical, we would expect to see three sets of dots in the graphic for the Iris data, each consisting of 50 points for each of the three species. Because there is no attribute other than position assigned to the splitting variable on a common axis, we would not be able to distinguish the points in these clouds visually. In a dynamic graphics system, we could observe behavior (selection, editing, etc.) which would indicate this splitting.

ELEMENT: *point.dodge.symmetric*(*position*(*bin.dot*(species*sepalwidth)))

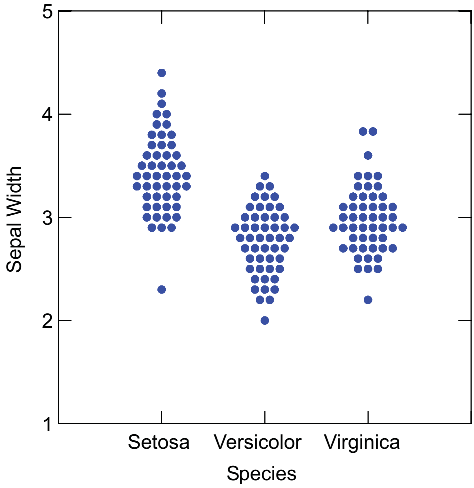


Figure 8.26 *Two-dimensional grouped dot plot*

Any use of a categorical variable in a specification splits graphs. Nesting splits because nesting variables are required to be categorical. A simple example would be

ELEMENT: *point*(*position*(sepalwidth/species))

This would produce three *point* clouds, one for each set of 50 points derived from each of the species. For a multiple nesting like a/b/c, we must count all existing combinations of b and c to determine the number of splits. If nesting is combined with blending and/or crossing, then do this count before considering splits due to crossing and blending.

Categorical variables used for aesthetic attributes split graphs:

ELEMENT: *point*(*position*(sepalwidth*sepallength), *color*(species))

This produces three *point* clouds, one for each set of 50 points within each value of species, and each a different color.

A continuous variable produces a visual result similar to that for a categorical variable but it does not split. We say that it *shades*. For many graphics, we cannot tell whether a variable splits or shades unless we know in advance whether that variable is continuous or categorical or we can explore the graphic using interactive tools that let us identify whether subgroups belong to a common class or to separate classes.

Finally, when we want to force splitting without using a variable for any other purpose, we can specify splitting explicitly by adding *split*() to the aesthetic methods, for example:

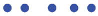
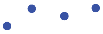
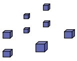
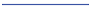
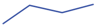
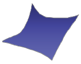
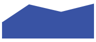
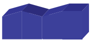
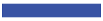
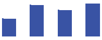
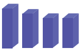
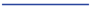
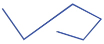
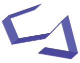
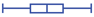
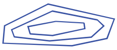
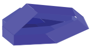
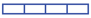
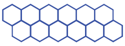
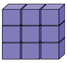
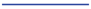
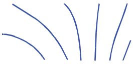
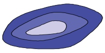
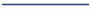
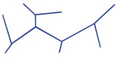
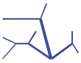
ELEMENT: *point*(*position*(sepalwidth*sepallength), *split*(species))

This method is not often used, because it leaves no aesthetic attribute to signal that splitting has happened. Figure 20.1 in the last chapter reproduces an historical graphic that requires this operation. In the original graphic, the same color was used to represent different split groups — not ordinarily a good idea.

8.2 Summary

Table 8.3 summarizes the graphs in this chapter for 1D through 3D. The pictures in the cells are graphical exemplars for the graphs shown in Table 8.1. The 1D rendering environment leaves us few opportunities to discriminate between graph types. We may use thickness in 1D when this does not measure a data attribute. For example, we can choose a symbol shape in a 1D graphic to represent some aspect of our data, but the center of all the symbols must still fall on a line and their size has no data meaning. Similarly, the thickness of *interval*, *schema*, and *polygon* has no data meaning in 1D. The 2D graphics provide an extra degree of freedom to represent data variation. Graphics such as *area*, *contour*, *path*, and *edge* require at least two dimensions to be useful. While 1D graphics are most constrained, 3D ones are least. The *line* graph becomes a *surface* and the *area* becomes a *volume*. In like manner, the *polygon* graph partitions a 3D space so that each polytope encloses a volume. The *contour* graph partitions 3D space similarly.

Table 8.3 **Geometric Graphs**

	1D	2D	3D
		Functions	
Point			
Line			
Area			
Interval			
Path			
Schema			
		Partitions	
Polygon			
Contour			
		Networks	
Edge			

8.3 Sequel

Most business and scientific graphics are displayed in rectangular coordinates. The next chapter covers coordinate systems that alter the appearance of graphics. Some of these coordinate transformations are so radical that we may fail to recognize the same geometric object embedded in different coordinate systems.

9

Coordinates

The word **coordinate** derives from the Latin *ordinare*, which means to order or arrange. Coordinates are sets that locate points in space. These sets are usually numbers grouped in tuples, one tuple for each point. Because spaces can be defined as sets of geometric objects plus axioms defining their behavior, coordinates can be thought of more generally as schemes for mapping elements of sets to geometric objects.

The most familiar coordinates are Cartesian. A point is located on a Cartesian plane, for example, by its distances from two intersecting straight lines. The distance from one line is measured along a parallel to the other line. Usually, the reference lines (axes) are perpendicular. Most popular graphics, such as line or bar charts, are drawn using Cartesian coordinates. The same real numbers behind these graphics can be mapped to points along circles, curves, and other objects, however. This chapter examines functions that transform a set of coordinates to another set of coordinates.

There are many reasons for displaying graphics in different coordinate systems. One reason is to simplify. For example, coordinate transformations can change some curvilinear graphics to linear. Another reason is to reshape graphics so that important variation or covariation is more salient or accurately perceived. For example, a pie chart is generally better for judging proportions of wholes than is a bar chart (Simkin and Hastie, 1987). Yet another reason is to match the form of a graphic to theory or reality. For example, we might map a variable to the left-closed and right-open interval [0, 1) on a line or to the interval $[0, 2\pi)$ on the circumference of a circle. If our variable measures defects within a track of a computer disk drive in terms of rotational angle, it is usually better to stay within the domain of a circle for our graphic. Another reason is to make detail visible. For example, we may have a cloud with many points in a local region. Viewing those points may be facilitated by zooming in (enlarging a region of the graphic) or smoothly distorting the local area so that the points are more separated in the local region.

In the figures in this chapter, axes often become curves or change their orientation or scaling or direction after a change of coordinates. This is an important consequence of coordinate transformations. Axes, or guides, share geometry with graphics and are therefore subject to the same transforming effects as *bars*, *boxes*, and other graphics. Not everything in a guide changes, however. In particular, text may rotate or move in other ways, but the placement and orientation of text is a special problem created by some coordinate systems. We must be able to read text. Because of this constraint, we cannot draw an image of a graphic and its text and then transform the entire image. The technical problem of text placement is not trivial and may account for why so few graphics and geometry programs handle text well.

There are many ways to organize this material, because coordinates are basic to analysis, geometry, algebra, and other areas of mathematics. In applying these concepts to graphics, however, we are most interested in the appearance of a result and its effect on the perception of variation being represented in a graphic. Therefore, we will tend to organize concepts by geometry.

Most of this chapter concerns continuous transformations, but we will cover discrete examples briefly. We will begin with the classic transformation groups involved in mapping the plane to itself. Then we will discuss polar and other general planar transformations. Next, we will cover projections to the plane, including global map projections. We will conclude with triangular, spherical, cylindrical, and parallel coordinates. We will omit detail in some of the specifications when it is not needed to make clear the geometry.

None of these treatments pretends to be comprehensive or abstract. The focus here is on practical methods for representation and organization of this material in a form that lends itself to efficient computer code for the task. Introductory core and tangential references for some of the topics discussed in this chapter are Preparata and Shamos (1985), Rogers and Adams (1990), Foley *et al.* (1993), Emmer (1995), Banchoff (1996), and Gomes and Costa (1998).

9.1 Transformations of the Plane

If (x, y) and (u, v) are elements, respectively, of two sets S_1 and S_2, then the set of equations

$$\begin{cases} u = g(x, y) \\ v = h(x, y) \end{cases},$$

where g and h are functions, transforms (x, y) to (u, v). As with all functions, we call (u, v) the **image** of (x, y) and $T(x, y)=(g(x, y), h(x, y))$ a **mapping** of S_1 to S_2. Ordinary graphics such as *intervals* and *polygons* take on radically different appearances under different planar transformations. Furthermore, if we don’t recognize this visually profound effect of shape transformations, we are likely to think a particular chart involves a new type of graphic or a different source of variation when, in fact, it is a simple coordinate transformation of a popular rectangular graphic.

The first part of this section is organized in terms of the class hierarchy of planar transformations in Figure 9.1. The CLASS column shows each class of transformation inheriting from its more general parent. For example, an isometry is a similarity, but a similarity is not an isometry. The TRANSFORMATION column indicates the methods within the transformation class. The INVARIANCE column indicates the feature of graphical objects unchanged after transformation. Finally, the IMAGE column shows the effect of each transformation on a *dino* graphic (you didn't see one in Chapter 8?).

The **isometry** group is the set of transformations that preserve distance between points. These operations obey the axioms of Euclidean geometry. The three isometric transformations are the **rigid transformations**: translation, rotation, and reflection. Translation moves an object vertically or horizontally without changing its shape, size or orientation. Rotation rotates an object around a point (usually its center) without changing shape or size. Reflection inverts an object horizontally or vertically without changing its size or shape, like looking in a mirror.

The **similarity** group is the set of transformations that change the size of an object. The transformation name, **dilation**, suggests enlargement. Nevertheless, dilation includes both shrinking and enlargement. The name of the group implies that two objects of the same shape but of different sizes and at different locations are nevertheless similar.

The **affine** group is the set of transformations that cause a dimension to stretch independently of the other. It also includes a shear, which is like turning Roman into Italic letters. The word affine implies that linearly stretched and sheared objects, regardless of their size and location, share an affinity of form. We call stretch and shear *affine* transformations.

The **projective** group is the set of transformations that is most easily visualized by thinking of a light source shining on graphics drawn on a transparent plane and projecting an image on another plane. This transformation preserves straight lines but can modify angles. The best physical model for imagining this group of transformations is a slide projector that we can move to any angle and location relative to a white wall. A picture projected on the wall from an oblique or acute angle will fan out across the wall; straight lines in the picture (sidewalks, buildings, etc.) will remain straight on the wall.

The **conformal** class covers conformal mappings. Conformal mappings preserve local angles in graphics, but may distort global shape considerably. The conformal class, like the affine class, is a parent of the similarity class. The conformal class is not a parent of the affine class, however, because affine transformations do not preserve angles. Similarity transformations (unlike affine transformations) are angle preserving. In fact, a conformal map looks like a similarity map at the local level. "Small" objects maintain their shape under a conformal transformation. "Big" objects are bent, sometimes considerably, so that straight lines become curves.

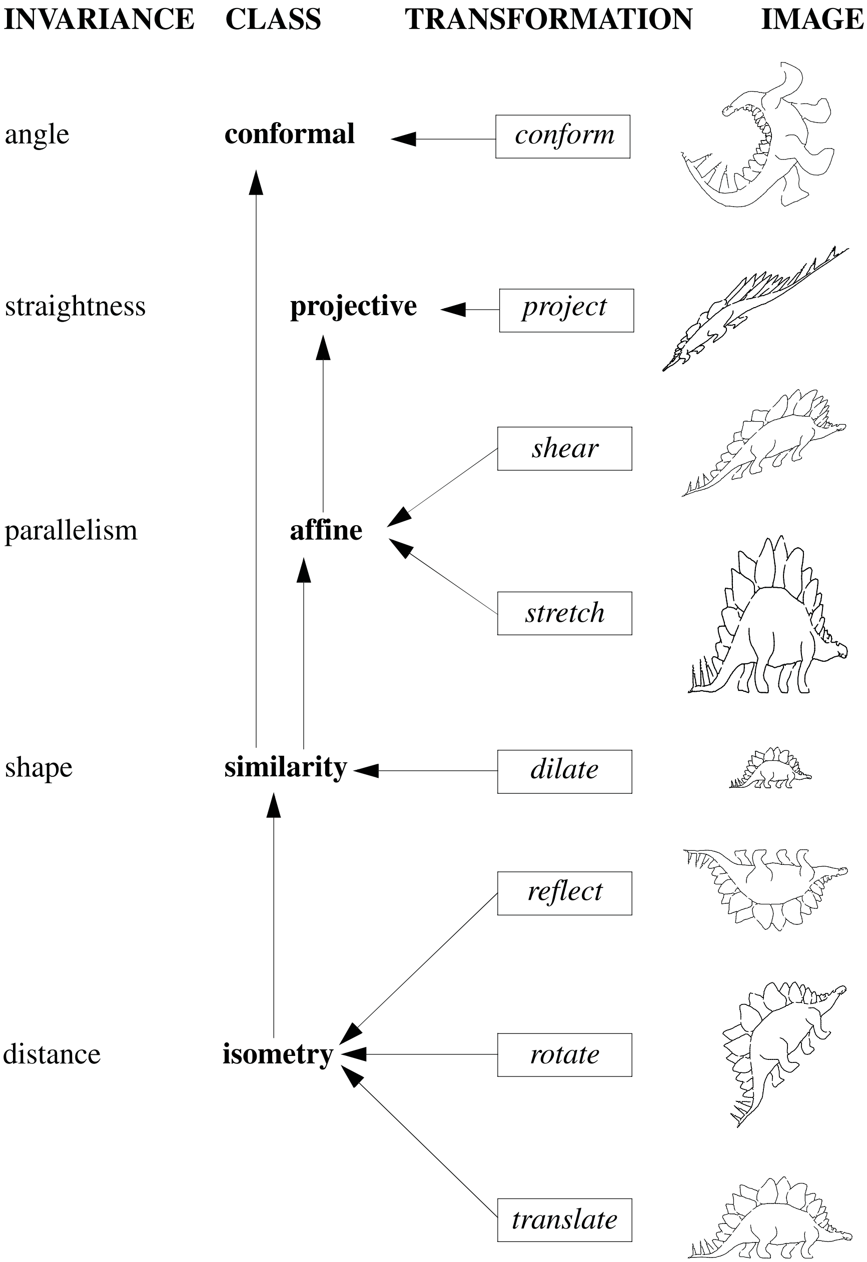


Figure 9.1 *Hierarchy of planar transformations*

9.1.1 *Isometric Transformations*

A **metric space** is a set S together with a function

$$d\colon S \times S \to [0, \infty)\ \text{, where}$$

$$\begin{aligned} &d(x, y) = 0 \quad \Leftrightarrow \quad x = y \\ &d(x, y) = d(y, x) \\ &d(x, z) \le d(x, y) + d(y, z) \end{aligned}$$

Although this definition is general enough to be applied to objects other than real numbers, we will assume d is a distance measure and x, y, z are points in a space defined on the real numbers. Thus, (1) zero distance between two points implies that the points are the same, and if two points are the same, the distance between them is zero, (2) the distance between the point x and the point y is the same as the distance between the point y and the point x, and (3) a **triangle inequality** among distances exists for any three points x, y, and z.

An instance of a metric space is the n-dimensional Euclidean space consisting of n-tuples $(x_1, \ldots, x_n)$ of real numbers x_i, with distance metric

$$d(x_j, x_k) = \left(\sum_{i=1}^{n} (x_{ij} - x_{ik})^2 \right)^{1/2}$$

If S_1 and S_2 are metric spaces with distance functions d_1 and d_2, then a function $g\colon S_1 \to S_2$ is an **isometry** transformation if and only if

$$d_2((g(x)), g(y)) = d_1(x, y) \quad \text{for all} \quad x, y \in S_1$$

Isometries on the plane involve translation, rotation, and reflection. All of these preserve distance. While there are formal proofs for this assertion, the simplest thing is to look at the pictures.

9.1.1.1 Translation

Translation sends the coordinates (x, y) to $(x+a, y+b)$. We will not show a figure for this, because translation is nothing more than moving a graphic right or left, up or down, or a combination of both, without changing its orientation. The dinosaur in Figure 9.1 has been translated from somewhere off the page (in space, not time), perhaps from Philadelphia.

The most frequent use for translation is in paneled graphics. As we will show in Chapter 11, we use translation to arrange a set of frames in a table of graphics. By composing translations with other coordinate transformations, we can produce even more unusual arrangements of multigraphics.

9.1.1.2 Rotation

Rotation sends the polar coordinates (r, θ) to $(r, \theta{+}c)$. This is equivalent to sending (x, y) to $(\cos\theta x - \sin\theta y, \sin\theta x + \cos\theta y)$. The dinosaur in Figure 9.1 is rotated 45 degrees. Figure 9.2 shows a bar chart rotated $\theta = 270$ degrees counter-clockwise. This is the coordinate transformation most commonly done to produce horizontal bar graphics. Notice that not everything is rotated in the graphic, however. Because the *rotate*() function operates on the view instead of the frame, the location of text rotates to follow the location of associated elements (axes in this case), but its orientation can be governed by other aesthetic considerations or constraints. The *position*() function signifies that the coordinates apply to the position aesthetic. This is almost always the case, although it possible to do coordinate transformations on other aesthetics.

ELEMENT: *interval*(*position*(gov*birth))

COORD: *rotate*(*dim*(1, 2), *angle*(270))
ELEMENT: *interval*(*position*(gov*birth))

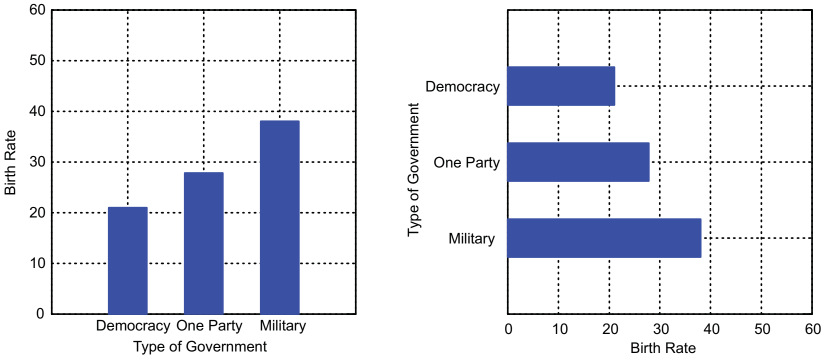


Figure 9.2 *270 degree orthogonal rotation*

Figure 9.3 shows an unusual graphical application of the rotation transformation. This example is taken from Tukey (1977). Tukey devised this graphic to represent simple additive models involving effects computed from a row-by-column two-way table. The model he graphs is

$$fit = row + column$$

where *fit* is the predicted value for a cell in the table. Tukey plots *fit* against a "forget it" dimension (*row* – *column*) through the transformation:

$$(row, column) \rightarrow ((row + column), (row - column))$$

This is proportional to the rotation $(\cos\theta x - \sin\theta y, \sin\theta x + \cos\theta y)$, where θ is –45 degrees, x is the *column*, and y is the *row*.

The data are mean monthly temperatures for three places in Arizona. Tukey's fitted values that determine the raw coordinates in Figure 9.3 are based on row and column medians of the means. The motivation for these calculations is to enable both simple paper-and-pencil methods and robustness of the fit. The size of the plotting symbols is proportional to the residuals of the fit. Crosses (+) indicate positive residuals and circles (o) indicate negative.

Tukey calls the horizontal direction of the rotated graphic a "forget-it" dimension. The vertical dimension that shows the fitted average temperatures across locations and months defines the principal variation of interest. Tukey has produced a scale of temperature based on joint variation among places and months. We will present in Chapter 11 another example of this approach in the context of a procedure that Duncan Luce and Tukey developed, called **conjoint measurement** (Luce and Tukey, 1964).

To keep this example simple, we omitted from the specification the overlay to produce the vertical temperature scale on the right. We also omitted a dilation constant of $1/\sqrt{2}$ (due to using cosines and sines). Related applications motivated by Tukey's graphic can be found in numerous sources, including Velleman and Hoaglin (1981) and Hsu (1996). See, for example, Figure 15.24. Tukey's graphic reminds us to turn our heads occasionally so that we can see hidden relationships.

TRANS: ar = *abs*(residual)
TRANS: sr = *sign*(residual)
COORD: *rotate*(*dim*(1, 2), *angle*(-45))
ELEMENT: *point*(*position*(month*city), *size*(ar), *shape*(sr))

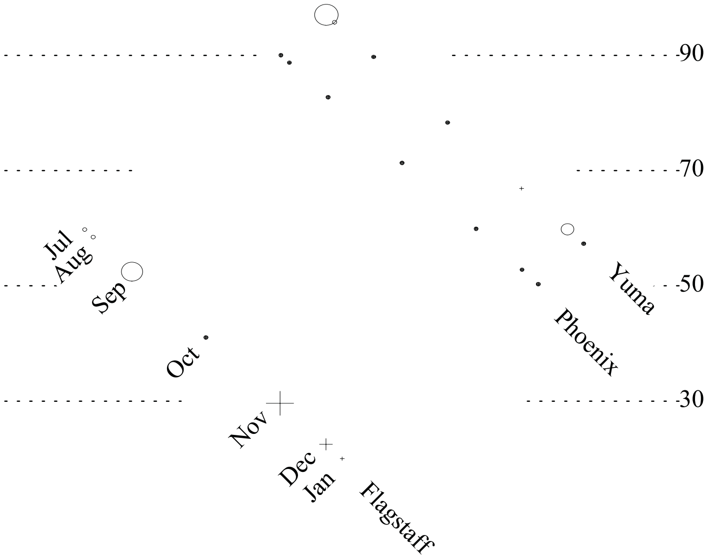


Figure 9.3 *Tukey additive two-way plot*

9.1.1.3 Reflection

Reflection sends (x, y) to ($-x$, y) or to (x, $-y$). This operation reverses the vertical or horizontal orientation of a graphic. The dinosaur in Figure 9.1 is upended by negating the second coordinate. In Figure 9.4, we have used the *reflect*() method to accomplish a vertical reflection. The last argument (2) ties the reflection to the second dimension.

Vertical reflection produces stalactites from stalagmites. To switch similes, reflection is a way to make icicles out of trees. When the tree display used in cluster analysis is turned upside-down so that the leaves are at the bottom and the root is at the top, it is called an **icicle plot**.

ELEMENT: *interval*(*position*(gov*birth))

COORD: *reflect*(*dim*(2))
ELEMENT: *interval*(*position*(gov*birth))

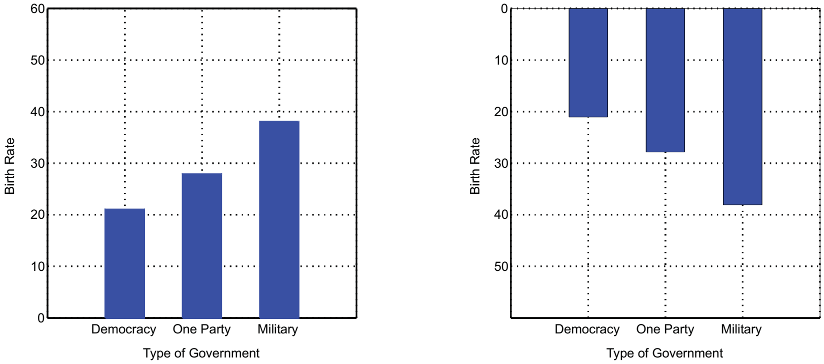


Figure 9.4 *Vertical reflection*

Figure 9.5 illustrates the composite transformation of *reflect* followed by *rotate*. This composite transformation is useful enough to merit its own function, which we call *transpose*(). A transposition is a flip of a graphic around its northeast–southwest diagonal, similar to a matrix transpose, but on the other diagonal.

A reflection followed by a rotation is still an isometry. Indeed, any sequence of $T_1(T_2(T_3(...)))$ is still an isometry if each T_i is a *reflect*(), *translate*() or *rotate*(). However, the result of the sequence *rotate*(*reflect*()) is not equivalent to that of the sequence *reflect*(*rotate*()). Even within their own class (*e.g.*, isometry) planar transformations are not commutative. As we shall see, these transformations are equivalent to matrix products and matrices are not commutative under multiplication. Compare Figure 9.5 to Figure 9.6.

ELEMENT: *interval*(*position*(gov*birth))

COORD: *reflect*(*dim*(1), *rotate*(*dim*(1, 2), *angle*(270)))
ELEMENT: *interval*(*position*(gov*birth))

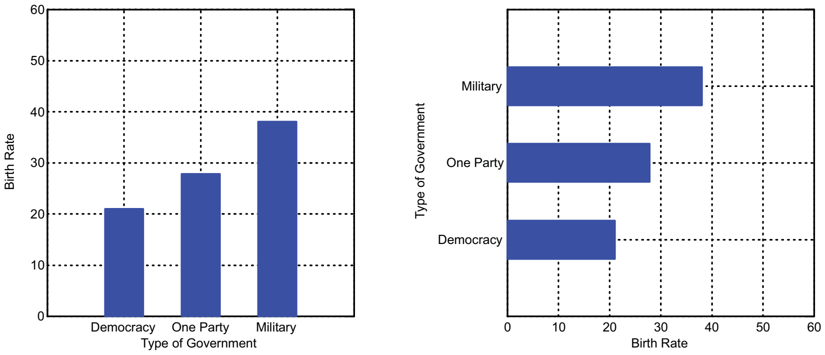


Figure 9.5 *Reflection followed by rotation*

ELEMENT: *interval*(*position*(gov*birth))

COORD: *rotate*(*dim*(1, 2), *angle*(270), *reflect*(*dim*(1)))
ELEMENT: *interval*(*position*(gov*birth))

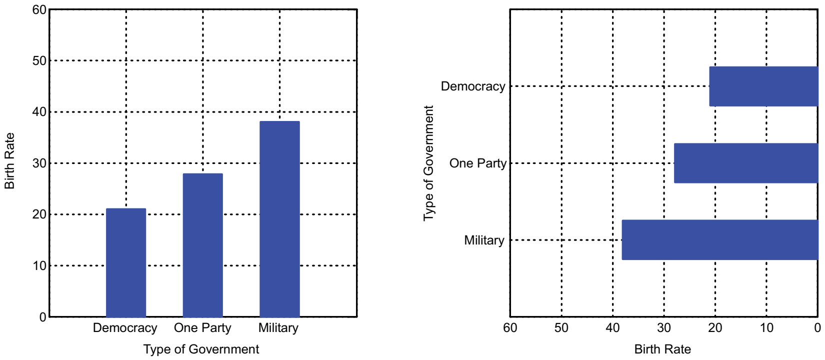


Figure 9.6 *Rotation followed by reflection*

As we have seen, the *transpose* transformation is easily mistaken for a rotation, but there is an even more subtle consequence that is easily overlooked. This involves the role of the range and domain in a graph. In Figure 9.5, birth is assigned to the range and gov is assigned to the domain. That is, the graph and its resulting graphic are designed to treat birth rates as a function of type of government. Whether or not this is true in real life is another matter.

What if we decided to reverse these roles? Figure 9.7 shows the difference. This operation is a *pivot* of the frame. We have reassigned the range to gov and the domain to birth in the specification. The resulting graphic expresses type of government as a function of birth rates. Note that the range need not be continuous. In this case a categorical range works fine. If we wanted a summary measure (instead of one bar for every value), we could use the mode. In any case, the graphics and summary measures inside a frame do not change the following fundamental rules of a specification:

$$x*y \neq y*x$$

$$x*y \neq \mathit{transpose}(y*x)$$

$$x*y = \mathit{pivot}(y*x)$$

Figure 9.7 shows that careless use of pivoting may have unintended consequences. To get what we want, we need to pay attention to the model rather than to the appearance of the graphic.

ELEMENT: *interval*(*position*(birth*gov))

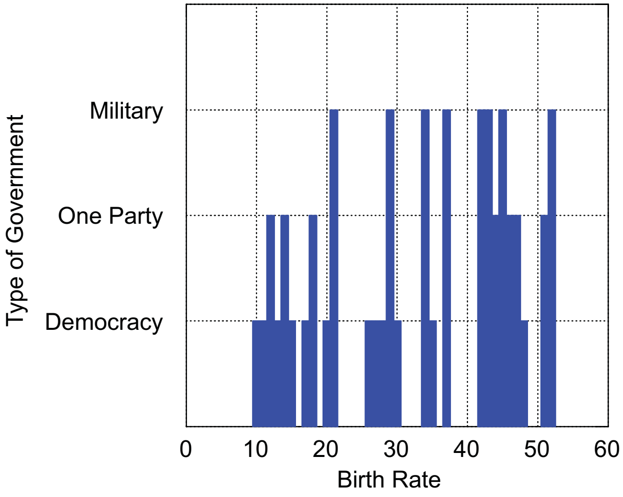


Figure 9.7 *Pivot of* gov*birth

We have adopted the name *pivot* from tables and database terminology. A *pivot* of a two-way table in spreadsheets and other computer software is an exchange of rows and columns. Tables aren't ordinarily thought of as having a range and domain, but statisticians who have developed models for analyzing variation in tables know well the difference between a log-linear model and a logistic regression model. The former model adopts the cell entries for the range and the cell margins (averages or other functions of rows and averages or other functions of columns) for the domain. The latter model adopts one margin for the range and the other for the domain. Sharing terminology between tables and graphs is not motivated by convenience or analogy, however. Tables *are* graphs. We will discuss in Chapter 11 the isomorphism between tables and graphs that makes clearer why pivoting a graph is the same operation as pivoting a table. This has important implications for spreadsheet technology, particularly those spreadsheets that produce graphics. Graphics algebra takes the guesswork out of producing a pivoted graphic from a pivoted table.

Figure 9.8 illustrates these important points in a pair of graphics that differ only subtly. In both graphics, the clouds look identical because *point* graphics look the same across range-domain exchanges. In the left plot, however, we have embedded a logarithmic regression smoother

$$E[y] = a + b\log(x)$$

where y is assigned to data and x is assigned to dist. In the right plot, we have embedded the same smoother except y is assigned to dist and x to data because of the pivot. The differences between the two curves are due to the different regression models. On the left, estimates for data are conditioned on dist and on the right, estimates for dist are conditioned on data. In the left plot, residuals (differences between the points and the regression line) are calculated vertically and in the right, horizontally. In the left plot, the range is on the vertical axis and domain on the horizontal. In the right plot, the range is on the horizontal and the domain on the vertical.

The graphic consequences of these specification rules are more than aesthetic. Figure 9.8 presents a Shepard diagram from a multidimensional scaling (MDS) of Morse code confusion data in Rothkopf (1957). The variable data contains the confusions (similarities) among the 26 Morse codes for the letters of the alphabet (numbers have been omitted here). The variable dist contains corresponding distances between points in the MDS configuration representing the letters. The smaller the residuals in the Shepard diagram, the better the fit of the MDS configuration to the original data.

Roger Shepard, the inventor of nonmetric multidimensional scaling and the person after whom this plot was named, oriented the plot as in the left panel of Figure 9.8 because the problem had been posed in the classical framework of fitting a theoretical configuration of points to a set of observed data values (Shepard, 1962). Subsequently, Kruskal (1964) and others formulated the problem as a multidimensional minimization of a loss function defined on the distances instead of the data. Rather than change the orientation of the Shepard diagram, Kruskal changed the assignments of the range and domain and represented his residuals horizontally. Later computer programs finally transposed the plot, presumably to reduce confusion over this matter.

This example is related to the statistical procedure known as **inverse regression** (Brown, 1986). In linear inverse regression, we employ the linear model $X = \alpha_0 + \alpha_1 Y$ to predict X, as opposed to the model $Y = \beta_0 + \beta_1 X$ to predict Y. If the ordinary least squares estimates of the parameters of these models are a_0, a_1, b_0, and b_1, then the prediction for a value of X at a given value y_h is $\hat{X} = a_0 + a_1 y_h$ for inverse regression and $\hat{X} = (y_h - b_0)/b_1$ for ordinary regression. This model was originally proposed for machine calibration problems and is now chiefly of historical interest.

ELEMENT: *point*(*position*(dist*data))
ELEMENT: *line*(*position*(*smooth.log*(dist*data), *color*("red"))

COORD: *pivot*(*transpose*())
ELEMENT: *point*(*position*(dist*data))
ELEMENT: *line*(*position*(*smooth.log*(dist*data)), *color*("red"))

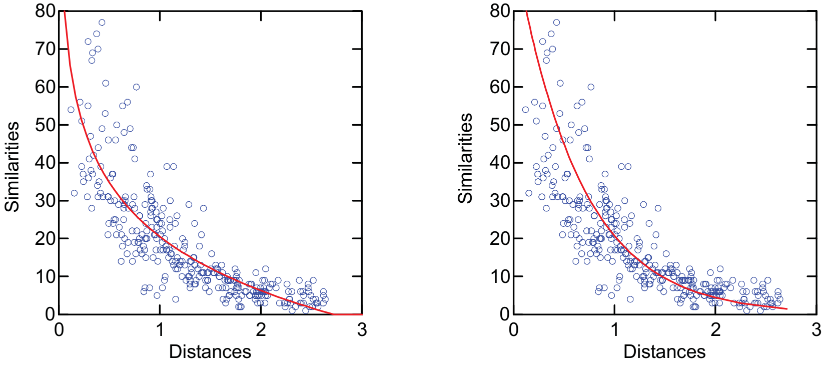


Figure 9.8 *Reflection and rotation exchanges range and domain*

Sometimes we transpose a graphic simply to make it more readable. This happens frequently with graphics that involve many categories on an axis. In English and other languages that are written horizontally, it is more convenient to orient a categorical axis vertically so that category names can be written easily. Figure 9.9 shows an example using the cities data from Chapter 5. Compare this to Figure 5.6.

DATA: p1980 = "1980"
DATA: p2000 = "2000"
SCALE: *log*(*dim*(2), *base*(10))
COORD: *transpose*(*dim*(1, 2)))
ELEMENT: *point*(*position*(city*(pop1980+pop2000)), *color*(p1980 + p2000))

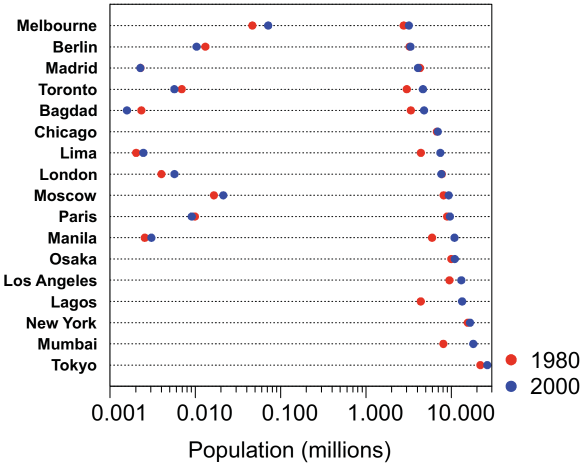


Figure 9.9 *Blended dot plot*

9.1.2 Similarity Transformations

A transformation g is a **similarity** if and only if there is a positive number r such that

$$d_2((g(x)), g(y)) = rd_1(x, y) \text{ for all } x, y \in S_1$$

Similarities on the plane involve isometries as well as dilation.

9.1.2.1 Dilation

Dilation sends polar coordinates (ρ, θ) to $(c\rho, \theta)$ or rectangular coordinates (x, y) to (cx, cy). The dinosaur in Figure 9.1 is shrunk by 50 percent. Figure 9.10 illustrates a dilatation for the countries data. The dilatation works like a photo magnifier or reducer. The multigraphics found in Chapter 11 require dilation transformations in order to size frames properly within an array of graphics.

Another application of dilation is for zooming in to reveal detail or zooming out to expose global structure. It is important to distinguish this **graphical zoom** from a **data zoom**. In a graphical zoom, we enlarge or reduce everything, including the axes and text. This operation is achieved through the dilation transformation. The graphical zoom is best thought of as an optical manipulation. It enables us to examine small areas of a graphic to analyze detailed structure. In a data zoom, however, we reduce or enlarge a frame by adjusting its bounds in data units. The physical size of the frame graphic (the box demarcated by the axes) and the size of the other graphics inside the frame (points, lines, bars) do not change. Instead, a data zoom-in subsets the data and a data zoom-out embeds the data in a wider range than usual. This data zoom operation has consequences for embedded graphics; they must be recalculated based on the subset of the data in the frame. A graphical zoom, by contrast, requires no recalculations; only the image is transformed. We will discuss the subtleties of this problem at length in Section 9.1.8.1 later in this chapter.

ELEMENT: *interval*(*position*(gov*birth))

COORD: *dilate*(*factor*(.5)))
ELEMENT: *interval*(*position*(gov*birth))

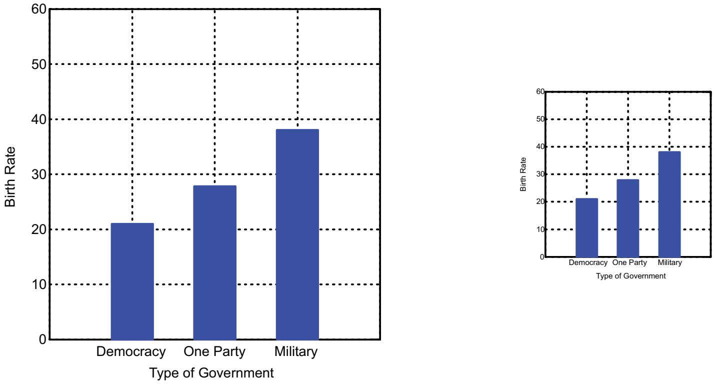


Figure 9.10 *Dilation transformation*

9.1.3 Affine Transformations

The n-tuple coordinate for a point in a space can be represented by the vector $\mathbf{x} = (x_1, \ldots , x_n)$. Vector notation allows us to express simply the affine class of transformations:

$$\mathbf{x}^* = \mathbf{xT} + \mathbf{c} \ ,$$

where $\mathbf{x}^*$, $\mathbf{x}$, and $\mathbf{c}$ are row vectors and $\mathbf{T}$ is an n by n transformation matrix. In this notation, $\mathbf{xT}$ is the image of $\mathbf{x}$. If $\mathbf{c} = \mathbf{0}$, we call this a **linear map**. The linear subset of the affine class includes rotation, reflection, and dilation, as well as stretch and shear. If $\mathbf{c} \neq \mathbf{0}$, we call it an **affine map**. This adds translation to these operations.

Let's first review the matrix form of the isometric and similarity transformations we have seen so far. Beginning with **T**, we can see that an identity transformation results from making **T** an identity matrix:

$$\mathbf{T} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Rotation involves the more general matrix

$$\mathbf{T} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$$

where θ is the angle of rotation.

Reflection involves an identity matrix with one or more diagonal elements signed negative, *e.g.*:

$$\mathbf{T} = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}$$

Any negative diagonal element will reflect the corresponding dimension of **x**. This particular **T** matrix reflects both dimensions.

Dilation involves a matrix of the form

$$\mathbf{T} = \begin{bmatrix} a & 0 \\ 0 & a \end{bmatrix}$$

where a is a real number.

Finally, translation involves a row vector of the form

$$\mathbf{c} = \begin{bmatrix} u & v \end{bmatrix}$$

where u and v are real numbers.

The affine class permits **T** to be a real matrix of the form

$$\mathbf{T} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

where a, b, c, and d are real numbers. Stretch and shear are produced by two types of this matrix.

9.1.3.1 Stretch

Stretch is the transformation that sends (x, y) to (ax, dy), so

$$\mathbf{T} = \begin{bmatrix} a & 0 \\ 0 & d \end{bmatrix} .$$

The dinosaur in Figure 9.1 is stretched by a factor of 2 on the second coordinate (vertical axis).

The stretch transformation varies the **aspect ratio** of a graphic. This is the ratio of the physical height to the physical width of the frame graphic. Often this ratio is chosen to make a graphic fit in a page layout or on a computer screen. This practice of convenience more often degrades than improves the accuracy of the perception of the information underlying the graphic. As Cleveland, McGill, and McGill (1988) have shown, many conventional prescriptions for aspect ratios in a plot ("make it square," "make it a Greek golden rectangle," etc.) have no empirical or theoretical justification. Instead, aspect ratios should be determined by the perceptual considerations of the content in the frame, namely, the shape of the graphics. Cleveland has shown that for line plots, perception of *relative* changes in slope is most accurate near a 45-degree orientation. (Sensitivity to *absolute* slope differences near threshold is highest near verticals or horizontals, but this is not relevant to this context.)

Figure 9.11 shows a stretch transformation of a time series graphic. The two graphics in this figure are rescalings of the sunspot data from Andrews and Herzberg (1985). The aspect ratio on the left makes a global absolute slope component of approximately 45 degrees. The extreme shear transformation on the right highlights local, high-frequency detail in the plot. It was computed by setting the median absolute slope of the line segments (approximately 500 in number) to unity, following a procedure outlined in Cleveland, McGill, and McGill (1988). The choice of aspect ratio must be guided by the information we wish to communicate. Sometimes more than one graphic is needed to highlight the important frequency components. In a dynamic graphics system, we could connect aspect ratio to a controller so that the user could explore these variations in real time. The simplest implementation of such a controller would be on the frame itself, so that we could resize by dragging vertically or horizontally.

ELEMENT: *line*(*position*(year*spots))

COORD: *stretch*(*factor*(2.0, 0.111))
ELEMENT: *line*(*position*(year*spots))

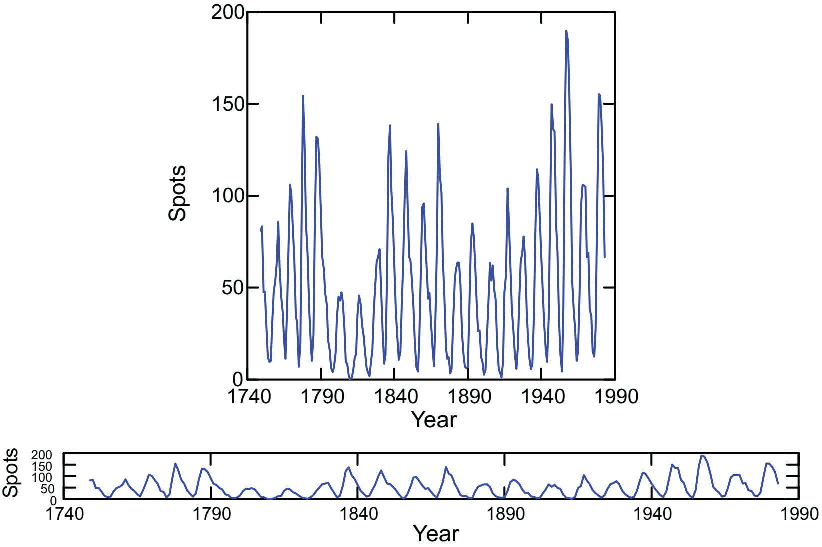


Figure 9.11 *Stretch transformation*

9.1.3.2 Shear

Shear is the transformation that sends (x, y) to ((ax+cy), (bx+dy)), so

$$\mathbf{T} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

The dinosaur in Figure 9.1 was produced by the matrix whose elements are $a = .96$, $b = .3$, $c = .3$, $d = .96$. Figure 9.12 shows the same shear transformation on a data graphic. The data are principal components of the sociometric ratings of US cities (Boyer and Savageau, 1996). The components have been rotated using an oblique rotation method called **oblimin** (Harman, 1976; Clarkson and Jennrich, 1988). This rotation fits basis vectors through bundles of component vectors without a restriction that the basis vectors be orthogonal. Computer packages typically graph oblique rotations in the manner of the left panel of Figure 9.12. Making the basis orthogonal helps us to see the separation between components, but conceals the dependency among oblique factors. The right panel shows a shear transformation applied to the graphic underlying the left pane. It makes clear the dependency between factors and thus encourages us not to make independent unitary interpretations.

Applications of oblique transformations in factor analysis sometimes show oblique axes running through the center of the vectors in an oblique cross. This graphic can be helpful for thinking of an oblique rotation as a fitting of a non-orthogonal basis under some loss function. As the right panel of Figure 9.12 shows, however, it can be helpful sometimes to place the axes at the edge of the plot to keep clutter outside the center of the frame, especially when there are many vectors.

```
DATA: zero = constant(0)
ELEMENT: edge(position(link.join(factor1*factor2 + zero*zero)),
         label(name))

DATA: zero = constant(0)
COORD: shear(matrix(0.96, 0.3, 0.3, 0.96))
ELEMENT: edge(position(link.join(factor1*factor2 + zero*zero)),
         label(name))
```

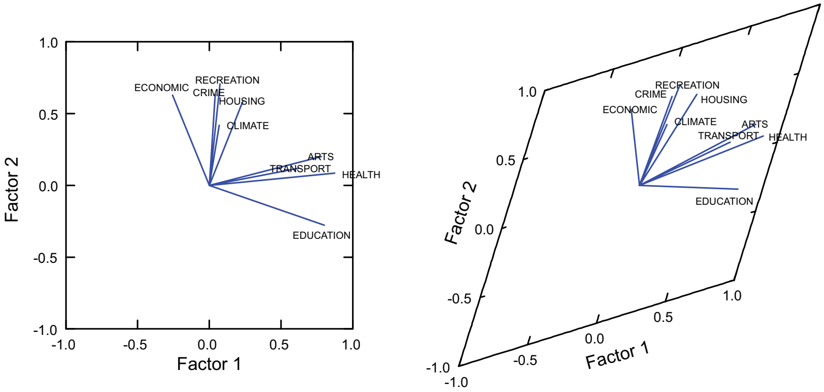


Figure 9.12 *Oblique factor rotation (shear)*

9.1.4 Planar Projections

A planar projection is the mapping of one plane to another by perspective projection from any point not lying on either. Figure 9.13 illustrates this mapping spatially. For every point in the image on the lower plane in the figure, there is a single corresponding point in the domain on the upper plane.

As the figure suggests, we may use a similar model to produce a perspective projection that creates 2D perspective views of 3D objects in computer graphics. Planar projections are more restrictive than 3D-to-2D projections, however. They share the *composition* behavior of other planar transformations. We can, in other words, project a projection and stay within the projective class. In the 3D-to-2D projection, it is possible to have more than one point in the domain for a single point in the image.

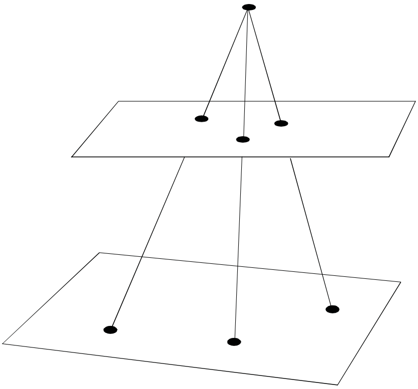


Figure 9.13 *Projection of one plane on another*

To notate projections, it is helpful to adopt **homogeneous coordinates**. We combine the **T** and **c** matrices into one general square matrix **A**:

$$\mathbf{A} = \begin{bmatrix} a & b & p \\ c & d & q \\ u & v & s \end{bmatrix}$$

The elements a, b, c, and d are from the **T** matrix and u and v are from the **c** vector that we used for affine transformations. The elements p, q, and s are for projection. To make this system work, we need to express **x** in homogeneous coordinates by augmenting our coordinate vector by one element:

$$\mathbf{x} = (x, y, h)$$

If $h = 1$, then our Cartesian coordinates are simply $x = x/h$ and $y = y/h$. This re-parameterization makes the general projective transformation

$$\mathbf{x}^* = \mathbf{xA}$$

This matrix equation produces the following homogeneous coordinates:

$$\mathbf{x}^* = ((ax + cy + u), (bx + dy + v), (px + qy + s))$$

If we renormalize after the transformation so that the third coordinate is unity, we can retrieve (x^*, y^*) as the Cartesian coordinates from the projection. To see what projection adds to the affine class, we should notice that the third column of **A** produces a different scaling of x and y, depending on their values. And because all the transformations are linear, the straightness of lines is preserved in the class.

9.1.4.1 Project

The projected dinosaur in Figure 9.1 was produced by the coordinate transformation

$$(x, y) \rightarrow (1/x, y/x) .$$

This projection involves a transformation used in the statistical technique called **weighted least squares**. We will present an example.

Figure 9.14 shows a scatterplot relating diastolic blood pressure to age. Neter, Wasserman, and Kutner (1990) devised this dataset to illustrate a popular technique for dealing with unequal variances (**heteroscedasticity**) in linear regression. As the left panel of the figure shows, the variation around the fitted line is larger for older people than for younger. This violates an assumption of equal variances (**homoscedasticity**) needed for conventional tests of hypotheses involving the regression coefficient relating blood pressure and age. The panel on the right of the figure shows the distribution of data we would like to see for testing these hypotheses appropriately.

One simple approach statisticians have used to ameliorate this problem is to transform the data before fitting the line. This transformation involves weighting the values of age and blood pressure according to the corresponding blood pressure value. We start by noticing that the envelope of the variation about the line in the left panel is like a fan with straight sides. Assuming we have a random sample of values, this pattern suggests that the population variance of blood pressure at a given level of age increases linearly with age, *i.e.,*

$$\sigma_i^2 = \sigma^2 x_i$$

Neter, Wasserman, and Kutner illustrate some statistical and graphical approaches to confirming this supposition by examining the distribution of the data. If we believe the data fit this model, then we can attempt to equalize the conditional variances by dividing both variables by the value of age for each pair. In other words, we transform a model we are almost certain is false to one that appears to be true:

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i \quad \rightarrow \quad \frac{y_i}{x_i} = \beta_0' + \frac{\beta_1'}{x_i} + \varepsilon_i'$$

The graphic in the right panel of Figure 9.14 is a fit of this latter model. It is plotted using the transformed data (**age/bp** and **1/bp**). The result resembles the classic textbook example of linear regression. The fan shape has been transformed into a stripe; the variance of *y* appears to be constant across all values of *x*. The regression line runs through the center of this stripe, as it should.

ELEMENT: *line*(*position*(*smooth.linear*(age*bp))
ELEMENT: *point*(*position*(age*bp))

TRANS: inverseage = *inverse*(age)
TRANS: bpbyage = *ratio*(bp, age)
ELEMENT: *line*(*position*(*smooth.linear*(inverseage*bpbyage))
ELEMENT: *point*(*position*(inverseage*bpbyage))

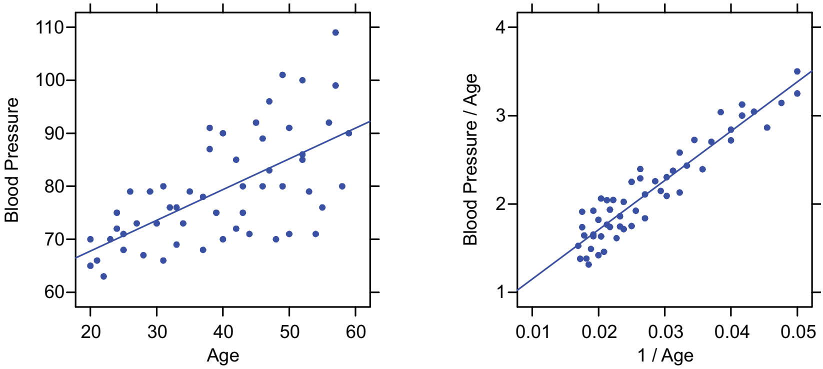


Figure 9.14 *Weighted least squares via transformation of variables*

A problem students often have with this approach is understanding what was done to the data. It is difficult to relate the plot in the right panel to the one in the left. The cloud is reversed horizontally and the scale values are not in the original metric. If we parameterize this model as a projection, however, we can produce a graphic in the original metric transformed. Note that the weighted model can be viewed as the coordinate transformation

$$(x, y) \rightarrow (1/x, y/x)$$

The projection matrix for expressing this transformation is

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

This takes $\mathbf{x}$ in homogeneous coordinates to $\mathbf{x}^* = (1, y, x)$, which produces the result we want in Cartesian coordinates after dividing through by x. Figure 9.15 shows the result. We have included the **A** matrix in the parameter list as the array *a*[]. Now we see explicitly that the transformation reflects the x axis because of inversion and fans out blood pressure for younger ages. Notice that the regression line passes through the same observations in all these plots. The ordinary fit of the line under heteroscedasticity is unbiased. It is the *variances* we have adjusted.

Data-modeling statisticians have devised more elaborate methods to deal with these problems, including iterative reweighting, maximum likelihood, and generalized least squares. While Figure 9.15 may look unfamiliar even to some experts at first glance, however, it is no different in spirit from polar coordinates, log scales, and other coordinate transformations we routinely use to produce more familiar scientific graphics. Coordinate transformations allow us to see what was done to our data.

COORD: *project*(*matrix*(..))
ELEMENT: *line*(*position*(*smooth.linear*(age*bp)))
ELEMENT: *point*(*position(*age*bp))

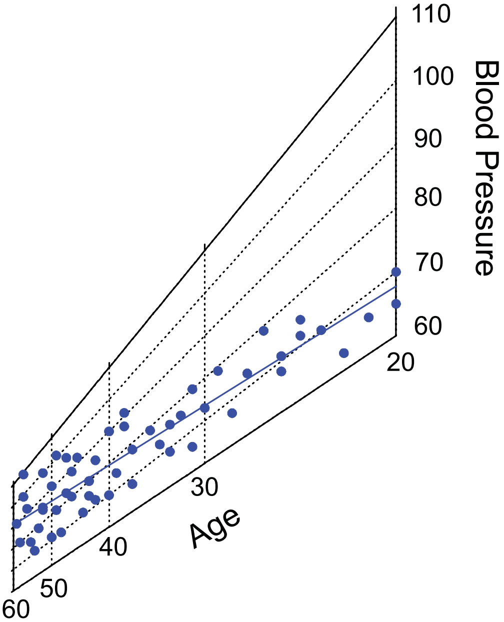


Figure 9.15 *Projection*

Figure 9.15 highlights an interpretive problem with transformations. Statistical modeling via transformation or weighting is a widely used method for dealing with unusual distributions. The problem is that we most often want to state our conclusions in terms of the original variables. Sometimes we can escape the problem by keeping the transformation and not inverting it to explain our result. Miles per gallon, for example, can be transformed to gallons per mile in order to make it more normally distributed. This form of the index may be more useful anyway. Paul Velleman (Velleman and Wilkinson, 1994) has suggested this transformation for routine use and has noted that Europeans use this form (liters per kilometer), presumably because it enables travelers to compute more easily their gas needs on a trip. If we transform, do statistics, and invert our transformation, however, we must be extremely careful in our interpretations of the result, particularly when random error is included in our models. We will examine this problem further in Section 9.1.8.

9.1.5 Conformal Mappings

We need to generalize our coordinates once more in order to move to the next level of the planar transformation hierarchy. By working on the complex plane, we can define functions that would be messy or difficult to understand in the real domain. A complex number $z = x + iy$ may be represented by a vector $\mathbf{z}$ on the complex plane whose coordinates are $Re(z) = x$ and $Im(z) = y$. Coordinate transformations on (x, y) can then be expressed in the form

$$w = f(z) = u(z) + iv(z)$$

where $u(z)$ and $v(z)$ are real functions of z and w is the image point of z under f.

First, as we did with the affine and projective classes, let us dress the child of the conformal class in the clothes of this new notation. Similarity transformations can be expressed in the complex formula

$$w = az + b$$

where w, a, b, and z are all complex. We can see this by noting that

$$(a_1 + ia_2)(x + iy) = (a_1x - a_2y) + i(a_2x + a_1y) + (b_1 + ib_2)$$

which is the same set of operations involved in the similarity subclass of the projective transformation

$$\mathbf{xA} = (x, y, 1) \cdot \begin{bmatrix} a_1 & a_2 & 0 \\ -a_2 & a_1 & 0 \\ b_1 & b_2 & 1 \end{bmatrix}$$

The projective matrix notation tells us that the complex constant b is involved in translation and the complex constant a is involved in rotation and dilation of the plane represented in z, since the submatrix

$$\begin{bmatrix} a_1 & a_2 \\ -a_2 & a_1 \end{bmatrix} = r \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} ,$$

where r is a real number. There is another way to show the rotational role of the complex constant a. Euler's formula

$$e^{i\theta} = \cos\theta + i\sin\theta$$

locates a point on the unit circle at angle θ on the complex plane. It tells us that any complex number can be expressed as

$$z = re^{i\theta} .$$

We can thus re-express the complex constant a and define a similarity transformation as

$$w = re^{i\theta}z + b ,$$

which rotates z by θ and dilatates it by r.

A conformal mapping adds a peculiar geometric characteristic to a similarity transformation: local angles (at the intersection of two curves) are preserved, but straight lines may become curves. A planar mapping is conformal if every point on the plane is transformed so that all possible infinitesimal vectors emanating from that point are rotated and dilated by the same amount in the image. This local rotation and dilation means that very small squares remain squares in the image, but large squares can be distorted considerably. The paradoxical beauty of this transformation is that locally it looks like a similarity but globally it looks like a nonlinear warping.

The conformal dinosaur in Figure 9.1 was produced by the transformation

$$w = \frac{1-z}{1+z}$$

Figure 9.16 shows several examples of conformal mappings of a chessboard pattern of square tiles. We have set the domain of these mappings to the interval $[-\pi,+\pi]$ on both x and y. Several of these transformations are a subclass of the Möbius transformation

$$w = \frac{az+b}{cz+d} ,$$

where all the constants and variables are complex. This transformation has inspired a variety of basic applications in physics, fluid dynamics, electromagnetic fields, and other areas. Needham (1997) offers a glimpse into this world from a geometric perspective and illustrates its application to vector flows and other graphics in physics. Running graphics through variations of this transformation to see what they look like can be addictive.

```
COORD: conform(matrix(..))
ELEMENT: polygon(position(x*y))
```

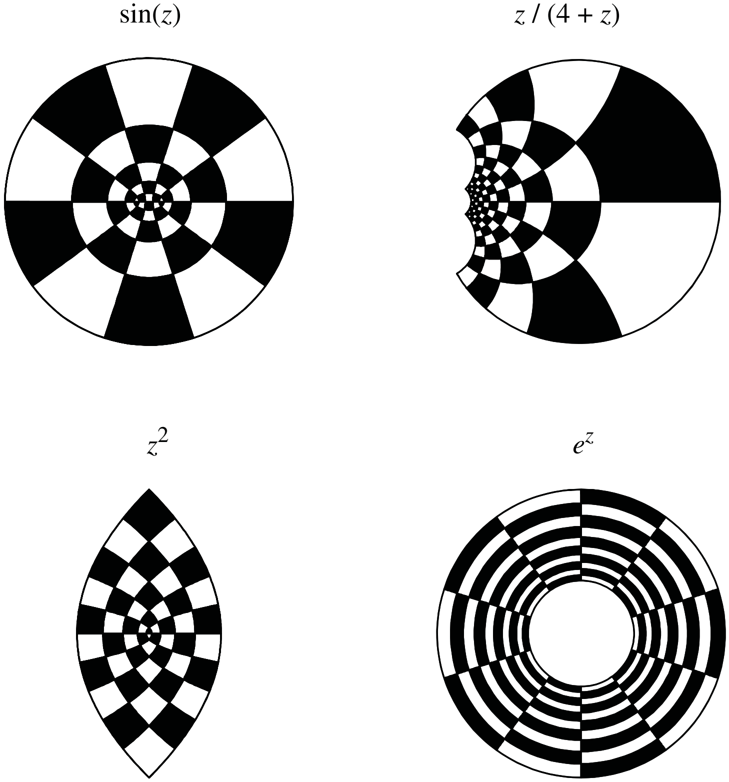


***Figure 9.16** Conformal mappings of a chess board*

9.1.5.1 Conform

Figure 9.16 may seem a looking-glass world for most of the graphics in Chapter 8, but there are practical applications of planar conformal maps. We will omit the vector flow diagrams used in physics and instead show a simple graphic from the field of meteorology. Figure 9.17 is a graphic relating humidity to wind direction for a ground-level site. The data comprise hourly meteorological measurements over a year at the Greenland Humboldt automatic weather station operated by NASA and NSF. These measurements are part of the Greenland Climate Network (GC-Net) sponsored by these federal agencies. Data like these are available at numerous sites on the World Wide Web.

COORD: *conform*(*matrix*(..))
ELEMENT: *line*(*position*(*smooth.loess*(direction*humidity)))
ELEMENT: *point*(*position*(direction*humidity))

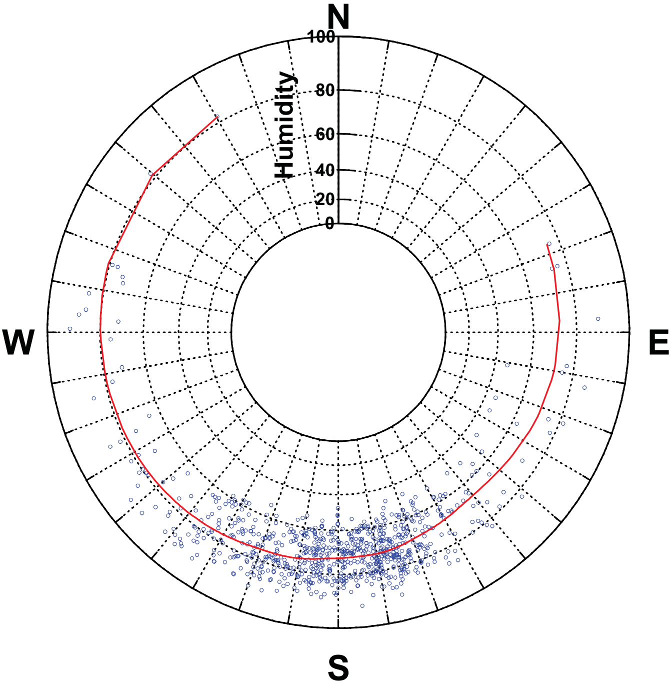


Figure 9.17 *Conformal mapping*

On first glance, Figure 9.17 resembles a polar wind chart with a *loess* smoother. The wind direction is represented by a polar angle from 0 to 360 degrees and humidity is represented by the outer section of the radius. The conformal mapping (e^z) underlying the graphic is not a polar transformation, however. Notice that the radial dimension (humidity) is exponentially spaced. High humidities are spaced farther apart than low. This nonlinear scale transforms the negatively skewed distribution of humidity to a more normal one. The other feature distinguishing this from a polar chart is that zero humidity is not mapped to the center of the circle. In the complex exponential transformation, the center of the circle corresponds to negative infinity. This graphic is not a pie. Its center is a black hole.

The *loess* robust smooth shows that humidities tend to be lower for winds coming from a south-easterly direction. The highest humidities prevail in the west. The cloud reveals that there are few wind measurements over the year in the northerly sectors.

9.1.6 Polar Coordinates

If (u, v) represents the polar coordinates (ρ, θ) of a point, then the polar coordinate function $P(u, v) \rightarrow (x, y)$ corresponds to the case $u = \rho$, $v = \theta$, where $x = \rho\cos\theta$ and $y = \rho\sin\theta$. There are numerous important applications of the polar transformation in graphics. Mathematicians and scientists usually deal with polar coordinates measured on the real numbers such that one revolution corresponds to an interval of 2π radians.

Figure 9.18 shows an example for the cosine function with period $\pi/3$. The range of this graph is on the interval $[-1, +1]$ and the domain is on the interval $(-\infty, \infty)$. Our graphic shows a domain only on the interval $[0, 2\pi)$. This means that we can see only one cosine curve at each petal even though there would be an infinite number of these curves at each petal if we allowed the domain to cycle more than once around the circle (see the *cycle* parameter in Section 6.2.2). There are not many good ways to fix this representation problem. We could plot the function in cylindrical coordinates (see Section 9.3.3) with the cylindrical axis assigned to the same variable as θ.

We could treat polar coordinates as an exception to the way all other scales are handled in this system. That is, we could interpret angular values absolutely as radians. This would make sense if all our graphics were mathematical or engineering applications involving radians. We have chosen not to do this, however, so that we can hide scaling details when doing coordinate conversions. This makes it easy, for example, to represent yearly time in polar coordinates. In the polar coordinate conversion, therefore, we align 0 radians with the minimum scale value in data units (degrees, radians, proportions, etc.) and 2π radians with the maximum. The *cycle* parameter, together with *min* and *max* parameters in the scale functions allows us to create polar graphs with more than one revolution if we wish.

A note about the *polar*() function in the COORD specification: the arguments are reversed from the order given for $P(u,v)$ above. That is, the first dimension is taken to be the domain, which is assigned to θ. The second dimension is taken to be the range, which is assigned to ρ. This is in keeping with the general order of algebraic specifications within a frame: the factors are ordered as $domain_1$, $domain_2$, ... , *range*.

As we will show in this section, applications of polar coordinates range far beyond technical and mathematical graphics. The polar transformation is useful whenever data lend themselves to circular arrangements. This includes directional data (vector wind, compass bearings), rotational data (defects on disk drives), astronomical time (daily, monthly, annual), periodic waveforms (radio signals), and proportions (the humble pie chart). Sometimes, circular arrangements offer simpler parameterizations or structures for making sense of a phenomenon. Polar models of facial expressions, for example, provide the most parsimonious summaries of behavioral observations (see Figure 10.46).

DATA: x = *iter*(0, 6.28, 0.01)
TRANS: y = *cos*(6*x)
SCALE: *interval*(*dim*(1), *min*(0), *max*(6.28), *format*(*format.pi*)
COORD: *polar*()
ELEMENT: *line*(*position*(x*y))

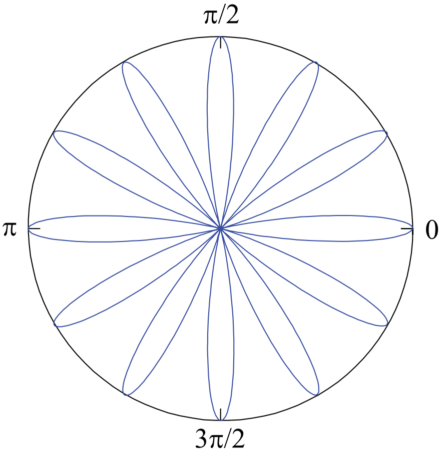


Figure 9.18 *Polar cosine*

Polar plots are often a means to a geometric end. In these cases, we are happy to limit our domain to one revolution because our goal is to represent objects in a circular arrangement. Figure 9.19 shows a polar dinosaur. The coordinates for the tail-to-head dimension have been scaled to vary between 0 and 2π and we have oriented the graphic to make 0 at the top by transposing it after the polar transformation. This dinosaur is hibernating.

COORD: *transpose*(*polar*())
ELEMENT: *edge*(*position*(*link.mst*(x*y)))

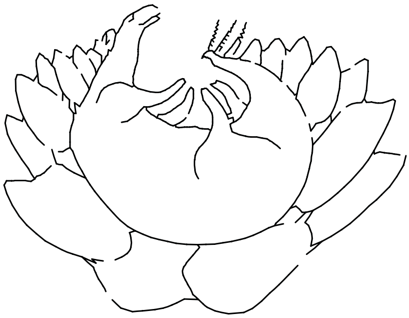


Figure 9.19 *Polar dinosaur*

The specification for the graphic reveals how we created these dinosaur transformations. Instead of working with bitmaps, we digitized the outlines into (*x, y*) coordinates computed at closely spaced intervals. Then we added the minimum spanning tree (MST) version of the *link* graph to "connect the dots." Occasional irregularities and gaps are due to the resolution of the dot spacing we chose and the MST algorithm's compulsion to go to the nearest adjacent dot.

Any circular directional variable is an obvious candidate for polar representation. Compass direction, for example, varies between 0 and 360 degrees. There are other reasons to send graphics through a polar transformation, however. The most popular application in graphics is the pie chart, which we will discuss first. Pies lend themselves to proportion-of-whole representations for valid visual-processing reasons. Polar coordinates also provide a useful medium for fitting a lot of information in a small space. Polar trees allow the highest resolution in the area of their leaves, which makes them popular among geneticists and others who must cluster large sets of objects.

There are several different *polar* coordinate methods needed for statistical graphics. The conventional *polar* function requires two arguments. We use this one for embedding graphics in two-dimensional frames, such as in scatterplots and mathematical graphs of polar functions. The other polar functions take only one argument. The *polar.theta* function assigns its argument to θ and sets ρ to unity. The *polar.rho* function assigns its argument to ρ and sets θ to unity. The *polar.rho.plus* function works like *polar.rho* except 1 is added to ρ before the transformation. This pushes the range outside the unit circle. This function is used for producing rose diagrams (see Section 9.1.6.3).

9.1.6.1 Polar.theta

As we have seen in Chapter 2, a pie chart is a stacked bar in polar coordinates. The *polar.theta* function assigns its only argument (the position attribute) to θ, and assigns the radius (ρ) to a constant that determines the size of the pie.

Figure 9.20 shows a pie chart for the females in the ACLU data. Notice that the labels for the slices are not part of an axis, scale, or other guide. They are attached to each slice through the *label*() aesthetic function. We will discuss this further in Chapter 12, where we will cover guides. The appearance of a graphic can sometimes deceive us. We must analyze text carefully to determine whether it is part of a guide or part of a graphic.

It seems odd to go through a function to produce the most popular chart of all. Simplicity is in the eye of the beholder, however. Once we learn to bake pies in round and square pans, we can graduate to other shapes, as the next example will show.

COORD: *polar.theta*()
ELEMENT: *interval.stack*(*position*(*summary.proportion*(response)),
label(response), *color*(response))

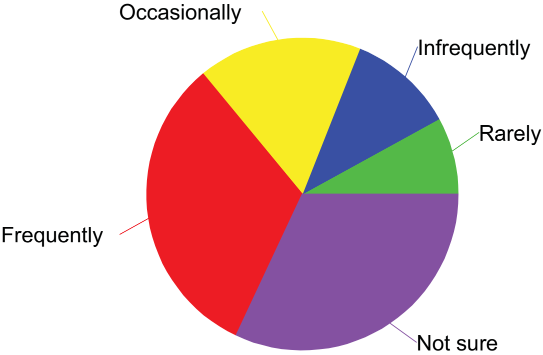


Figure 9.20 *Pie chart*

9.1.6.2 Polar.rho

A circular pie chart is a variation on the divided bar in polar coordinates. It bakes a pie in a bundt cake pan. The *polar.rho* projection assigns the position attribute to ρ and wraps θ around the circle.

Figure 9.21 shows an example of the chart for the bias data. This graphic is occasionally used by newspaper and magazine marketing departments to represent spheres of circulation among different readerships. The perceptual problem with the graphic is that areas are confounded with the radial variable.

COORD: *polar.rho*()
ELEMENT: *interval.stack*(*position*(*summary.proportion*(response)),
label(response), *color*(response))

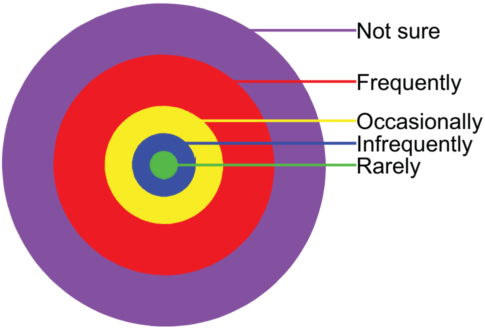


Figure 9.21 *Circular pie chart*

9.1.6.3 Polar.plus

The *polar.plus* and *polar.rho.plus* functions are for rendering graphs along the circumference of a circle. This is needed when essential features of graphics get jammed together near the center of the polar domain. Graphics in this form are seen most frequently in ecology and spatial statistics applications, where circular distributions are analyzed (*e.g.*, Upton and Fingleton, 1989). Figure 9.22 shows a polar dot plot and rose histogram for the wind direction data used in Figure 9.17.

```
COORD: polar.rho.plus()
ELEMENT: point.dodge(position(bin.dot(direction)))

COORD: polar.rho.plus()
ELEMENT: interval(position(summary.count(bin.rect(direction))))
```

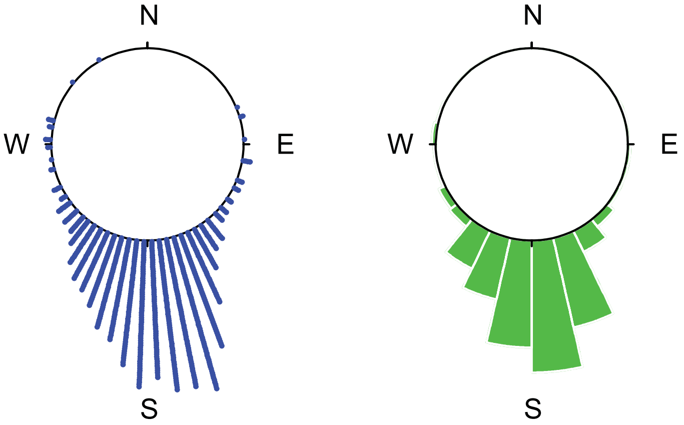


Figure 9.22 *Polar dot plot and histogram*

The wind rose (*rosa ventorum*) has been drawn for centuries by cartographers. Predating the compass rose, it appeared on navigational charts as early as the 13th century. Wainer (1995) shows an example by Léon Lalanne from 1830. Lalanne drew by hand his estimates of prevailing ocean wind strength and direction.

We can construct a similar graphic by computer today. Figure 9.23 shows a statistically based smoothing of the Greenland wind direction data. We have drawn the axes to resemble Lalanne's chart. This rose results from embedding a kernel smoother *density* graphic in a polar frame. Because we used the *polar* function in this figure, much of the detail near the center is lost.

This graphic would benefit from a nonlinear transformation of the radial scale. Nevertheless, it is interesting that coordinate-based applications of relatively recent technology (kernel smoothing) can benefit from clever cartographic insights of the past. One thing to note: the radial distance from the center represents the proportion of measured times that the wind was blowing in a given direction, not the strength of the wind itself.

COORD: *polar*(*transpose*())
ELEMENT: *line*(*position*(*smooth.density.kernel.epanechnikov*(direction)))

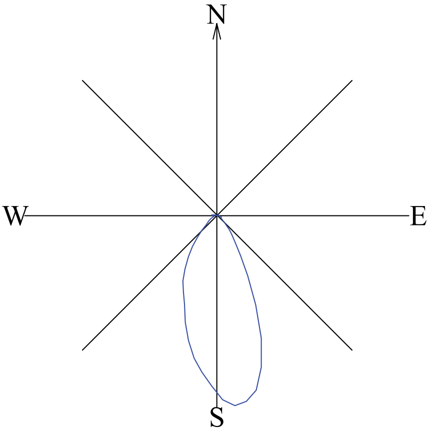


Figure 9.23 *Rose kernel density*

9.1.6.4 Polar

The standard *polar* two-argument function has numerous applications. Figure 9.24 shows a Nightingale rose chart and the bars from which it is constructed. We have used the female data from the ACLS dataset in Chapter 2. The left panel shows the rectangular coordinate graphic as an ordinary bar chart. The bar thicknesses are set to full width so that the bars touch each other. This makes the segments in the polar graphic in the right panel span the entire circle. Notice that, unlike Figure 9.20, the polar angles are constant for every bar. While this figure seems to be a type of pie chart, it is in fact a fundamentally different graphic based on multiple bars instead of a partitioned (stacked) bar. This chart derives its name from a graphic devised by Florence Nightingale (Wainer, 1995). Attractive as it may be, the polar bar chart confounds area with radius. Square rooting the radii only partly ameliorates this confounding.

ELEMENT: *interval*(*position*(response), *color*(response), *size*(2))

COORD: *polar*()
ELEMENT: *interval*(*position*(response), *color*(response), *size*(2))

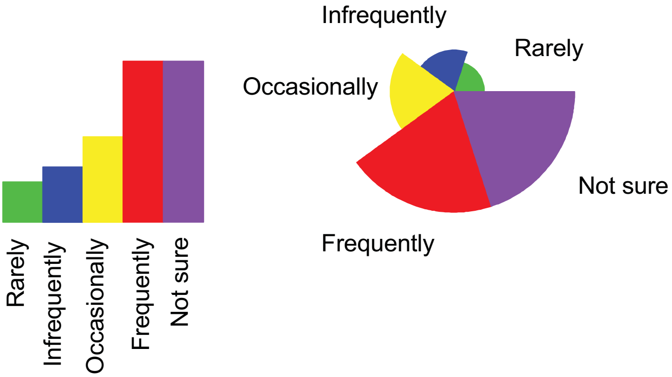


Figure 9.24 *Bar graphic and Nightingale rose*

Figure 9.25 shows a reflection of the Nightingale rose. We have included this rather atrocious chart to illustrate further the composition of coordinate functions. Notice that the *reflect* function reverses the range scale before mapping to polar coordinates.

COORD: *polar*(*reflect*(*dim*(2)))
ELEMENT: *interval*(*position*(response), *color*(response), *size*(2))

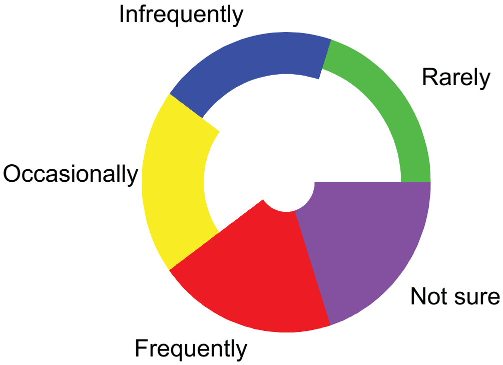


Figure 9.25 *Reflected Nightingale rose*

Figure 9.26 shows a polar transformation of a dual stacked bar. As in the previous example, polar coordinates do not help to elucidate the structure, but this example illustrates the roles the dimensions play in the coordinate transformation. Compare these results to the un-transposed version in Figure 8.19. What would the right panel look like if we did not use *transpose*()?

COORD: *transpose*()
ELEMENT: *interval.stack*(*position*(*summary.proportion*(gender*response)),
color(response), *label*(response))

COORD: *polar*(*transpose*())
ELEMENT: *interval.stack*(*position*(*summary.proportion*(gender*response)),
color(response), *label*(response))

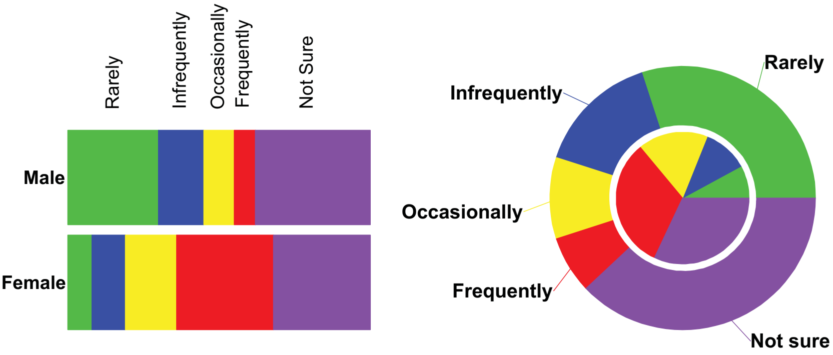


Figure 9.26 *Polar divided bar of ACLS data*

Polar coordinates can be used to enhance detail in areas where it is important. When there are many leaves, or terminal nodes, a cluster tree becomes unwieldy. Placing the tree in polar coordinates leaves room for displaying more leaves. Figure 9.27 shows a polar tree on the cities data. Notice that straight lines become curves in the polar representation, although we could take special steps to prevent this behavior if we wished to design the software that way.

Polar trees are often used by biologists to display results of cluster analyses or large genetic trees. Recently, computer scientists have taken an interest in similar layouts for displaying large trees in browsers and other windows. The fish-eye coordinate transformation (see Section 9.1.8) can be used to show more detail in a region of the tree. Lamping, Rao, and Pirolli (1995) describe a hyperbolic tree transformation for embedding large trees within a circle. Munzner (1997) extended this model to 3D.

COORD: *polar*()
ELEMENT: *edge*(*position*(*link.join*(xnode*ynode),*label*(city)))

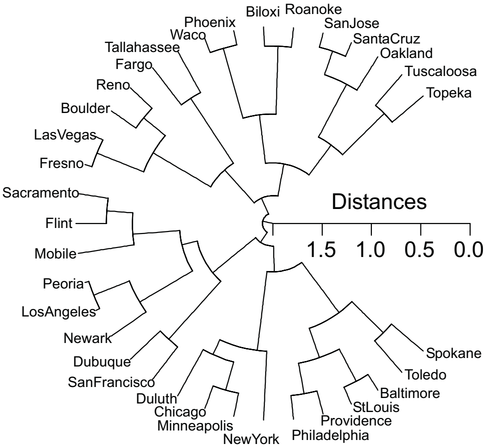


Figure 9.27 *Polar cluster tree*

Polar coordinates have many uses for displaying seasonal data. Time series are usually displayed linearly. This is the format in which we are accustomed to viewing stock market data, for example. This format facilitates scale look-up and decoding of prices. When seasonal characteristics are of interest, however, strip charts of annual time series are less effective as a display. Figure 9.28 shows a three-year time series of the daily closing price of the stock of SPSS. We have omitted the stock price scale because the focus of interest here is the trend. We have also specified a three-year cycle for the time scale; see Chapter 6 for information on the format of the *time*() scale function. Polar coordinates make it easy to compare local features across seasons, especially at the end/beginning of years. Moreover, the plot enables a global assessment not possible with strip charts of time series: if the stock is everywhere increasing over its previous season's price, the *line* graphic will appear as a spiral. Seasonal comparison is most prevalent in the language of companies when they present quarterly results to the brokers who set earnings numbers. Wadsworth *et al.* (1986) have made this same observation in the context of process control monitoring.

Carlis and Konstan (1998) propose polar visualizations for periodic series, although they employ a spiral for anchoring the data (see Figure 9.31). This method can be effective for series where the inspection of trend components is less critical than seasonal. Stock series would not be appropriate for their method, but they present other series for which it makes sense. Tufte (1983) shows historical examples.

SCALE: *time*(*dim*(1), *cycle*(3))
GUIDE*: axis*(*dim*(1), *format*(*format.month*))
COORD: *polar*()
ELEMENT: *line*(*position*(date*price))

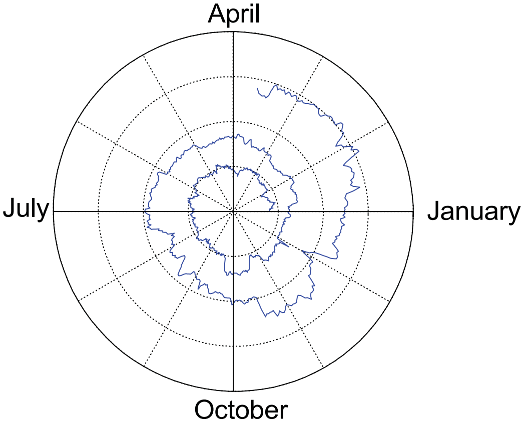


Figure 9.28 *Polar time series (SPSS stock)*

A favorite graphic among quality control engineers, particularly in Japan, is the **radar plot**. Like the spiral time series, this display facilitates rapid discrimination of a single important feature: is an ordered set of numbers pairwise larger than another ordered set of the same size? The radar plot makes this relation easy to detect because the convex hull of the larger set completely contains the hull of the smaller. This display generalizes easily to more than two sets of numbers. General Motors used this graphic in 1997 advertising to demonstrate that the envelope of performance features for its C5 Corvette — handling, economy, acceleration, safety, etc. — substantially exceeded the numbers for its previous model. Figure 9.29 shows this type of plot for the US weather data. The two polar profiles show average summer and winter temperatures for eight regions of the US. Not surprisingly, average summer temperatures are higher than winter in all regions. Some radar plots involve different scales for each variable in the plot. These are instances of polar parallel coordinate plots rather than polar profiles. See Figure 9.61 for an example.

There is one technical detail concerning the shape of Figure 9.29. Why is the outer edge of the *area* graphic a set of straight lines instead of arcs? The answer has to do with what is being measured. Since region is a categorical variable, the line segments linking regions are not in a metric region of the graph. That is, the segments of the domain between regions are not measurable and thus the straight lines or edges linking them are arbitrary and perhaps not subject to geometric transformation. There is one other problem with the grammatical specification of this figure. Can you spot it? Undo the polar transformation and think about the domain of the plot. We cheated.

DATA: s = *string*("Summer")
DATA: w = *string*("Winter")
COORD: *polar*()
ELEMENT: *area*(*position*(region*(summer+winter)), *color*(s+w))

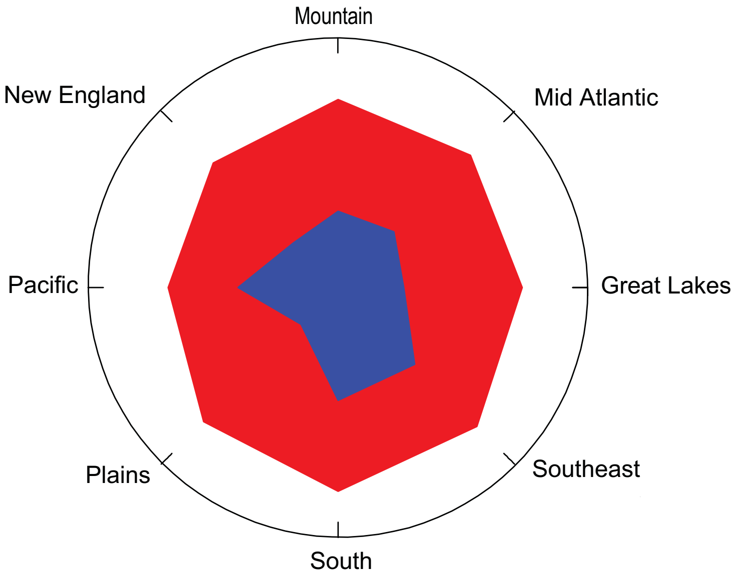


Figure 9.29 *Radar plot*

The polar domain offers numerous opportunities for contouring and other graphs that represent additional dimensions. Figure 9.30 shows a transposed polar plot of barometric pressure by wind speed by wind direction. The pressure values have been smoothed by distance-weighted least squares onto the surface represented by wind speed and direction. The contours have been colored by level of barometric pressure. Lower pressures (blue) correspond to southeasterly high speed winds, while higher pressures (red) correspond to westerly low-speed winds. The plotted speeds are the highest gust recorded each hour, so they have a ceiling, as evidenced by the uniform boundary of values in the cloud near 11 meters per second at the bottom of the circle.

When contouring in the polar domain, we must be careful in the way we compute contours. If we contour in rectangular coordinates and then transform the contours, there will be discontinuities at the boundaries, especially with polynomial and related smoothers. A similar problem exists when contouring on the sphere. To solve the problem, we must do our calculations in the transformed metric.

```
COORD: transpose(dim(1, 2), polar())
ELEMENT: point(position(direction*speed))
ELEMENT: contour(position(
                    smooth.quadratic.cauchy(direction*speed*pressure)),
                    color.hue())
```

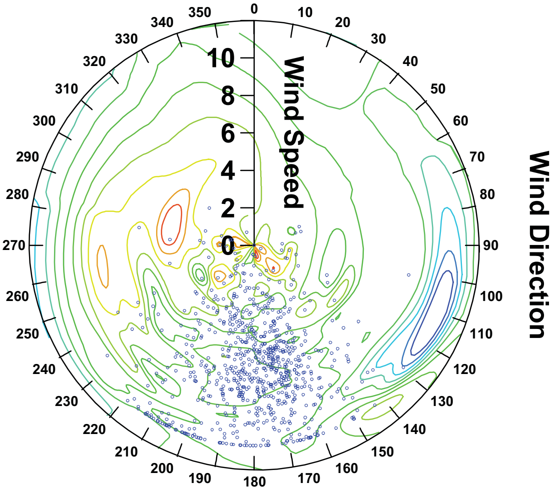


Figure 9.30 *Barometric pressure (contours) by wind speed by direction*

9.1.7 Inversion

Complex inversion is a turning inside-out of the plane. It is most easily understood by considering the transformation $(\rho, \theta) \rightarrow (1/\rho, \theta)$ in polar coordinates, where radii are converted to their reciprocals. More generally, we define the **complex conjugate** of $z = x + iy$ to be $\bar{z} = x - iy$. Then the transformation of the complex plane $z \rightarrow 1/\bar{z}$ is a geometric inversion. The invariance of this transformation is that circles remain circles (notice that θ is unchanged and ρ is simply inverted in the polar representation of the transformation).

Figure 9.31 shows an inversion of a spiral. Notice that the points outside the unit circle are turned in and the points inside are turned out. This coordinate transformation would be useful for highly skewed data or whenever we wish to expose the detail in the center of the polar coordinate world. Inverting an image suggests some interesting possibilities. Inverting a face is best reserved for Halloween.

DATA: x = *iter*(0., 18.86, 1000)
TRANS: y = x / *8* – *.2*
SCALE: *linear*(*dim*(1), *min*(0), *max*(18.86), *base*(10), *cycle*(3))
COORD: *polar*()
ELEMENT: *point*(*position*(x*y), *color.hue*(x))

DATA: x = *iter*(0., 18.86, 1000)
TRANS: y = x / *8* – *.2*
SCALE: *linear*(*dim*(1), *min*(0), *max*(18.86), *base*(10), *cycle*(3))
COORD: *inverse*(*polar*())
ELEMENT: *point*(*position*(x*y), *color.hue*(x))

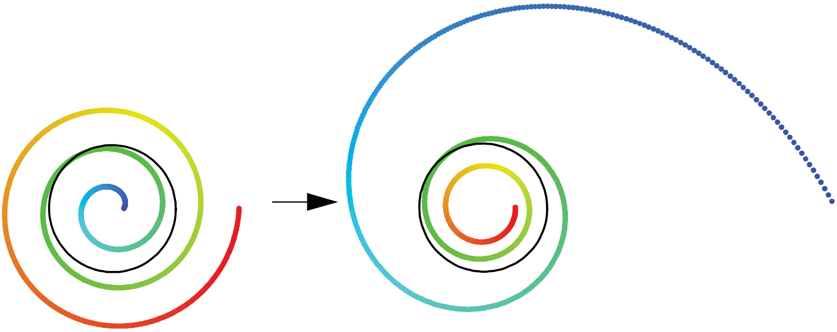


Figure 9.31 *Inversion of a spiral*

9.1.8 Bendings

This section covers coordinate transformations that bend the plane like a sheet of plastic. Bendings stretch the plane along *x* or *y* or in both directions. We will first discuss single bending, most frequently used to straighten curves and linearize scales. Then we will discuss double bends, used to compress or dilate sections of the plane. Bendings involve the continuous planar transformation

$$(x, y) \rightarrow (f(x), g(y)) = (u, v)$$

One way to visualize this class of transformations is to think of a frame graphic with axes and grid lines and note that the grid lines remain parallel after transformation and the axes remain perpendicular. Bendings involve no shear. This is because *u* and *v* depend only on *x* and *y*, respectively.

9.1.8.1 Inverting a Scale Transformation

The logarithmic planar transformation is

$$(x, y) \rightarrow (log(x), y) \text{ , or}$$

$$(x, y) \rightarrow (x, log(y)) \text{ , or}$$

$$(x, y) \rightarrow (log(x), log(y))$$

where the *log* function is

$$log: x \rightarrow ln(x)/ln(k) \;\; , \quad k > 0 \, , \;\; x > 0$$

Many graphs use aggregation functions in their position computations. However, the *mean* of the *log* is not the *log* of the *mean*. Nor is the *sd* of the *log* the *log* of the *sd*. In general, any aggregation must be computed *after* nonlinear transformations in order for the results to be statistically correct. Thus, we should not use a *log* coordinate transformation to deal with skewness in our data. Instead, we should use a *log* scale to address this problem. We may *antilog* our results, however, if we wish to view them in the original metric. Figure 9.32 shows how this works.

SCALE: *log*(*dim*(2), *max*(2000))
COORD: *exp*(*dim*(2))
ELEMENT: *point*(*position(summary.mean*(exposure*brainweight)))
ELEMENT: *interval*(*position*(*region.spread.sd*(exposure*brainweight)),
color(“red”))

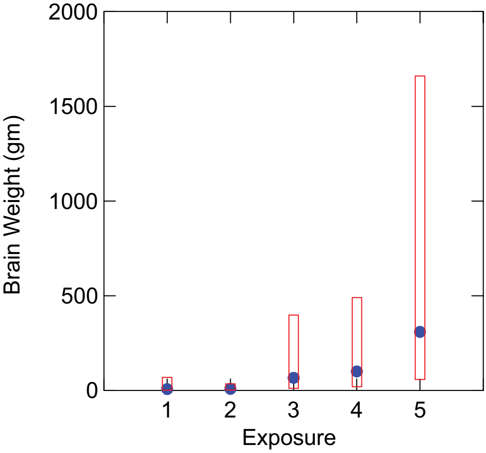


Figure 9.32 *Log error bars on raw scale*

In Figure 9.32 we have inverse-transformed the *log* scale to the raw metric by exponentiating the graphic of the sleep data shown in Figure 6.7. We have restricted the range to 2,000 by leaving the log *base* parameter to missing (default natural log) and setting the *max* to 2,000. Now the standard deviation bars are asymmetrical because they have been computed on the logged data. This may seem odd until we consider that the raw distributions are asymmetric, so conventional *mean* and *sd* calculations make sense only after *log* transformation of the brain weight values. Computing this way, and assuming the logged values are normally distributed, we would expect the bars to cover approximately 68 percent of the values (the central two-standard deviation spread for a normal distribution). That would not be true if the calculations were done on the raw data.

Figure 9.32 is misleading. The two largest brain weights in the dataset (5,712 and 4,603) belong to elephants, yet the vertical scale suggests that the data range between 0 and 2,000 grams. One might argue that we are faithfully representing the standard deviations and means, but we would reply that this misrepresents the data. It also violates our fundamental rule for a statistical graphics system: the frame should always cover the data on which the graphs are based. The range of any dimension should not depend on the graphs that live in the frame. Graph dependence is dangerous.

Defining the range independently of summary functions means that the displayed range covers *all* the data included in the calculation of *all* the means. If the data are positively skewed, as in this example, the points will tend to congregate at the bottom of the display unless the data are transformed to be roughly symmetric about the means. Figure 9.33 shows the same data on the proper scale. The cloud graphic in the left panel shows why Figure 9.32 is misleading.

Statisticians are accustomed to doing these transformations before plotting and analysis because statistics like means are otherwise unrepresentative of their data batches. That is why we logged the data before plotting. If the data were not logged, and if we restricted the range to less than 1000, say, then the dot summaries would have excluded humans as well as elephants and their locations would have been different. This can be disconcerting to those users who might want to place the points to conserve white space or to make them look pretty. The only possible answer to this desire is to point out that the summary would be misleading. It is a fundamental principle (some would call it a drawback) of the system presented here that displayed ranges and domains define the behavior of graphics, not the other way around. If a graph summarizes data in this system (*e.g.*, regression lines, confidence intervals, etc.) this summary is based on data within the bounds of the frame only. We would claim that the alternative — clipping white space around a summary graphic for aesthetic purposes — is a form of lying with graphics (Huff, 1954). We have spent some time on this issue because it is so widely misunderstood. After designing this behavior for SYSTAT's graphics, we heard from some users who thought this was a bug. We began to realize that this preference for the geometry of summaries over the geometry of data may be a worrisome and pervasive problem in scientific practice. Some scientists are publishing graphics that conceal the range and variability of their data. Graphics programs that encourage them to do this thoughtlessly are promoting scientific malpractice. There are exceptions to this statement, as there are to all generalizations. But we believe we are usually better served by showing, rather than hiding, our data.

ELEMENT: *point*(*position*(exposure*brainweight))

SCALE: *log*(*dim*(2))
COORD: *exp*(*dim*(2))
ELEMENT: *point*(*position*(*summary.mean*(exposure*brainweight)))
ELEMENT: *interval*(*position*(*region.spread.sd*(exposure*brainweight)),
color(*color.red*))

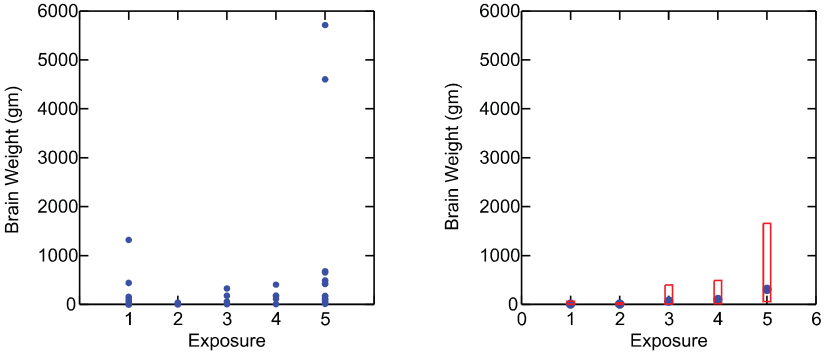


Figure 9.33 *Cloud and error bars on raw scale*

After looking at Figure 9.33, a persistent sceptic might argue that elephants and humans are outliers in this dataset, so we might as well exclude them and *then* restrict our range. This would solve our data-clipping problem in Figure 9.32 but would also be wrong. Figure 9.34 shows normal curves superimposed on dot plots of these data in the logged metric. The value most outlying from its group is the brain weight of the *smallest* animal (.14 gm); the lesser short-tailed shrew sits at the far left end of the dot density for the group corresponding to a value of **exposure** equal to 2. The brain weights of the elephants (at the right end of the scale in the group with an **exposure** value equal to 5) are *not* remarkable relative to their **exposure** cohorts. Logging often has this effect. Values that we suspect are outliers in the raw metric are not necessarily outliers in the transformed metric. The problem is not with our measurements; it is with the way we think about scales.

COORD: *transpose*()
SCALE: *log*(*dim*(1), *base*(10))
ELEMENT: *point.dodge*(*position*(*bin.dot*(exposure*brainweight)))
ELEMENT: *line*(*position*(*smooth.density.normal*(exposure*brainweight))

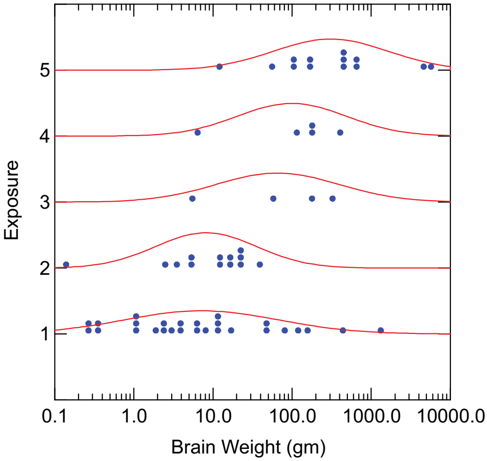


Figure 9.34 *Transformed densities*

9.1.8.2 Lensing and Fisheye Transformations

The *fish* transformation expands a graphic away from an arbitrary locus, usually the center of the frame or viewing area. This class of transformations has received a lot of attention from computer interface designers because of the need to make the best use of limited screen "real estate" when navigating through dense networks and graphical browsers (Furnas, 1986; Sarkar and Brown, 1994; Leung and Apperly, 1994). Smooth versions of fisheye functions look like the transpose of the graphics in Figure 6.12. This is because we wish to perform the opposite of those transformations: instead of lengthening the periphery and shortening the center so as not to be distracted by extreme variation, we wish to push the center toward the periphery in order to see more variation in the center. A broad class of functions will serve; one example is the logit cumulative distribution function $f(x) = e^x / (1 + e^x)$. We can improve speed by using the function

$$fish\colon\ x \rightarrow 2^x / (1 + 2^x)$$

which allows us to use bit-shifts in integer arithmetic instead of exponentiating floating-point numbers.

Figure 9.35 shows how to make our dinosaur put on weight by a simple transformation. The important feature to notice is the independence of the transformation on *x* and *y*. The center of the dinosaur is pushed to the edges of a square, not a circle.

COORD: *fisheye*()
ELEMENT: *edge*(*position*(*link.mst*(x*y)))

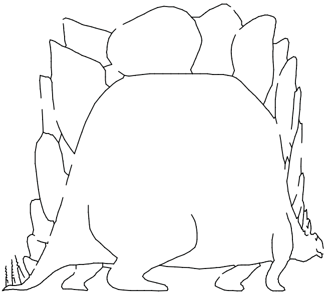


Figure 9.35 *Fisheye dinosaur*

Figure 9.36 shows a fisheye transformation of a circular Gaussian scatterplot cloud with a message embedded in the center (patterned after a dataset constructed by David Coleman). This transformation is especially suited for dynamic displays where the center of magnification (a constant subtracted from *x* before the transformation) can be moved with a mouse or joystick. Carpendale, Cowperthwaite, and Fracchia (1997) discuss applications for 3D graphics.

ELEMENT: *point*(*position*(x*y))

COORD: *fisheye*()
ELEMENT: *point*(*position*(x*y))

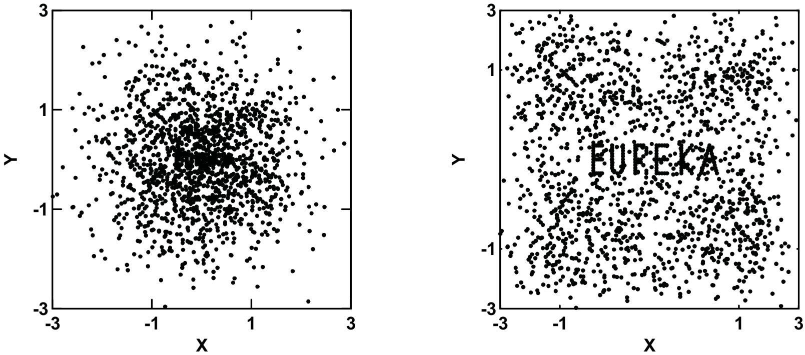


Figure 9.36 *Fisheye transformation*

9.1.9 Warpings

Warpings involve the continuous planar transformation

$$(x, y) \rightarrow (f(x, y), g(x, y)) = (u, v)$$

This introduces shear into the general bending transformation, because u and v both depend on x and y. Many global and local transformations fit this model, but we will focus on two simple examples that reflect what cartographers sometimes call **rubber sheeting**.

9.1.9.1 Parametric Warping

The left two panels of Figure 9.37 are reproduced from D’Arcy Thompson’s *On Growth and Form* (Thompson, 1942, page 1064). As Gould (1980) explains, Thompson attempted to offer an alternative to Darwin’s principle of natural selection by attributing morphology mainly to the effect of physical forces on biomatter. This idea led Thompson to look for mathematical similarities in the shapes of organisms — simple transformations that could relate one biological shape to another. John O'Connor and Edmund Robertson illustrate Thompson’s ideas in the MacTutor archive at the School of Mathematics and Statistics, University of St. Andrews. They propose a simple quadratic map for most of Thompson’s transformations:

$$x \rightarrow a_1x + a_2y + a_3x^2 + a_4y^2 + a_5xy = u$$

$$y \rightarrow b_1x + b_2y + b_3x^2 + b_4y^2 + b_5xy = v$$

Figure 9.37 shows Thompson’s drawing of a different warping transformation of the species Diodon (pufferfish) to the species Orthagoriscus (sunfish). He describes the transformation as follows:

> I have deformed its vertical coordinates into a system of concentric circles and its horizontal coordinates into a system of curves which, approximately and provisionally, are made to resemble a system of hyperbolas.

In the rightmost panel of the figure, we have implemented Thompson’s transformation on his drawing of the pufferfish. The result is not far from Thompson’s drawing of the sunfish superimposed on the same grid. Other than the tail, the fit is quite close. Except for the size. We need to do a rather large dilation of the pufferfish to get a fish weighing several thousand pounds.

COORD: *circular.hyperbolic*()
ELEMENT: *point*(*position*(x*y))

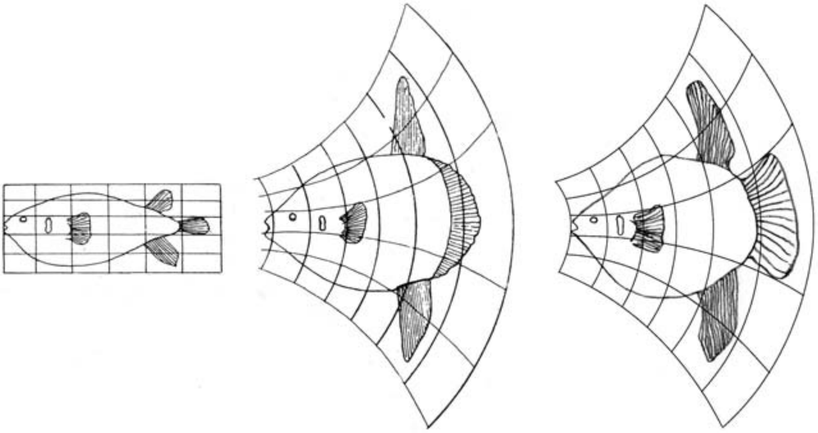


Figure 9.37 *Thompson's growth transformation (Diodon left, Orthagoriscus middle, transformed Diodon right)*

9.1.9.2 Dynamic Time Warping

Suppose we have two sequences,

$$A = \langle a_1, \ldots, a_m \rangle \text{ and } B = \langle b_1, \ldots, b_n \rangle ,$$

each representing an acoustic measurement of a non-repeating segment of a bird song. Both songs are produced by the same bird, but the two series are a different length. How do we compute a measure of similarity between the two series that allows for local stretching of the time scale (by repeating indices) in order to match the shapes of the two series as closely as possible?

Sakoe and Chiba (1978) devised one solution to this problem. First, based on some distance function, compute a distance matrix

$$\mathbf{D} = d(a_i, b_j) \ , \ i = 1, \ldots, m \ , \ j = 1, \ldots, n$$

Then compute a path through **D** from $i = 1, j = 1$ to $i = m, j = n$ such that the sum of the path weights $d(a_i, b_j)$ is minimum *and* the path is monotonically increasing. Because the algorithm is best implemented through dynamic programming (recursively computing the path at each point), the procedure is called **dynamic time warping** (DTW). Sankoff and Kruskal (1983) cover applications of this algorithm to a variety of areas. Zhang and Lu (2004) review applications to shape analysis. Ratanamahatana and Keogh (2004) discuss applications in data mining.

Figure 9.38 shows an example. The data (Legrand, 2002), represent a formant for the word "right." A formant is a resonant region of a spectrum associated with a wind instrument or voice. The red trace is the formant for the word spoken slowly and the blue is for the word spoken rapidly. Computing a measure of agreement based on dynamic time warping allows both traces to be recognized as the same word. An ordinary cross-correlation would fail to match similar segments of the curves.

ELEMENT: *line*(*position*(x*y))

COORD: *dtw*()
ELEMENT: *line*(*position*(x*y))

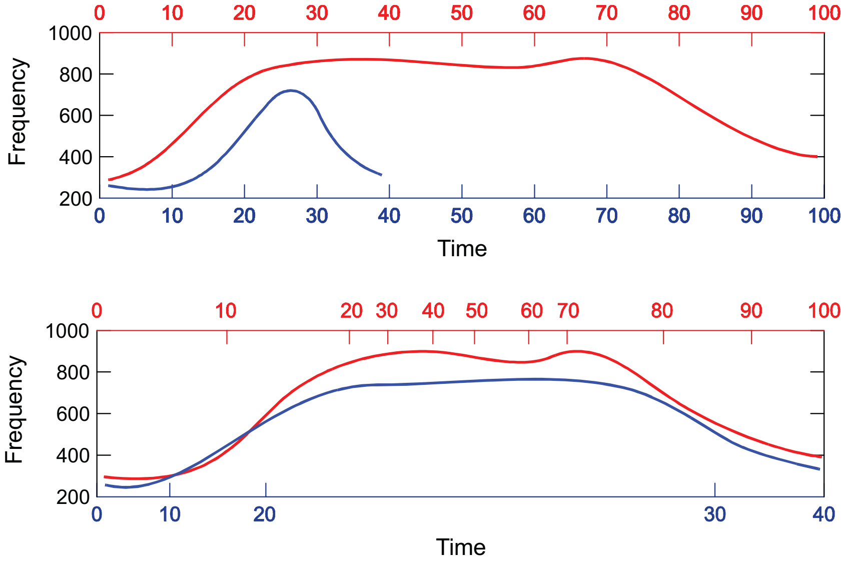


Figure 9.38 *Dynamic time warping of formant*

9.1.9.3 Locally Parametric Warping

Because of terrain, congestion, road conditions, and other factors, the set of all points at the end of one hour's travel time from a point located on a road map does not lie on a circle centered at that point on the map. Tobler (1993, 1997) proposed to analyze travel-time geometry as a two-manifold of variable curvature. From this analysis, one could construct polar geodesic maps showing **isochrones** (contours of equal travel time) centered on a selected location.

Tobler collected data on a number of measures for this type of spatial modeling, including road travel times, airline travel costs, parcel shipment costs, and estimates of distances by individuals. Tobler's non-Euclidean spatial methodology brings to mind Saul Steinberg's famous *New Yorker* cover that shows a native New Yorker's parochial view of the United States. Steinberg's map compresses almost all of the country west of the Hudson river.

Figure 9.39 contains a map for airline travel costs using a similar concept but different methodology from Tobler's. This map is not centered at any location, but instead reflects the distortion of the plane due to variations in fares within all pairs of cities. It is designed so that the airline fare can be approximated by computing a straight-line distance between any two points.

To make this graphic, we began by visiting several travel sites on the Internet to collect the cheapest round-trip airline fares on September 1, 1998 for all pairs of 36 continental US cities. We then used nonmetric multidimensional scaling to compute a configuration of cities embedded in a two-dimensional Euclidean space. The stress for this solution was .19 and the Shepard diagram revealed a roughly linear relation between prices in dollars and distances between cities in the MDS solution. If the airline fares had been perfectly proportional to distances between cities, then the stress would have been zero and the MDS solution would have been similarity transformable (see Figure 9.1) to a map of the US on the Euclidean plane.

Next, we used Procrustes rotation (Borg and Groenen, 1997) to align the MDS configuration as closely as possible with the geographic map. The Procrustes rotation produces a similarity transformation (translation, rotation, reflection, dilation) minimizing the sum of the squared differences between the transformed source and target coordinates. This computation produced a set of airline-fare-spaced city "knots" that we could use to pin down the distortions in a map. The following equation shows the transformation we used. Let θ_i be the longitude of the ith city ($i = 1, \ldots, 36$) and let ϕ_i be its latitude. Let r_i be the coordinate of the ith city on the first rotated MDS dimension and let s_i be its coordinate on the second rotated MDS dimension. For any point at coordinates (x, y) on the geographic plane, we computed new coordinates (u, v) with an inverse distance-weighted average of the displacements of the cities:

$$\begin{cases} u = x + \Sigma w_i(\theta_i - r_i)/\Sigma w_i \\ v = y + \Sigma w_i(\phi_i - s_i)/\Sigma w_i \end{cases}, \text{ where}$$

$$w_i = 1/((x - \theta_i)^2 + (y - \phi_i)^2 + \alpha)$$

The summation is taken over the 36 cities. The parameter α is a small value chosen to prevent division by zero when a point is located at one of the cities.

The map reveals that Midwesterners pay relatively higher fares to get out of town. The "distance" from New York to Los Angeles is roughly the same as that from Duluth to Los Angeles. And, cruel fate, Midwesterners have to sacrifice their first-born (relatively speaking) to get to Florida! For recent work on flexible warping algorithms for choropleth maps, see Keim, North, and Panse (2004) and Gastner and Newman (2004).

DATA: longitude, latitude = *map*(*source*("US states"))
COORD: *warp*(*r*(), *s*())
ELEMENT: *polygon*(*position*(longitude*latitude))

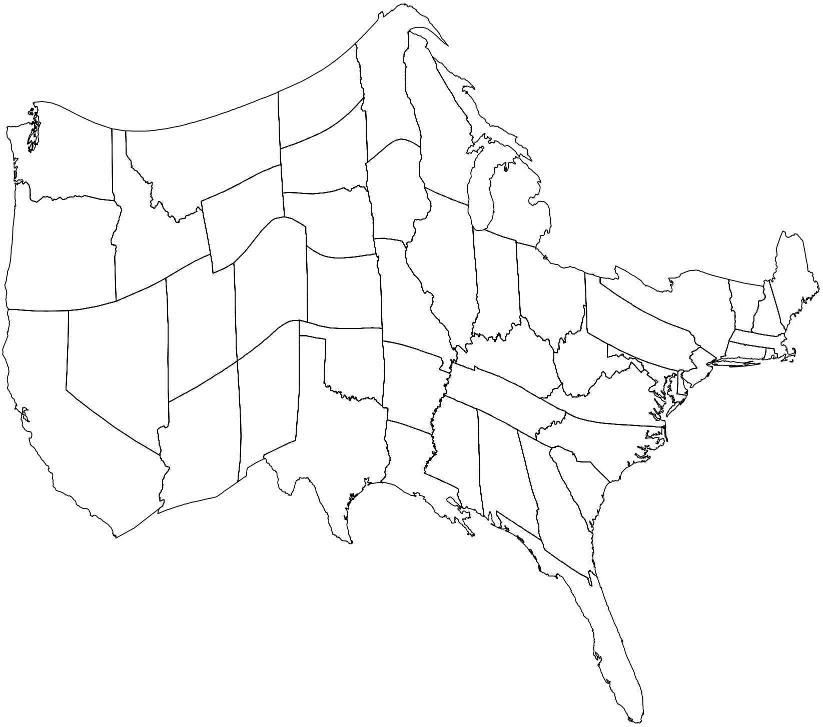


Figure 9.39 *US Airline Pricing Map*

9.2 Projections onto the Plane

So far, we have examined coordinate transformations that help to enhance the perception of patterns and structures in two dimensions. Now we are going to look at transformations that allow us to explore 3D and higher-dimensional worlds through a 2D window. This enterprise is doomed from the start. If we attempt to view pairs of dimensions separately, we will have trouble detecting second-order or higher relations between pairs not in the same view. If we try to find linear combinations of dimensions that reveal structures, we will have difficulty orienting ourselves in the full dimensional space. And, if we use some nonlinear projection or compression method to characterize an entire structure, we risk distorting our view enough to prevent accurate interpretation. In short, 2D windows into higher-dimensional spaces can give us a glimpse into another world but may either confuse us with apparent complexity so that we overlook global coherence or mislead us with apparent simplicity that leads us to incorrect global inferences. The complexity of representing structures in high-dimensional spaces has been called the **curse of dimensionality** (Bellman, 1961). This curse does not prevent us from devising methods for circumventing it in specific instances, however. We will begin with the 3D issues and then move to higher dimensions.

9.2.1 Perspective Projections

We saw in Figure 9.13 how planar linear projection works. Figure 9.40 illustrates linear projection of a 3D object onto a plane. Rogers and Adams (1990) show how to construct the projection matrix from the coordinates of the projection point, projection plane, and object. This is the perspective planar projection used in computer graphics libraries. It is not exactly the model for realizing an image on our retina and definitely not the model for how images are processed in our visual cortex. But it does provide a result good enough to allow our visual system to use its tricks to reconstruct a 3D scene from a picture. Pinker (1997) explains these tricks to non-psychologists in a beautiful work of general science writing.

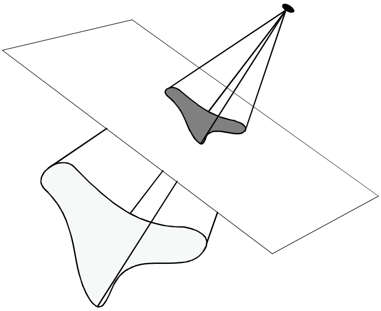


Figure 9.40 *Perspective projection onto plane*

Figure 9.41 contains a perspective projection of a 3D compound object. The points, axes, grid lines, and text are all put through the same projection. These are the countries data we used in Chapter 1. We have plotted annual percapita health care expenditures in adjusted US dollars against death and birth rates for selected countries. The heaviest investment in health care is in those countries with low birth and death rates. In general, countries with lower death rates (except for the lowest) tend to have heavier investments in health care. As birth rates increase, however, per capita investment declines.

COORD: *project*(*matrix*(..))
ELEMENT: *point*(*position*(birth*death*health), *shape*(*shape.cube*))

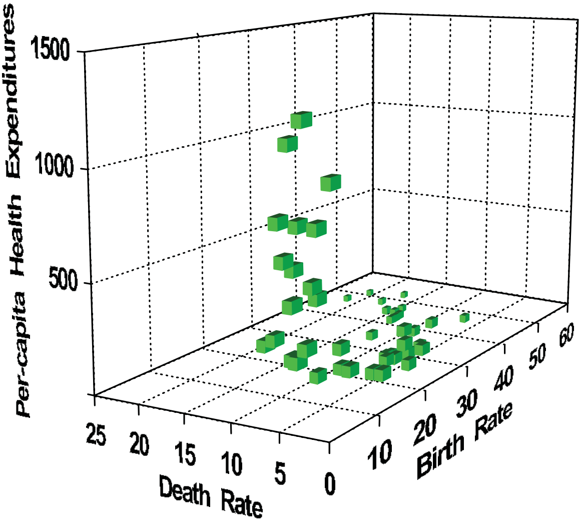


Figure 9.41 *Health expenditures by birth and death rates (enhanced)*

Why do we display data in 3D? As we will argue in Chapter 10, there is often an effective 2D alternative to a specific 3D display. Unfortunately, 3D graphics often elicit more perceptual illusions and make it difficult to use scale look-up in perspective to recover data values. Nevertheless, 3D graphics can be useful when we encounter **configurality** among variables (Meehl, 1950). While it is possible to define configurality for some functions using partial derivatives or parallelism of level-curves, it is perhaps simpler and more general to describe it in ordinary language and to illustrate it with these data. For three variables *x*, *y*, and *z*, configurality occurs when the way *z* changes across values of *y* at some level of *x* is different from the way *z* changes across values of *y* at some other level of *x*. There is a duality in this definition. Configurality also implies that the way *z* changes across values of *x* at some level of *y* is different from the way *z* changes across values of *x* at some other level of *y*. Statisticians often describe this condition as an **interaction** between *y* and *x* with respect to *z*, but we mean something more general (which is why we used the more general word "way" rather than "rate"). My definition of configurality assumes that an interaction cannot be removed by the coordinate transformations in this chapter; if it could, we could do a transformation and resort to a 2D plot instead (see Abelson, 1995).

Consider Figure 9.42. If there were no configurality among health expenditures (z), death rates (y), and birth rates (x), then this figure would be an adequate representation of the relation between health expenditures and death and birth rates. We could look at the left panel, for example, and conclude that health expenditures decline precipitously with increasing birth rate regardless of death rate. Similarly, we could conclude that health expenditures hit a peak for death rates slightly less than 10 per 100,000 regardless of birth rates. Comparing Figure 9.41 to Figure 9.42 reveals that these descriptions are misleading, however. The high health expenditures for low death rate countries are spread across a large range of birth rates, for example.

```
DATA: b = "Birth Rate"
DATA: d = "Death Rate"
ELEMENT: point(position((birth/b + death/d)*health))
```

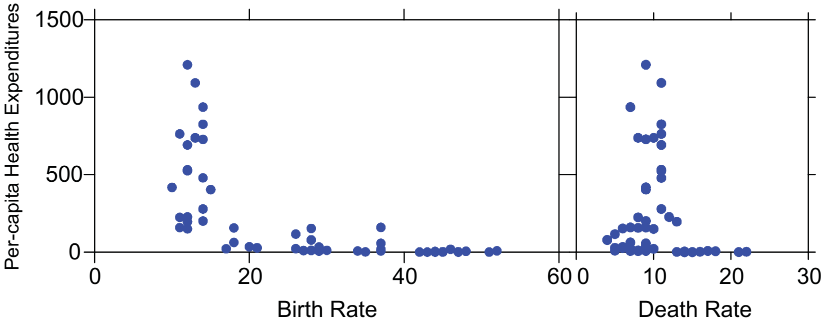


Figure 9.42 *Health expenditures by birth and death rates*

We can graphically model configurality by **blocking** (or **stratifying**) on the x or y variables so that we can see the z variable at various levels of one or the other. This is shown in Figure 9.43. We cut the distribution on birth rate into three equal intervals and then plotted health expenditures against death rate within each interval. Now the change in the relationship is clearly apparent. We could construct a similar plot by stratifying on death rate and plotting health expenditures against birth rate. Becker, Cleveland, and Shyu (1996) developed the **trellis display** to produce such plots automatically.

TRANS: birthgroup = *cut*(birth, 3)
ELEMENT: *point*(*position*(death*health*birthgroup))

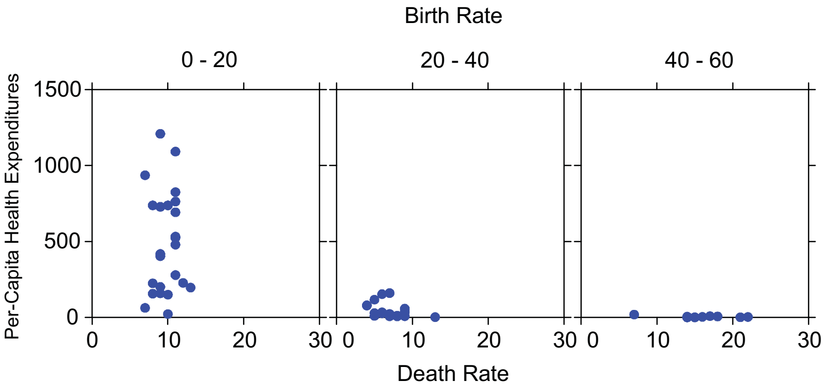


Figure 9.43 *Health expenditures jointly by birth and death rate*

We will discuss faceted displays like Figure 9.42 and Figure 9.43 in Chapter 11. These can be effective for dealing with configurality. And they facilitate scale look-up to help recover data values. There is some price we pay for this stratification, however. First of all, we may have too few cases left in some of the strata, particularly if we stratify on more than one variable. Cleveland (1995) has a clever way of dealing with this problem. He calls it **shingling**, which is cutting the distribution into overlapping groups. This provides more cases in each cell and provides more continuity in the changes across cells.

We are left with a second, more difficult problem, however. Faceted displays are designed to answer one type of question, and 3D displays another. To illustrate this problem, we need to distinguish **variable world** from **object world**. (We might say dimension world rather than variable world, but we will assume that there is one variable per dimension in this discussion). Variable world requires variable descriptions, *e.g.*, “health expenditures tend to decrease with increasing birth rate.” Object world requires object descriptions, *e.g.*, “the points spiral in a ram’s horn pattern from low and thin in the rear of the plot to high and thick in the foreground.” Engineers alternate between these two worlds when they speak of rotating an object or rotating axes. Mathematicians alternate between these two worlds when they describe a function either as $z = x^2 - y^2$ or as a saddle.

We believe that if we have a three-variable system, we are better served by displaying the data in 3D when we are interested in object world and we do better with facets when we are interested in variable world. The ram’s horn that we see in Figure 9.41 is not readily discernible in any faceted plot of these data. Furthermore, this ram’s horn is memorable as a single object. We would expect that, following results in Wilkinson and McConathy (1990), we would recognize and recall configural relationships in 3D plots better than we would in faceted displays because they more closely fit the way we deal with objects in the physical world. However, we would decode values more accurately when presented with a faceted graphic. There are limits, however. Cleveland (personal communication) has stated that effective 3D graphics require **coherence**. If a geometric graphic does not cohere in a single recognizable thing, 3D representation won't help us much. There are enough problems when parts of objects remain hidden behind other parts in a 3D plot. If points, lines, and other primitives do not cohere, we are better served by a faceted display.

Other questions: How are trellis displays related to linear projections? Would it make sense to facet a display with nonorthogonal or nonlinear projections? There is one sleight-of-hand we played in Figure 9.41. What did we do and does it help or hurt our perception of the structure? Hint: look at the size of the boxes.

9.2.2 Stereo Pairs

Our minds perceive depth through a variety of strategies. Illumination, texture, occlusion, contiguity, surface color, and other features of a scene all provide clues to the 3D orientation of objects in that scene. One important optical mechanism for depth measurement is called **parallax**. If we observe an object in space from two different points of view, then we have a triangle whose base lies between the observation points and whose apex lies at the object. Figure 9.44 shows two examples. The baseline distance and the angles at each end of the baseline are sufficient information to compute the dotted line perpendicular to the baseline in the figure. This technique has been used in astronomy and surveying for centuries, and it is not surprising that it should play a role in the visual system through the physical separation and orientation of the eyes.

In order to use parallax to elicit a sense of depth from a 2D scene, we have to present a different perspective projection to each eye separately so that the parallax effect will resemble that found in viewing a real 3D scene. In other words, the same object must be projected to different coordinates on each retina as it is in Figure 9.44. Sir Charles Wheatstone, the English physicist, found a way to do this in 1938. His **stereoscope** projected a separate image to each eye through the use of mirrors. Each image was produced from a photograph taken with a camera located at one of two different positions in front of a scene. The stereoscope artificially reproduced the optical projection that takes place when we view a natural scene. The inexpensive contemporary equivalent of Wheatstone's stereoscope is the plastic ViewMaster stereo slide viewer. ISSCO, a mainframe technical graphics company, used ViewMasters to present some spectacular visualizations in the early 1980s before virtual reality technology was generally available.

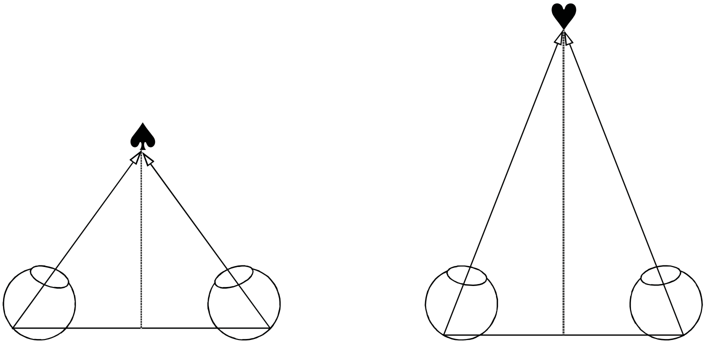


Figure 9.44 *Parallax*

If we do not have a stereoscope, we can project a separate 2D image on each retina by crossing our eyes. To see how this works, we begin with Figure 9.45. The left panel shows two eyes aimed at a common focal point represented by a red dot. We have drawn an arrow from the center of the retina to the focal point. Two objects, a spade and a heart, are viewed by the eyes. We have drawn dotted lines from these objects to the retinas in order to indicate where they are projected onto the retinal surface.

The blue eye in the middle represents the **cyclopean eye** (named after Ulysses' one-eyed nemesis). This is not a real eye, but instead a schematic representation of the result of the visual system's merging separate patterns of retinal stimulation into a common perceptual image. Because spade and heart are projected to similar coordinates on each separate retina, they coalesce in the cyclopean image and we see them as single objects. The arrows below the eyes show how the spade and heart are merged.

The right panel of Figure 9.45 shows the same eyes presented with a single heart closer than the point of focus. This time, the object projects to a different part of each separate retina. The visual system cannot merge the sources of stimulation into a single cyclopean image, so we see double in this circumstance. You can verify this by focusing on your right index finger held at arm's length while you hold your left index finger at half the distance. The left finger doubles. If you switch your gaze to the foreground, the finger in the background doubles.

If we do the same thing with a double graphic that looks like Figure 9.46, then we get a double of doubles. If we focus our gaze at the correct distance, and hold the page at half that distance, then the doubling of doubles overlaps in the center and we see a single middle composite image and two surrounding separate images. This means that the central overlapping image is compiled from both graphics, mimicking the result that happens when we look at a real scene at the focused depth. By plotting each graphic from a different perspective (by a few degrees), we trick our visual system into blending them stereoscopically, as if they were real projections on each eye.

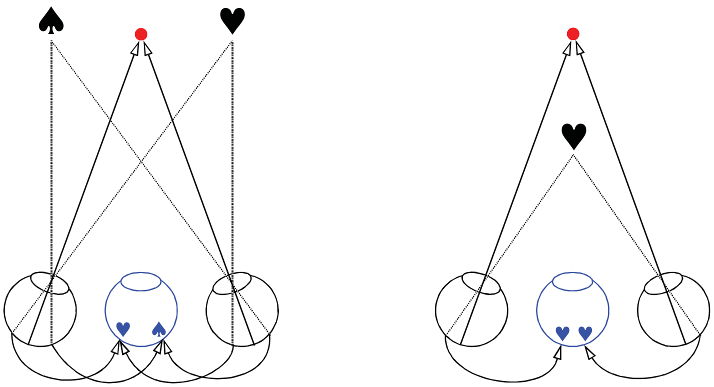


Figure 9.45 *Cyclopean vision*

Not everyone can view these graphics without mechanical assistance. We find it easiest to place the page two feet from our eyes (not two eyes from our feet), and force the image to double by focusing on our index finger held about 8 inches from our eyes. Then we move the index finger away while carefully transferring our concentration to the central composite image. Remember to do this only with Figure 9.46. If you focus on Figure 9.45 by mistake, you will permanently cross your eyes and lose your driving privileges (except in Boston). Actually, the eye-crossing lockup is a myth, but if you gaze too long on Figure 9.45, you may turn into a post-modernist.

Three-dimensional virtual reality systems use mechanical stereoscopic devices based on cathode-ray tubes or liquid-crystal displays to present a separate image to each retina. Goggles with separate displays for each eye can present images controlled by separate processors. The most intense psychological experience, however, is offered by the CAVE immersive environment (Cruz-Neira, Sandin, and DeFanti, 1993). This apparatus consists of separate color projectors aimed at the walls of a room large enough to hold three or four people. Each person wears polarizing glasses controlled by the same computers running the projectors. The perspective images are switched at a high enough rate to blend in a single perception, like a movie. Because the environment is immersive, allowing other people to move about the room, it provides the highest level of virtual reality currently obtainable. Recent advances in desktop computers and LCD color projector technology are bringing this capability that once cost over a million dollars to smaller laboratories and homes.

COORD: *project*(*dim*(1, 2, 3), *matrix*(..))
ELEMENT: *point*(*position*(birth*death*health), *shape*(*shape.cube*))

COORD: *project*(*dim*(1, 2, 3), *matrix*(..))
ELEMENT: *point*(*position*(birth*death*health), *shape*(*shape.cube*))

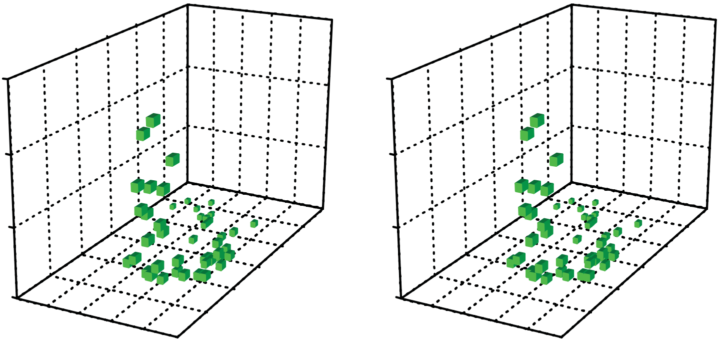


Figure 9.46 *Stereo pair*

9.2.3 Triangular (Barycentric) Coordinates

If three variables are constrained to sum to a constant, the set of all possible values they may take lies on a plane. If all these values are constrained to be positive, they are bounded by a triangle. Figure 9.47 shows an example of this triangle. Triangular coordinates are especially useful for representing mixtures of three variables. Perhaps the best known application is the *CIE* color diagram which shows perceptible mixtures of the three color primaries red, green, and blue (Levine, 2000). Other interesting applications of barycentric coordinates are discussed in Mosteller and Tukey (1968).

A formula for computing this projection is

$$\begin{cases} u = \left(2\tan\left(\frac{\pi}{3}\right)x + \tan\left(\frac{\pi}{3}\right)y\right) / (x + y + z) \\ v = y / (x + y + z) \end{cases}$$

The tangent function is on 60 degrees, the angle of the vertices of an equilateral triangle. This triangle is shown tilted in Figure 9.47. The denominator of the expression $(x + y + z)$ controls the scaling of the triangle.

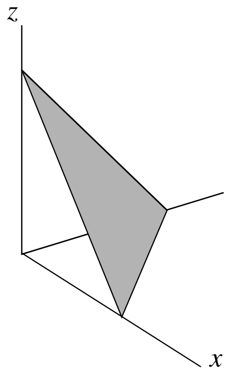


Figure 9.47 *Triangular coordinate plane*

Nielsen *et al.* (1973), cited in Andrews and Herzberg (1985), reported percentages of three components — sand, silt, and clay — in core borings from 20 different plots in a soil field. Figure 9.48 shows a triangular coordinate plot of their data. To read the values on a given axis, choose the grid lines that are parallel to the axis that shares an apex at the zero end of the axis. For example, the **silt** axis grid lines are horizontal and the **clay** grid lines tilt 30 degrees to the right.

COORD: *triangular*(*dim*(1, 2, 3, 4))
ELEMENT: *point*(*position*(sand*silt*clay), *color.hue*(plot))

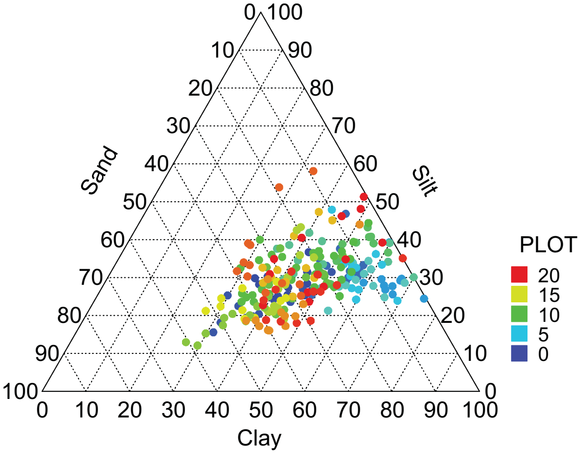


Figure 9.48 *Triangular coordinates plot of soil samples*

We can see that most of the samples contain fairly high percentages of clay and lower percentages of silt and sand. Furthermore, the plots with lower index numbers tend to have higher concentrations of clay and somewhat lower percentages of sand.

Triangular coordinates are especially important for displaying the results of mixtures experiments. When engineers seek to find optimal combinations of three ingredients (on some price, quality, or performance criterion), a triangular coordinate plot provides a convenient summary of the model they fit to predict the criterion. Cornell (1990) presents the results of an experiment designed to measure the suppression of the population of mites on strawberry plants after spraying with a mixture of three different pesticides. The surface plotted with contours represents the seasonal average mite population per leaf after a period of spraying. Cornell fit these numbers with a variety of parametric models, all of which fairly closely resemble the nonparametric fit in Figure 9.49.

Notice that the data points were chosen to provide support for estimating parameter values of the fitted function. The contours of the fitted function follow a valley running from the lower right apex to the middle of the **x1** axis. Cornell selected more data points along this valley in anticipation of finding a surface roughly resembling this form. Since collecting data at each mixture point can be costly, careful planning and iterated experimentation is needed to converge on a model that identifies the optimal combination of ingredients.

COORD: *triangular*(*dim*(1, 2, 3, 4))
ELEMENT: *point*(*position*(x1*x2*x3))
ELEMENT: *contour*(*position*(*smooth.quadratic.cauchy*(x1*x2*x3*mites)),
    color.hue())

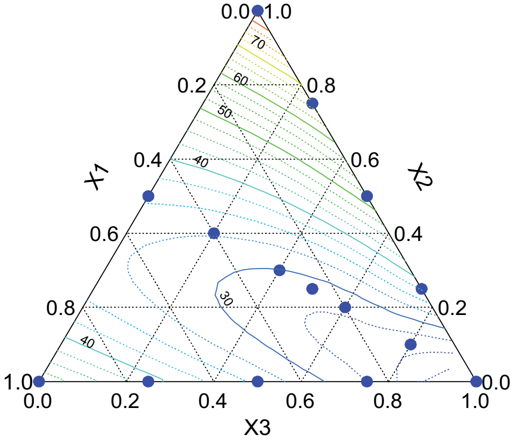


Figure 9.49 Contours in triangular coordinates

9.2.4 Map Projections

This class of mappings involves the continuous transformation

$$(x, y, z, ...) \rightarrow (f(x, y, z,...), g(x, y, z,...)) = (u, v)$$

It includes everything from linear projections of the hemisphere onto the plane to nonlinear and piecewise projections of the whole globe onto the plane.

Map projections have a long history in cartography, beyond the scope of this book (see Maling, 1992; Snyder, 1989). Spherical projections are basic to non-Euclidean geometry, another area beyond the scope of this book. Mapping the sphere to the plane has graphical applications beyond representing the earth in a 2D map. Map projections are useful for representing graphically any statistical distribution on the sphere.

Most cartographic projections can be classified according to their projective surface: planes, cylinders, or cones. Figure 9.50 shows three normal planar projections of the Eastern hemisphere. we have projected latitudes between –75 and 75 degrees and longitudes from 0 to 180 degrees. The light-ray model is realistic, but there is one small anomaly that you should be able to detect even if you are not a geographer. Hint: “Wrong Way” Corrigan would have liked our map. Double hint: Australians are used to being down-under, but not backwards. We needed to do this to maintain the physical model.

The **gnomonic** projection assumes a light source located at the center of the sphere and a projection plane tangent to the surface (or parallel to this plane) at a selected point on the sphere. It can project only a hemisphere. As the figure shows, area distortions are severe at the poles. Longitudes project to straight lines and latitudes (except the Equator) to curves. The **stereographic** projection places the light source on the surface of the globe and the projection plane tangent to the point on the opposite side of the globe. This important geometric projection maps the whole sphere (except for one point) to the plane. It figures in geometric theory as well as geography (Stillwell, 1992). The **orthographic** projection places the light source at infinity. This projection produces a result that resembles the view of the Earth from the moon. All three of these planar projections are a form of perspective projection (see Section 9.2.1).

Figure 9.51 shows cylindrical and conical projections. By bending the plane, these methods help to represent more of the global surface and, in some cases, distort area less. Normal cylindrical methods project longitudes and latitudes to straight lines. The classic Mercator cylindrical projection is especially useful for navigation because compass bearings plot as straight line segments on the plane. Standard nautical maps still use the Mercator projection. The conical methods are most suited for regional maps. In the normal conical projections, longitudes plot as oblique lines and latitudes as straight lines or curves.

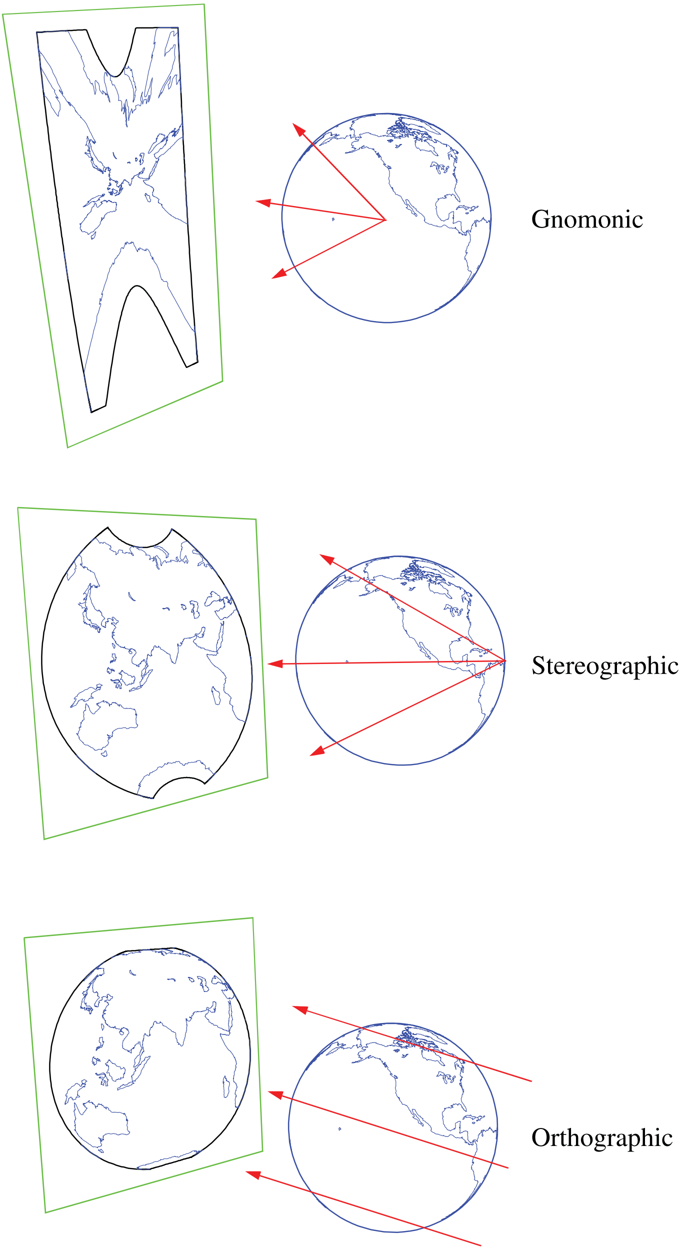


Figure 9.50 *Planar map projections*

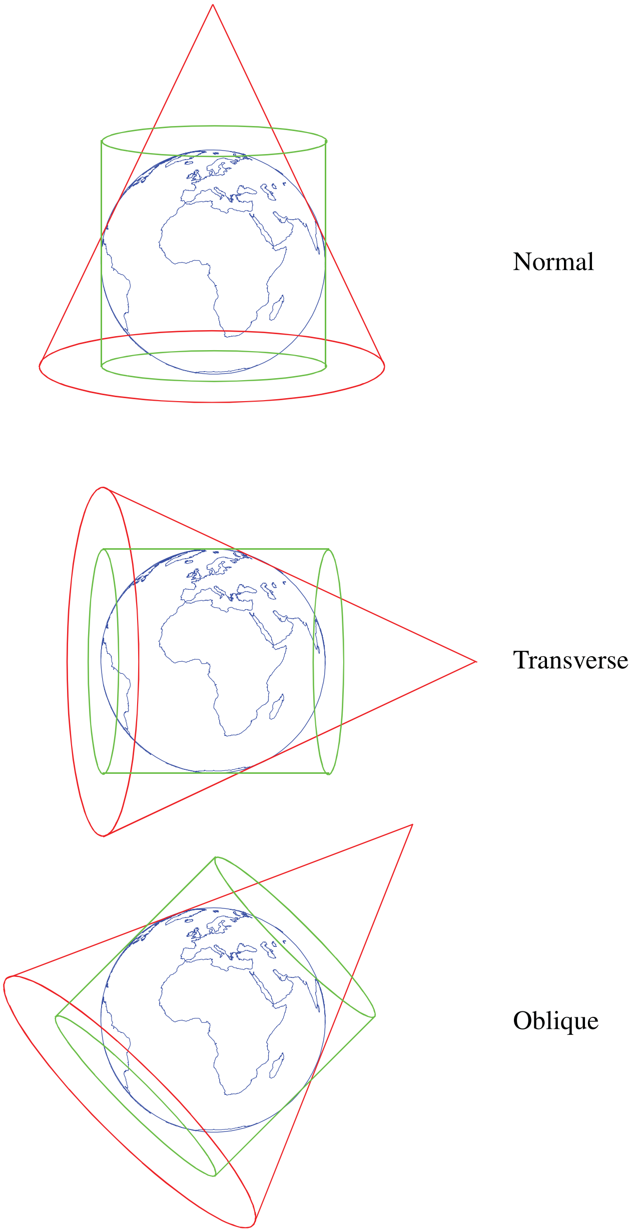


Figure 9.51 *Cylindrical and conical map projections*

9.2.4.1 Global Maps

Figure 9.52 shows several global map projections. The **Peters**, **Miller**, and **Mercator** projections are recognizable as cylindrical because the longitudes are parallel. The Peters projection favors the Southern Hemisphere in a politically motivated attempt to counter the influence of centuries of maps positioned transversely on the Northern Hemisphere. The **Robinson** projection (Robinson, 1974) achieves more balance. Unlike the others, the Robinson projection is not the result of a single trigonometric function; it is a smooth piecewise blending. Robinson has been a projection used by *The National Geographic* in many of its maps because it favors the temperate zones without discriminating against the Southern Hemisphere. Sadly, the Robinson projection discriminates against penguins.

We have included axes on all the maps to reveal global distortions. Instead of repeating the specification four times, we have substituted <*projection*> for the projection name in the specification.

```
DATA: longitude, latitude = map(source("World"))
COORD: project.<projection>()
ELEMENT: polygon(position(longitude*latitude))
```

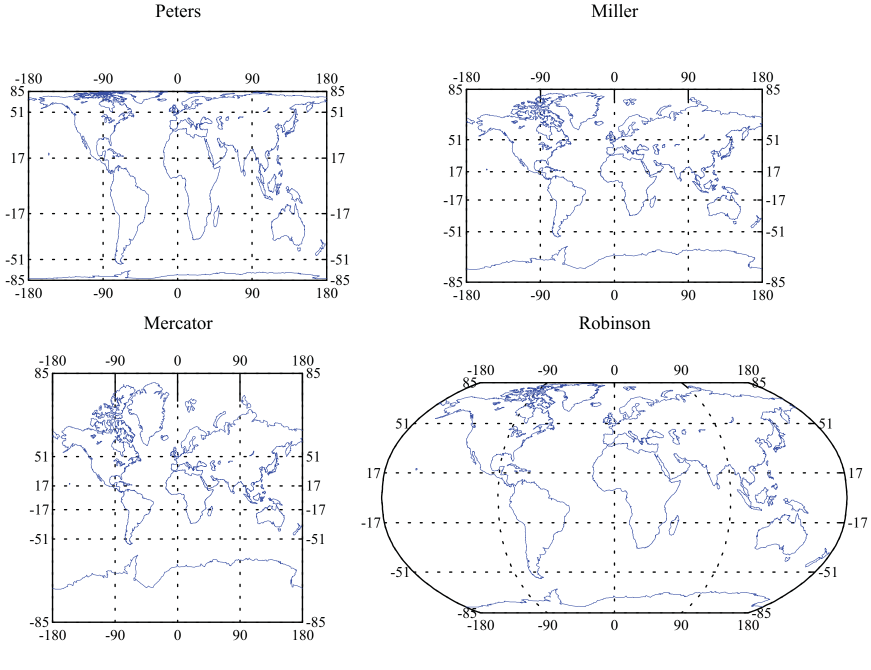


Figure 9.52 *Global map projections*

9.2.4.2 Local Maps

Local maps suffer less from global distortion. Figure 9.53 illustrates several.

DATA: longitude, latitude = *map*(*source*("US states"))
COORD: *project.<projection>*()
ELEMENT: *polygon*(*position*(longitude*latitude))

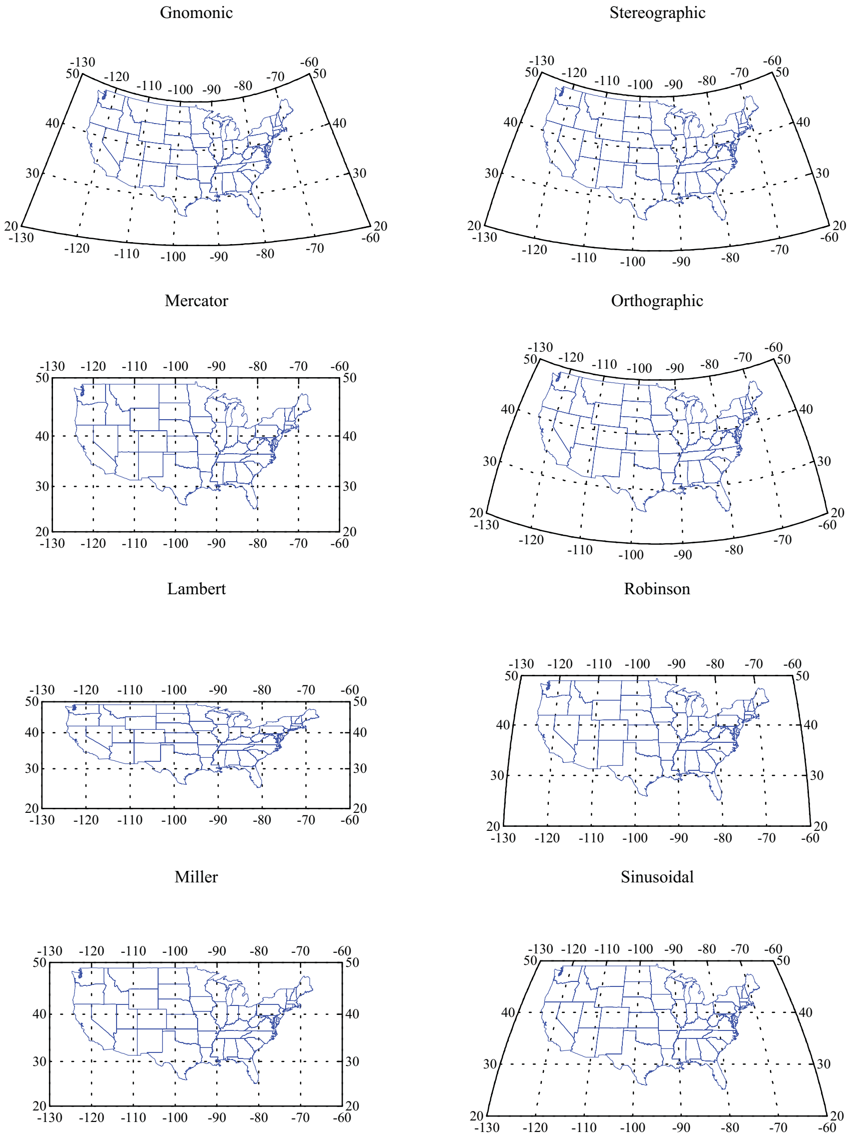


Figure 9.53 *Map projections of US*

9.2.4.3 Statistical Graphics on the Sphere

Map projections transform the geometry of graphics. Except for symbols, which maintain their shape regardless of coordinate systems, graphic shapes are modified by coordinate transformations. Figure 9.54 shows an example. The data were taken from a compilation of worldwide carbon and nitrogen soil levels for more than 3,500 scattered sites. These data were compiled by P.J. Zinke and A.G. Stangenberger of the Department of Forestry and Resource Management at the University of California, Berkeley. The full dataset is available at the US Carbon Dioxide Information Analysis Center (CDIAC) site on the World Wide Web (*cdiac.esd.ornl.gov*).

We logged the carbon values because they are positively skewed in the data. They appeared to be normally distributed after logging. We used log base 2 for two reasons: the scale is easily read as doublings and the range of colors is better covered on a round log$_2$ scale than on a round log$_{10}$ scale. The high carbon soil sites in the Pacific Northwest and in Florida are easily identified. They are twice as high as the average levels in the rest of the country.

The sampling grid for these data was not uniform. We have used a *polygon* graphic to represent the carbon levels in bins delineated by latitude and longitude boundaries. The stereographic spherical projection changes the shapes of these bins from spherical rectangles to planar arcs. This is not a cartographic map. It is a statistical distribution measured in geographic coordinates. Unfortunately, rectangular bins do not represent equal sampling areas on the surface of a sphere. Carr *et al*. (1992) comment on the usefulness of hexagonal bins for this purpose.

TRANS: carbon = *log.2*(carbon + 1)
COORD: *project.stereo*()
ELEMENT: *polygon*(*position*(*bin.rect*(lon*lat)), *color.hue*(carbon))

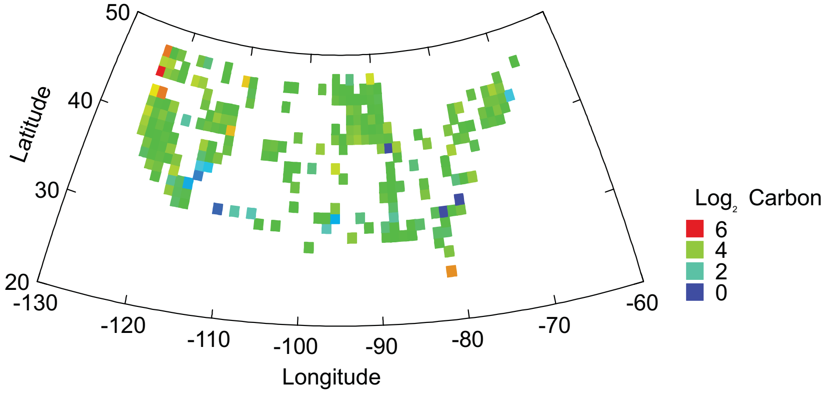


Figure 9.54 *Tiles of soil samples*

9.2.5 Higher-Dimensional Projections

Projections to the plane in higher dimensions present more severe problems than we have already encountered. We tend to encounter both fragmentation (inability to keep adjacent objects near each other on the plane) and distortion (preserving lines, circles, angles, etc.). We will discuss projections from high-dimensional spaces in more detail in Section 9.4

9.3 3D Coordinate Systems

So far, we have considered 2D, 3D, and higher-dimensional objects. We have visualized all these objects through the plane, sometimes directly and sometimes through projections. Some higher-dimensional coordinate systems are interesting in themselves, apart from their projected representations. We have already seen some of these — for example, the spherical coordinates used for mapping the globe.

The coordinate systems in this section resemble polar and certain other non-rectangular 2D coordinates. In order for us to visualize objects in these systems, we will have to use 2D perspective projections, but this is only because they are presented in a book. Virtual reality systems — especially immersive environments — could be used to represent these structures in a 3D setting.

The focus of our interest in this section is thus on non-rectangular 3D coordinates, including spherical, triangular, and cylindrical. As the examples will show, these coordinates are useful for applications that involve both spatial and non-spatial statistics. While geography has motivated the most popular applications, other theoretical structures are best represented in these non-rectangular coordinate systems.

9.3.1 Spherical Coordinates

If (u, v, w) represents the spherical coordinates (ρ, θ, ϕ) of a point, then a spherical coordinate function $S(u, v, w) \rightarrow (x, y, z)$ corresponds to the case $u = \rho,\ v = \theta,\ w = \phi$ where $x = \rho \sin\phi \cos\theta$, $y = \rho \sin\phi \sin\theta$, and $z = \rho \cos\phi$. Spherical coordinates are useful for representing points on a sphere when ρ is a constant, and bundles of vectors at a common origin when it is not.

Contouring on the sphere presents special problems because distances must be calculated on the surface rather than in rectangular coordinates. There are technical problems with the drawing of other graphics as well. Figure 9.55 shows an example of tiling the sphere using geographic data. McNish (1948) reports magnetic declinations for 22 locations on the globe. We have used inverse distance smoothing on the surface of the sphere to represent the average declination at different areas on the globe. The *spherical.phitheta*() coordinate function takes two arguments only: ϕ and θ. The radius is assumed to be constant.

This function is similar to the way we handled polar coordinates in Section 9.1.6.1 when only one argument is used to make a pie chart. Spherical coordinates for three arguments are handled with *spherical*().

```
DATA: longitude, latitude = map(source("World"))
COORD: spherical.phitheta(p(), t())
ELEMENT: polygon(position(smooth.mean.cauchy(lon*lat*declination)),
         color.hue())
ELEMENT: polygon(position(longitude*latitude))
```

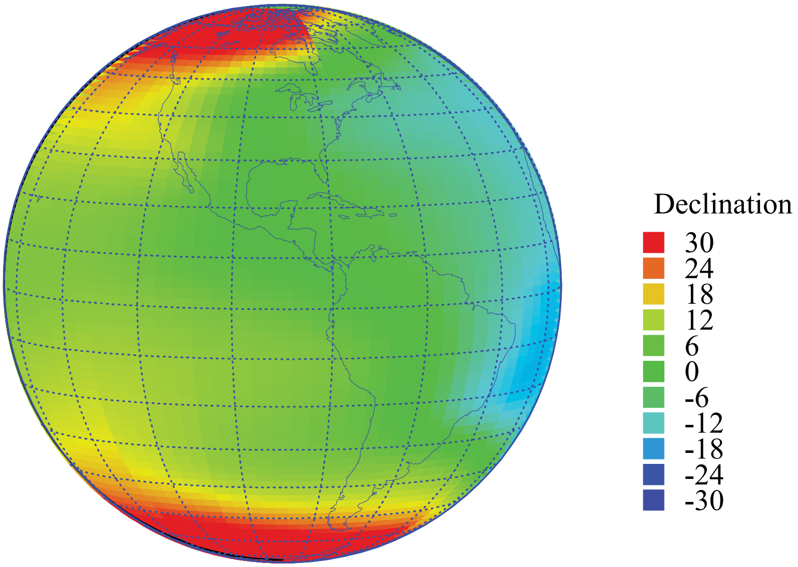


Figure 9.55 *Spherical distribution of magnetic declination*

9.3.2 Triangular-Rectangular Coordinates

Figure 9.56 shows a 3D triangular coordinate plot whose first three dimensions are embedded in 2D triangular coordinates and whose fourth dimension (the vertical axis) is represented by a rectangular coordinate system. This representation allows us to show a surface as a function of mixtures of three ingredients. This type of surface was represented by contours in Figure 9.49.

We need a different coordinate function here, however, because the graphic involves a projection from a 4D space to a 3D. The *tri4*() function computes this projection by computing triangular coordinates on the first three dimensions and rectangular on the fourth. We have used the soil data from Figure 9.48 and let the fourth axis represent the depth of the core samples. The plot shows that there is a higher percentage of silt and lower percentage of sand toward the surface, and there is a higher percentage of sand and lower percentage of silt deeper down.

COORD: *rect*(*dim*(4), *triangular*(*dim*(1, 2, 3)))
ELEMENT: *point*(*position*(sand*silt*clay*depth), *color.hue*(plot))
ELEMENT: *surface*(*position*(*smooth.mean.cauchy*(sand*silt*clay*depth)))

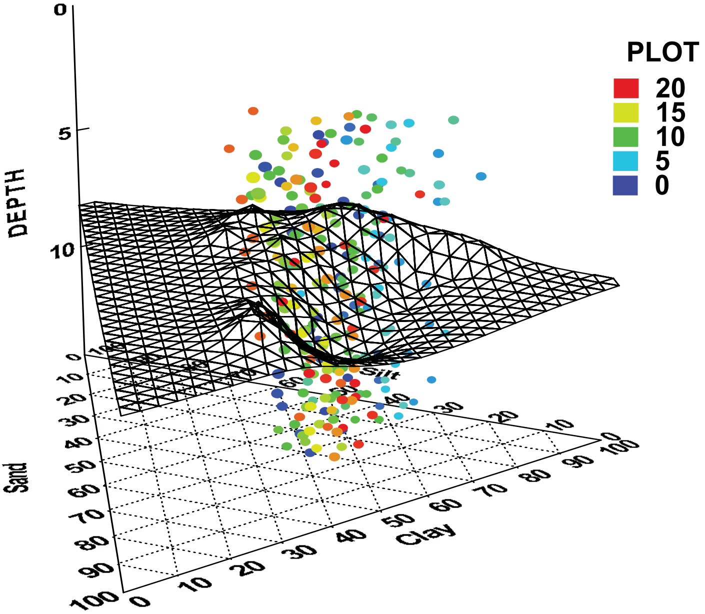


Figure 9.56 *3D triangular/rectangular coordinates*

9.3.3 Cylindrical Coordinates

If (u,v,w) represents the cylindrical coordinates (ρ, θ, ζ) of a point, then a spherical coordinate function $S(u, v, w) \rightarrow (x, y, z)$ corresponds to the case $u = \rho$, $v = \theta$, $w = \zeta$, where $x = \rho \cos\theta$, $y = \rho \sin\theta$, and $z = \zeta$. Cylindrical coordinates are useful for representing cylinders, spirals, and cones.

Krumhansl (1979) analyzed the perception of musical pitch in a tonal context. She presented her multidimensional scaling results in a 3D graphic showing the configuration of musical tones lying on the surface of a cone. Figure 9.57 shows the specification and graphic based on her model.

COORD: *cylindrical*(*dim*(1, 2, 3))
ELEMENT: *point*(*position*(pitch*tone*chroma), *label*(note),
shape(*shape.sphere*), *texture*(*texture.silver*))
ELEMENT: *surface*(*position*(*smooth.linear*(pitch*tone*chroma)),
transparency(.8), *texture*(*texture.silk*), *color*(*color.mauve*)

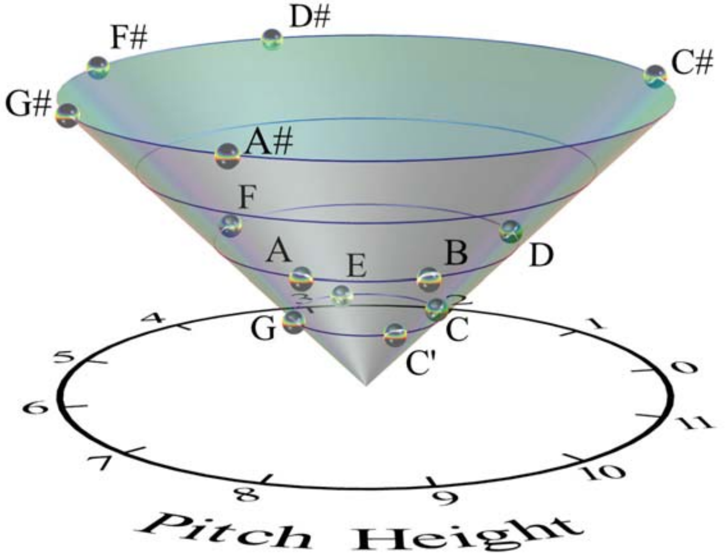


Figure 9.57 *Cylindrical plot of pitch perception (courtesy of H.F. Wessler)*

We have used Krumhansl's data to illustrate the usefulness of a cylindrical coordinate representation for certain configurations. The linear smooth based on a single parameter predicts her results quite well. An equivalent parameterization would be to think of her tone-pitch model as a **modular surface**. The modulus of a complex number is $r = |z|$, the length of z in its vector representation. In this form, her surface is simply $f(z) = |z|$.

Krumhansl's original article contains a simple line drawing that we think is superior to the representation in Figure 9.57. The original figure shows a black-and-white line-drawing outline of the cone with a vertical cut (like an open shirt collar) to highlight the depth of the cone and the fact that the notes do not span a full 360 degrees in one octave. This is probably a case where realism in scientific visualization contributes little to the comprehension of the graphical information, as Becker and Cleveland (1991) have argued.

9.4 High-Dimensional Spaces

The methods we use to represent 2D and 3D spaces graphically do not generalize well to higher dimensions. The curse of dimensionality introduces peculiar problems. Near-neighbor sets of data points are relatively compact in 2D but not generally compact in high-dimensional spaces. Search algorithms that work efficiently in 2D or 3D become intractable in high dimensions. Hastie *et al.* (2001) discuss these problems further.

Three general approaches to representing high-dimensional data graphically have been taken in the visualization and data mining literature. The first is linear and nonlinear projection onto a 2D space. The second is representation by sets of continuous functions of the data. The third is representation through recursive partitioning, which produces nested coordinate spaces.

9.4.1 Projection

One scheme is to employ a linear or nonlinear projection from p-dimensions to a few. This may cause loss of information because a projection onto a subspace is many-to-one. Also, projection is most suitable for displaying points or $\{V, E\}$ graphs. It is less suitable for many other geometric chart types. Nevertheless, some low-dimensional projections have been designed to capture structures contained in subspaces, such as manifolds, simplices, or clusters.

Linear projection from high-dimensional spaces into low-dimensional spaces has a long history in many fields, such as psychology, biology, and astronomy. Despite the differences in names for these methods in various fields, they all depend on the singular decomposition of a real rectangular matrix of the form $\mathbf{X} = \mathbf{UDV}^{\mathrm{T}}$. The columns of **UD** are generally called the **principal components** of **X**. Plotting the first two or three principle components gives us the linear projection we seek. If the first few singular values (diagonal elements of D) are nonzero and the rest near zero, then we conclude that the major variation in the high-dimensional space lies on a hyperplane that we have captured in the projection. Examples can be seen in Section 9.1.3.2, and Section 16.5.4.2.

Several approaches to nonlinear projection have been taken. The first is to relax global structure in order to reveal local. Banchoff (1996) summarizes a variety of applications involving the visualization of higher-dimensional geometric objects. Shepard and Carroll (1966) present a numerical relaxation method for representing on the plane nonlinear configurations of points in higher dimensions.

A second approach is to compute interesting 2D projections and present them singly or jointly. Friedman and Stuetzle (1981) and Friedman (1987) developed loss functions for such problems and a method for minimizing them called **projection pursuit**. Their algorithm has been used, for example, to locate configurations of points bounded by a triangle when projected to a plane. Asimov (1985) and Buja and Asimov (1986) developed a method for computing a series of 2D projections that follow a continuous path through a higher-dimensional space. Animating these projections with a video player in a **Grand Tour** allows a viewer to perceive aspects of higher dimensional structure.

A third approach focuses on identifying low-dimensional manifolds embedded in high-dimensional space. A simple example would be a surface that spirals (like a Swiss roll) or twists (like DNA). Such manifolds are not unwrappable by global warpings or nonlinear projections. By using nearest-neighbor graphs or locally-weighted distance functions, however, we can relax the distortions produced by large distances in a configuration of points. Tenenbaum *et al.* (2000) devised an iterative method of this type. Roweis and Saul (2000) developed a locally linear approximation for this purpose. He and Niyogi (2002) implemented an eigendecomposition of the Laplacian matrix derived from a *k*-nearest neighbor graph on the points.

Probably the most widely used nonlinear projection method is **multidimensional scaling** (MDS). Figure 9.58 shows a multidimensional scaling of the world countries data. We have selected five variables: birth rate, death rate, plus per-capita annual investment in education, health care, and military. Through the multidimensional scaling projection, we are collapsing a five-dimensional space to a two-dimensional space.

```
COORD: project.mds(dim(1, 2))
ELEMENT: point(position(birth*death*education*health*military),
         color(gov))
```

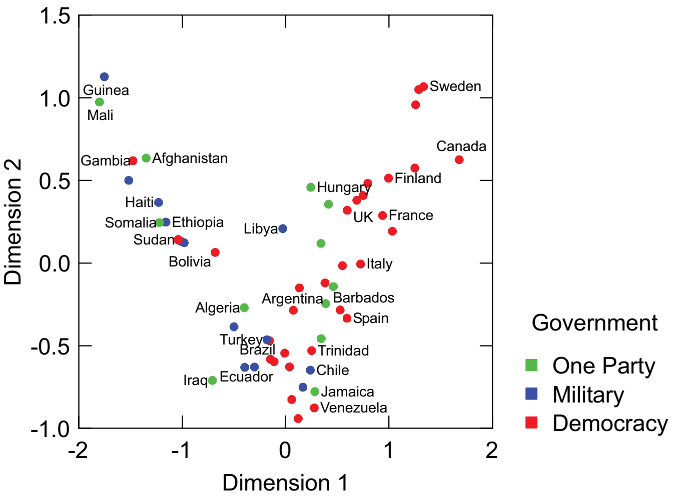


***Figure 9.58** Multidimensional scaling of world countries*

Many analysts attempt to interpret the axes of a multidimensional scaling as if they were principal components or simple linear scales, but this is usually a mistake. Figure 9.58 is a good illustration of this pitfall. Because the points cover the ranges of both axes, it would appear that we have two relatively independent dimensions in this plot. In fact, we have only one. There is an almost perfect "V" pattern in the distribution of points, which indicates that the solution is topologically one-dimensional. This is a well-known phenomenon in nonmetric, iterative MDS. One-dimensional data tend to distribute in a horseshoe or "V" pattern when scaled this way in two dimensions.

If we trace a path along the "V" in Figure 9.58, we can see that the countries run from liberal/socialist democracies in the upper right to conservative/militaristic autocracies in the upper left. Thus, there is little point to interpreting the axes or trying to understand the meaning of the composite variables. The essential information is concerned with what countries are similar to each other and what is the overall shape of the manifold in which the countries are embedded.

9.4.2 Sets of Functions.

A second possibility for dealing with high-dimensional data is to map a set of n points in $\boldsymbol{R}^p$ one-to-one to a set of n functions in $\boldsymbol{R}^2$. A particularly useful class of functions is formed by taking the first p terms in a Fourier series as coefficients for $(x_1, ..., x_p)$. Another useful class is the set of Chebysheff orthogonal polynomials. The most popular class is the set of p piecewise linear functions with $(x_1, ..., x_p)$ as knots, often called **parallel coordinates**. An advantage of function space representations is that there is no loss of information, since the set of all possible functions for each of these types in $\boldsymbol{R}^2$ is infinite. Orthogonal functions (such as Fourier and Chebysheff) are useful because zero inner products are evidence of linear independence. Parallel coordinates are useful because it is relatively easy to decode values on particular variables. A disadvantage of functional representations is that manifolds and distances are difficult to discern.

9.4.2.1 Fourier Functions

Andrews (1972) proposed using a Fourier function for summarizing data. The function Andrews used is

$$f(t) = \frac{x_1}{\sqrt{2}} + x_2 \sin(t) + x_3 \cos(t) + x_4 \sin(2t) + x_5 \cos(2t) + \dots$$

where $\boldsymbol{x}$ is a p-dimensional variate and t varies continuously from $-\pi$ to π. Figure 9.59 shows an Andrews Fourier plot of the countries data. See Figure 10.43 for an example of polar Fourier plots, called *blobs*.

COORD: *fourier*()
ELEMENT: *line*(*position*(birth*death*education*health*military),
color(gov))

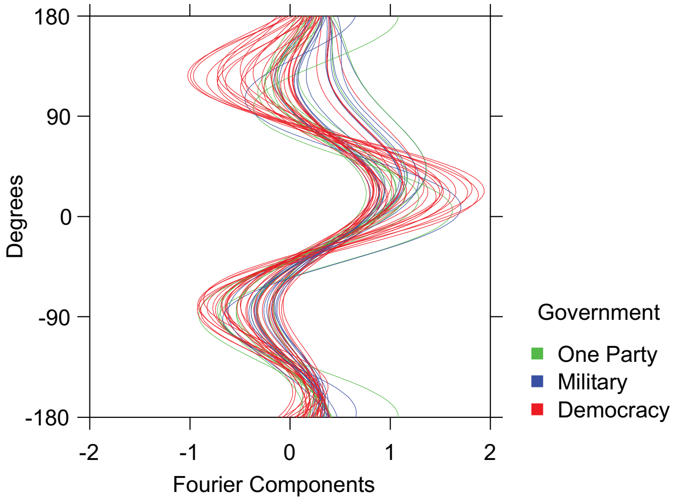


Figure 9.59 *Fourier coordinates*

9.4.2.2 Parallel Coordinates

A popular chart that has existed for over a century represents multivariate data with profiles drawn in a simple line graph. For example, psychologists and educators have traditionally represented multidimensional test scores with diagnostic profiles. The tests are ordered on a horizontal axis and the standardized scores on a vertical axis. By standardizing the scores, all the separate scales are given a common domain. Horizontally oriented zigzag lines are used to connect the scores so that more than one profile can be represented in the same plot.

Inselberg (1984) generalized this idea by assigning a separate, parallel axis to each dimension. Even though the parallel coordinate plot is almost indistinguishable from a profile plot, what made the contribution novel was the recognition of the dual between a high-dimensional space and the 2D parallel coordinate space. Wegman (1990) went further and examined the relation between estimating densities in high-dimensional space and in parallel coordinate space.

Figure 9.60 contains a parallel coordinate plot on the world countries data. The *line* graphic is one of the few that is of any use in this coordinate system. Most other graphics go to points on each line. Even if we colored them, they would be difficult to recognize.

COORD: *parallel*()
ELEMENT: *line*(*position*(birth*death*education*health*military),
color(gov))

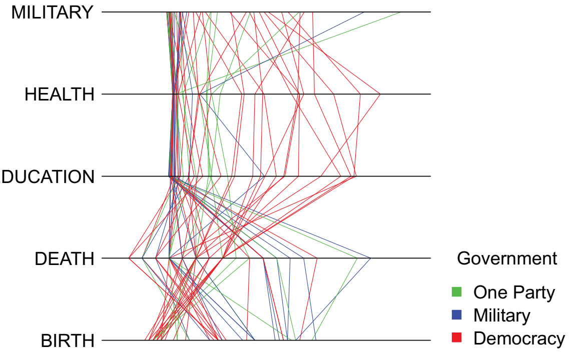


Figure 9.60 *Parallel coordinates*

9.4.2.3 Polar Parallel Coordinates

We can send parallel coordinates through a polar transformation to produce something called a **spider web** or **star plot**. Figure 9.61 shows an example using the data from the previous Figure 9.60. This plot is analogous to the radar plot shown in Figure 9.29.

COORD: *polar*(*parallel*())
ELEMENT: *line*(*position*(birth*death*education*health*military),
color(gov))

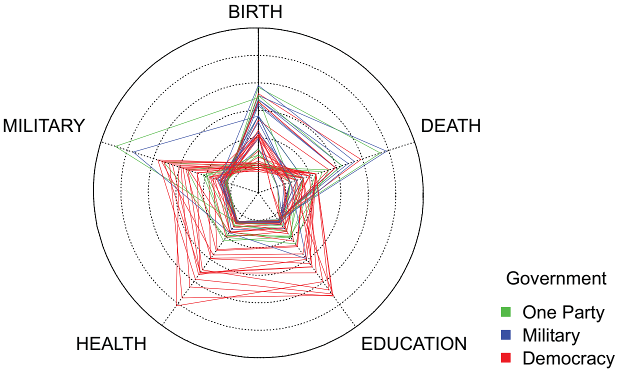


Figure 9.61 *Parallel coordinates in polar form*

9.4.2.4 Recursive Partitioning.

A third approach to representing high-dimensional data is recursive partitioning. We choose an interval $[u_1, u_2]$ and partition the first dimension of $\boldsymbol{R}^p$ into a set of disjoint intervals each of size $(u_2 - u_1)$, in the same manner as histogram binning. This yields a set of rectangular subspaces of $\boldsymbol{R}^p$. We then partition the second dimension of $\boldsymbol{R}^p$ similarly. This second partition produces a set of rectangular subspaces within each of the previous subspaces. We continue choosing intervals and partitioning until we finish the last dimension. We then plot each subspace in an ordering that follows the ancestry of the partitioning. Recursive partitioning layout schemes have appeared in many guises: $\boldsymbol{R}^p \rightarrow \boldsymbol{R}^3$ (Feiner and Beshers, 1990), $\boldsymbol{R}^p \rightarrow \boldsymbol{R}^2$ (Mihalisin *et al.*, 1991), $\boldsymbol{R}^4 \rightarrow \boldsymbol{R}^2$ (Becker *et al.,* 1996).

There are several modifications we may make to this scheme. First, if a dimension is based on a categorical variable, then we assume $(u_2 - u_1) = 1$, which assures one partition per category. Second, we need not partition a dimension into equal intervals; instead, we can make $[u_1, u_2]$ adaptive to the density of the data (see Section 7.2.4.2). Third, we can choose a variety of layouts for displaying the nodes of the partitioning tree. We can display the cells as an *n*-ary tree, which is the method used by popular decision-tree programs. Or, we can alternate odd/even dimensions by plotting horizontally/vertically. This display produces a 2D nested table, which has been variously named a *mosaic* (Hartigan and Kleiner, 1981) or *treemap* (Johnson and Schneiderman, 1991).

Like simple projection, this method can cause loss of information because aggregation occurs within cells. Nevertheless, it yields an interpretable 2D plot that is familiar to readers of tables. Because recursive partitioning works with either continuous or categorical variables, there is no display distinction between a table and a chart. This equivalence between tables and graphs has been noted in other contexts (Shoshani, 1997; Pedersen *et al.*, 2002). With recursive partitioning, we can display tables of charts and charts of tables.

9.5 Tools and Coordinates

A **brush** is a bounded region inside a frame that is movable via a translation controller such that all points inside it are highlighted in other graphics based on the same data (Cleveland and McGill, 1988). This linking is accomplished through the relational key index (case number) associated with a variable set as defined in Chapter 2. This tool allows a user to select points in one plot, for example, and see them highlighted in another. In a continuous mapping, we would expect to see points that are close together in one graphic (inside the brush region) close in the other, but this is not generally the case with statistical graphics. Two cars may have similar acceleration but differ in weight or fuel economy. A brush helps us to perceive these second-order relationships.

In a rectangular coordinate scatterplot, a brush is usually a square. It demarcates intervals on two orthogonal scales. What shape would this brush assume in triangular coordinates? Did you guess triangle? What shape would this brush have in polar coordinates? Hint: it's not a circle. A brush does not change size or shape when moved in rectangular coordinate space. It is translated. What happens in polar coordinates? What is the polar equivalent of movement along the vertical rectangular axis? Horizontal? Diagonal? Try sending a square brush through the various other coordinate transformations in this chapter.

Some writers have suggested that circular brushes would be more appropriate than square ones for rectangular coordinates; a circle is the iso-distance contour on the Euclidean plane. How would a circular brush behave in polar coordinates? Would this make more sense than the transformed square? Finally, how do coordinate transformations affect the behavior of other selection tools? Some software implements a lasso tool that allows the user to select points within any closed region, for example. Would that require any special programming attention under coordinate transformations?

9.6 Sequel

So far, we have graphs that describe functions, but we cannot perceive them. The next chapter presents functions that link graphs to aesthetic attributes and thus make graphs perceivable as graphics.

10

Aesthetics

The term **aesthetics** derives from the Greek αἴσθησις, which means perception. The derivative modern meanings of beauty, taste, and artistic criteria arose in the 18th century. We have chosen the name *aesthetics* to describe an object in our graphical system because of its original connotations and because the modern word *perception* is subjective rather than objective; perception refers to the perceiver rather than the object. Aesthetics turn graphs into graphics so that they are perceivable, but they are not the perceptions themselves. A modern psychologist would most likely call aesthetics in this sense *stimuli*, *aspects*, or *features*, but these words are less precise for our purposes.

The preceding chapters have discussed the components of the system for producing graphs, but up to this point we having nothing to show. Without aesthetics, graphs are invisible, silent, indeed imperceptible. Aesthetics are functions that govern how a graph is represented as a visible or otherwise perceivable graphic.

Since this book focuses on developing statistical graphics systems that can transmit measurable multidimensional information to perceivers, the cognitive and perceptual psychological research on scaling aesthetics is relevant. The field of perception research is wide and long-standing. Its contemporary roots are in 19th-century medicine and philosophy. Its recent growth has been nourished by neuroscience. For an introduction, see Shiffman (1990), Levine (2000), or Anderson (1995).

To make a statistical graphics system, we need to map qualitative and quantitative scales to sensory aspects of physical stimuli. Each dimension of a graph must be represented by an aesthetic attribute such as color or sound. We will first examine problems in the mapping of continuous scales. Then we will examine similar issues with categorical scales. Next, we will look at problems introduced when we work with multiple dimensions. Then we will discuss the role of realism in constructing graphics.

Finally, we will discuss specific aesthetic attributes. We have extended the classification system of Bertin (1967, 1977), who, while not a psychologist, has formalized attributes in a system that can be related to psychological theories of perception. Indeed, while Bertin's work is based entirely on visual displays, his variables apply, with small modifications, to other sensory modalities. For a psychological perspective on Bertin's work, see Kosslyn (1985). For a review of cartographers' extensions of Bertin's system, see MacEachren (1995)

10.1 Continuous Scales

When we map real numbers to the intensity of a physical stimulus, how do we know that magnitude will be perceived as a linear function of the values? We know from physics, for example, that the brightness of a light source decreases as the square of its distance from a receptor. Even if we measure brightness at the retina, however, neural processes may nonlinearize, truncate, or otherwise filter the signal. We will examine this problem in the next section and then consider its implication for the use of aesthetic attributes.

10.1.1 Psychophysics

The field of **psychophysics** is devoted to relating the magnitude of a physical stimulus to the intensity of a perception (Stevens, 1985; Falmagne, 1985). The German biologist and physicist Gustav Theodor Fechner coined this term in his *Elemente der Psychophysik* (1860). Although Fechner spent most of his career on metaphysics and religion and his psychophysical work focused on human perception, there is nothing in psychophysical theory that would limit it to humans. Psychophysics in the general sense involves mapping the intensity of a quantitative stimulus (light, sound, motion) to the response of a sensing system (human, insect, computer).

Fechner built his theory on an observation of one of his medical professors, Ernst Heinrich Weber. Weber had discovered that the change in magnitude of a stimulus needed to produce a **just noticeable difference** (JND) in sensation appeared to be a constant. For example, if we place a kilogram weight in each of your hands you will most likely perceive no difference in weight between the two. If we add more weight to one hand, you will just begin to notice the difference at around 1.1 vs. 1 kilograms. Similarly, if we place ten kilograms in each of your hands, we will need to add approximately one kilogram to one hand for you to begin to notice the difference. In both cases, the ratio of the difference to the magnitude is 0.1. This ratio differs for individuals and for kinds of stimuli, but the phenomenon of constancy within experiment is remarkable.

Figure 10.1 (adapted from Levine, 2000) shows a graph of a function based on Weber's observation. Each unit increase in sensation (δS) occurs after the stimulus intensity increases by a JND (δI). In other words,

$$\delta S = k\left(\frac{\delta I}{I}\right)$$

The left panel of the figure shows the graph for a fixed value of δS. The units of the scales are arbitrary. It is the shape of the function that led Fechner to his formulation.

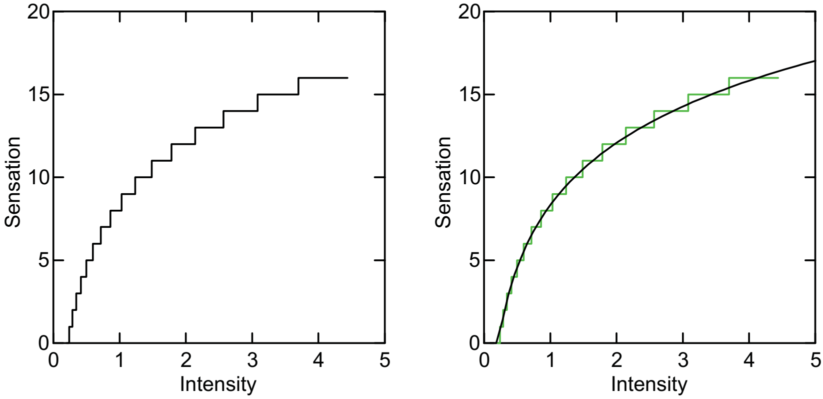


Figure 10.1 *JND's following Fechner–Weber law*

Fechner integrated the function

$$\delta S / \delta I = k / I$$

to yield

$$S = k\, log(I) + c$$

Fechner called this Weber's law, but it really was Fechner's, based on Weber's observation. This logarithmic function is plotted in the right panel of Figure 10.1, with $c = 8$ and $k = 5$.

Boring (1950) discusses the assumptions Fechner made, including the potential for negative sensation, since his log function had no limits. This and other questionable assumptions caused a sensation of its own and led to controversies which have persisted for over 100 years. Nevertheless, Fechner's unique achievement was to infer from Weber's simple observation the shape of the psychophysical function without having to measure perceived intensity.

Weber's discovery and Fechner's formulation prompted other researchers to consider measuring sensation directly. Plateau (1872) had artists paint a gray patch midway between black and white under different levels of illumination. On observing that all the patches were virtually the same gray, he proposed the power function

$$S = kI^p$$

S.S. Stevens (1961) tested Plateau's function with numerous experiments employing **direct scaling**. Instead of asking subjects to report when a JND occurred, Stevens requested direct numerical estimates of the ratio of intensities of different stimuli. His model fit observed data across a wide variety of subjects, intensities, modes of stimuli, and measurement methods.

For Weber–Fechner, equal stimulus ratios produce equal sensation differences. For Plateau–Stevens, equal stimulus ratios produce equal subjective ratios. The reason for the discrepancy between their models is that they begin with a different latent variable. Both infer sensation rather than measure it directly. Because of this difference in definition and because both are based on a latent variable, scientists are still unable to reject conclusively one or the other model, despite numerous claims to the contrary. In the end, neither model may be correct, or even more likely, neither may be of much use to a theory of perception (Ekman, 1964; Gregson, 1988; Lockhead, 1992).

For our purposes, these functions tell us that intensity may not be linearly related to sensation. Figure 10.2 shows power-function psychophysical curves for a variety of stimuli using data from Shiffman (1990), Stevens (1961), and other sources. Note that electric shock has an exponent greater than 3. For this and other reasons, shock would not be a good aesthetic candidate for representing a dimension in a statistical graphic.

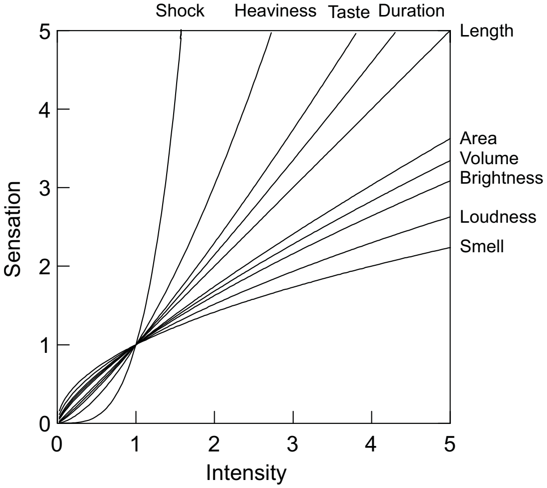


Figure 10.2 *Psychophysical functions for various stimuli*

10.1.2 Consequences for Attributes

As we have seen, different aesthetic attributes will more-or-less nonlinearize our signal. Should we adjust for this by including the Stevens exponents in the scaling of graphics in an automated statistical graphics system? The recommendation has been made at various times by both psychologists and statisticians, but it would be a mistake. Unfortunately, even numbers themselves behave like other stimuli in psychophysical experiments (Schneider *et al.*, 1974). So does numerosity (Krueger, 1982). If our goal for a statistical graphics system is to communicate accurately quantitative or qualitative information, should we root all our displayed numbers? This would certainly be a radical change in the output of statistical packages! Most certainly, however, the presence of bias in human information processing does not imply that we should normalize the physical world to an inferred perceptual world. As we will see in Section 10.3, nonlinearities have their uses in perceiving objects. There are ways to accommodate them.

One way is to ensure that the range of variation in an aesthetic attribute does not exceed too many orders of magnitude in intensity. If there is nonlinearity, clamping limits the psychophysical response function to a roughly linear segment. We might worry that our effect is too subtle in these circumstances, but we have to acknowledge the speed–accuracy trade-off in the perceptual process. If our goal is to emphasize differences, then we should be more willing to accept bias.

Another way is to live with the bias. With positively skewed data, we can turn a sow's ear into a silk purse. Using circle areas to represent magnitude of a third variable on a scatterplot, we will de-emphasize extremely large values because their areas will be less salient (the Stevens exponent for area is .8). If this is the effect we wish to achieve, particularly with positively skewed data, then a **bubble plot** like this might be the best representation, despite admonitions to the contrary (Cleveland, Harris, and McGill, 1981).

Finally, we can choose attributes that are closer to a Stevens exponent of 1 when we are concerned about linearity. This approach underlies Cleveland's (1985) recommended hierarchy of graphical elements. Figure 10.3 displays this hierarchy, derived from a series of graphical element perception studies by Cleveland and his associates. In most of these experiments, Cleveland used a paradigm not unlike direct scaling. He simply compared subjects' numerical judgments with the generating stimulus magnitude. The rank order of Cleveland's elements corresponds roughly to the rank of the absolute differences between their Stevens exponents (averaged across different studies) and 1. Additional variation contributing to Cleveland's results is probably due to the effects of visual illusions (especially with angular judgments).

Cleveland's hierarchy has sometimes been recommended as a general criterion for evaluating statistical graphics (*e.g.*, Wallgren *et al.*, 1996). Several researchers have found that Cleveland's results are contextual, however. Spence and Lewandowsky (1991), for example, found support for the predictions of the hierarchy only in the initial, pre-attentive stages of processing where rapid and direct evaluation of magnitude was required. Simkin and Hastie (1987) found that accuracy of comparative, magnitude, and ratio judgments depends on types of graphic elements and types of tasks. And Carswell (1992) found support for the hierarchy mainly when attention was focused on a portion of a graphic. Kosslyn (1994) discusses these and other findings concerning the hierarchy at greater length. It must be remembered, however, that

Cleveland's original intention was to evaluate different elements when isolated, out of context. Furthermore, the task was deliberately restricted to magnitude and ratio comparisons. Such restriction is often desirable when attempting new research in an ill-defined field. Like all psychophysical approaches that isolate stimuli in order to examine their psychometric functions, however, the results apply only to certain restricted, indeed artificial, situations. As we shall see in Section 10.4, cognitive psychologists have recently turned away from psychophysics and toward a more integrated, ecological approach for just this reason.

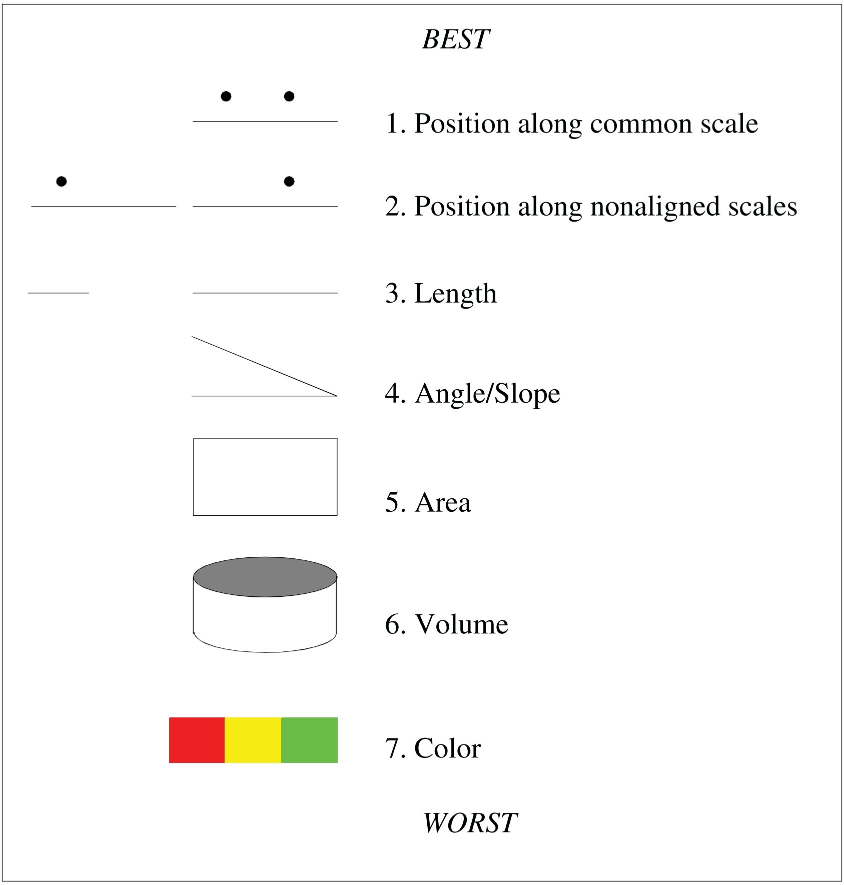

BEST
1. Position along common scale
2. Position along nonaligned scales
3. Length
4. Angle/Slope
5. Area
6. Volume
7. Color
WORST

Figure 10.3 *Cleveland graphic elements hierarchy*

10.2 Categorical Scales

If we assign the values of an attribute, such as color, to a set of categories, how do we know that a person will perceive categories? For example, we might assign two shades of red to a scale distinguishing males and females on a gender variable. Is it enough simply for these shades to differ by at least one JND, or will we do better assigning, say, red and green? Are there color categories at all, or is color perceived simply as a continuum in three dimensions? Indeed, does perceptual categorization itself exist, or do we simply respond to a continuum of perceived similarity among stimuli?

We take answers to many of these questions for granted when we place legends on graphics or use tick marks to separate categories on an axis. The system described in this book depends on mapping real numbers (for continuous scales) or integers (for categorical scales) to perceptual attributes. If the perceptual result is the same in both cases, then this aesthetic distinction cannot be supported. For answers, we need to consult the field of categorical perception.

Categories enable us to recognize objects in the world. Consider two berries, one black, one blue, both otherwise indistinguishable to the eye. The chokeberry is poisonous, the blueberry delicious. Color categorization allows us to distinguish them. We identify the poisonous berry as different from the tasty even though they appear identical on most available sensory attributes (size, shape, firmness, etc.). The consequences of our mistakes in categorization can be themselves categorical: we eat and die (or die and are eaten).

We should not assume that the ability to make effective categorical judgments in the world implies a categorical perceptual mechanism, however. We need instead to examine whether there is experimental evidence for categorical processing and then see if it can affect the assignment of attributes to categorical scales. First, we will review the evidence for innate categorical processing and then examine learned categories.

10.2.1 Innate Categories

Aristotle made categorization basic to his logic. His categories — substance, quantity, quality, relation, place, time, position, state, action, and affection — formed the basis of simple propositions whose truth could not be determined by logic. Although Aristotle did not claim his categories were self-evident in all cases, they were nevertheless taken to be real by most philosophers and theologians until the medieval Nominalists challenged their meaning. Categories, the Nominalists claimed, are by-products of expressions of similarity and difference between particular things, but they have no reality in themselves. There is an echo of this distinction in recent psychological theorizing on whether **prototypes** (Realist) or **exemplars** (Nominalist) underlie categorization in memory.

We can evade the philosophical question of whether categories are real and still ask if categorization can be perceptually innate. The Gestalt psychologists were driven by the belief that in perception, the whole is different from the sum of its parts (Wertheimer, 1958). Several Gestalt principles suggest an innateness to visual categorization. The **principle of proximity**, for example, states that things that are close to each other seem to belong together. The dot pattern in Figure 10.4, for example, is seen as comprising four pairs and a single rather than three triples.

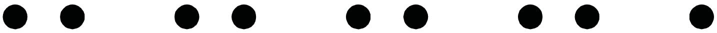


Figure 10.4 *Grouping by proximity*

Figure 10.5 shows that this principle of proximity is neurological, organized elsewhere in the visual system than in the stimulus or retinal image itself. In both halves of this figure, the dots are closer to their neighbors horizontally than vertically. Viewed separately, these figures show three rows of three dots. Viewed as a stereogram, however, three columns emerge because the differences in the perceived depths of the dots are greater than the vertical differences. Proximity applies to the perceived rather than retinal image. (Not everyone can view the stereogram easily. It requires crossing the eyes until the images blend into a third, central square. See Chapter 9).

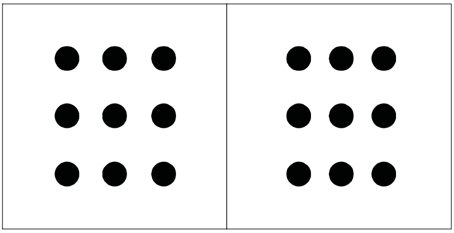


Figure 10.5 *Stereogram from Kaufman (1974)*

Developmental psychologists have uncovered evidence which suggests innate categorization (Bornstein, 1987). A categorical response to color exists in humans shortly after birth, well before language is available to name the categories (Bornstein, Kessen, and Weiskopf, 1976). This result is consistent with the findings on color categorization in the cat visual system (deValois and Jacobs, 1968). Eimas (1974) has found similar categorical boundaries in the processing of sound. From an evolutionary perspective, pre-wired categorical perception would make sense. Some skills may be too critical for survival to leave them to learning alone.

10.2.2 Learned Categories

Categories do not have to be innate to shape perception. There is also evidence that categories learned on a dimension influence perception of stimuli invoking that dimension. Whorf (1941) is best known for inferring this phenomenon for language. Observing that Eskimos have many different words for snow, while English speakers have only one, Whorf concluded that they perceive this category differently from English speakers.

Although Whorf's hypothesis has not been widely verified by anthropologists, there is laboratory evidence for learned perceptual categorization effects (Harnad, 1987; Gibson, 1991; Goldstone, 1994). Burns and Ward (1978), for example, found that expert musicians recognized categories in pitch that novices could not detect. Categorical perception is especially pronounced in the recognition of speech. Lisker and Abramson (1970) used a computer to generate sounds continuously varying between the voiced and unvoiced phonemes [b] and [p]. English–speaking listeners showed a sharp switch from recognizing one to the other. Figure 10.6 summarizes their result.

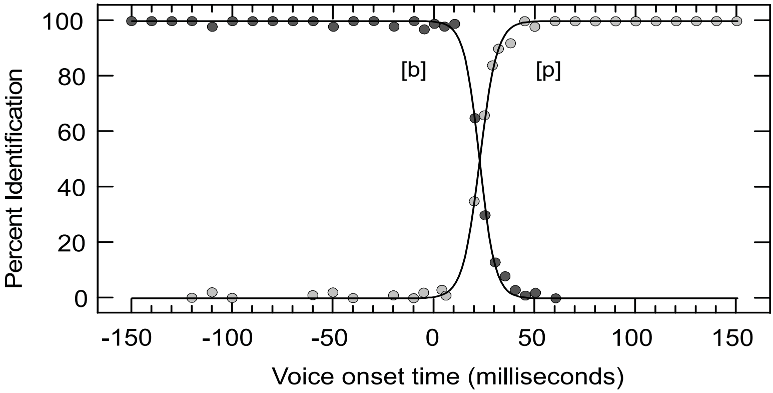


Figure 10.6 *Percentage identification of [b] vs. [p] as function of voice onset time (adapted from Lisker and Abramson, 1970)*

10.2.3 Scaling Categories

Although it was not done in the original study, we fit logistic distributions to the two sets of data in Figure 10.6. The fit of the functions to these data is extremely good. The curves show that, although it is a discrete concept, categorization can be modeled with a continuous probability distribution. **Signal detection theory** has been used to parameterize categorical responding based on continuous probability distributions (Swets, Tanner, and Birdsall, 1961). From another perspective, Massaro (1992) has argued that the observation of sharp boundaries between categories is not sufficient evidence for concluding that categorical perception exists. He has modeled sharp identification boundaries using a Fuzzy Logical Model of Perception (FLMP).

One can always fit a continuous distribution with a steep slope parameter to model a categorical response. Nevertheless, the slopes of the curves in Figure 10.6 are substantially steeper than those for most continuously varying stimuli. As with the Weber–Fechner vs. Plateau–Stevens debate, it is not likely that the continuous-categorical scaling question will be resolved soon by mathematical modeling. What matters most for our purposes is that perceivers bring a different set of perceptual biases to categorical stimuli than they do to continuous.

10.2.4 Consequences for Attributes

That perceivers may respond to stimuli categorically creates several problems for categorical scaling of graphics. We may have the misfortune, for example, to assign two categories to two different colors that are perceived as belonging to a single color category. In general, natural category boundaries for a stimulus may interfere with the values we assign to our artificial categories. This is more likely to happen when we attempt to represent a large number of categories in a graphic. With a small number of categories, we can avoid problems by choosing as wide a range as possible for the stimulus dimension.

Even if our assignments are perceived as one-to-one, some categories may dominate our categorical scale by appearing more salient. One way this can occur is because of their perceived frequency. Tversky and Kahneman (1974) have shown that base rates can overwhelm other evidence in attributions and quantitative judgments involving categories. Rips and Collins (1993) show that category frequency can influence category judgments even more than perceived similarity. We need to be careful that this salience-bias does not conflict with the categories we are attempting to emphasize in graphing our data.

Category ideal types (prototypes) may subsume or draw other categories to themselves and thus change our intended categorization. This can occur at the earliest perceptual stages or the latest. Rosch (1975) showed, for example, that category judgments are faster for instances that are close to typical category members than atypical. Stevens and Coupe (1978) introduced nonlinear boundaries on hypothetical maps and were able to distort subjects' distance judgments and category memberships on the remembered maps. Tversky and Schiano (1989) induced a variety of similar distortions in memory for maps by manipulating categorical dimensions.

Finally, as we have seen, categories can underlie apparently continuous perceptual dimensions, through either innate or learned processes. For example, simple cells in the visual cortex are organized to detect, among other things, orientation. The most common receptive field maps of these cells involve vertical, horizontal, and diagonal patterns (Hubel and Wiesel, 1962). Stimuli presented at angles near these directions will tend to cause firing in these receptors. Thus, our best angular sensitivity is near these canonical orientations. This may account for the relatively poor performance of angle/slope in the Cleveland hierarchy (see Figure 10.3) and for why slope is most accurately judged at 45 degrees in statistical line charts (Cleveland, McGill, and McGill, 1988).

10.3 Dimensions

What happens when we combine several scales in a single display? Can we represent one quantitative dimension with color and another with orientation and expect a perceiver to respond to both dimensions? If so, how many dimensions can we represent without producing perceptual chaos? Five or six quantitative dimensions are commonplace in scientific visualization. Does this make psychological sense?

10.3.1 Integral Versus Separable Dimensions

The Gestaltists were interested primarily in **wholistic perception**, in which multidimensional stimuli were perceived in an unanalyzable whole. At the other pole were classical psychophysicists, who believed that the perception of multidimensional stimuli involved a summation or aggregation of separate unidimensional percepts. While this is an oversimplification, the synthesis of these positions came from a few researchers who believed that complex stimuli could be analyzed in their parts without resorting to simple functions of dimensional attributes such as summation or weighted linear composites. Wendell Garner (Garner, 1970, 1974; Garner and Felfoldy, 1970) has been one of the pioneers in this effort.

To Garner, “a configuration has properties that have to be expressed as some form of interaction or interrelation between the components, be they features or dimensions” (Garner, 1981). The analysis of configuration implies more than paying attention to covariation, however. A configural property exists in addition to, not as a consequence of, other properties such as dimensions. Specifically, Garner distinguished **integral dimensions** — such as hue vs. brightness — from **separable dimensions** — such as size vs. texture. Integral dimensions are not as easily decomposable by perceivers as separable.

This distinction has several important practical consequences, supported by numerous studies. First, discriminations between classes defined by multiple integral stimuli are more difficult than those between classes defined by separable stimuli when the perceiver must attend to only one dimension. Selective attention to only one dimension, in other words, is more difficult with integral stimuli. Second, discrimination between classes using all dimensions together is easier for integral stimuli than for separable. Third, redundant cues in classification improve performance for integral stimuli and degrade performance for separable.

The implications of these findings for assignment of aesthetic attributes to dimensions in graphics are several. First, if we wish to facilitate comparisons among and within subgroups, it is better to assign relatively uncorrelated variables to separable aesthetic dimensions. Wainer and Francolini (1984) show, for example, that using integral dimensions in color choropleth maps makes it almost impossible to decode quantitative information separately for the color dimensions. Figure 10.7 demonstrates this phenomenon for symbols. The data are a subset of the barley yield dataset from Fisher (1935). As part of a larger experiment, three strains of barley (Manchuria, No. 475, Wisconsin) were grown at three Minnesota sites (Duluth, Crookston, Waseca). The left panel of the figure uses a double symbol to represent each of the nine combinations of strain and site. Because the symbols are configural (the pair constitutes a "super symbol" with its own recognizable shape), distinguishing the trend in growth within site or within strain is extremely difficult.

The right panel uses symbol and size to represent strain and site, respectively. We are able to focus on symbol type to notice that growth increases across sites both years, on average, within strains as they are ordered. Similarly, we can focus on size to notice that growth increases by strains within sites, as they are ordered. There are better ways to organize a display of these data showing this additive growth increase by strain and site (*e.g.*, Cleveland, 1996), but when multiple coding of symbols is necessary, this example shows that the choice of attributes is critical.

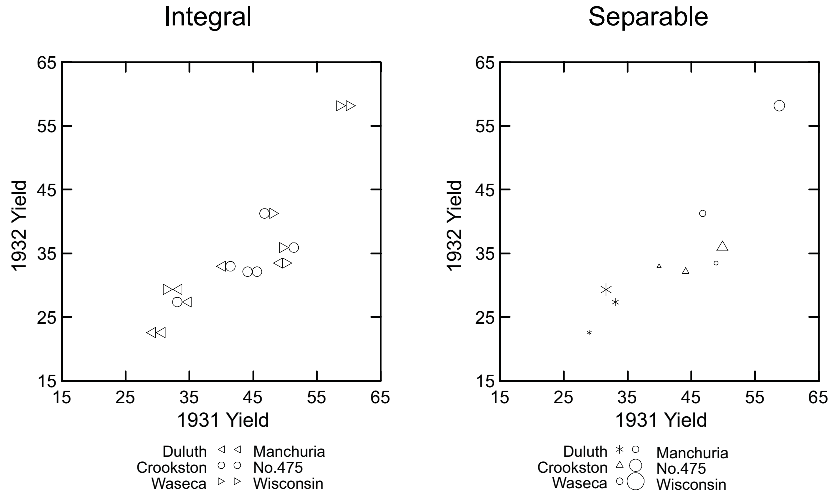


Figure 10.7 *Using separable dimensions for independent discrimination*

If we do need to discriminate groups jointly on several variables, then integral dimensions can help us facilitate the discrimination. Figure 10.8 shows a FACES display (Chernoff, 1973) of the cities data used in Chapter 7. Each variable is assigned to a different feature of the face. The rating of a city's health-care facilities is assigned to the curvature of the mouth, for example, and pleasantness of climate is assigned to angle of the eyebrow. Section 10.6.3.3 covers the *glyph* functions needed to produce the specification. We have inserted them for reference, but you should ignore them for now.

We perceive each face as a whole. The primary component of our perception is emotional, but there are structural elements as well, such as the length and width of the nose. The face is integral because we cannot separate these components easily. A sad face with a long nose (Orlando) does not have the same emotional impact as a sad face with a short nose (Anchorage). This wholeness confounds the measured variables but helps us to recognize the similarity between Los Angeles and San Francisco, or San Diego and Oakland. Because the dimensions are integral, there is less load on working memory for making these comparisons and recognizing clusters. In other words, the display offers us a way to do perceptual clustering (Tidmore and Turner, 1977).

Since its original presentation by Chernoff, FACES have been widely misunderstood by statisticians and data analysts. Some have dismissed them for their lack of seriousness. For example, Haber and Wilkinson (1982) proposed a real-time multivariate FACES display for command-and-control facilities where a limited number of different rapid decisions needed to be made in a multi-cue environment. The proposal received good technical ratings but was ultimately rejected for fear of it receiving a "Golden Fleece" award. Rapid, multi-attribute decision-making with a limited response repertoire is exactly where we would expect FACES to excel.

Others, particularly statisticians, consider FACES subjective, not realizing that facial perception is more consistent and innate than many other types of complex object perception. Specific neuronal responses have been found for both physical and emotional facial dimensions (Young and Yamane, 1992). The recognition of emotional expressions is well developed in human newborns and infants (Haith, 1980; Johnson and Morton, 1991). Faces are readily categorizable into prototypes (Reed, 1972). And faces appear to be the most memorizable stimulus in psychological research. Standing (1973) has demonstrated recognition memory for up to 10,000 faces viewed in brief sessions. Bahrick *et al.* (1975) found extraordinary memory persistence for faces vs. other types of stimuli. Spoehr and Lehmkuhle (1982) summarize other studies showing the perceptual uniqueness of faces. Finally, Wilkinson (1982) showed that FACES outperform other graphical glyphs when used as cues in multivariate similarity judgments.

There are two caveats concerning FACES, however. First, FACES have little utility when decoding information for separate variables is required. This problem is shared by all graphic methods which map quantitative variables to highly integral stimuli. Second, the assignment of quantitative variables to facial features is critical (Chernoff, 1975; De Soete, 1986). This is a problem with configural stimuli in general, but especially so for FACES. Assigning variables with relatively large variance to facial features with relatively small perceptual salience (or vice versa) can result in false-negative or false-positive classification errors. The most successful recent automated facial recognition models (*e.g.*, Turk and Pentland, 1991) distill facial images to a relatively small number of **eigenfaces**. This research suggests that the overall facial configuration is more important to classification than isolated features. A promising approach to assignment would be to ensure that the eigenstructure of the quantitative variables matched as closely as possible the eigenstructure of the perceived set of faces.

ELEMENT: *point*(*position*(col*row), *shape.glyph.face*(health..arts), *label*(city))

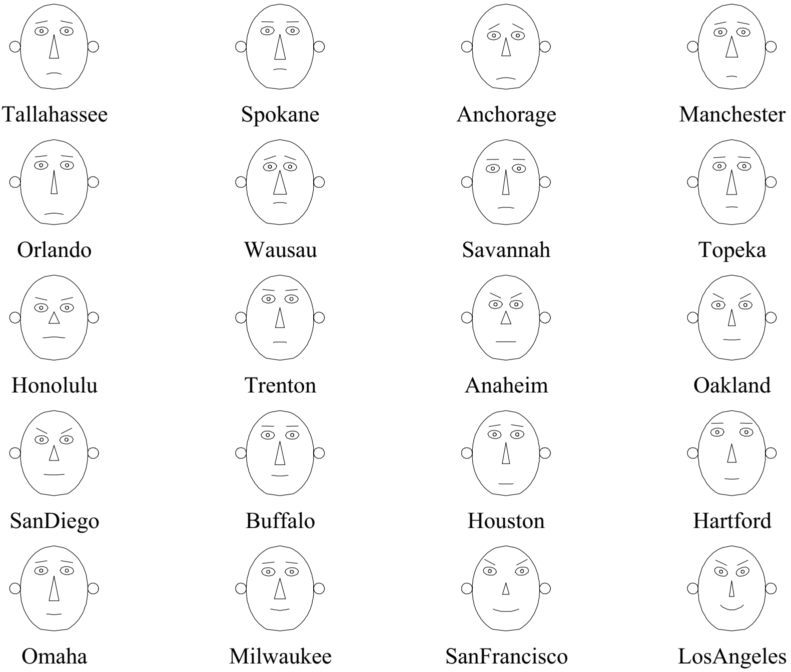


Figure 10.8 *Using integral dimensions to classify*

The fundamental misunderstanding among many critics of FACES, however, has most to do with a failure to appreciate the difference between integral and separable dimensions. To repeat, selective attention is *more* difficult with integral than separable stimuli but discrimination using all dimensions is *easier* for integral stimuli than for separable. We can see this by comparing FACES to glyphs based on more separable perceptual dimensions.

Figure 10.9 shows the same data displayed with bar glyphs. Each city's value on each variable is now measurable by the height of each bar. The bar glyphs make it easier to compare the health rating of Honolulu to that of Trenton, but it is more difficult to recognize that the two cities are quite different in their profile across all nine variables. Try comparing the faces for these cities in Figure 10.8 with the bars in Figure 10.9.

ELEMENT: *point*(*position*(col*row), *shape.glyph.bar*(health..arts), *label*(city))

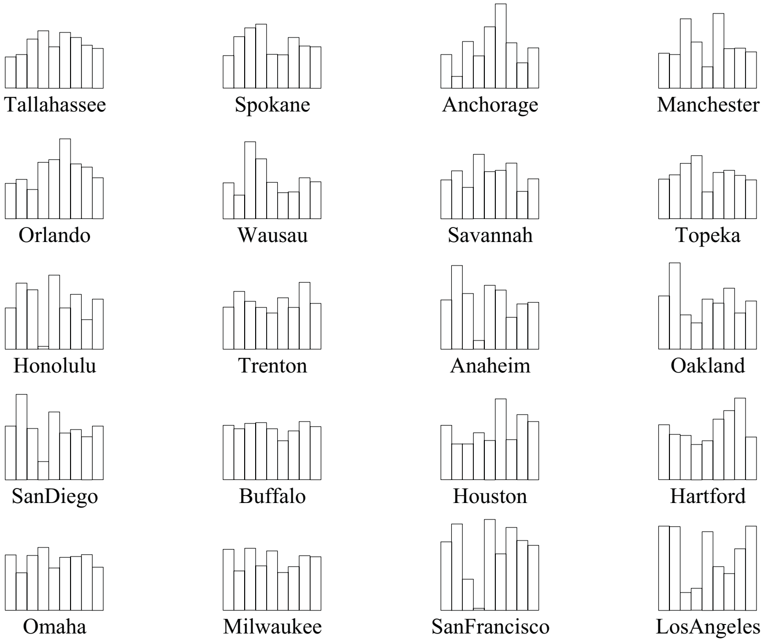


Figure 10.9 *Using separable dimensions to classify*

10.4 Realism

Perceptual psychology has had a long tradition of searching for principles of organization in the mind of the perceiver. In part, this has been a consequence of its phenomenological origins in Wundt's introspectionist psychology laboratory in Leipzig in the late nineteenth century. For almost a century, perceptual psychologists resisted the pressure of the behaviorists to explain perceptual constancies by looking to the regularities of the stimulus rather than those of the perceiver.

In reaction to the phenomenological tradition, James J. Gibson (Gibson, 1966, 1979) argued that the researcher must pay more attention to the perceiver interacting with a structured environment. Before imputing constancies to the perceiver, we must look for constancies in the environment. Gibson rejected the laboratory tradition of isolating stimuli to the point of making them impoverished. As an alternative, Gibson was interested in the persistent aspects of objects and their arrangements on the retina that allow us to perceive effectively and adapt to a changing world. Gibson's evolutionary perspective has influenced artificial intelligence approaches to perception, where machines must recognize and interact with their world (Marr, 1982; Winston, 1984).

Gibson helped stimulate a new trend in perceptual and cognitive psychology. Supported by new methodologies, advances in neuroscience, and the maturing of cognitive and computer science, theorists have enlarged their laboratories and turned their attention to the real world. Irving Biederman (1972, 1981) has analyzed schemas needed to comprehend creatures, objects, and their interactions in real scenes and pictures. Stephen Kosslyn (Kosslyn, 1980, 1983; Kosslyn, Ball, and Reiser, 1978) has examined the mental images used in navigating and remembering real-world situations. And Roger Shepard (Shepard and Metzler, 1971; Shepard and Cooper, 1983) has elucidated mental rotations and other cognitive operations which mirror a spatial world and derive from our need to interact with that world. Lockhead (1992) reviews this trend in perception research and Glenberg (1997) reviews recent evolutionary developments in the related field of memory.

This ecological focus offers several implications for representing multiple quantitative dimensions in graphical (aesthetic) systems. First of all, there is a new theoretical foundation for statistical graphics to achieve richer, more realistic expressions. For years, some writers on statistical graphics (*e.g.*, Tufte, 1983; Wainer, 1997) adopted Mies van der Rohe's adage, **less is more**. For Tufte, this means not wasting ink. The primary message of these writers has been that statistical graphics should be spare, abstract designs. For this formalist Bauhaus school of graphics, intent on minimizing the ink/data ratio, the major nemesis is the Rococo school of Holmes (1991), *U.S.A. Today*, and other popular pictographers. The controversy centers on **chartjunk**: 3D bar charts, pseudorealism, and pictographs. However witty the criticisms by the formalists and however sincere their aesthetic motivations, it must be remembered that these are basically *ad hoc* and unsupported by psychological theory.

Psychological research has a lot to say about the use of aesthetic/sensory features in graphic perception, but offers little support for the simple idea that less is more or that a small ink/data ratio facilitates accurate decoding.

On the contrary, there is some evidence that chartjunk is no less effective than abstract displays in conveying statistical information accurately (Lewandowsky and Myers, 1993). Pseudo 3D, drop shadows, and pictographs add redundant dimensions to multidimensional graphics. As we have seen, Garner's and others' research shows that redundant cues do not impair, and can actually improve, performance for integral dimensions. Additional dimensions degrade performance only when they are irrelevant to and vary independently of the dimensions needed for proper classification. Wilkinson and McConathy (1990) found, for example, that meaningful pictograms can contribute to memory for graphics. In short, there may be good reasons to dislike chartjunk, and Tufte's graphics are indisputably beautiful, but the crusade against chartjunk is not supported by scientific research and psychological theory.

From a somewhat different perspective, but still more Bauhaus than Rococo, Becker and Cleveland (1991) have argued that the field of scientific visualization has incorporated too much virtual reality and not enough abstract formal structure. Making a statistical graphics scene realistic, they claim, does not improve its information processing potential or its capability to convey information accurately. In light of the psychological research, however, we should emphasize the *complement* of their statement: realism is not bad, but ignoring the constraints of realism is *really* bad. In other words, what matters in statistical graphics (or in the assignment of aesthetic features to quantitative or qualitative dimensions), is whether we construct configurations that satisfy the rules of attributes in the real world. If we violate these rules (see Biederman, 1981, for examples), our message will be distorted or meaningless.

This principle implies, for example, that we should exercise care to avoid perceptual illusions when we use cues from the 3D world to convey a 2D graphical scene to our senses. Many of the classical visual illusions are probably due to 3D cues mistakenly applied to 2D scenes (Gregory, 1978; Coren and Girgus, 1978). Huff (1954), Hochberg and Krantz (1986), and Kosslyn (1994) provide a variety of examples of distorted graphics which invoke these illusions. Consistent with Cleveland's hierarchy shown in Figure 10.3, most of these illusions involve angles or features related to the processing of 3D scenes. Perhaps the best (or worst) example of a graphic eliciting these illusions is the 3D pie chart, which allows perspective illusions to interfere with angular part/whole judgments.

Most of all, we should avoid representations that not only do not exist in the real world, but contradict or grossly exaggerate real-world phenomena. Figure 10.10 shows an example of this problem. In an effort to improve Chernoff's FACES, Flury and Riedwyl (1981) added an asymmetrical dimension to the glyph in order to represent paired data. Each face in Figure 10.10 represents a strain of barley in the Fisher (1935) dataset; the left half of each face represents the yields for 1931 and the right half represents the yields for 1932.

This display is simply confusing. Indeed, humans are sensitive to asymmetries in faces (Troje and Bülthoff, 1996), but this is a subtle configural phenomenon usually associated with emotions (Ekman, 1984) and the range of sensitivity is far narrower than that implemented in the asymmetrical FACES glyphs. Recognizing and recalling facial asymmetry (as evidence of lying, for example) is a skill requiring intensive training (Ekman *et al.*, 1988). More severe asymmetry is associated with paralysis, trauma, or congenital defects. In Figure 10.10, the face of No. 457 is suffering from a case of Bell's palsy.

Asymmetrical faces present viewers with a glyph based on distracting analogues to the real world and with a perceptual task that introduces unnecessary complexity into pairwise comparisons. Half a face is no better, however. Following Flury and Riedwyl, Tufte (1983) suggested using only half of Chernoff's FACE glyph. Tufte advised that symmetries should be avoided in statistical graphics because they introduce unnecessary redundancies. While this may be true for some abstract stimuli, it makes little sense to enlist a wired-in perceptual mechanism and then defeat it by radical surgery.

ELEMENT: *point*(*position*(col*row),
shape.glyph.face2(university31..duluth32), *label*(site))
.

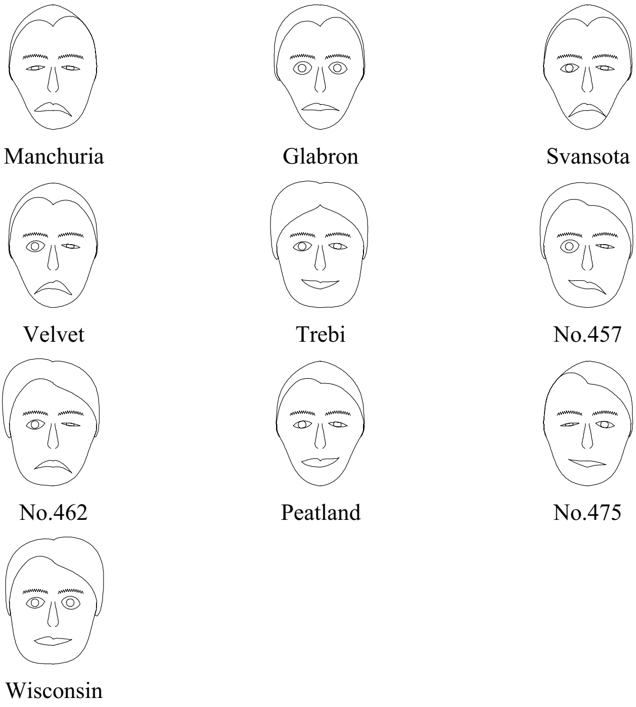


Figure 10.10 *Asymmetrical FACES (1931, 1932) of Fisher barley data*

Figure 10.11 shows the same barley data represented by pairs of symmetrical faces. The pairings clearly make it easier to compare faces between and within pairs. Grouping faces as siblings also allows us to consider larger families, as arise in repeated measurements. The reason comparison of siblings is easier than teasing apart halves of faces is that it resembles a real-world task. Faces are perceived as wholes. We are rarely called upon to make a second-order judgment that depends on the difference between the left and right side of a face (including its laterality). If there is asymmetry, we perceive it to have either a single emotional or congenital explanation. However, we are frequently called upon to make judgments concerning siblings and family membership. We might notice in Figure 10.11 that older (on the right) siblings tend to look unhappier or have smaller eyes. Similarly, we can easily make comparisons between families. We notice, for example, that the Trebi family is well fed and that the Svansota family is not thriving.

```
ELEMENT: point(position(row1*col1),
         shape.glyph.face2(university31..duluth31), label(site))
ELEMENT: point(position(row2*col2),
         shape.glyph.face2(university32..duluth32), label(site))
```

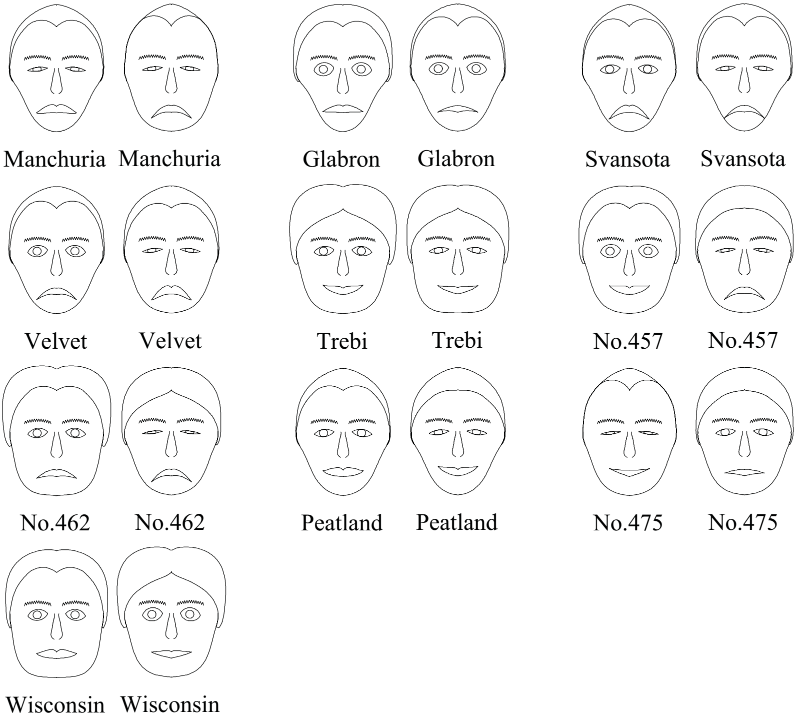


Figure 10.11 *Paired (1931, 1932) symmetrical FACES of Fisher barley data*

10.5 Aesthetic Attributes

The remainder of this chapter covers assignments of quantitative dimensions to aesthetic attributes. Table 10.1 summarizes these aesthetic attributes. We have grouped these attributes in five categories: form, surface, motion, sound, and text. This is not intended to be an exhaustive list; other attributes, such as odor, can be devised. Seven of these attributes are derived from the **visual variables** of Bertin (1967): position (*position*), size (*taille*), shape (*forme*), orientation (*orientation*), brightness (*valeur*), color (*couleur*), and granularity (*grain*). Bertin's *grain* is often translated as *texture*, but he really means granularity (as in the granularity of a photograph). Granularity in this sense is also related to the spatial frequency of a texture.

Some of the major attribute functions have several methods. These are indented in the table. For example, *color*() is an attribute function that normally indexes a categorical color table or scale. If one wishes to specify only hue, then one may use *color.hue*(). One can also specify only brightness by using *color.brightness*(). Custom color scales can be constructed as other named methods of the *color*() attribute function, such as *color.spectrum*(). We will not discuss these in more detail. Similarly, *shape*() specifies a table of indexed shapes (such as symbols). If one wants to reference different types of shape modification, one can use *shape.polygon*() for a general polygon, *shape.glyph*() for a multivariate glyph shape, or *shape.image*() for a general image such as a bitmap.

Aesthetic attribute functions are used in two ways. Most commonly, we specify a variable or blend of variables that constitutes a dimension, such as *size*(population) or *color*(trial1+trial2). Or, we may assign a constant, such as *size*(3) or *color*("red"). The *position*() function cannot take constants as arguments, but it is the one attribute that can accept more than one dimension, as in *position*(x*y).

Table 10.1 **Aesthetic Attributes**

Form	Surface	Motion	Sound	Text
position	*color*	*direction*	*tone*	*label*
size	*hue*	*speed*	*volume*	
shape	*brightness*	*acceleration*	*rhythm*	
polygon	*saturation*		*voice*	
glyph	*texture*			
image	*pattern*			
rotation	*granularity*			
resolution	*orientation*			
	blur			
	transparency			

These aesthetic attributes do not represent the aspects of perception investigated by psychologists. This lack of fit often underlies the difficulty graphic designers and computer specialists have in understanding psychological research relevant to graphics and the corresponding difficulty psychologists have with questions asked by designers. Furthermore, these attributes are not ones customarily used in computer graphics to create realistic scenes. They are not even sufficient for a semblance of realism. Notice, for example, that pattern, granularity, and orientation are not sufficient for representing most of the textures needed for representing real objects. Instead, these attributes are chosen in a trade-off between the psychological dimensions they elicit and the types of routines that can be implemented in a rendering system. Specifically,

- An attribute must be capable of representing both continuous and categorical variables.
- When representing a continuous variable, an attribute must vary primarily on one psychophysical dimension. In order to use multidimensional attributes such as color, we must scale them on a single dimension such as hue or brightness, or compute linear or nonlinear combinations of these components to create a unidimensional scale.
- An attribute does not imply a linear perceptual scale. As Section 10.1.2 indicates, few attributes scale linearly. In fact, some attributes such as hue scale along curvilinear segments in two- or three-dimensional space. All linear scales are unidimensional but not all unidimensional scales are linear.
- A perceiver must be able to report a value of a variable relatively accurately and effortlessly when observing an instance of that attribute representing that variable.
- A perceiver must be able to report values on each of *two* variables relatively accurately upon observing a graphic instantiating two attributes. This task usually, but not necessarily, requires selective attention. This criterion probably isn't achievable for all of our attributes and may not even be achievable for any pair of them. But any attribute that is clearly non-separable with another should be rejected for our system. It is too much to expect, of course, that higher-order interactions among attributes be nonexistent. Much of the skill in graphic design is knowing what combinations of attributes to avoid. Kosslyn (1994) offers useful guidelines.
- Each attribute must name a distinct feature in a rendering system. We cannot implement an attribute that does not uniquely refer to a drawable (or otherwise perceivable) feature. An attribute cannot be mapped to a miscellaneous collection of widgets or controls, for example.

Bertin's books are a rich source of examples for many of these attributes and their various combinations. Discerning the full breadth of his classification scheme requires careful reading and testing on specific scenarios. We have made a few modifications to Bertin's scheme. These can be seen by comparing our Table 10.2 at the end of this chapter with the figure on page 230 of Bertin (1981). We have reorganized four of Bertin's graphic variables (*size*, *shape*, *orientation*, and *granularity*) into two sets of parallel components under form (*size, shape, rotation*) and texture (*granularity, pattern, orientation*).

The motivation for this separation is the following. Graphics must respond independently to messages concerning their aesthetic attributes. At the same time, these independent responses must function in concert so that the graphic produced is consistent and coherent. This requires making a few of Bertin's categories independent, by employing what computer designers call **orthogonalization**. Bertin uses *size*, *shape*, and *orientation* to characterize both the exterior form of objects (such as symbol shapes) and their interior texture pattern (such as cross-hatching). This appears natural and parsimonious (especially when looking at his examples) until one has to write a computer program to implement these attributes. It becomes apparent in architecting the attribute functions that Bertin has confounded form and texture attributes. Because Bertin uses *shape* to determine both the shape of an area and the pattern of a texture, it is impossible to produce a circular symbol filled with a triangular mesh pattern or a dashed line consisting of images of the French flag. Bertin has his reasons for these strictures, although they may have more to do with preventing chartjunk (making dashes out of flags) than with spanning the perceptual and representational space.

We have avoided this confounding of attributes by separating form and texture. These modifications are discussed in more detail in the sections below. None of this implies, however, that attributes orthogonalized in a design sense are not correlated in the way they are perceived by our visual system. Orthogonalization in design means making every dimension of variation that is available to one object available to another. How these variations are perceived is another matter.

Many aesthetic attributes, even ones such as size or position that are usually considered visual, need not be perceived visually. There is nothing in the definition of a graphic given in Chapter 2 to limit it to vision. Provided we use devices other than computer screens and printers, we can develop graphical environments for non-sighted people or for those unable to attend to a visual channel because, perhaps, they are busy, restrained, or multiprocessing. Touch, hearing, and other senses can be used to convey information with as much detail and sensitivity as can vision. While three sensory modalities — vision, sound, and touch — can accommodate all the attributes discussed here, taste and smell involve multidimensional processing as well (Levine, 2000). The psychophysical components of these latter senses are not as well understood, however. Nor are they practical in current display environments. We will not discuss these specific implementations further.

10.5.1 Position

Spatial position refers to location in a (multi-) dimensional space. Bertin restricts his analysis to a sheet of paper or the plane (*plan*), but spatial position in a graphics system need not be restricted to even three dimensions. For example, a frame can be represented (with loss of generality) by a variety of projections into 3D, 2D, or 1D space. A positional attribute simply requires that values on a quantitative scale map to coordinates in a space.

Continuous variables map to densely distributed locations on a positional dimension. Categorical variables map to a lattice. These positions are ordered, but the ordering may or may not have meaning in terms of the scale of the measurements represented by the variable. Some projections may send two different coordinates to the same position, rendering them indistinguishable (see Chapter 9 for more information). This is commonplace in maps, where hemispheres may overlap in spherical projections, or in mixture coordinates (such as triangular), where the projection is a subspace of the data. Sometimes position is used simply to keep objects from overlapping and has no other purpose in a layout. Other times, position is used to place objects near to each other. We see this in Figure 10.11.

Cleveland (1985) rates position on a common scale as the best way to represent a quantitative dimension visually. This reflects the research finding that points or line lengths placed adjacent to a common axis enable judgments with the least bias and error. This recommendation requires a major qualification, however: it depends on how far a point, line, or other graphic is from a reference axis. If a graphic is distant from an axis, the multiple steps needed to store and decode the variation can impair the judgment (Simkin and Hastie, 1987; Lohse, 1993; Kosslyn, 1994). Grid lines can reduce this problem somewhat.

10.5.2 Size

Bertin defines size variation in terms of length or area. The extension of this definition to three dimensions would be volume. Cleveland ranks area and volume representations among the worst attributes to use for graphing data (see Figure 10.3). Indeed, area and volume both have Stevens exponents substantially less than 1 (see Figure 10.2). Figure 10.12 reveals this clearly. The circles in the upper row have diameters proportional to the integers 1 to 5. Those in the lower row have areas proportional to the same integers. It is self-evident that the areas are not perceived in a linear relationship to the scale values.

Size for lines is usually equivalent to thickness. This is less likely to induce perceptual distortion than it does for symbols, although extremely thick lines present problems of their own. Thick lines must be mitred and fill patterns must be handled carefully. Size can be used to great effect with *path*, however. A notable example is the famous graphic by Minard featured in Chapter 20.

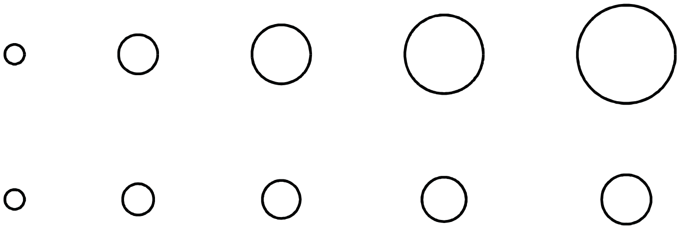


Figure 10.12 *Symbol size: diameters (upper) and areas (lower)*

Areas can change their size only if their perimeters are unconstrained. An area is defined by a perimeter that is determined by data or by a tiling or by some other enclosure element. Thus, size for area is a data attribute, not an arbitrary value we may change for aesthetic purposes. We can modify the coarseness of a texture inside an area, however. For example, we may use *texture.granularity*() to produce Bertin's "size" effects for areas.

Surfaces can change size in a manner similar to the way lines do. That is, lines may become thicker or thinner and so may surfaces. The ribbon element used in the left panel of Figure 8.4, for example, is relatively thick. Modifying the thickness of surfaces presents problems similar to those involved in modifying thickness of lines.

Finally, solids may change size in any dimension not constrained by data or geometry. As with areas, we cannot modify size of solids to suit arbitrary aesthetic goals. Size of volumes must be driven by data. Size distortion is worse for volumes, however. Figure 10.13 shows a size series for a set of boxes. The upper row shows boxes increased in size from left to right by magnifying each dimension proportionally to the integers 1 to 5. The lower row shows increases in volume according to the same series. The size of the upper boxes increases as a cubic function while the size of the lower follows a linear function.

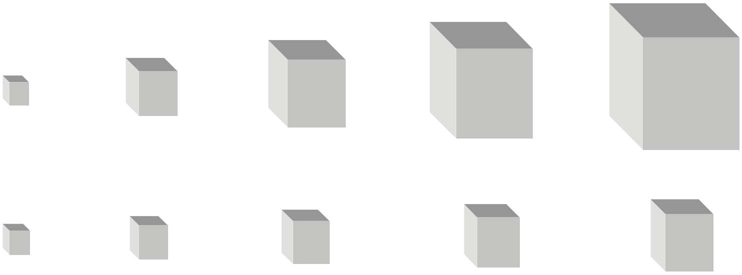


Figure 10.13 *Solid sizes: diameters (upper) and volumes (lower)*

For objects that have rotational symmetry, like circles, we can map size to the diameter rather than area (the upper rows of Figure 10.12 and Figure 10.13). The result is not always undesirable, especially for negatively skewed data or data with short upper tails. Conversely, representing size through area or volume should probably be confined to positively skewed data that can benefit from the perceptual equivalent of a root transformation. Otherwise, we might do best by following Cleveland's advice and avoiding size as an attribute.

Some designers assign size to only one dimension of an object, as in the width of a rectangle. This can confound perceptual dimensions, making selective attention to the size component difficult. Bar charts, for example, use the height of *interval* graphics to represent a variable. Their width is usually held constant. If we vary their widths according to an additional variable, we may run into trouble. The following example illustrates this.

Figure 10.14 shows four different displays of the same data with various uses of size and other aesthetic attributes. The data are from Allison and Cicchetti (1976). They represent ratings of animals, on a 1 to 5 scale, of their exposure to predators while sleeping (exposure) and likelihood of being eaten by predators in their environment (predation). Needless to say, the habitat presumed in this study is not urban. As the New Yorker Woody Allen once said after visiting the countryside in Connecticut, "I am *at-two* with nature."

The clustered bar graphic in the upper-left corner represents exposure and predation using separate bars. It is difficult to discern the relationship between the two variables in this plot. The vertical axis is sorted on the predation values, but we cannot sort both variables in the same frame. The dot plot in the upper right panel has a similar problem, even though it separates the variables into two sub-frames. The graphic in the lower-left corner attempts to represent joint variation by varying the height and width of the rectangles used for bars. Height and width of rectangles are configural dimensions, however. This makes it difficult to perceive the dimensions separately. Notice how difficult it is to perceive that Pig has the same value on both variables, while Kangaroo is discrepant on the same variables.

The scatterplot in the lower-right corner does not suffer from these problems, It is easy to discern the relative rankings of the animals on both variables. Furthermore, it is easy to locate clusters of animals (*e.g.*, goat, pig, kangaroo) in the scatterplot. The moral of this story is that configurality can cause problems with *size* and other aesthetics. It is best to let aesthetics vary on only one dimension.

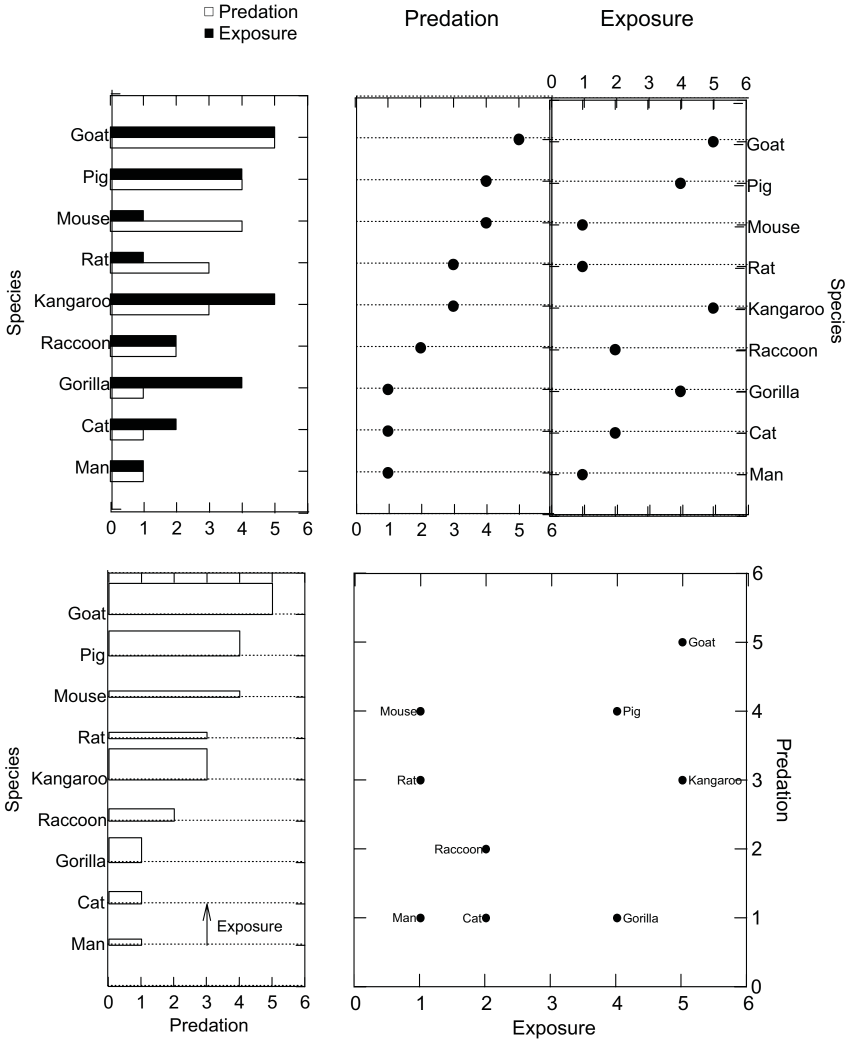


Figure 10.14 *Sized bars (lower left) versus other representations*

10.5.3 Shape

Shape refers to the exterior shape or boundary of an object. Symbols are the most obvious example, but every graph has the potential for taking on a different shape. Figure 10.15 shows several examples of shape variation in symbols. The top row shows a **morph** of a hexagon into a circular shape. Morphing is the technique needed to vary shape along a continuous dimension. The second row shows another morph of an ellipse into another ellipse. This example is problematic, however, because it is not rotationally invariant. Shape must vary without affecting size, rotation, and other attributes. The graphics in the second row could be used for representing negative and positive variation, but it is not clear that they would work as well as sized plus and minus signs.

The bottom row shows categorical shape variation. Several researchers have investigated optimal symbol shapes for categorization (Lewandowsky and Spence, 1989; Cleveland 1993). The symbols in this row of Figure 10.15 were selected for this purpose.

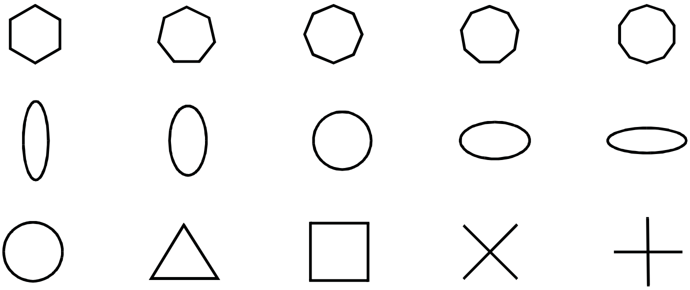


Figure 10.15 *Symbol shape: continuous (upper 2) and categorical (lower)*

Figure 10.16 shows several examples of continuous and categorical shape variation for lines. The left panel varies the roughness of the line to create a continuous shape dimension. The right modifies the outer contours of the line to create categorical shape variation.

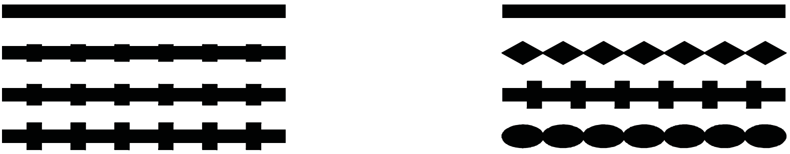


Figure 10.16 *Line shape: continuous (left) and categorical (right)*

Areas can change shape only if their perimeters are unconstrained by a positional variable. The *polygon*() graphic can be set to a hexagon, for example, in order to tile a surface or it can be set to the outline of a state in order to create a geographic map. If we want to produce some of the "shape" effects Bertin shows for areas, then we can use *texture.pattern*().

Surfaces can change shape in a manner similar to lines (*e.g.*, bumpy surfaces), although taking advantage of this behavior is a dubious practice. Like areas, surfaces can change shape in other ways if their shape is unconstrained by positional variables.

Figure 10.17 shows continuous and categorical shape variation for solids. Some solids, like interval graphics, are constrained on one or more axes for the purpose of representing size variation. This usually leaves at least one physical dimension free to vary in shape. The solids in Figure 10.17 have constant height but vary in their shape along the other two dimensions.

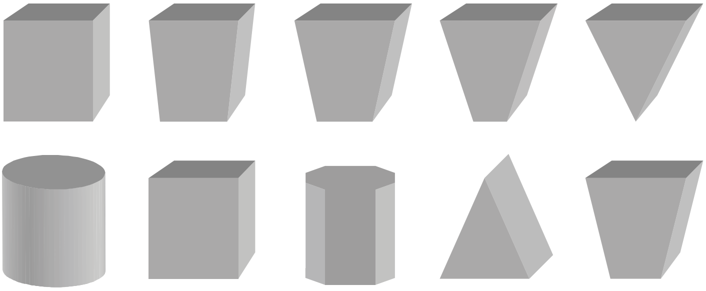


Figure 10.17 *Solid shape: continuous (upper) and categorical (lower)*

10.5.4 Rotation

The rotation of a graphic is its rotational angle. Figure 10.18 illustrates rotation variation for 2D and 3D objects. Lines, areas, and surfaces can rotate only if they are positionally unconstrained. We can produce the "orientation" effects Bertin shows in these graphics by using *texture.orientation*() instead.

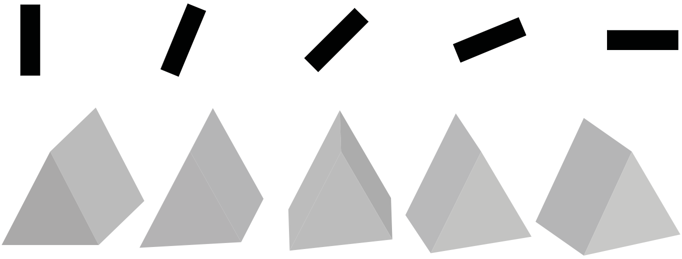


Figure 10.18 *Rotation: symbol (upper) and solid (lower)*

10.5.5 Resolution

The resolution of a graphic is a function of the amount of information contained in its frame. There are many ways to measure information. Most are based on a measure of **entropy** (Shannon, 1948):

$$H(x) = -K\sum_{i=1}^{n} p_i \log p_i \ ,$$

where x is a random variable with n possible states, p_i is the probability associated with the ith state, and K is a positive constant.

Consider the rightmost panel of Figure 10.19. If we assume this picture has n possible states (based on a finite set of pixels each with a finite number of darkness values) and each state has an associated probability (assume it is equal in this case), then we can see that the entropy of this picture is the lowest in the series, according to Shannon's formula.

It is not coincidental that the greater the entropy, the less compressible is the binary file representing the picture. Shannon proved that if one wishes to code a given signal into a sequence of binary symbols (bits) so that a receiver of the binary sequence can reconstruct the original signal perfectly, then one needs at least $H(x)$ bits to do so. Having more pixels does not guarantee greater entropy, however, since multiple pixels could have the same gray-scale value. The point is, *decreasing* the number of pixels cannot *increase* entropy or information.

The *resolution* aesthetic modifies the appearance of geometric elements by grouping the units on which statistical calculations are based. With lower resolution, statistical functions operate on groups of similar cases (analogous to pixels) rather than on single cases. The result of this operation is evident in histograms (see Figure 10.49). We usually modify resolution through the *bin*() statistical function, but when there are no variables to bin (as with images) we can use the aesthetic function *resolution*() instead.

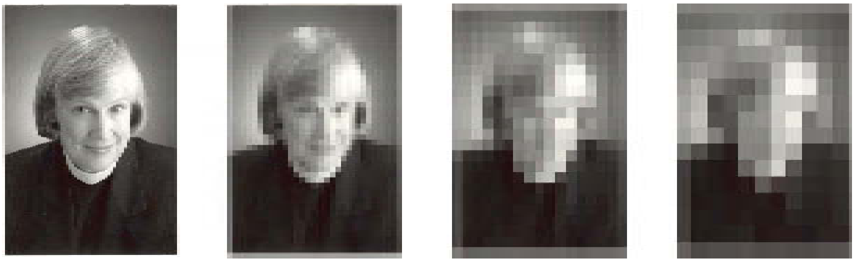


Figure 10.19 *Resolution from high on left to low on right (photograph courtesy of Stuart–Rodgers Photography, Evanston, IL)*

10.5.6 Color

Color is a psychological phenomenon, a fabrication of the visual system (Levine, 2000). The physical stimulus for color is light. Light has no color; it is electromagnetic energy of different wavelengths. We see color because we have three different photoreceptors in our retinas sensitive to light of different wavelengths. Thomas Young proposed this mechanism in 1802, assuming a mixture of the output of three primary receptors for red, green, and violet would produce any visible color. Hermann von Helmholtz formalized Young's conjecture in 1866 by delineating hypothetical excitation curves for each of the three types of receptor fibers Young postulated.

Because our color perceptual system is three-dimensional, we can represent all visible colors with any three non-collinear axes in that space. Computer monitors employ an RGB model, named for its use of *red*, *green*, and *blue* as a basis. This basis provides a wide range of color variation using comparable additive weights, and corresponds roughly to the sensitivity of the photoreceptors in the retina. Printers use a CYM model, named for *cyan*, *yellow*, and *magenta*. When used as pigment on an opaque surface, the colors in the CYM model absorb red, blue, and green light, respectively. This makes the CYM model *subtractive* for light, while the RGB model is *additive*. A third model called HLS refers to *hue*, *lightness*, and *saturation*. This model derives from Newton's analysis of the spectrum. The Munsell color solid for artists (*hue* = hue, *value* = brightness, *chroma* = saturation) is related to this system. See Travis (1991) for a review of color theory. Sacks (1995) discusses historical issues in the context of an intriguing study of a monochromatic perceiver.

Figure 10.20 shows the color cube for the RGB model. We have displayed both the front (on the left) and the back (on the right) of this cube in order to show the full color range available in this model. There are several things to note about this cube. First, it does not represent the full range of perceivable colors. Color displays are nevertheless quite good at giving us the impression of real-world colors. Second, it does not represent the metric space of perceived colors. We cannot assume that because two hues are near each other on the surface of the cube that we perceive them as similar or conversely, that if two are distant we perceive them as much different. Third, the colors on the surface of the cube are fully saturated. If we displayed the interior, we would see less saturated colors, with the center being gray.

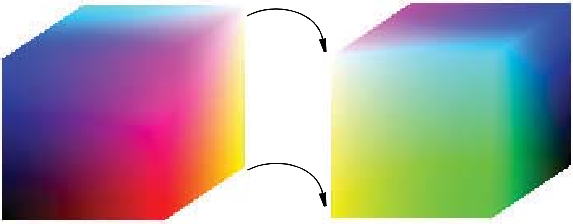


Figure 10.20 *RGB color cube (front left, rear right)*

Figure 10.21 shows the RGB cube space with annotations. The values of the tuples at the vertices of the cube are shown, as well as the primary (*red*, *blue*, *green*) and secondary (*magenta*, *cyan*, *yellow*) colors generated.

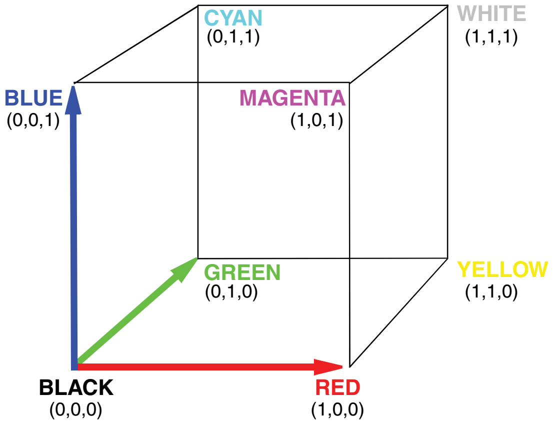


Figure 10.21 *RGB color space*

How do make a color scale? As with other scales, we require a mapping. In short, we are mapping a 3D color space to a 1D color space through a connected path. Figure 10.22 illustrates how this works. We wrote a little Java program to generate linear color scales for the figure. We have indicated in red on the little cubes where the paths are located for several popular scales. Some of these paths are rather nonlinear. Remember, this implies nothing about the linearity of the scale for perceptual purposes. Perceptual linearity means that a plot of perceived color differences against metric color differences (derived by measuring the color locations on the scale with a ruler) would be linear for all intervals on the scale.

The *brightness* scale is simply a diagonal path through the cube. Because it involves an equal mixture of all three primaries, it is a gray scale. The *heat* scale was developed by Levkowitz (1997) to be a perceptually linear temperature scale. Notice that it includes green and blue components toward the end in order to end up white. The *rainbow* scale is perhaps the most popular in use today. It runs through all the spectral colors on the cube. By staying on the edges, it does not compress color subcomponents disproportionately. If you are good at mental rotations, see if you can rotate the cube in this figure to be congruent with the projection in Figure 13.14 (which is close to a rainbow scale). The *circular* scale closes the path; it is rarely used, but can be valuable for circular variables. Finally, the *bipolar* scale is useful for representing bipolar variables. We ran the center through black to indicate zero on the scale, although we could have run it through gray (in the center of the cube) instead. Also, we could pick other hues to fit the application.

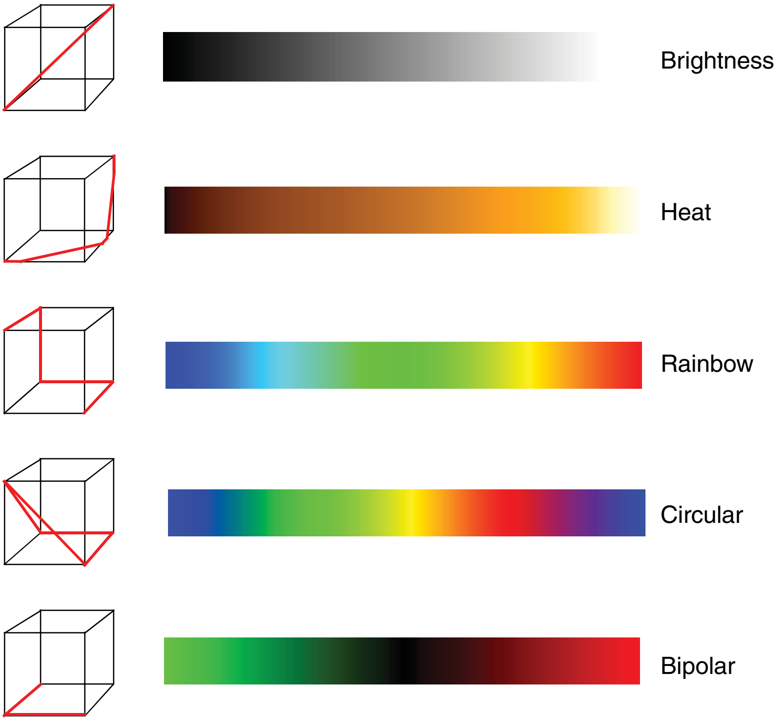


Figure 10.22 *Color scales based on paths through RGB color space*

10.5.6.1 Perceptual Issues

Trichromatic color theory accounts for three-dimensional color perception, but it fails to accommodate some curious color phenomena. One of these is **color afterimages**. Staring at a green patch for about 30 seconds leaves us with the perception of a red patch when we look at a white background. Similarly, a red patch leaves us with green, and a blue, yellow. Another curious phenomenon involves **color naming**. When asked to give additive mixture names to spectral colors, we employ more than three primaries (Judd, 1951). To cover the spectrum, we need names such as “reddish blue” and “yellowish green.” This phenomenon occurs across cultures as well (Berlin and Kay, 1969). For these and other reasons, it would appear that an **opponent process theory** is needed to account for human color perception. The modern form of this theory is integrated with trichromatic theory in a stagewise model. Trichromatic theory involves the initial integration of receptor signals, while opponent process involves later stages. The three opponent mechanisms are white-black, red-green, and blue-yellow.

We must be careful not to assign multiple color components to multiple variables. Even using two at once is dubious (see Wainer and Francolini, 1980). In general, it is safer to use a single color dimension such as *color.hue*, or the parent function *color*() to index a specific color scale (Brewer *et al*., 1997). Travis (1991) and Brewer (1994, 1996) present strategies for effective color representation. Olson and Brewer (1997) discuss ways to construct scales for the color-vision impaired.

10.5.6.2 Brightness

Brightness is the luminance, or lightness/darkness of a patch. Figure 10.23 shows a brightness scale for five patches. Brightness can be used to represent categorical dimensions, but only with a few categories.

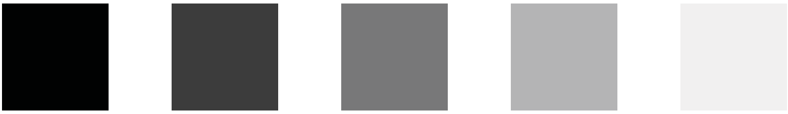


Figure 10.23 *Brightness variation*

10.5.6.3 Hue

Hue is the pure spectral component (constant intensity) of a color. Figure 10.24 shows five different hues: red, yellow, green, blue, purple. Hue is particularly suitable for representing categorical scales. Boynton (1988) names 11 basic colors: red, yellow, green, blue, white, gray, black, orange, purple, pink, and brown. These are especially suited for categorical scales, although Kosslyn (1996) advises against using all 11 at the same time.

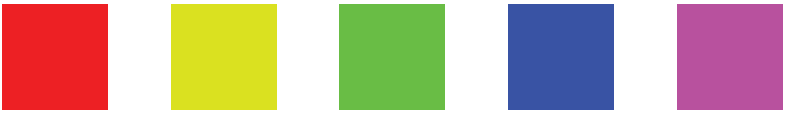


Figure 10.24 *Hue variation*

10.5.6.4 Saturation

Saturation is the degree of pure color (hue) in a patch. Figure 10.25 shows five different saturation levels for a red patch, from gray (lacking any hue) to red (pure hue). The brightness of the patches should be constant. MacEachren (1992) recommends saturation for representing uncertainty in a graphic.

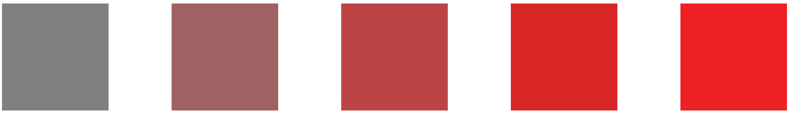


Figure 10.25 *Saturation variation*

10.5.7 Texture

Texture includes pattern, granularity, and orientation. Pattern is similar to **fill style** in older computer graphics systems, such as GKS (Hopgood *et al.*, 1983) or paint programs. Granularity is the repetition of a pattern per unit of area. Bertin describes it as "photographic reduction." Orientation is the angle of pattern elements. The word *orient* derives from the Latin word for *sunrise* (in the East). Conversely, the word *occident* is derived from the Latin word for *sunset* (in the West). Thus, *orientation* is alignment relative to the East.

A mathematical definition of texture is the spatial distribution of brightness values of the 2D image of an illuminated surface. This definition underlies the texture perception research of Julesz (1965, 1971, 1975). Spatial distribution can be represented in several ways. One of the most common is through the **Fourier transform,** which decomposes a grid of brightness values into sums of trigonometric components. This decomposition is orientation-dependent. Rotate the image, and another decomposition results. Another representation is the **auto-correlogram**, used by Julesz to characterize the spatial moments of a texture. The correlogram can be calculated in ways that make it orientation independent. These and other spatial functions have been used to construct machine texture-perceivers (see Watt, 1991 for a general discussion of texture/form issues). This topic also gives us the opportunity to cite our favorite title for a statistical paper on this or any subject, Besag (1986).

Texture alone can be a basis for form perception. Two gray areas that have the same overall level of brightness can be discriminated if their texture is different. The letter "R" rendered in a gray sand texture is readable against a background rendered in a gray woven cloth texture. Surprisingly, this form perception occurs even when the outlines of the form itself do not exist on the retina. Julesz demonstrated this with an invention of his called the **random-dot stereogram**. This display has been popularized recently in books whose pages have paired patches of computer-generated dots that reveal hidden figures when the reader fuses both images in the mind's eye. Julesz's research shows that a form emerges when the spatial distributions of the separate retinal images are processed in the visual cortex. Papathomas and Julesz (1988) demonstrate some graphical applications of the random-dot stereogram.

Form, pattern, and granularity interact. The top row of Figure 10.26 (adapted from Julesz, 1981) shows how. In the leftmost patch, we see a backwards "R" embedded in a field of regular "R"s. English (Indo-European) readers recognize this as a shape violation. They can distinguish this on the basis of form alone without resorting to texture perception. Others can recognize the figure from a single form comparison with any adjacent letter. Still others can recognize the figure by paying attention to the texture of the whole patch. For the middle patch, either form or texture detection picks up the array of backwards "R"s embedded in the lower right corner. The right patch, however, invokes purely texture perception, at least if held at the proper distance from the eye.

The interaction shown in this example helps explain Bertin's use of shape and orientation to describe both form and texture. Given a continuum, Bertin's taking an opportunity to reduce the number of basic constructs makes sense. As we have explained earlier in this chapter, however, implementing a renderer requires us to keep these concepts separate but parallel.

The bottom row of Figure 10.26 shows how random orientation can defeat texture-based perception of form. Each patch is a randomly oriented version of the patch in the upper row. In the left patch, it is possible to discern the backwards R by serially scanning the forms. It is still possible to do this in the middle patch, although quite difficult. It is almost impossible to do so in the rightmost figure. The random orientation masks the spatial boundaries of the sub-patch so that it is no longer detectable as it is in the patches in the top row.

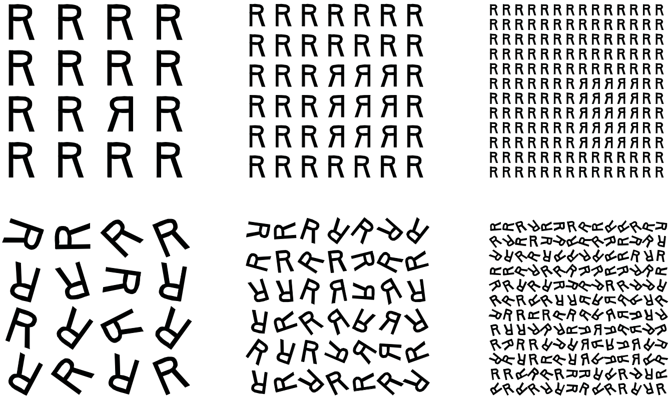


Figure 10.26 Granularity and orientation affect perception of form

10.5.7.1 Granularity

The top row of Figure 10.26 illustrates continuous granularity variation. Figure 10.27 shows another example. This is a grating of constant brightness that varies in spatial frequency. Optometrists use these patterns to measure resolution of the visual system. Less grainy patterns (having fewer low-frequency spatial components) are more difficult to resolve.

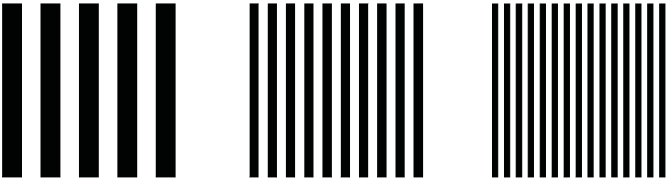


Figure 10.27 One-dimensional granularity

Figure 10.28 shows several degrees of granularity variation for lines. Notice the similarity between this example and the ones in Figure 10.27. Each row of Figure 10.28 can be considered to be a vertically compressed pattern like those in Figure 10.27. Both figures contain, in fact, 1D texture maps.

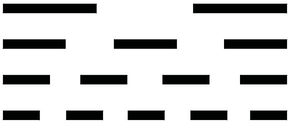


Figure 10.28 *Line granularity*

10.5.7.2 Pattern

Figure 10.29 shows continuous and categorical variation in pattern. The continuous examples (upper row) make use of increasing degrees randomness in a uniform spatial distribution. The categorical examples employ different shapes for their elements. The luminosity (brightness) of each patch is controlled by maintaining the same proportion of black pixels in each.

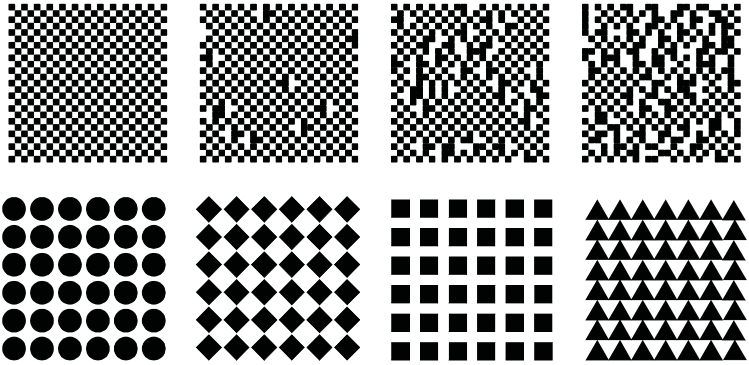


Figure 10.29 *Pattern: continuous (upper) and categorical (lower)*

Line patterns can be varied by filling thick lines with different patterns. A dashed line, for example, can be constructed by filling the interior of a thick line with a one-dimensional grid pattern, as in Figure 10.28. With apologies to Tufte, this is also the method we use for making lines out of flags. We fill the polygon defined by a thick line with a bitmap image of a flag. Some countries have laws against this sort of thing; check with your local authorities before using flags in graphics.

10.5.7.3 Orientation

Figure 10.30 shows several degrees of orientation variation. Orientation affects other components of texture, so it is not always a good idea to use it for representing a variable. Notice how variation in texture orientation introduces a visual illusion, making the lines seem not parallel. Tufte (1983) has excellent examples illustrating why use of orientation — whether for lines or for areas — introduces visual vibration, Moire patterns, and other undesirable effects.

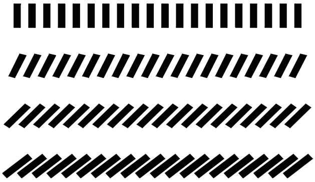


Figure 10.30 *Line texture orientation*

10.5.8 Blur

Blur describes the effect of changing focal length in a display. It is implemented by filtering a bitmap. As MacEachren (1992) discusses, blur is an attribute ideally suited for representing confidence, risk, or uncertainty in a process. Wilkinson (1983b) used blur to represent sampling error in histograms. Figure 10.31 shows five different blur levels.

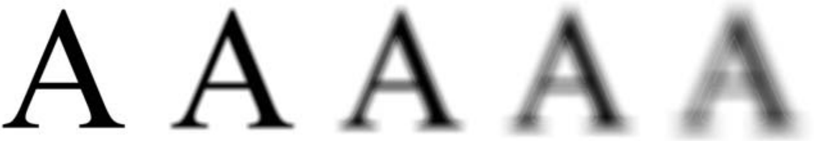


Figure 10.31 *Blur variation*

10.5.9 Transparency

Transparency, like blur, is an attribute suited for the display of uncertainty. It is implemented by blending layers of a color or gray-scale bitmap. Figure 10.32 shows five different transparency levels.

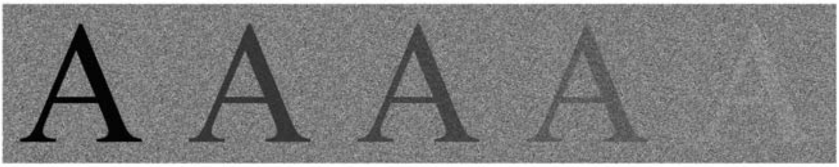


Figure 10.32 *Transparency variation*

10.5.10 Motion

Animation is discussed in Cleveland and McGill (1988) and Earnshaw and Watson (1993). It is implemented in a wide range of visualization software. For obvious reasons, a book is not the ideal format for presenting animation. For a glimpse, flip the pages and watch the page numbers in the corner change.

10.5.11 Sound

Sound can be used to display graphics in different ways. One approach is to use pitch, amplitude, texture, and other waveform features to represent separate quantitative dimensions (Bly, 1983; Mezrich *et al.*, 1984; Bly *et al.*, 1985, Fisher, 1994). This is the approach taken when a sound attribute is added to other dimensions in a visual graphic. The other approach is to treat every object in a graphic as a sound source and embed all objects in a virtual spatial environment (Julesz and Levitt, 1966; Blattner *et al.*, 1989; Bregman, 1990; Smith *et al.*, 1990; Hollander, 1994). In this **soundscape** approach, the metaphor of a symphony or opera performance is not inappropriate. A scatterplot cloud can be represented by a chorus of "singers" distributed appropriately in space. Thus, instead of using sound to represent a quantitative dimension, we can use sound to paint a real scene, a sonic image that realizes the graphic itself. An added benefit of this method is that motion can be represented in time without interfering with other dimensions of the signal. Sighted people must be reminded that the potential dimensionality of a sound environment is at least as large as a visual. Soundscape technology now makes this feasible, particularly for development environments like Java, where soundscape capability is built into 3D graphics foundation classes. Krygier (1994) reviews issues in the use of sound for multidimensional data representation. See also Shepard (1964), Kramer (1994), and Figure 9.57 for further information on the structure of sound perception. In the end, however, you can listen to this book for a long time before you hear any sound.

10.5.12 Text

Text has not generally been thought of as an aesthetic attribute. We classify it this way because reading involves perceptual and cognitive processing that helps one to decode a graphic in the same way that perceiving color or pattern does. The *label*() text attribute function allows us to associate a descriptive label with any graphic. It places text next to a point, on top of a bar, or near a line, for example. We will not present a separate example for the *label*() function in this chapter. Instead, it appears in numerous graphics throughout this book (*e.g.*, Figure 10.33, the next example). The *label*() attribute function allows us to associate with the graphic a descriptive text constant (*e.g.*, "New York") or a value that is automatically converted from numeric to text (*e.g.*, "3.14159").

10.6 Examples

The following examples illustrate selected aesthetic attributes. Each is set by using one of the attribute functions in a graphing function parameter list.

10.6.1 Position

Position is used so often as the primary attribute for a graphic that we can fail to notice the variations we can produce by altering it. One application is to embed multiple graphics in a common frame by using different positional variables. This section outlines a variety of applications.

10.6.1.1 Embedding Graphics

Ordinarily, multiple elements in a frame share the same algebraic expression on the same variables inside the *position*() aesthetic function. Sometimes we wish to embed graphics in a common frame using different variables, however. As long as the variables share scale specifications (categorical or continuous), these embeddings are meaningful.

As part of a study of lay self-diagnosis, Wilkinson, Gimbel, and Koepke (1982) collected co-occurrences of symptoms within disease classifications of the *Merck Manual*. We computed a multidimensional scaling of these selected symptoms in order to provide a framework for analyzing self-diagnosis from the same symptoms. Figure 10.33 shows the result of that scaling.

```
ELEMENT: point(position(dim(1)*dim(2)), shape(cluster), label(symptom))
ELEMENT: polygon(position(bin.voronoi(mean(1)*mean(2))))
```

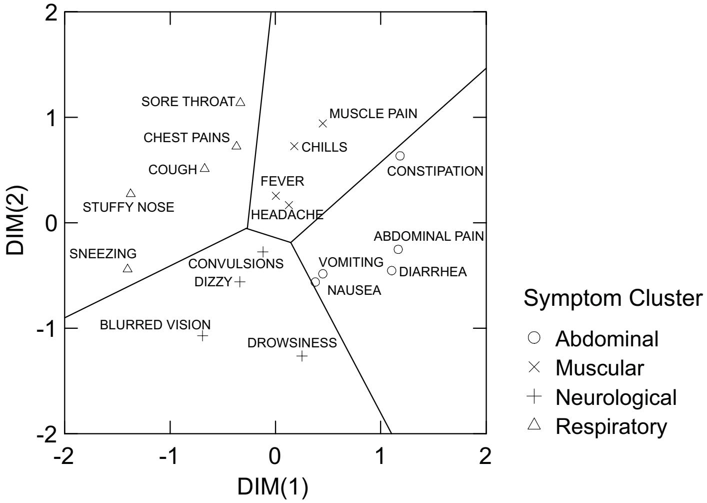


Figure 10.33 *Disease symptoms*

The coordinates of the 18 symptoms and their labels are fixed by the dim(1) and dim(2) variables in the first *position*() specification. Separate ratings of the symptoms yielded four clusters: respiratory, neurologic, muscular, and abdominal. Figure 10.33 shows a Voronoi tessellation (see Section 10.6.3.5) superimposed on the graphic. This graphic was positioned by using two variables containing only the means of the four clusters on the two dimensions. The boundaries almost perfectly separate the clusters in a radial pattern that closely resembles the hand-drawn graphic in the original article. Note that the positional frame is set by the union of dim(1) and mean(1) and dim(2) and mean(2), respectively. This works the same way a blend does.

10.6.2 Size

The *size*() attribute is most applicable to *point* graphics, but it has interesting applications elsewhere. Figure 20.1, for example, shows how *size*() can be used to control the segment-by-segment thickness of a path. The *size*() attribute can also be used to control the width of bars. The most popular application of *size*() is the *bubble* plot.

10.6.2.1 Bubble Plots

Figure 10.34 shows a bubble plot using symbol size to represent the reflectivity (albedo measure) of the planets in our solar system. The frame plots distance from the Sun (normalized so that the Earth–Sun distance is 1 unit) by mean temperature (in Kelvin). We have logged both scales so that the planets align roughly along the diagonal and the discrepancy of Venus is highlighted. The data are from the NASA Web site (*nssdc.gsfc.nasa.gov*).

SCALE: *log*(*dim*(1), *base*(10))
SCALE: *log*(*dim*(2), *base*(10))
ELEMENT: *point*(*position*(distance*temperature), *size*(albedo),
label(planet))

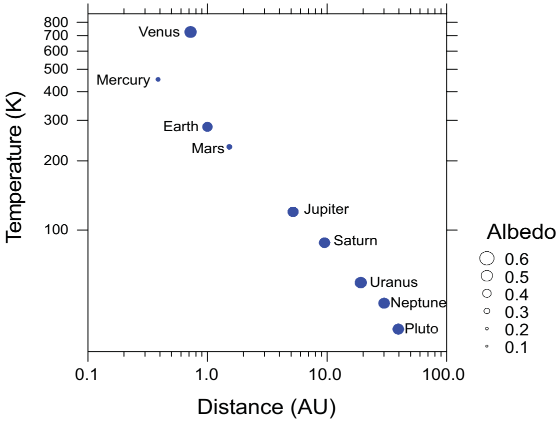


***Figure 10.34** Bubble plot of planet reflectivity (albedo)*

Another application of the *size*() attribute is in assessing statistical assumptions via graphics. The Pearson correlation coefficient, for example, measures the standardized linear association between two random variables that vary jointly. It is calculated by standardizing both variables to have zero mean and unit standard deviation and then averaging the cross-products of the standard scores. Certain extreme cases can contribute disproportionately to this computation, so it is often useful to look for such cases before trusting the correlation as a measure of association on a given set of data. One straightforward index of **influence** of a case on the computations is to compute the Pearson correlation with and without the case and examine the difference. This can be done for all cases and the influence measure used to size plotting symbols.

Figure 10.35 shows an example. We have plotted torque against horsepower for the cars dataset used in Figure 8.2. These performance statistics are usually linearly correlated among production automobile engines. One case stands out at the top of the plot, however. It is the Ferrari 333P race car. Some race car engine designers sacrifice torque for horsepower because they wish to favor top speed (at high RPM's) over acceleration. They compensate for this bias by making the car bodies as light as possible.

We have inserted two *point* clouds into the frame, one in blue to reveal each point and one in red to represent the influence function. The filled circles denote *negative* influence (correlation increased when the case was omitted) and the hollow circles denote *positive* (correlation decreased when the case was omitted). The filled circle for the Ferrari shows that it is attenuating the correlation by more than .10 even though it is in the upper right quadrant of the plot.

TRANS: influence = *influence.pearson*(torque, hp)
TRANS: polarity = *sign*(influence)
ELEMENT: *point*(*position*(torque*hp), *color*(*color.blue*))
ELEMENT: *point*(*position*(torque*hp), *size*(influence), *color*(*color.red*),
texture.pattern(polarity))

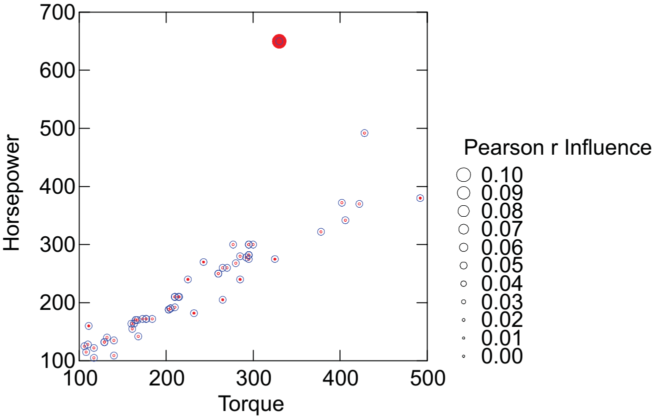


Figure 10.35 *Pearson correlation influence scatterplot*

Figure 10.36 shows how to use size to represent the counts in the GSS dataset. Note the contrast between this result and that in Figure 8.25. There, small counts were distinguishable; here, large counts are distinguishable. We could change this perception by transforming and threshholding the size scale, but representing the entire range would be problematic.

ELEMENT: *point*(*position*(sex*bible), *size*(count))

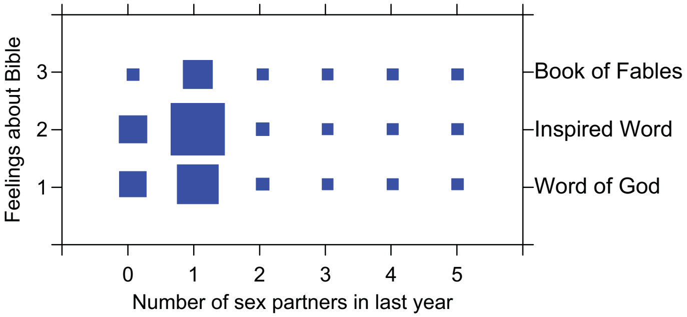


Figure 10.36 *Representing counts with size aesthetic*

10.6.3 Shape

The *shape*() attribute function is most often used to determine the shape of plotted symbols in *point* graphics. It also affects the shape of bars and other graphics.

10.6.3.1 Symbol Shapes

We are accustomed to seeing dot plots constructed from dots. There is nothing in the algorithm to prevent us from using other symbol shapes, however. Figure 10.37 shows an example of a dot plot using squares. The plot is computed on the sleep data from Allison and Cicchetti (1976), containing brain weights of 62 animals. We have logged the scale and used square dot shapes to emphasize the uneven spacing of the dots. For the singletons, the center of the square is placed exactly above the scale value it represents. This makes dot plots look more like histograms.

SCALE: *log*(*dim*(1), *base*(10))
ELEMENT: *point.dodge*(*position.bin.dot*(brainweight), *shape*(*shape.square*))

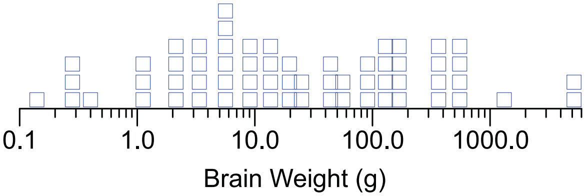


Figure 10.37 *Logged dot plot*

Figure 10.38 shows a scatterplot of the distribution of king crabs captured in a 1973 survey off Kodiak Island, Alaska. The data, posted on Statlib (*www.lib.stat.cmu.edu/crab*), were provided by the Alaska Department of Fish and Game. Each symbol is located at one of the survey sites marking a string of crab pots resembling the ones used by the commercial fishing fleet. The pots were left in the water for approximately a day, removed, and the crabs counted.

The symbols in Figure 10.38 are used to characterize the distribution by sex at the sites. Yields of more than 10 males at a site are marked with a vertical line, yields of more than 10 females are marked with a horizontal line, yields with more than 10 of both sexes are marked with the union of these symbols (a plus), and all other yields (including no crabs) are marked with a dot. This plot is patterned after several featured in Hill and Wilkinson (1990). Notice that the waters southwest of the island had a dearth of female crabs that year.

The choice of symbol shapes affects how distributions in a scatterplot are perceived. The obvious choice of male and female symbols (♂♀) is not a good one, because there is little perceptual contrast between the two. See Cleveland (1993) and the accompanying discussion of issues involving symbol choices. We have used vertical and horizontal line symbols in Figure 10.38 in order to maximize the orientation contrast, following a suggestion made in a comment on Cleveland (Wilkinson, 1993b), and also to highlight the potential union of crabs at a site.

```
DATA: longitude, latitude = map(source("Kodiak"))
ELEMENT: point(position(lon*lat), shape(group))
ELEMENT: polygon(position(longitude*latitude))
```

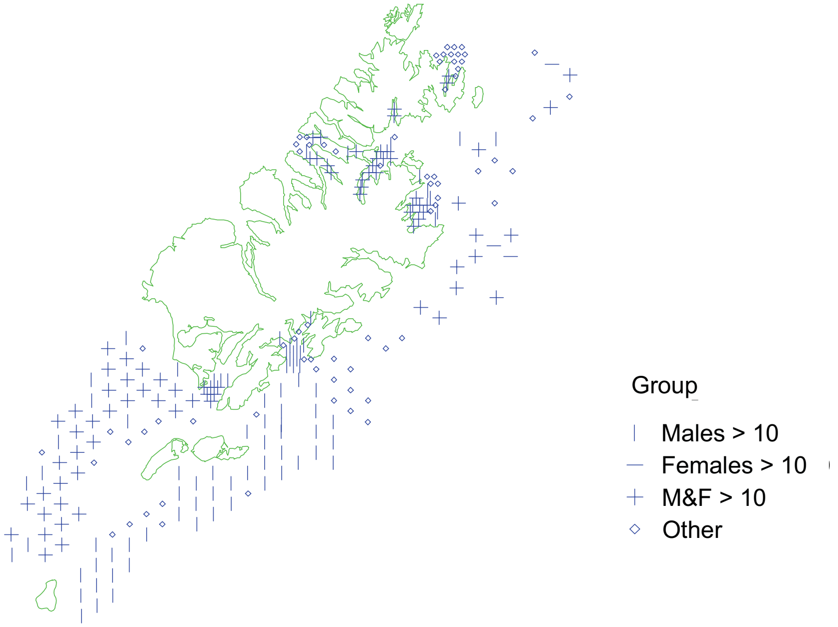


Figure 10.38 King crabs near Kodiak Island by gender

10.6.3.2 Polygon Shapes

Symbol shapes need not be determined by a fixed repertoire of symbols. They can also be set by arbitrary polygons. Figure 10.39 shows a scatterplot of winter and summer temperatures using state outlines derived from a shape file. We got the idea for this plot from Woodruff *et al*. (1998). It is a clever application, although labeling the points with the names of the states using *point*(*label*(state)) would make it easier to identify them. Perhaps this graphic would find its best use in a secondary school geography contest.

ELEMENT: *point*(*position*(winter*summer), *shape.polygon*(state))

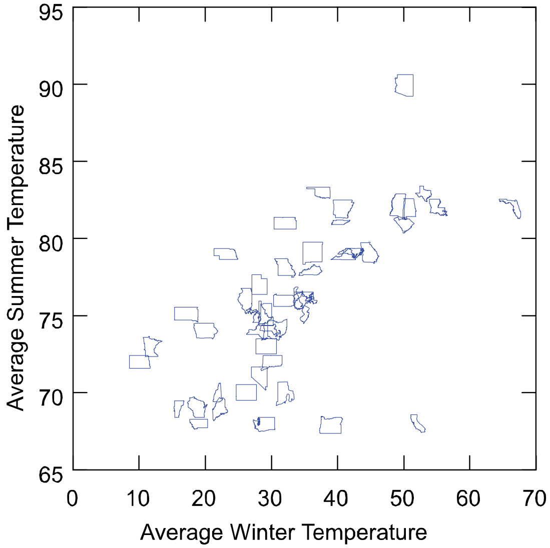


Figure 10.39 *Scatterplot of states*

10.6.3.3 Glyphs

Symbol shapes can be determined by even more complex algorithms. The following sections discuss *glyph*() functions. These are used to determine the shape of *point* graphics based on one or more variables extrinsic to the frame. Glyphs are geometric forms used to represent several variables at once (Fienberg, 1979; Carr *et al.*, 1992; Haber and McNabb, 1990).

glyph.face

Herman Chernoff (1973) invented the FACES display. Figure 10.8 shows an example of Bruckner's (1978) revision of Chernoff's FACES, and Figure 10.10 shows an example of Flury and Riedwyl's (1981) version. Consult those figures to see how the *glyph.face* (Bruckner) and *glyph.face2* (Flury and Riedwyl) act on the *shape* attribute of *point* to produce the glyphs. The figures earlier in this chapter show them plotted in a rectangular array, but some of the examples here will show how glyphs can be used in other configurations.

glyph.bar

Figure 10.9 shows an example of the *glyph.bar* function. This creates a histogram profile for each row in the dataset across the specified variables.

glyph.profile

The *profile* glyph is constructed similarly to a *bar*, except the bar heights are connected with a single profile line. Figure 10.40 shows an example.

ELEMENT: *point*(*position*(col*row), *shape.glyph.profile*(health..arts), *label*(city))

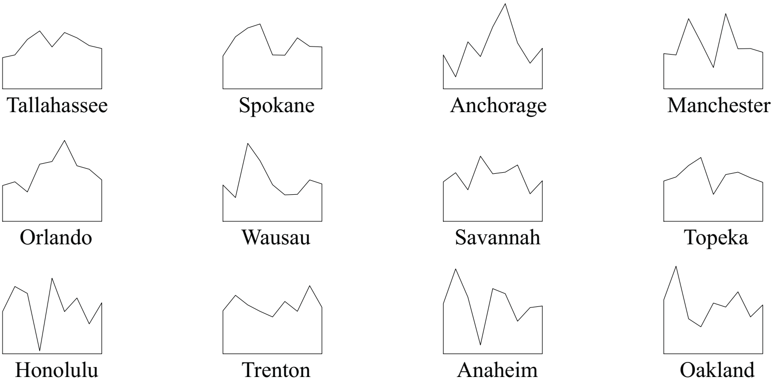


Figure 10.40 *Profile glyphs*

glyph.star

The *star* function is simply a *profile* function in polar coordinates. Figure 10.41 shows an example.

ELEMENT: *point*(*position*(col*row), *shape.glyph.star*(health..arts), *label*(city))

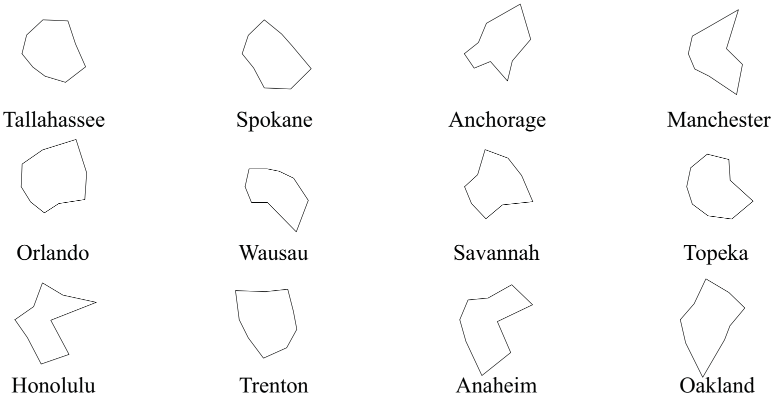


Figure 10.41 *Star glyphs*

glyph.sun

The sun glyph substitutes rays for the perimeter of a star. Figure 10.42 shows an example that uses a coordinate scaling procedure developed by Borg and Staufenbiel (1992). Instead of plotting evenly spaced polar rays, Borg and Staufenbiel compute a set of row vectors taken from the first two columns of the matrix **V** in the singular value decomposition $\mathbf{X} = \mathbf{UDV}^{\mathrm{T}}$ of the data matrix **X**. These vectors are proportional to the variable coordinates of a 2D **biplot** (Gabriel, 1971, 1995; Gower, 1995). For each observation, Borg and Staufenbiel scale the length of the vectors to be proportional to the normalized values of the variables. This produces a set of stars varying only in the length of their rays.

An advantage of this procedure is that, unlike the other glyphs, it does not depend on the order of the variables in the function parameter list. Furthermore, variables that covary in the data are represented by vectors that have relatively small mutual angles. This reduces the emphasis given to redundant variables. Borg and Staufenbiel compared their method to other popular glyph displays and found that judgments of similarity based on sun glyphs were more accurate on generated data than judgements based on the other displays.

ELEMENT: *point*(*position*(col*row), *shape.glyph.sun*(health..arts), *label*(city))

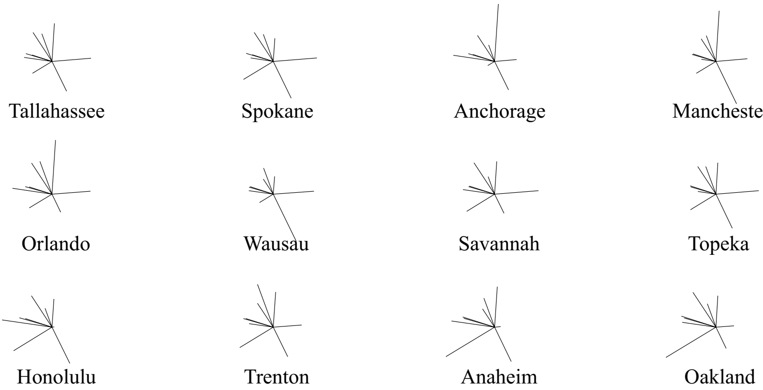


Figure 10.42 *Sun glyphs*

glyph.blob

Andrews (1972) introduced a Fourier transform on rows of a data matrix that allows one to plot a separate curve for each row. Cases that have similar values across all variables have comparable wave forms. Section 9.4.2.1 presents Andrews' method. The function Andrews used is

$$f(t) = \frac{x_1}{\sqrt{2}} + x_2 \sin(t) + x_3 \cos(t) + x_4 \sin(2t) + x_5 \cos(2t) + \ldots$$

where $\boldsymbol{x}$ is a p-dimensional variate and t varies continuously from $-\pi$ to π. The *blob* glyph plots Andrews' function for each case in polar coordinates. Figure 10.43 shows an example. The shape of the blobs depends on the order variates are entered in the Fourier function. Thus, blobs share a variable-ordering problem with most of the other glyph methods.

ELEMENT: *point*(*position*(col*row), *shape.glyph.blob*(health..arts),
label(city))

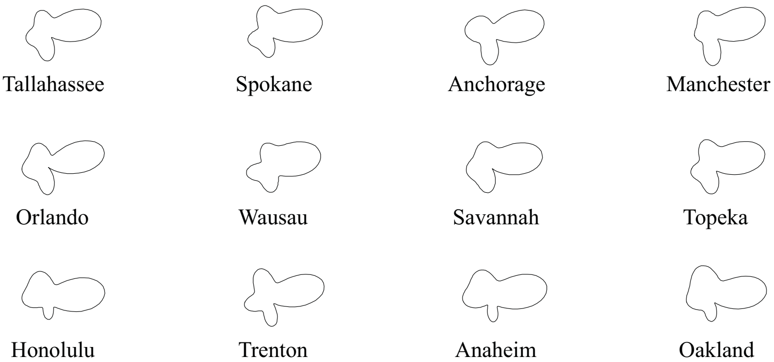


Figure 10.43 *Fourier blobs*

glyph.therm

The *therm* glyph resembles a thermometer. Ordinarily, it is useful for representing only one variable, in a mode that resembles the mercury level in a thermometer. Cleveland and McGill (1984b) recommend it for adding a variable to scatterplots (as opposed to using *size* or *shape* attributes) because of the opportunity for accurate linear scale decoding that it offers. Dunn (1987) proposed varying the width of the *therm* to display yet another variable. Figure 10.44 shows an example. We have superimposed the *glyphs* on a map of the continental US. The width of the thermometers is proportional to winter temperatures and the filled area is proportional to rainfall. Varying width is a clever idea, but it introduces configurality into the judgment of the symbols. It needs to be used with caution.

DATA: longitude, latitude = *map*(*source*("US states"))
ELEMENT: *polygon*(*position*(longitude*latitude), *pattern*(*pattern.dash*))
ELEMENT: *point*(*position*(lon*lat), *shape.glyph.therm*(rain, winter))

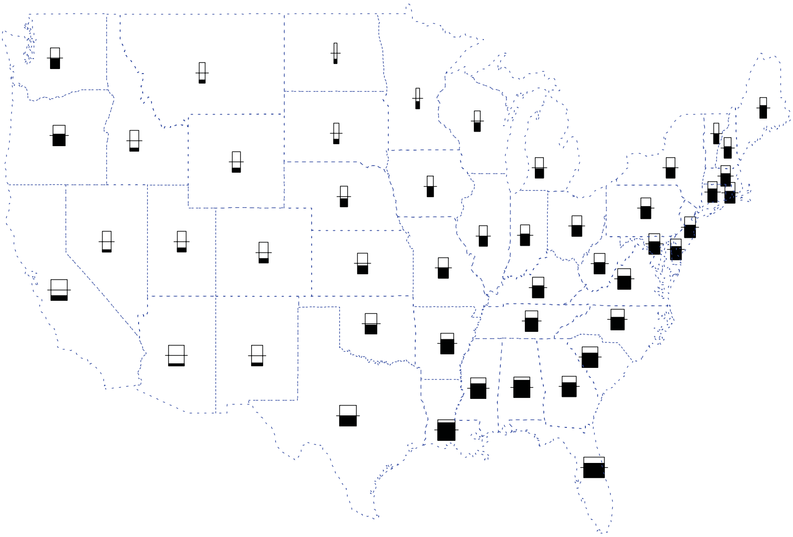


Figure 10.44 *Thermometer glyphs*

glyph.vane

The *vane* glyph uses direction and length and size of circle to represent three variables. Carr and Nicholson (1988) discuss the use of this and other “ray” representations. Many of them derive from the weather map symbols for wind vectors. These symbols resemble small flags that point in the direction of the wind.

Figure 10.45 shows an example involving continental US climate data. The size of the symbols is proportional to rainfall, the length of the vanes is proportional to summer temperatures, and the rotational angle is proportional to winter temperatures. The problem in using vanes this way is that the wind metaphor interferes with decoding values when we do not map the angles to wind direction and lengths to wind speed. See Figure 10.48 for comparison.

```
DATA: longitude, latitude = map(source("US states"))
ELEMENT: polygon(position(longitude*latitude), pattern(pattern.dash))
ELEMENT: point(position(lon*lat), shape.glyph.vane(rain, summer, winter))
```

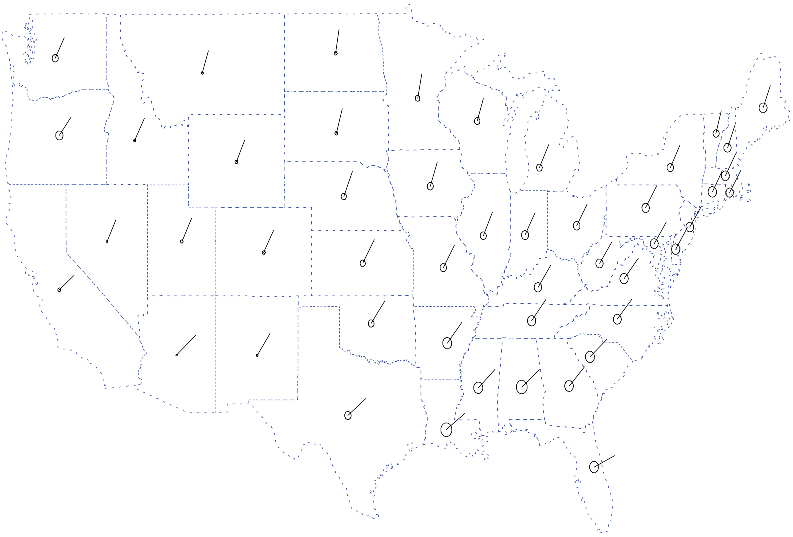


Figure 10.45 *Vane glyphs*

10.6.3.4 Shape set by Image

Images can be used to determine the shape of *points*, *bars*, and other graphics. Figure 10.46 shows a scatterplot of facial expressions adapted from psychometric research documented in Russell and Fernandez-Dols (1997). The configuration in this plot fits a pattern called a **circumplex** (Guttman, 1954), which is best interpreted through polar coordinates. In this example, the center of the configuration is marked by a face showing lack of emotion. Radial distance from this point in any direction represents *intensity* of emotion and polar angle represents *type* of emotion. These two polar variates can be more interpretively useful than the rectangular axis labels (sleepiness-activation and unpleasantness-pleasantness) shown here. When images have substantive meaning, as in this example, they can legend a graphic more effectively than any other type of symbol or guide.

DATA: face = *link*("faces")
ELEMENT: *point*(*position*(pleasantness*activation), *shape.image*(face))

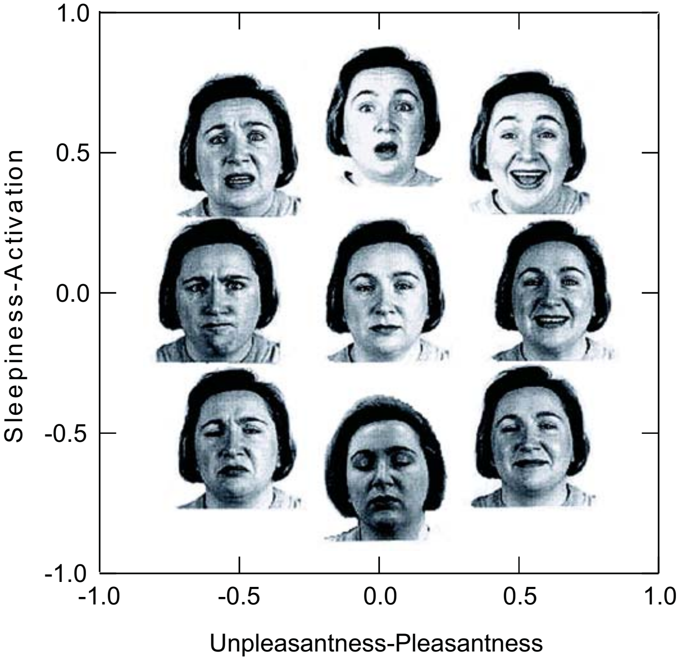


Figure 10.46 *Scatterplot of faces (images courtesy of James A. Russell)*

10.6.3.5 Bar Shapes

Figure 10.47 shows a shape variation on bars using a "pyramid" shape function. This form of bars is not recommended because the slopes of the bar sides change with their height, a confusing visual illusion.

ELEMENT: *interval*(*position*(*summary.mean*(gov*birth)),
shape(*shape.pyramid*))

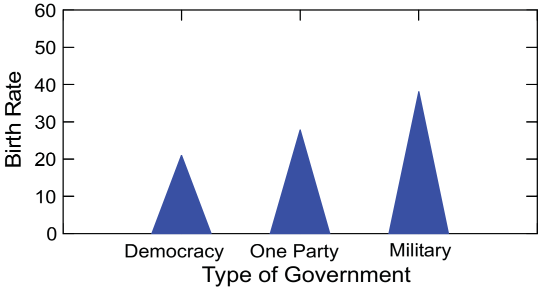


Figure 10.47 *Pyramid-shaped bars*

10.6.4 Rotation

Figure 10.48 shows how to use *rotation*() and *size*() to plot velocity of winds with an arrow symbol. The data are from the Seasat-A satellite scatterometer (*podaac.jpl.nasa.gov*) reported in Chelton *et al*. (1990).

DATA: longitude, latitude = *map*(*source*("World"))
COORD: *project.robinson*(*dim*(1, 2))
ELEMENT: *polygon*(*position*(longitude*latitude), *color*(*color.green*))
ELEMENT: *point*(*position*(lon*lat), *shape*(*shape.arrow*), *rotation*(direction),
size(strength))

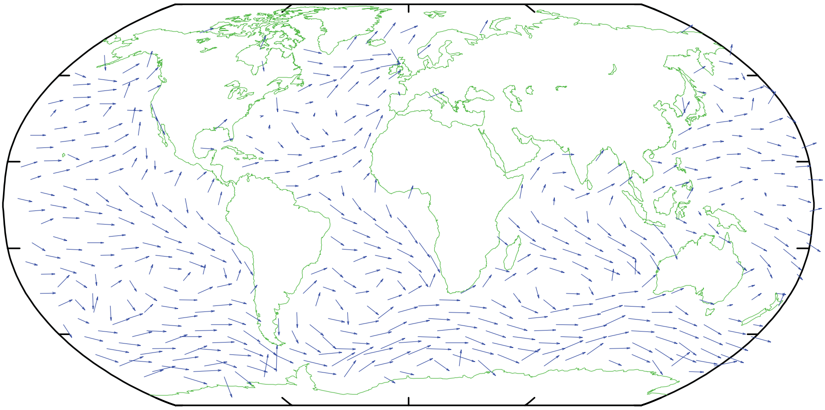


Figure 10.48 *Global prevailing winds*

10.6.5 Resolution

Figure 10.49 shows how to control the number of bars in a histogram. The data are standard normal pseudorandom numbers. Notice how the shape of the histogram changes under different resolutions.

ELEMENT: *interval*(*position*(*summary.count*(*bin.rect*(z, *bins*(b)))))

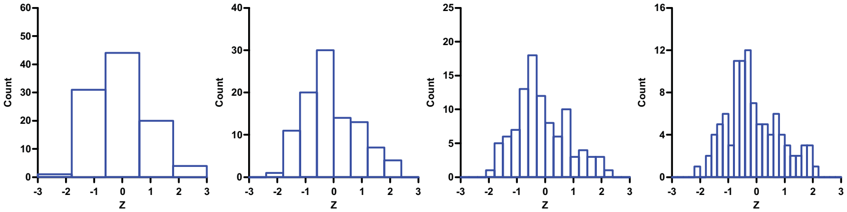


Figure 10.49 *Standard normal histograms*

10.6.5.1 Dot-Box Plots

Resolution need not affect the number of geometric elements in a graph. Dot plots, for example, are binned by placing nearly overlapping points at the same location and then stacking them like a tally. This reduces the data resolution but not the number of points. Figure 10.50 shows *schema* and *point* graphics grouped by a categorical variable (gov). This plot, suggested to us by Jerry Dallal when he superimposed separate SYSTAT graphics, resembles some variations in the box plot devised by Tukey (1977).

ELEMENT: *schema*(*position*(gov*birth))
ELEMENT: *point.dodge.symmetric*(*position*(*bin.dot*(gov*birth, *dim*(2))),
color(*color.green*))

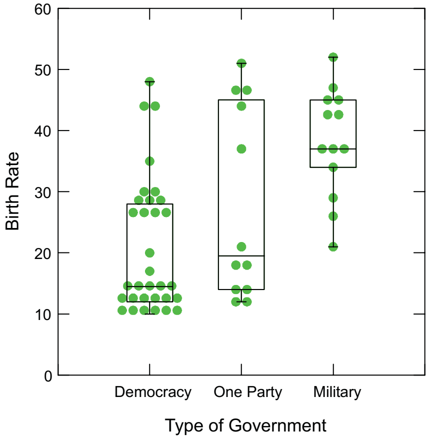


Figure 10.50 *Box and dot plots by group*

10.6.6 Color

Color can be especially effective for categorical coding. This section also features continuous color scales.

10.6.6.1 Representing Counts with Color

Figure 10.51 shows two uses of color on the GSS dataset. The upper panel uses *color.brightness* and the lower panel uses *color.hue*.

ELEMENT: *point*(*position*(sex*bible), *color.brightness*(count))

ELEMENT: *point*(*position*(sex*bible), *color.hue*(count))

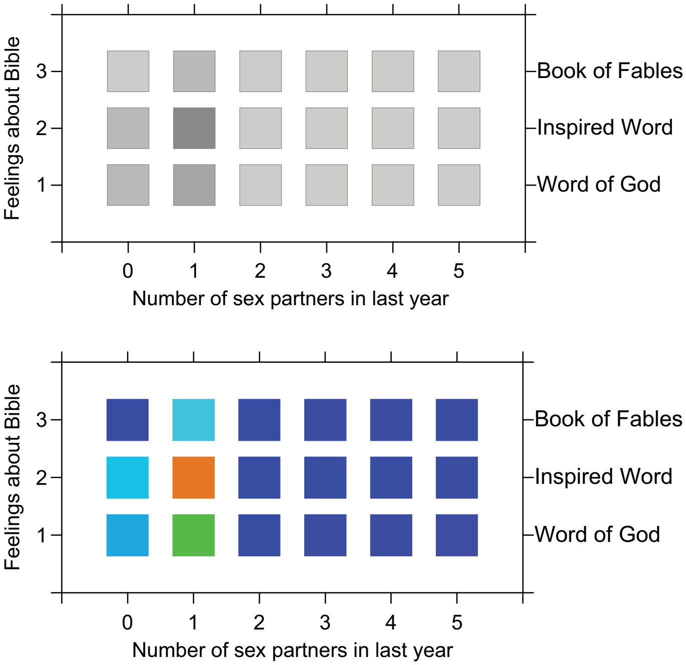


Figure 10.51 *Representing counts with color aesthetic*

10.6.6.2 Shading with Color (continuous color scales)

Figure 10.52 shows an example of shading (setting an attribute by a continuous variable) with the Iris data. The values of petalwidth determine the shades of color of the symbols plotted. We must use *color.hue*() instead of *color*() because we are mapping to a continuous scale rather than a table of color categories. Even though there are several species in the plot, there is only one cloud of points. No subgrouping occurs because species was not used to split this graphic. The shading shows that petal width varies more strongly with the *difference* between sepal length and sepal width than with their *sum*.

ELEMENT: *point*(*position*(sepallength*sepalwidth), *color.hue*(petalwidth))

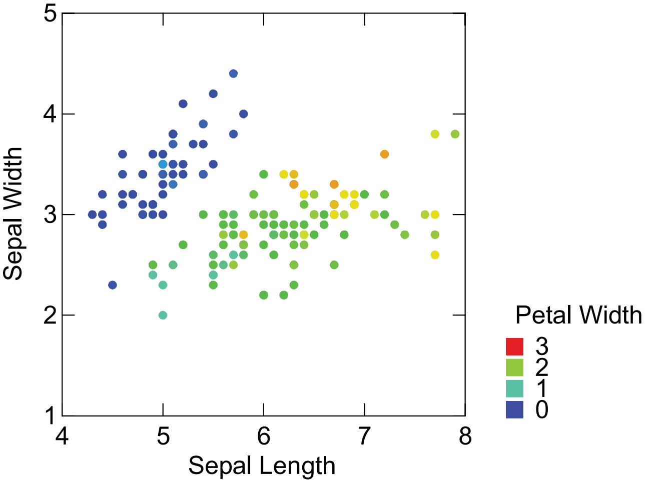


Figure 10.52 *Scatterplot shaded by color*

10.6.6.3 Splitting with Color (categorical color scales)

Exchanging the continuous shading variable in Figure 10.52 for a categorical splitting variable produces the graphic in Figure 10.53. There are now three ellipses and point clouds, one for each species. Without the *contour* graphic, we would have no way of knowing whether there are one or three clouds. This is because we could have treated species as continuous (by giving it numerical values), which would yield one *point* cloud shaded with three values.

ELEMENT: *point*(*position*(sepallength*sepalwidth), *color*(species))
ELEMENT: *contour*(*position*(*region.confi.mean.joint*(sepallength*sepalwidth)), *color*(species))

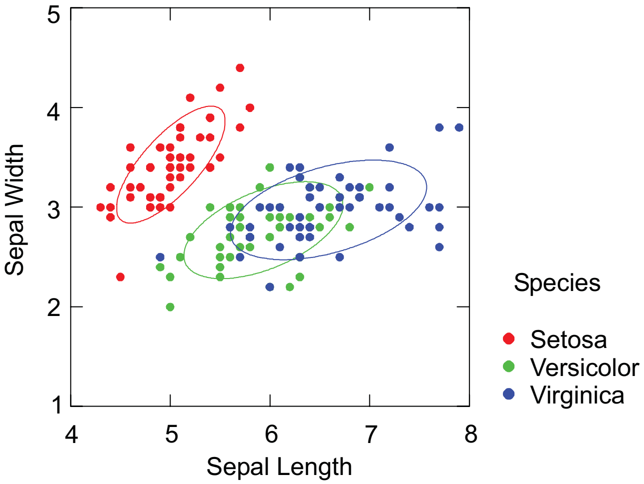


Figure 10.53 *Scatterplot split by subgroups*

10.6.6.4 Legending a Blend with Color

Blends usually require legends to distinguish the variables on which the blends are based. The *string*() data function fills a column with a specified string (see Section 3.1). For example, *string*() indexes a blend of variables *A* and *B* in the following manner:

$$\begin{array}{c} A \\ \boxed{\begin{array}{c} a_1 \\ a_2 \\ a_3 \\ a_4 \end{array}} \end{array} + \begin{array}{c} B \\ \boxed{\begin{array}{c} b_1 \\ b_2 \\ b_3 \\ b_4 \end{array}} \end{array} = \begin{array}{c} (A+B) \\ \boxed{\begin{array}{c} a_1 \\ a_2 \\ a_3 \\ a_4 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \end{array}} \end{array} \longleftrightarrow \begin{array}{c} string(\text{"a"})+string(\text{"b"}) \\ \boxed{\begin{array}{c} \text{a} \\ \text{a} \\ \text{a} \\ \text{a} \\ \text{b} \\ \text{b} \\ \text{b} \\ \text{b} \end{array}} \end{array}$$

The *string*() data function doesn't actually allocate storage to do this; it simply returns the appropriate string value whenever a blend is used. If there is no blend, then *string*() returns a string for as many cases as exist in the FRAME specification. We ordinarily use a *string*() variable to label, color, or otherwise describe a blend.

Figure 10.54 shows an example. Notice that the *color* attribute function operates on the *string*() function to add color to the *point* and *contour* graphics. Notice also that because *string*() creates a categorical variable, it splits the one cloud into two. Using a variable in a graphing function is like crossing with that variable.

```
DATA: s=string("Sepal width")
DATA: p=string("Petal width")
ELEMENT: point(position(sepallength*(sepalwidth+petalwidth)),
             color(s+p))
ELEMENT: contour(position(region.confi.mean.joint(
                                sepallength*(sepalwidth+petalwidth)),
             color(s+p))
```

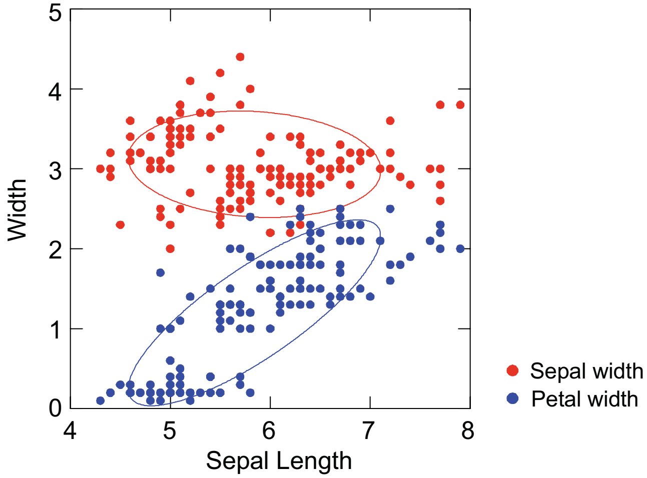


Figure 10.54 *Blended scatterplot*

For repeated measures and multivariate data, we often wish to plot values against a variable that represents the indices of the measures. For example, we might have measurements on subjects before and after an experimental treatment and we want a plot with measurements on the vertical axis and two values, *before* and *after* on the horizontal axis. Figure 10.55 shows an example of this type of design, using data from Estes (1994). The experiment involved a 320-trial learning series requiring subjects to classify artificial words as adjectives or verbs. The exemplar variable denotes whether words in the trials were first occurrences (new) or first repetitions (old). The trials were grouped in 4 blocks of 80 trials each.

The blend in the specification (vertical axis) is plotted against the names of the blend variables (horizontal axis). The exemplar variable is used to determine the hue of both graphics and the shape of the symbols in the *point* graphic.

```
DATA: b1=string("BLOCK(1)")
DATA: b2=string("BLOCK(2)")
DATA: b3=string("BLOCK(3)")
DATA: b4=string("BLOCK(4)")
ELEMENT: line(position(b1* block1+b2*block2+ b3*block3+b4*block4),
         color(exemplar))
ELEMENT: point(position(b1* block1+b2*block2+ b3*block3+b4*block4),
         color(exemplar), shape(exemplar))
```

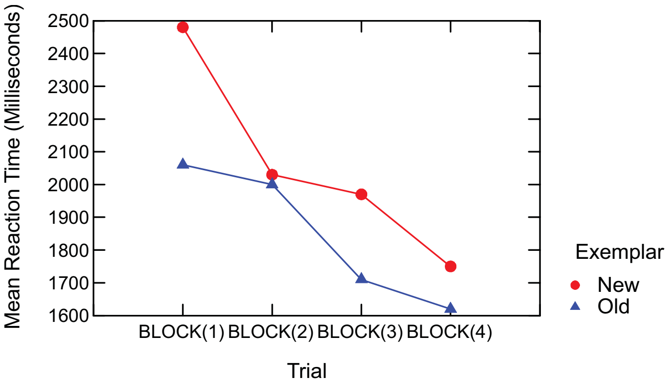


Figure 10.55 *Repeated measures experiment from Estes (1994)*

Figure 10.56 shows an application of color to distinguish groups in a nested design. The figure is based on data of Charles Darwin, reprinted in Fisher (1935). The design included 15 pairs of plants (self-fertilized and cross-fertilized) distributed across 4 pots. Plant pairs were nested within pots. Pots 1 and 2 contained 3 plant pairs each, Pot 3 contained 5, and Pot 4 contained 4.

Fisher used these data to illustrate the power of Student's *t*-test, arguing against Galton's analysis of the same data. In doing so, he noted the peculiar reversals of plant pair 2 in pot 1 and plant pair 4 in pot 4. Figure 10.56 reveals these reversals more readily than an examination of the raw numbers. The graphic is produced by the expression plant/pot*(self+cross). The vertical dimension is a blend of the two fertilization variables self and cross. This blending corresponds to the repeated measure in the design. The horizontal dimension is a nested factor plant/pot. Notice that the tick marks are spaced categorically (on integers) and indicate plant within pot. The sections within the frame are of unequal width because there are different numbers of plants within the pots. The horizontal axis is telling us that plant pair 1 in pot 1 is not the same as plant pair 1 in pot 2. Finally, the implicit variables s and c are used to color the dots according to the measure type (self or cross). Blending these creates a categorical variable set whose value is the string "Self" or "Cross" and whose index (1 or 2) is used to determine the color from a preset color table.

DATA: s = *string*("Self")
DATA: c = *string*("Cross")
ELEMENT: *line*(*position*(plant/pot*(self+cross)), *color*(s+c))

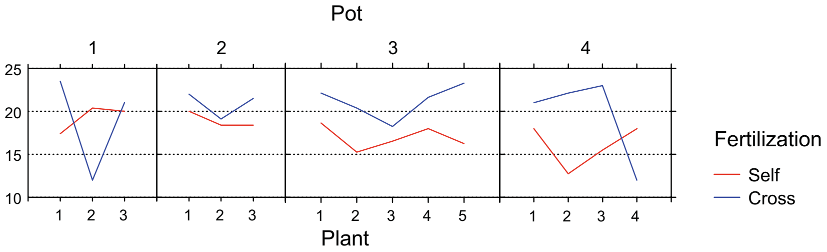


Figure 10.56 *Nested experimental design*

Figure 10.57 shows the dot plot of the Shakespeare data featured in Figure 5.7. This time we are coloring the dots according to the name of the blended variable, which we have encoded in two data strings (f and s).

DATA: f = *string*("First")
DATA: s = *string*("Second")
ELEMENT: *point.dodge*(*position*(*bin.dot*(first+second)), *color*(f+s))

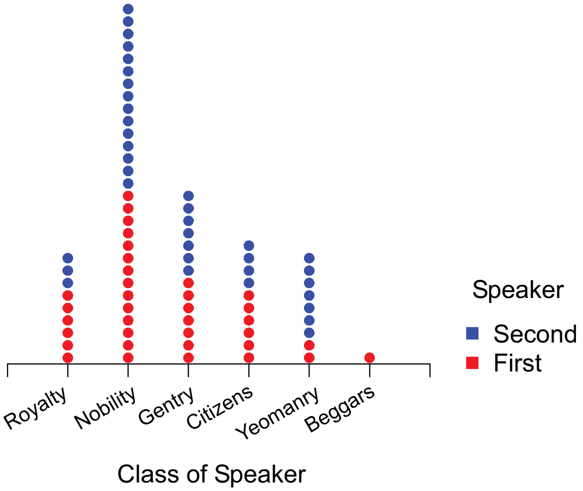


Figure 10.57 *Blended dot plot*

Several other widely used plots employ color to mark blends. Figure 10.58 shows how the *string*() function is used to create a **profile plot**. This plot has been used by individual-differences psychologists for almost a century to display profiles on psychometric tests (see Hartigan, 1975a). The vertical axis displays the subtest score and the horizontal axis shows the name of the subtest. In general, profile plots can be used to display repeated measurements of a single variable or a multivariate profile on several comparable measurements. In this example, we have used the Fisher Iris flower measurements and added color to the profiles in Figure 10.58 by referencing the species variable in the dataset. The three species are distinguished clearly in the plot. Compare this plot to the parallel coordinate plot shown in Figure 9.60. The difference between the two is that profile plots have a common measurement scale and parallel coordinate plots do not.

```
DATA: sw=string("SEPALWID")
DATA: sl=string("SEPALLEN")
DATA: pw=string("PETALWID")
DATA: pl=string("PETALLEN")
ELEMENT: line(position(sw*sepalwid+sl*sepallen+pw*petalwid+
                       pl*petallen), color(species))
```

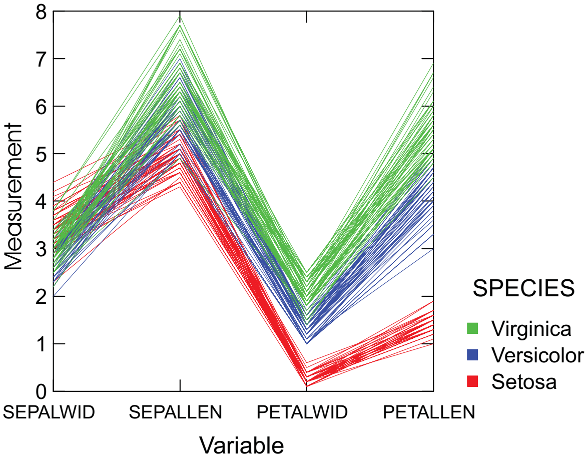


Figure 10.58 Profile plot of Iris data

10.6.6.5 Using Color in Matrix Plots

Figure 10.59 shows the entire Iris dataset plotted as a matrix using the row and column data functions. The layout of the cells is determined by the index variables. The rectangular cuts between datapoints (the lattice of row and column indices) create the tiles. These tiles are colored by *color.hue*(). The same graphic can be created through the *blend* operator. Can you write a specification to do this?

```
DATA: x = reshape.rect(species, sepalwid, sepallen, petalwid,
        petallen, "rowindex")
DATA: y = reshape.rect(species, sepalwid, sepallen, petalwid,
        petallen, "colname")
DATA: d = reshape.rect(species, sepalwid, sepallen, petalwid,
        petallen, "value")
ELEMENT: polygon(position(bin.rect(x*y), color.hue(d))
```

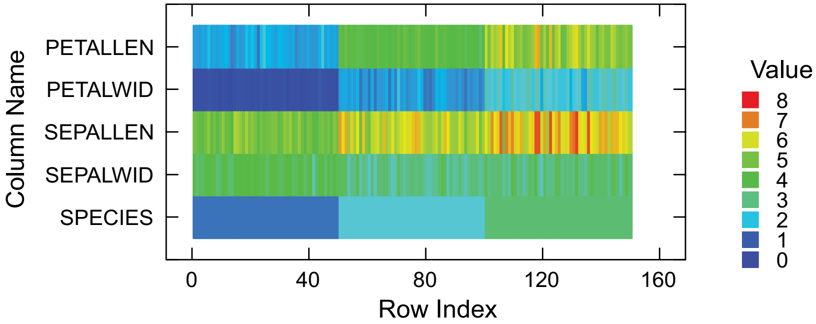


Figure 10.59 *Matrix plot of Iris data*

10.6.6.6 Choropleth Maps

Figure 10.60 shows a **choropleth map** representing the logged population of each county in the continental US. This map is based on the Bureau of the Census Summary Tape File 3A. Geographers have traditionally disparaged the spectrum, or rainbow, color scale we have used for representing population in this choropleth map, but recent perceptual research by Brewer (1996) fails to support this conventional opinion.

It is interesting to compare this map to two others: the population density map GE-70, prepared by the Geography Division at Census (reproduced in Tufte, 1983, and available as a poster from the US Government Printing Office), and the nighttime satellite composite photo of the US, available through NASA. Not surprisingly, the distribution of artificial light emanating from cities and towns matches the Census population distribution figures almost perfectly.

We logged the population of each county because the distribution of population in the US is highly skewed. Without logging, almost every county would be blue or dark green and only a few would be orange or red. An alternative approach would be to assign population a dimension and display it on a decimal log scale. Then the legend would show population on a log scale (see Chapter 6) rather than the log values themselves.

```
DATA: longitude, latitude = map(source("US counties"))
TRANS: lpop = log.10(population)
COORD: project.stereo(dim(1, 2))
ELEMENT: polygon(position(longitude*latitude), color.hue(lpop))
```

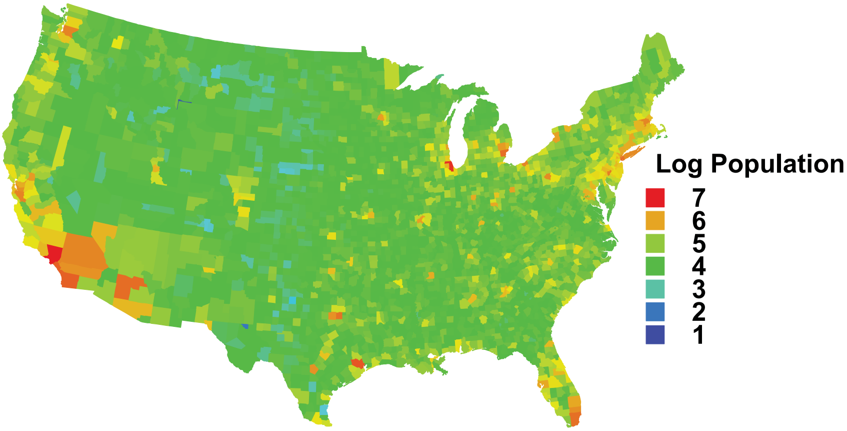


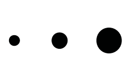
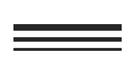
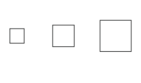
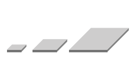
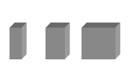
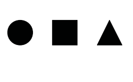
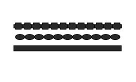
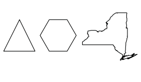
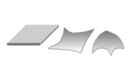
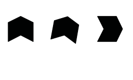
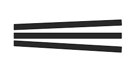
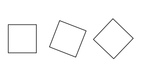
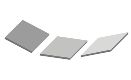
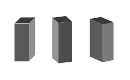
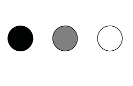
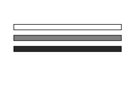
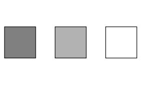
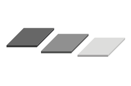
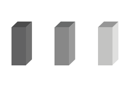
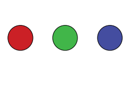
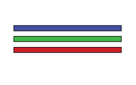
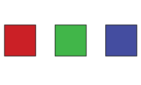
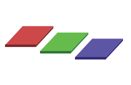
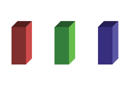
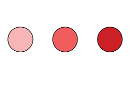
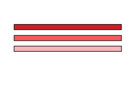
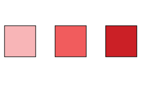
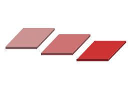
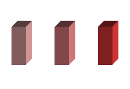
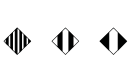
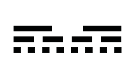
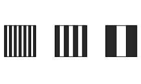
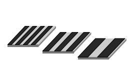
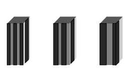
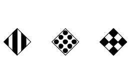
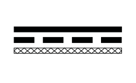
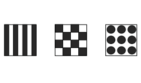
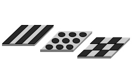
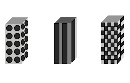
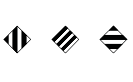
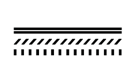
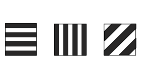
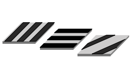
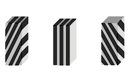
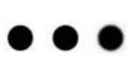
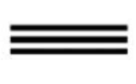
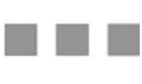
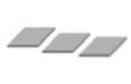
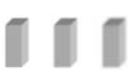
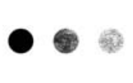
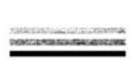
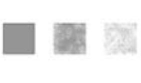
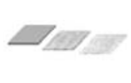
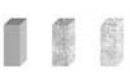
Figure 10.60 *Color map of logged county population*

10.7 Summary

The major attributes discussed in this chapter are summarized in Table 10.2. The columns represent geometric forms that are employed by different graph types: point, line, area, surface, and solid. The rows represent the attributes divided into four super-classes: form, color, texture, and optics.

Table 10.2 is relatively straightforward except for the subtable covering form attributes. In other sections of the table, we are free to vary an attribute independently of the other attributes. For example, the hue of any geometric object is free to vary independently of position, size, shape, and other attributes. With form attributes, however, we sometimes collide with the constraints of position. The size of a point is free to vary independently of its position because changing its size does not change its location. The size of a line, however, can vary only in its thickness unless its length is not constrained by a positional variable. The size of an area can be changed only if its boundary is not determined by a positional variable. The size of a *polygon*, for example, can be modified under certain circumstances, but the size of an *area* cannot. For similar reasons, the shape of an *area* can be changed only if it is not constrained by a positional variable. Similar rules apply to rotation of lines, areas, surfaces, and solids. In each case, we must look for a degree-of-freedom from positional constraint.

Table 10.2 **Aesthetic Attributes by Geometry**

	Point	Line	Area	Surface	Solid
Form					
Size					
Shape					
Rotation					
Color					
Brightness					
Hue					
Saturation					
Texture					
Granularity					
Pattern					
Orientation					
Optics					
Blur					
Transparency					

A *point* is rendered by a symbol, polygon, or image positioned at the coordinates of the *point*. It has the most degrees-of-freedom of any object because we can vary every attribute independently of position. Point objects maintain their appearance under different coordinate systems. For example, the shape of a *point* graphic does not change under a polar transformation.

A *line* is rendered by one or more symbols, polygons, or images positioned at the coordinates of points on that line. Dashed lines, for example, are rendered by placing a small number of rectangular polygons or symbols at points along a line. We change the size of a line by increasing the size of those symbols in a direction orthogonal to the line itself. This increases the thickness of the line. The shape of a line is determined by the shape of the object used to render it — polygon, image, or symbol. Its granularity is determined by the number of symbols used to render it (if a dashed shaped is used) or by the granularity of the pattern that fills the line polygon.

An *area* is rendered by one or more symbols, polygons, or images positioned at selected coordinates on a plane. We can vary the size of an area, its shape, or its rotation only if we have degrees-of-freedom to do so.

A *surface* is rendered by positioning one or more symbols, polygons, or images in a 3D space so that the coordinates of one of their vertices or their centroids lie on a surface. As with lines in 2D space, we vary size of surfaces by varying their thickness. For example, we could make the surfaces thicker in Figure 8.4 by adding a *size*() attribute to the graphics. We can vary shape by changing the shape of the polygons or elements used to render it. This could be used to make a surface bumpy or smooth, for example. We can vary the exterior shape of a surface if we have the degrees-of-freedom to do so. We can also modify the texture of a surface (independently of shape) by changing the texture map used to render it.

A *solid* is rendered similarly to a surface. Depending on how they are employed by a graphic, solids may have more than one degree-of-freedom available for varying attributes like size, shape, and rotation. Thus, we may change the size of one or more facets of a rectangular solid and we can vary its rotation along one or more dimensions depending on which facets are positionally anchored. We have shown this in the last column of Table 10.2. If a solid has no degrees-of-freedom for representing an attribute such as size, then we cannot vary such an attribute.

10.8 Sequel

We have so far concentrated on single graphics. Often, we want to create tables and other structures of graphics. The next chapter covers facets, which create such multiplicities.

11

Facets

The English word **facet** is derived from the Latin *facies*, which means face. A facet implies a little face, such as one of the sides of an object (*e.g.*, a cut diamond) that has many faces. This word is useful for describing an object that creates many little graphics that are variations of a single graphic. In a graphical system, facets are frames of frames. Because of this recursion, facets make frames behave like points in the sense that the center of a frame can be located by coordinates derived from a facet. Thus, we can use facets to make graphs of graphs or tables of graphs. Indeed, tables *are* graphs. This general conception allows us to create structures of graphs that are more general than the specific examples of multigraphics such as scatterplot matrices (Chambers *et al.*, 1983), row-plots (Carr, 1994), or trellises (Becker and Cleveland, 1996). We can also construct trees and other networks of graphs because we can link together graphic frames in the same way we link points in a network. And we can transform facets as well as frames to make, for example, rectangular arrays of polar graphics or polar arrangements of rectangular graphics. For a similar concept in the field of visualization, see Beshers and Feiner (1993).

All the machinery to make a faceted graphic can be derived from the definition of a frame itself in Chapter 2. In other words, a facet is simply another frame extended from the frame class. Its name denotes its role as a pattern maker for other frames. How patterns of frames are produced, organized, and displayed is a matter of attaching various schemes to their drawing methods. These schemes may be driven by explicit data (as in most of the examples in this chapter) or by implicit methods that involve simple iterations.

11.1 Facet Specification

Facets are embeddings. A facet of a facet specifies a frame embedded within a frame. A facet of a facet of a facet specifies a frame embedded within a frame embedded within a frame. Each frame is a bounded set which is assigned to its own coordinate system. To construct facets, therefore, we use the COORD specification.

For example, the three-dimensional frame

COORD: *rect*(*dim*(1, 2, 3))

consists of a 3D frame. The *rect*() function denotes rectangular coordinates. Adding graphics to this specification produces a single 3D plot. By contrast, the three-dimensional frame

COORD: *rect*(*dim*(3), *rect*(*dim*(1, 2)))

consists of a 2D frame represented by the first two measures embedded in the 1D facet frame represented by third variable. Adding graphics to this specification would produce a row of 2D plots.

More complex facet structures follow the same model. That is, a four-dimensional graph can be realized in four 1D frames, a 1D frame embedded in a 2D frame embedded in a 1D frame, two 2D frames, a 3D frame embedded in a 1D frame, and so on. A chain of nested COORD specifications is a tree of facets, with the first child of the second, the second child of the third, and so on.

11.2 Algebra of Facets

Trees and tables are alternate ways to represent facets. Table 11.2 lists a variety of facets sharing a common COORD specification. We have omitted the first variable set (say, x) associated with the first dimension in order to show the remaining terms in the frame that produce a table in each case. You may think of these examples as involving a dot plot of x for each combination of the facets determined by the variables in Table 11.1.

Table 11.1 contains the data on which Table 11.2 is based.

Table 11.1 **Data for Table 11.2**

a	b	c	d
Barb	Jean	Young	Short
Jean	Jean	Young	Short
Barb	Mark	Young	Short
Jean	Jean	Old	Short
Jean	Jean	Old	Tall
Jean	Jean	Old	Tall

The first row of Table 11.2 shows the simple expression a, which yields a tree with two branches and a table with two cells, one for each value of a. We have used a dot in each cell to represent the presence of a graphic within the cell. We could put a one-dimensional graphic in each of the two cells of this table with the expression x*a. The full specification for a one-dimensional scatterplot split into these two cells would therefore be:

COORD: *rect*(*dim*(1), *rect*(*dim*(2)))
ELEMENT: *point*(*position*(x*a))

The specification for a two-dimensional scatterplot split into these two cells would be:

COORD: *rect*(*dim*(3), *rect*(*dim*(1, 2)))
ELEMENT: *point*(*position*(x*y*a))

The second row of Table 11.2 shows a crossing a*b, which produces a 2x2 table in this case. We should not assume this operator always implies a cross-tabulation structure, however. As we will show in row 5 of Table 11.2, a crossing can look superficially like a nesting. Market-research, data-mining, and statistical tabulation packages fail to make this distinction and thus miss the fact that crossing is an aspect of a frame, not a layout. A frame and a view must be carefully distinguished. Notice that only three of the four cells are filled. The fourth cell has no dot, but there remains the possibility that data may be encountered that fit this combination of categories.

The third row shows the nested expression a/b. As the definition of nesting in Chapter 2 reveals, the term a*b is two-dimensional, while the term a/b is one-dimensional. In other words, there is only one scale of values above the three cells in row 3 of Table 11.2. We have made these reference values italic to indicate that they are nesting rather than crossing categories.

The distinction between row 2 and row 3 may appear subtle and of only surface importance. Imagine, however, the difference between the following two associations. The first to consider is the crossing of gender with marital status. The possible combinations are married men, married women, single men, and single women. The second is the nesting of pregnancy status under gender. The possible combinations are pregnant women, non-pregnant women, and non-pregnant men. Non-pregnancy has a different meaning for men than for women. Marital status by gender is two-dimensional because all possible combinations of the values are possible. Pregnancy status by gender is one-dimensional because only three combinations are possible and the values of status are not comparable across the gender variable. We could design a variety of physical layouts to convey the difference between nesting and crossing, but any layout for nesting must clearly show only the possible combinations. If a cell is left empty, we might imagine that it could be filled by an observation. This must never occur for a physical representation of an impossible category in a nested structure.

Row 4 shows a blending. A blending is a one-dimensional operation that collapses tied categories. We see only three cells because the name Jean within variable a is taken to be the same value as the name Jean within variable b.

Row 5 illustrates the subtlety with crossing that we mentioned in discussing row 2. It also illustrates the use of a constant value (**1**) as a place-holder. The a variable set is crossed with **1** to yield one row of two values. This set, in turn, is split into two by the values of b. The placement of the **1** is critical. The sequence of orientations in the faceted coordinate system is *h, v, h, v, ...* , where *v* is vertical and *h* is horizontal. The lattice on which this coordinate system is based has no limit. Remember, also, that the cross operator is not commutative.

Row 6 shows a tree and table for the expression (a + b) * c, which is equivalent to a*c + b*c. Notice, as in row 4, that there are only three levels for the blended dimension because of the presumed tie in data values.

Row 7 shows a tree and table for the expression (a + b)/c, which is equivalent to a/c + b/c. This operation blends two trees like the one in row 3.

Row 8 illustrates the expression a*b + c***1**. There is no row label for the two columns Y and O because the crossing set for this group is unity. Using a nesting operator in combination with crossing may produce row or column labels in the middle of the aggregated table, however (see row 12).

Table 11.2 **Tables and Trees**

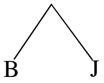
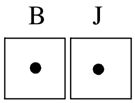
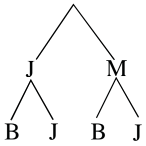
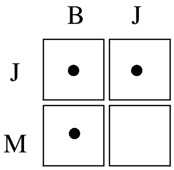
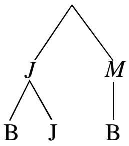
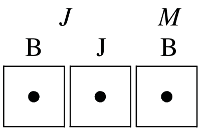
Expression	Tree	Table
1 a		
2 a * b		
3 a/b		

Table 11.2 **Tables and Trees (Continued)**

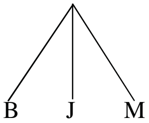
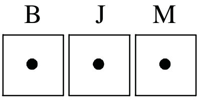
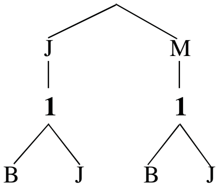
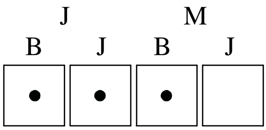
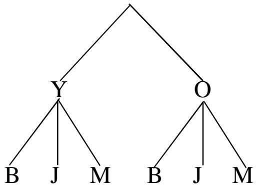
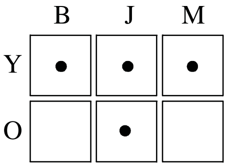
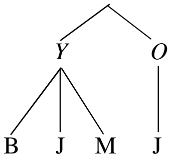
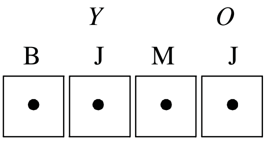
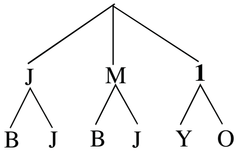
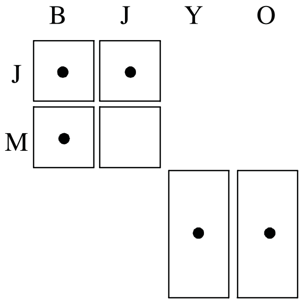
Expression	Tree	Table
4 a+b		
5 a*1*b		
6 (a+b)*c		
7 (a+b)/c		
8 a*b+c*1		

***Table 11.2* Tables and Trees (Continued)**

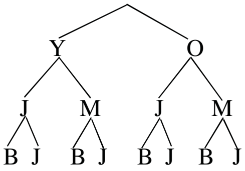
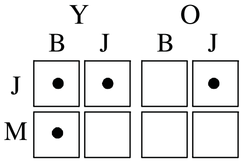
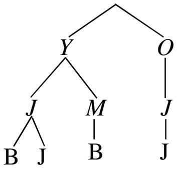
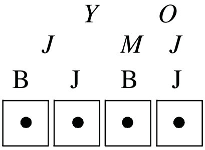
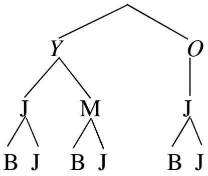
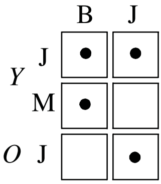
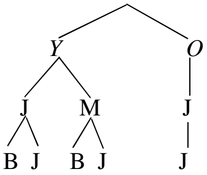
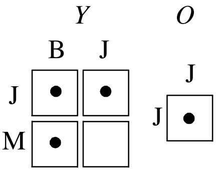
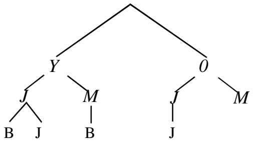
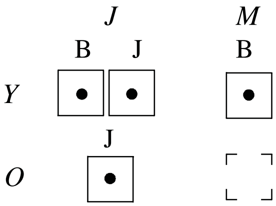
Expression	Tree	Table
9 a*b*c		
10 a/b/c		
11 a*b/c		
12 (a*b)/c		
13 a/(b*c)		

Table 11.2 Tables and Trees (Continued)

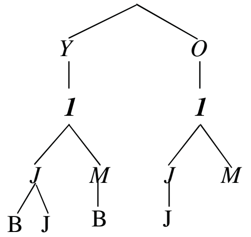
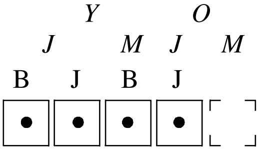
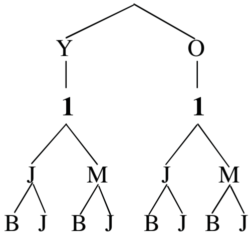
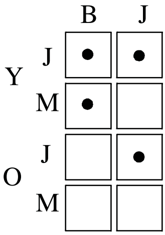
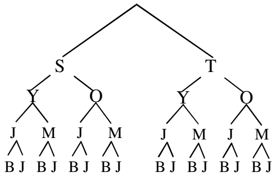
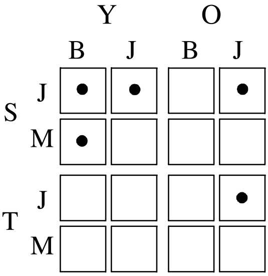
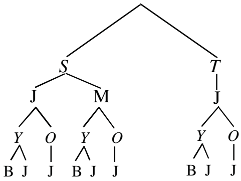
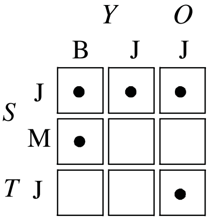
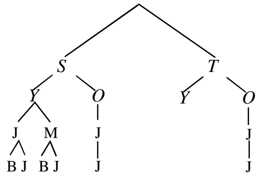
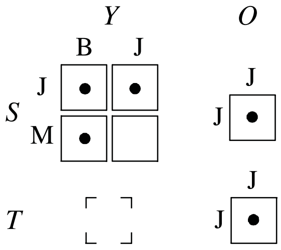
Expression	Tree	Table
14 a/(b***1***c)		
15 a*b***1***c	 Y O 1 1 J M J M B J B J B J B J	
16 a*b*c*d	 S T Y O Y O J M J M J M J M B J B J B J B J B J B J B J B J	
17 a/c*b/d		

Table 11.2 **Tables and Trees (Continued)**

Expression	Tree	Table
18 (a*b)/(c*d)	 S T Y O Y O J M J J B J B J J J	 Y O B J J S J M J J J T J

Row 9 illustrates a three-way crossing a*b*c. Because the COORD specification for Table 11.2 is 2D, we end up with two groups of 2x2 cross-tabulations. Many of the cells are empty, but the layout of the table signals that these combinations are theoretically possible.

Row 10 illustrates a three-way nesting a/b/c. Only combinations existing in the data appear in the tree and table.

Row 11 illustrates the expression a*b/c, which is a crossing with a nested factor. The layout shows a full crossing, but the vertical table axis contains italic labels to indicated that the cells fall under a nesting of b within c.

Row 12 shows a tree and table for the expression (a*b)/c. This example has an unusual structure that shares a number of subtleties with the example in row 8. We are nesting a crossing under variable c. To do this, we must examine each separate value of the nesting variable c for all combinations of the crossing variables a and b. As the tree and table show, there are two values of a and b available to cross under c = "Young" but only one value of a and b under c = "Old". Consequently, we produce a 2x2 and 1x1 table to nest under c.

Row 13 shows a tree and table for the expression a/(b*c). The b*c part of the term determines a two-way table under which the levels of a are nested.

Row 14 shows a tree and table for the expression a/(b*1*c). This time, the b*1*c part of the term determines a crossing aligned in a row because of the unity operator taking place for a crossed variable. The levels of **a** are nested within each of these levels of crossing.

Row 15 shows a table that, on the surface, appears to be a crossing of a nested variable with another. The corresponding expression a*b*1*c shows, however, that this is a three-dimensional rather two-dimensional table. Reinforcing this in the view is the lack of italics that would denote nesting.

Row 16 shows a simple four-way table. The expression a*b*c*d produces a table with many empty cells, but is fully crossed.

Row 17 shows a two-dimensional table produced by the nested expression **a/c * b/d**. This expression is a simple crossing of two nested variables.

Finally, row 18 shows a nesting of two crossings. Producing this table requires us to compute the crossing c*d and then examine every possible crossing a*b nested within each of the resulting combinations. The layout seems to violate the rule that a nesting cannot be represented as a crossing. There are no data under the combination T by Y. Nevertheless, the expression tells us that this is a meaningful combination even if there are no values for it in the data set. Therefore, a crossed layout is required. There are two dimensions in the resulting graph.

11.3 Examples

Multiplots are graphics faceted on extrinsic variables. For example, we might want to plot heart rate against blood pressure for different clinical populations or for males and females separately. These would be categorical multiplots. We could also make scatterplots of scatterplots, employing continuous variables to lay out the graphics. These would be continuous multiplots. The following sections show examples of both kinds.

11.3.1 One-Way Tables of Graphics

11.3.1.1 Tables of Scatterplots

Figure 11.1 shows a table of ordered categories. The data are from the sleep dataset. Each scatterplot of body weight against sleep is ordered by the danger-of-being-eaten index. This enables us to discern a trend in the orientation and position of the scatterplot cloud. The animals most in danger tend to have the lowest levels of sleep and highest body weights.

```
SCALE: log(dim(1), base(10))
COORD: rect(dim(3), rect(dim(1, 2)))
ELEMENT: point(position(bodyweight*sleep*danger))
ELEMENT: contour(
           position(region.confi.mean.joint(bodyweight*sleep*danger)))
```

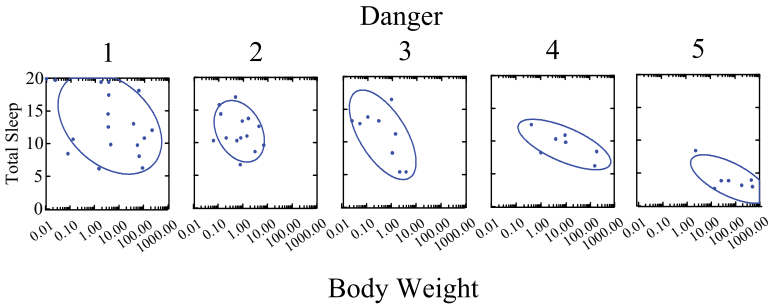


Figure 11.1 *Table of scatterplots*

11.3.1.2 Tables of 3D Plots

Figure 11.2 shows a table of 3D *interval* graphics. The data are from the king crab dataset used in Figure 10.38. Each bar measures the yield of crabs per pot at the sampled location. The plots are ordered by year, revealing the decline in yield over a six-year period.

```
DATA: longitude, latitude = map(source("Kodiak"))
COORD: rect(dim(4, 5), rect(dim(1, 2, 3)))
ELEMENT: polygon(position(longitude*latitude*yield*1*year),
         color(color.green))
ELEMENT: interval(position(lon*lat*yield*1*year), color(color.blue))
```

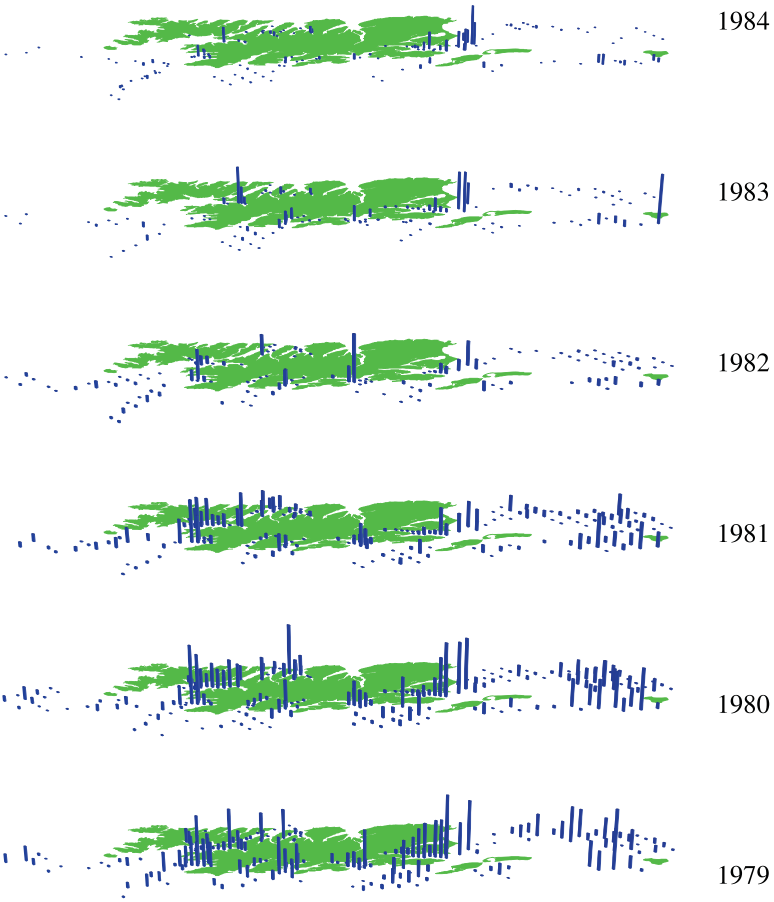


Figure 11.2 *Table of 3D bars*

11.3.2 *Multi-way Tables*

Multi-way tables are produced by categorical facet variables. The popular word for this is **cross-tabs**. The following examples show how to embed a variety of graphics within multi-way tables.

11.3.2.1 Simple Crossing

Figure 11.3 shows a two-way table of line plots using the barley dataset from Chapter 10. The barley yields by variety have been plotted in an array of site by year. The average yields have been ordered from left to right. This table follows a general layout that we prefer for tables of graphics. The top and left axes are devoted to the table variables and the bottom and right for the scale variables. The table values are ordered from top to bottom and left to right. This fits the format table viewers are accustomed to seeing. The scale variables are ordered according to the usual layout for a single graphic. If higher-way tables of graphics are needed, then the scale variables are always moved to the bottom and right sides of the tables. This makes table look-up a left-top scan for categorical information and a bottom-right scan for scale information. The regularity of this layout generalizes nicely to any number of facets.

The problem with this layout for this particular graphic is that the lengthy string values for variety on the horizontal axis are unreadable. Ordinarily, this scale would be numerical and there would be only a few tick marks. With a categorical variable, however, we need one tick per category. Thus, when we have a categorical variable on an inner frame, we often need to change the layout. We will fix this problem in the next figure.

COORD: *rect*(*dim*(3, 4), *rect*(*dim*(1, 2)))
ELEMENT: *line*(*position*(variety*yield*site*year))

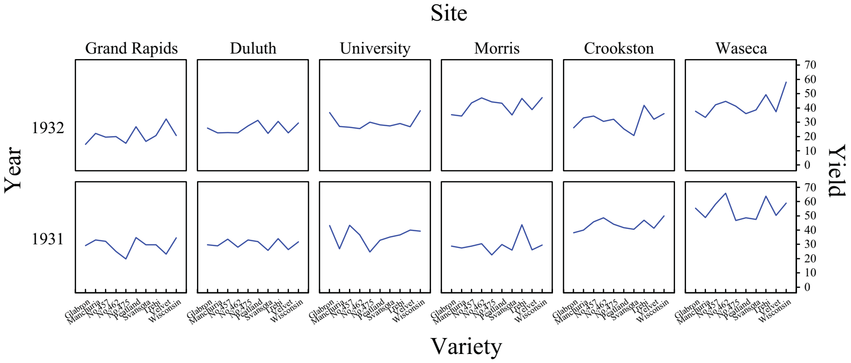


Figure 11.3 *Two-way table of graphics*

11.3.2.2 Transposed Tables

Figure 11.4 transposes the table in Figure 11.3 to provide more readable labeling, following a layout in Cleveland (1995). This makes the range of the inner frames (yield) horizontal and the domain (variety) vertical.

COORD: *transpose*(*rect*(*dim*(3, 4), *rect*(*dim*(1, 2))))
ELEMENT: *line*(*position*(variety*yield*site*year))

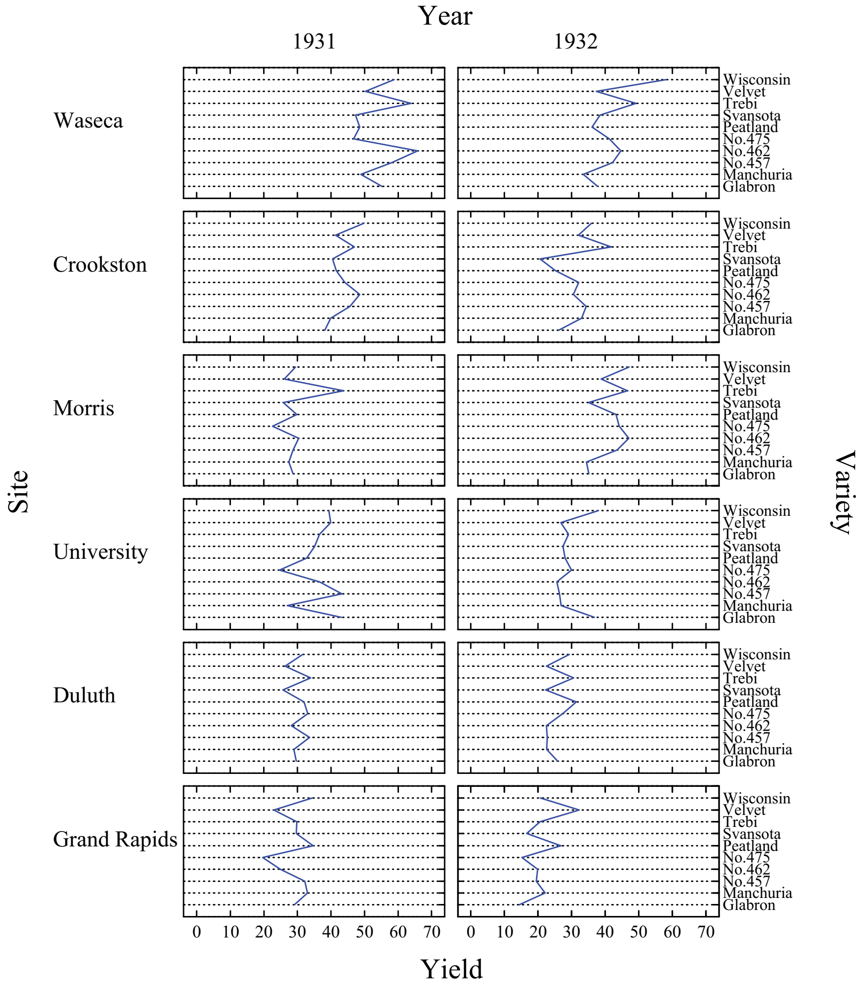


Figure 11.4 *Transposed barley data*

11.3.2.3 Blended Facets

Figure 10.56 showed a nested design based on the data from Darwin used in Chapter 10. We can rearrange the nested factor to produce another tabular display of the same data. Figure 11.5 shows an example. This time, we use the nested variable plant/pot to determine the color used for the *line* graphic. This causes *line* to split into a different number of graphics in each panel.

The first facet consists of the blend s*self + c*cross. This is plotted in rectangular coordinates. The second frame consists of **1***pot, which makes the panels stack vertically. The expanded expression is equivalent to the blend of s*self***1***pot + c*cross***1***pot.

```
DATA: s = string("SELF")
DATA: c = string("CROSS")
COORD: rect(dim(3, 4), rect(dim(1, 2)))
ELEMENT: line(position((s*self+c*cross)*1*pot), color(plant/pot))
```

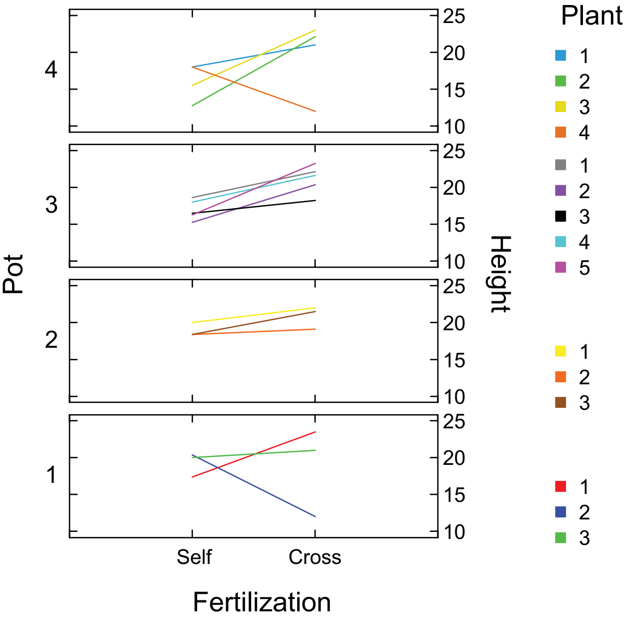


Figure 11.5 *Darwin data*

11.3.2.4 Tables of Mathematical Functions

Tables of graphics are often useful for visualizing different slices through higher-dimensional objects. Figure 11.6 shows an example of a 3D object we can tabulate. It is a beta density function (*bdf*) for parameters p and q, where $q = p$. We have truncated the vertical axis to show more detail, since the spiked tails in the rear go to infinity.

DATA: x, p = *mesh*(*min*(0, 0), *max*(1, 10))
TRANS: z = *bdf*(x, p, p)
ELEMENT: *surface*(*position*(x*p*z), *color.hue*(z))

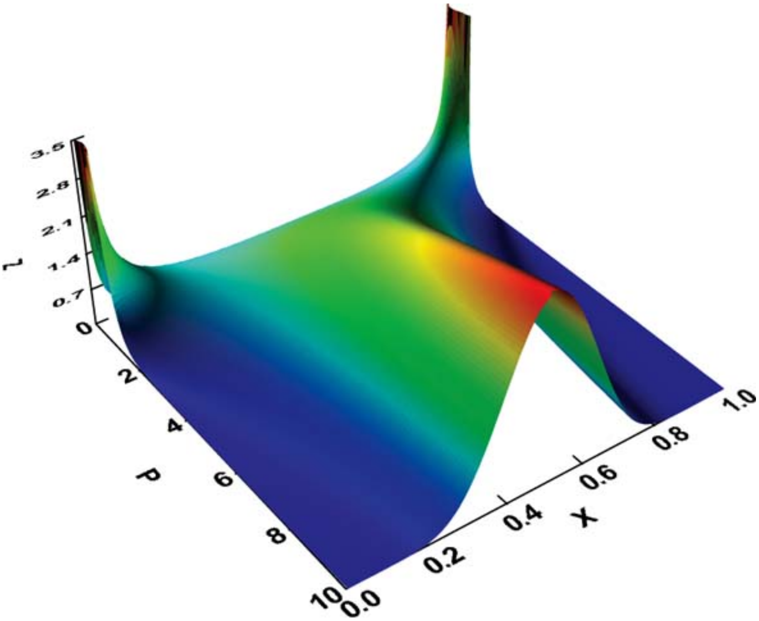


Figure 11.6 *Beta density*

Figure 11.7 shows a table of slices through this density. We vary the two beta parameters independently. The values of the *p* and *q* parameters determine the orientation of the outer frame. Slices of Figure 11.6 can be seen on the diagonal of the table from the lower left to the upper right.

DATA: x, p, q = *mesh*(*min*(0, .1, .1), *max*(1, 6.1, 6.1), *n*(100, 4, 4))
TRANS: z = *bdf*(x, p, q)
COORD: *rect*(*dim*(3, 4), *rect*(*dim*(1, 2)))
ELEMENT: *line*(*position*(z*x*p*q))

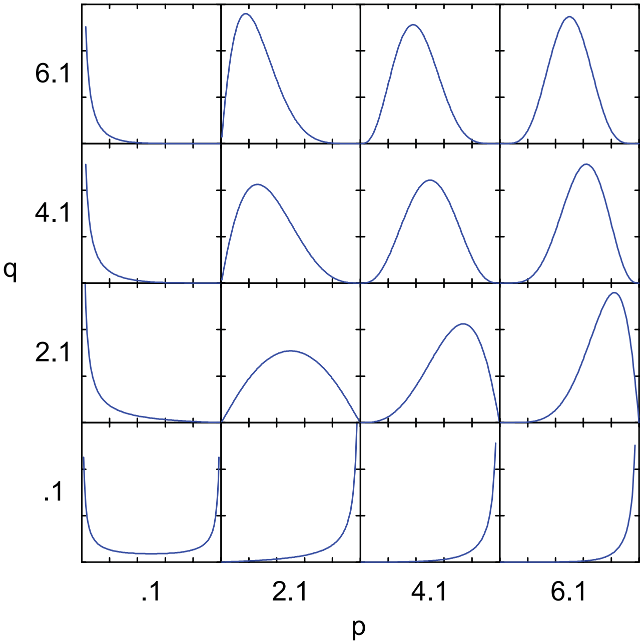


Figure 11.7 *Table of beta densities*

11.3.2.5 Nesting under Crossed Tables

Figure 11.8 shows a graphic nested within a crossed table. The data are from Lewis *et al*. (1982). This was a study of the fixation of six different types of tibial prosthetic components used in total knee operations. The authors used finite-element analysis to determine the maximum stress at several locations around the prostheses. The variables were **configuration** (type of prosthesis component), **material** (metal or polyethylene), **mode** (location and type of stress), and **stress** (meganewtons per square meter under 2000-newton total load).

DATA: configuration = *link*("prostheses")
COORD: *rect*(*dim*(2, 3), *transpose*(*rect*(*dim*(1))))
ELEMENT: *interval*(*position*(stress / (mode*material/configuration)))

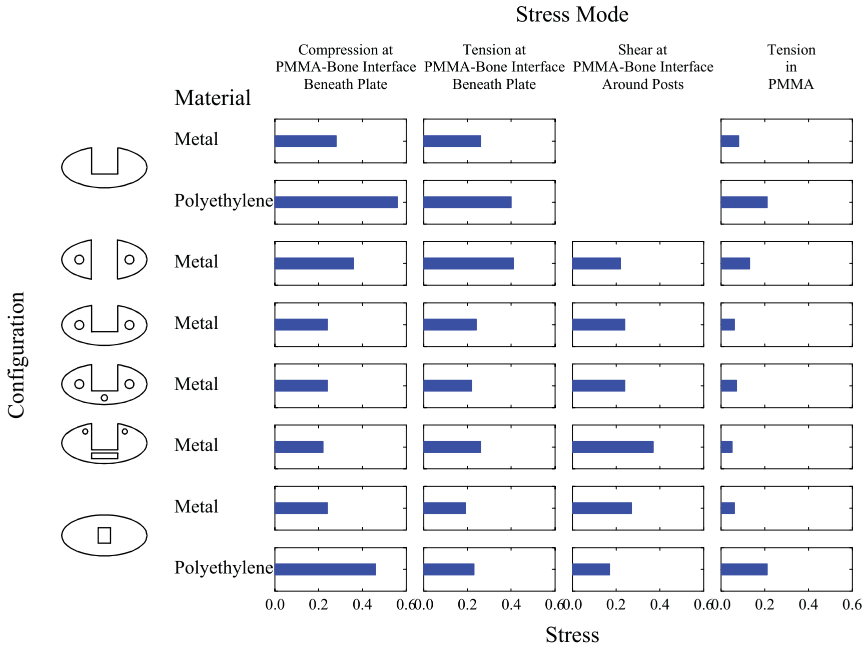


***Figure 11.8** Stress measures on total knee prostheses*

The material is nested under configuration because not all designs are available in both plastic and metal. To produce the outer crossing, the stress mode is crossed with material/configuration. To make the left margin more readable, the values of configuration are associated via metadata links with images of the prostheses. There are 32 potential graphics in the crossing (4 stress modes and 8 materials given configuration). The stress is not measurable at the Shear at PMMA-Bone Interface Around Posts value of mode, however. This is because the first configuration (top left in the figure) has no posts (represented in the other images by small circles to the right and left of the central notch). Consequently, there can be no bar graphic and no frame for two of the combinations. This situation is *not* like having missing stress data or zero values, however. The nesting tells us that the combination is not defined for stress.

This may seem to be a small point, but it is important. The original graphic in Figure 5 of Lewis *et al.* (1982) places all the bars in a regular grid of 32 frames and leaves these two cells empty, assuming the readers will realize that the measurement of stress under this combination makes no sense. This is confusing. To see why, we can draw an analogy from survey research. Pollsters are careful to distinguish several types of missing values for a response: "refuse to answer" *vs.* "no opinion" *vs.* "not asked" and so on. Their care shows that it is important to understand whether lack of information is due to a structural condition or to a failure to measure.

One remedy for the original journal graphic layout would be to place the words *Not Measurable* in these two empty cells. That would substitute for the two missing inner frames in the layout of Figure 11.8. This strategy would be satisfactory for a printed graphic, but it would not satisfy the structural problem in a live display system. The reason we need the nesting operator is so that a query to an area containing a missing frame yields a different response (concerning metadata, links to other graphics, *etc.*) than does a query to an area containing an empty frame.

There is another consequence of nesting that profoundly affects the functioning of a graphics system involving tables. Some market research programs compute statistical tests such as chi-square tests and *t*-tests on tables that violate basic assumptions needed for inference. This happens because these programs have no concept of statistical independence when they are given data to tabulate. A system that correctly implements the cross and nest operators can be designed to prevent these situations. The details of this implementation are beyond the scope of this book, but solving these problems can yield for the first time a system that is at once capable of producing complex tables and appropriate statistical tests.

One final point. With the crossing of mode*material/configuration, we are assuming that some possible variable (not stress) *could* be measured under all the combinations. This assumption is implicit in the layout of Lewis *et al.*'s Figure 5. It could be argued, however, that mode is not crossed but is instead nested under material/configuration because no measurement of any kind could be made around nonexistent posts. This argument highlights the role of algebra versus display. An automated system cannot tell us how to structure displays if extrinsic knowledge is required. Making specifications explicit, however, is the only way to insure that we will recognize the difference between two graphics that are visually similar but structurally different.

11.3.3 Continuous Multiplots

Continuous multiplots are rare, but have their uses. Scatterplots can be positioned on a map, for example, to show the relations among variables at different geographical locations. Scatterplots of scatterplots can sometimes reveal second-order relations among non-spatial variables as well.

11.3.3.1 Scatterplots of Scatterplots

Figure 11.9 shows a scatterplot of scatterplots. The spacing of the plots is determined by an additive linear model based on the means of beginning and current salary at a Chicago bank sued for discrimination (SPSS, 1996). We used the CONJOINT procedure in SYSTAT to compute the marginal scales. The unity constant (**1**) insures that the gender variable plots in a vertical orientation. The last facet is thus **1***gender. The salary spacing makes sense for education but is harder to defend for the other variables.

COORD: *rect*(*dim*(5, 6), *rect*(*dim*(3, 4), *rect*(*dim*(1, 2))))
ELEMENT: *point*(*position*(salbeg*salnow*education*race***1***gender))

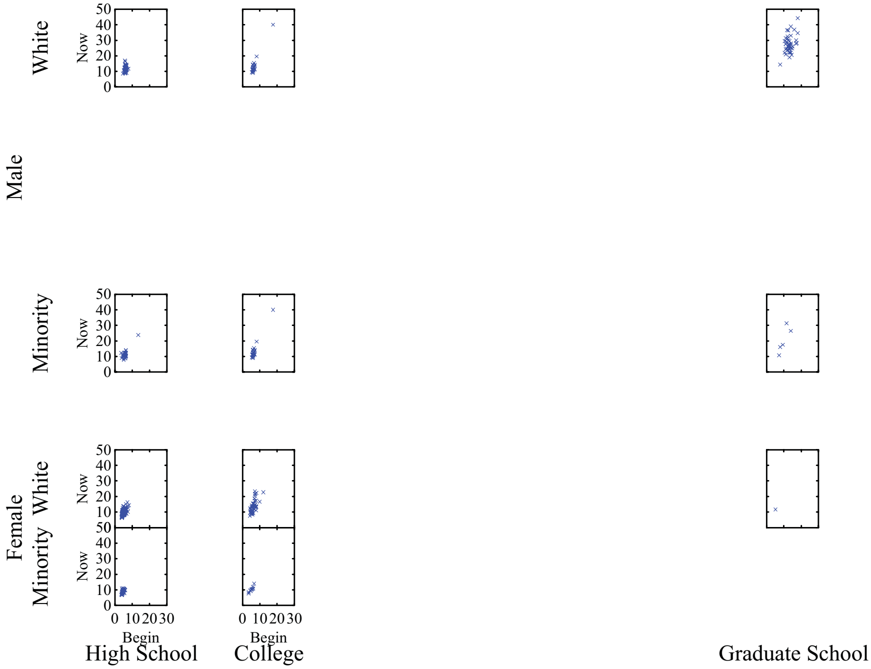


Figure 11.9 Bank salaries

11.3.4 Scatterplot Matrices (SPLOMs)

The **scatterplot matrix** (SPLOM) was invented by John Hartigan (1975b). The idea has been rediscovered several times since and has been most extensively developed by the research group originally at Bell Labs (Chambers *et al.*, 1983; Cleveland, 1985). The SPLOM replaces the numbers in a covariance or correlation matrix with the scatterplots of the data on which they were computed. Most SPLOM's are symmetric, but they can be constructed from rectangular sub-matrices as well. Hartigan included scatterplots in off-diagonal cells and histograms in the diagonal cells, but other graphics may be used.

Figure 11.10 is a SPLOM of the 1997 EPA emissions data for cars sold in the US. The variables are horsepower (HP), miles-per-gallon (MPG), hydrocarbons (HC), carbon monoxide (CO), and carbon dioxide (CO2). The emissions of pollutants are measured in per-mile weight. We use the exponentiation operator (^) to save space in the quadratic *position* specification.

```
DATA: s1 = "HP"
DATA: s2 = "MPG"
DATA: s3 = "HC"
DATA: s4 = "CO"
DATA: s5 = "CO2"
COORD: rect(dim(3, 4), rect(dim(1, 2)))
ELEMENT: point(position((hp/s1+mpg/s2+hc/s3+co/s4+co2/s5)^2))
```

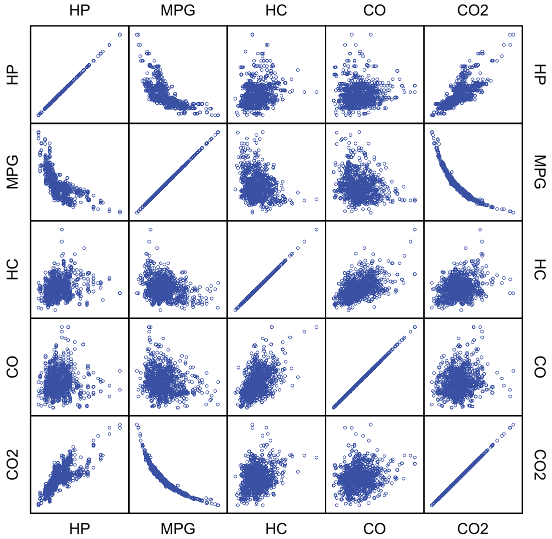


Figure 11.10 *Scatterplot matrix of EPA data*

11.3.5 Facet Graphs

Facets can be proxies for graphs themselves. By structuring the organization of a facet through a function, we can orient graphics in more complex or customized structures. See Butler and Pendley (1989) for an example.

Facet graphs extend graphing functions such as *link*(). They require data to construct graphs of graphs. Like maps, facet graphs use sets of variables (relational tables) that are linked through common keys or functions. For example, to make a tree of scatterplots, we need triples (*e.g.*, xnode, ynode, parent) for the tree and tuples (*e.g.*, **x**, **y**) for the scatterplot. The following examples illustrate how this works.

11.3.5.1 Trees

Organizational charts, clustering trees, prediction trees, and other directed graphs offer a superstructure for embedding graphics. Figure 11.11 shows a regression tree (Breiman *et al.*, 1984) for predicting accident rates from sociometric variables aggregated by states in the continental US.

We are assuming that the *tree*() coordinate system includes the annotation methods for delineating the branches. This is a rather heavyweight assumption, with a lot of accompanying code to accomplish it. The boxes around the dot plots are not the problem. It is the edges connecting them. Further work needs to be done on this idea.

```
COORD: tree(dim(2, 3), rect(dim(1)))
ELEMENT: point.dodge(position(bin.dot(accident*xnode*ynode)),
         color(leaf))
```

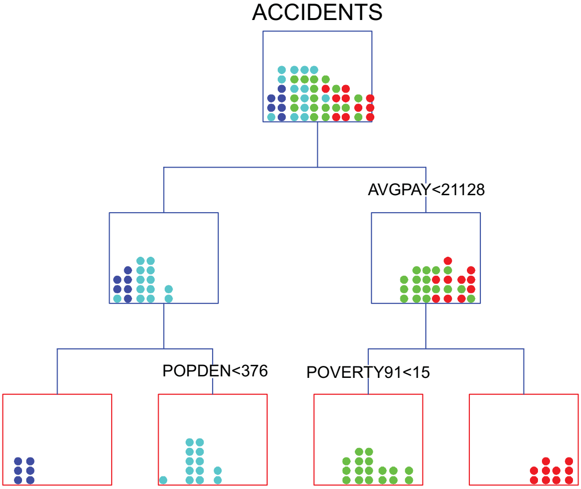


Figure 11.11 *Regression tree predicting accident rates*

Incidentally, this tree is called a *mobile* (Wilkinson, 1997). This display format gets its name from the hanging sculptures created by Calder and other artists. If the rectangles were boxes, the dots marbles, the horizontal branches metal rods, and the vertical lines wires, the physical model would hang in a plane as shown in the figure. Each box contains a dot density based on a proper subset of its parent's collection of dots. The scale at the bottom of each box runs from low accident rates on the left to high on the right. The dots are color coded according to which terminal (red) box they fall in. There are 46 dots altogether because two states have missing data.

11.3.5.2 Reflection: Population Pyramids and Dual Histograms

Reflecting facets enables us to take advantage of symmetry to contrast paired graphics. The most popular application of this method is a favorite plot of demographers, called a **population pyramid** (Cox, 1986). This plot places age histograms back-to-back, one for males and the other for females. Figure 11.12 shows an age–sex pyramid for the United States, based on the 1980 Census. The coordinate function *mirror*() implements a composite transformation consisting of reflection and translation. For an even number of categories on a facet (usually two), it reflects half of the graphics contained in the facet.

COORD: *transpose*(*mirror*(*dim*(3), *rect*(*dim*(1, 2))))
ELEMENT: *interval*(*position*(age*pop*sex), *color*(sex), *size*(2))

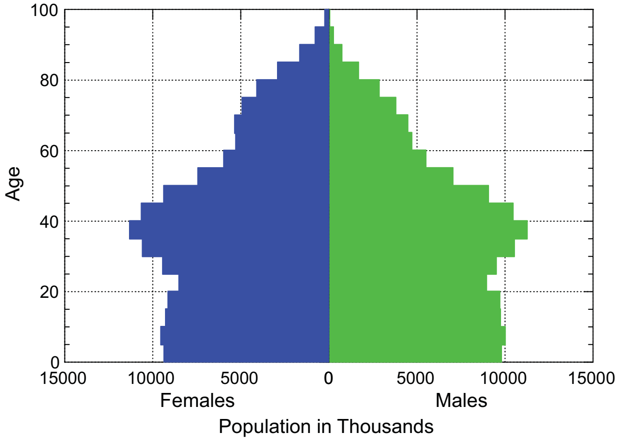


Figure 11.12 *Age–sex pyramid of 1980 US population*

Dallal and Finseth (1977) extended this idea to general dual histograms. Figure 11.13 shows a dual histogram for times in the 2004 Chicago Marathon. The panels are split by gender. An interesting aspect of the distribution is the discontinuity at four hours. Runners near that boundary evidently push themselves to come in under the limit. Roger Dubbs ran in this marathon; his time was 3:34:28.

COORD: *mirror*(*dim*(3), *rect*(*dim*(1, 2)))
ELEMENT: *interval*(*position*(hours*count*sex), *color*(sex))

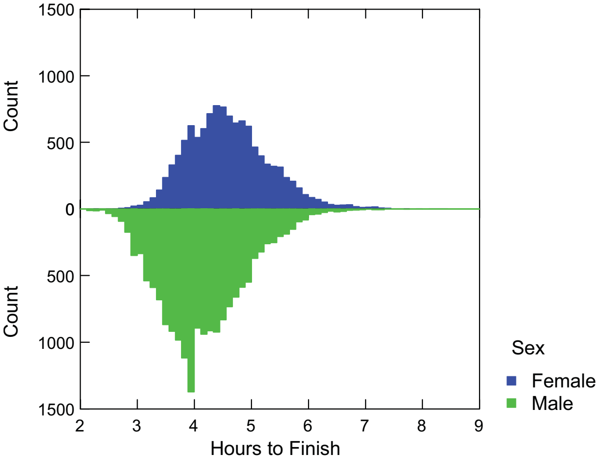


Figure 11.13 *Times for Chicago Marathon*

11.3.5.3 Tables of Polar Plots

Figure 11.14 shows a table of polar graphics of death-month by social group. The data are from Andrews and Herzberg (1985), contributed by C. O'Brien. The polar arrangement makes sense for years because it allows us to examine relations across the annual boundary.

COORD: *rect*(*dim*(3), *polar*(*dim*(1, 2)))
ELEMENT: *point*(*position*(month*death*group))
ELEMENT: *line*(*position*(month*death*group))

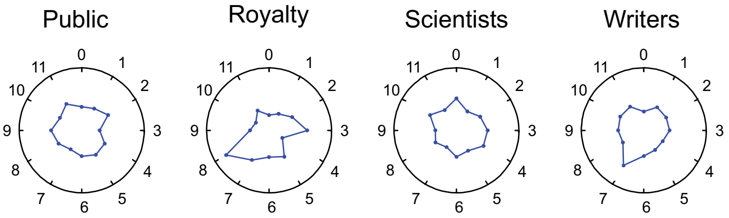


Figure 11.14 *Deaths as function of months from last birthday*

11.3.5.4 Polar Array of Polar Plots

Figure 11.15 shows a polar plot of polar plots of the Greenland wind data. This arrangement is particularly appropriate because wind direction and astronomical time (month of year) are intrinsically polar. The plot reveals cyclic trends that would be difficult to discern in a rectangular time-series plot. Changing the coordinate transformations from polar to rectangular would turn this into a trellis plot.

COORD: *polar.theta*(*dim*(3), *polar*(*dim*(1, 2)))
ELEMENT: *interval*(*position*(*summary.count*(*bin.rect*(month*direction, *dim*(2)))))

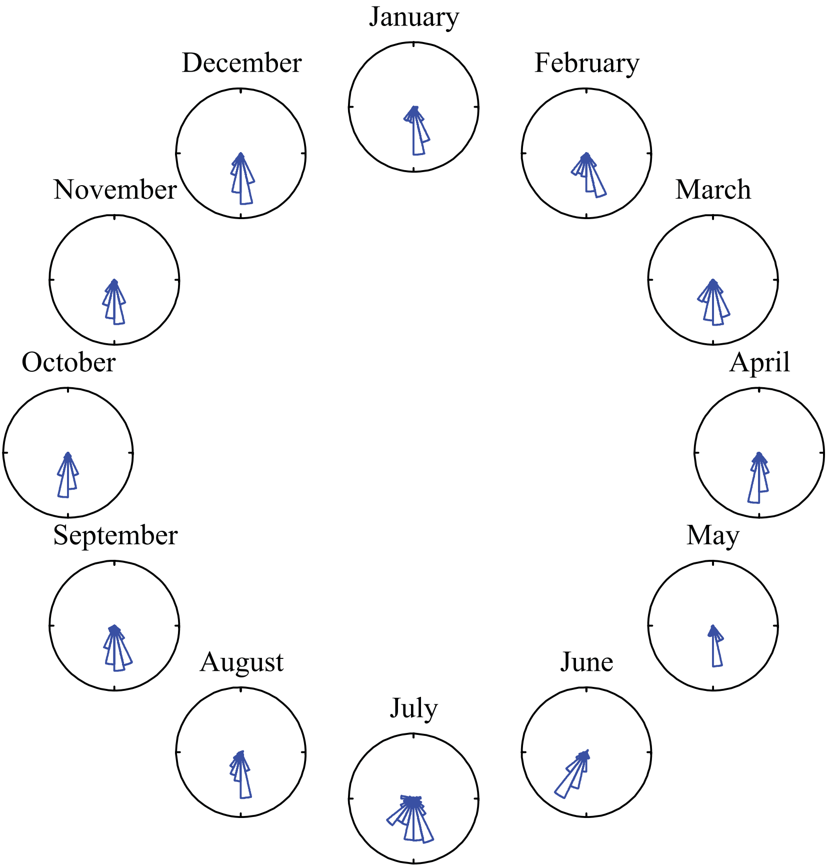


Figure 11.15 *One year of wind data*

11.3.5.5 Mosaics

The **mosaic** plot is a method for displaying categorical data in a contingency table whose cell areas are proportional to counts. Its application to contingency table models is due to Hartigan and Kleiner (1981, 1984), but it has a long history as a type of data display (Friendly, 2002). It is a type of tiling. Friendly (1994) presented a generalization of the mosaic that is equivalent to a set of crossed frames (see also Unwin *et al.*, 1996 and Theus, 1998). We will use a famous dataset to illustrate how the mosaic and its generalization works.

Figure 11.16 shows a 3D bar graphic representing proportions of survivors of the Titanic sinking, categorized by age, social class, and sex. The data are from (Dawson, 1995), who discusses various versions and their history. Simonoff (1997) fits a logistic regression model to these data that predicts survival from class, sex, and the interaction of class by sex. The two bar graphics were produced by a facet on sex, which aligns each on a horizontal facet dimension.

COORD: *rect*(*dim*(4), *rect*(*dim*(1, 2, 3)))
ELEMENT: *interval*(*position*(*summary.mean*(class*age*survived*sex)))

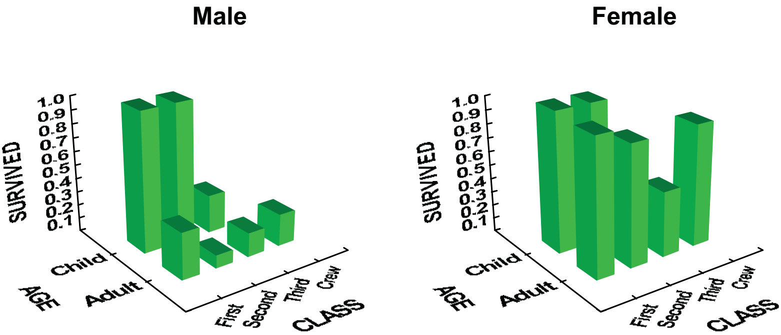


Figure 11.16 *3D bar graphic of Titanic survivor data*

Figure 11.17 shows a tiling of these data with the same faceting on sex. In this graphic, survived is used to determine the color of the tiles instead of the heights of bars. This graphic shows more clearly that there were no registered children among the crew. It also reveals the poignant fact that a relatively small proportion of the third-class passengers (even children) survived. Although color is not always suited for representing continuous variables (see Chapter 10), Figure 11.17 is preferable to Figure 11.16 because 3D bars tend to hide each other and the 3D display angle makes it difficult to decode the height of the bars.

COORD: *rect*(*dim*(3), *rect*(*dim*(1, 2)))
ELEMENT: *polygon*(*position*(class*age*sex), *color.hue*(survived))

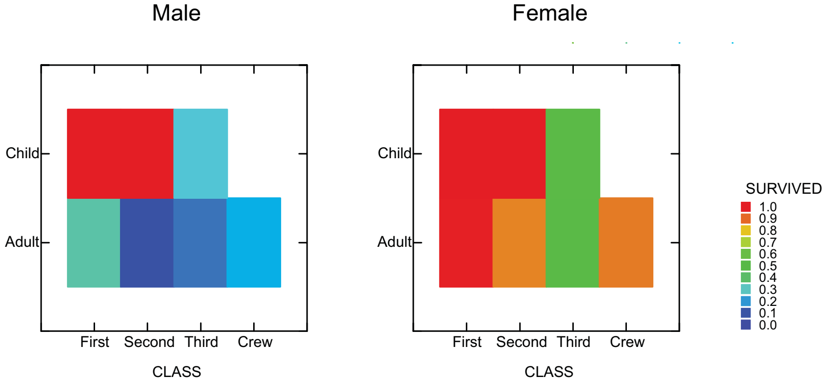


Figure 11.17 Tiled graphic of Titanic survivor data

The mosaic plot varies the area of each tile according to another variable, usually the count of the cases represented by that tile. Figure 11.18 shows how this works for the Titanic data. The innermost frame (**1*1**) holds one *polygon* graphic colored by survived. The remaining facet variables define coordinates that are modified by the *mosaic*() transformation. Eliminating axes leaves gaps between the tiles.

COORD: *mosaic*(*dim*(3, 4, 5, 6, 7), *rect*(*dim*(1, 2)))
ELEMENT: *polygon*(*position*(**1*1***age***1*1***sex*class),
color.hue(survived))

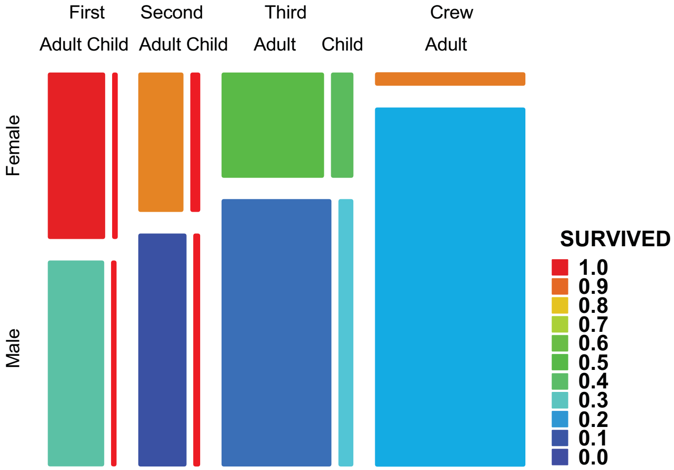


Figure 11.18 General mosaic plot of Titanic survivor data

11.3.6 Multiple Frame Models

For some tables of graphics, we have no choice but to set up multiple frames. Carr *et al*. (1998) present a paneled graphic called a linked micromap plot, designed for displaying spatially indexed statistical summaries. Figure 11.19 shows a variety of this plot.

```
DATA: longitude, latitude = map(source("US states"))
TRANS: octile = cut(popden, 8)
GRAPH: begin(origin(0, 0))
 COORD: rect(dim(3, 4), rect(dim(1, 2)))
 ELEMENT: polygon(position(longitude*latitude*1*octile), color.hue(popden))
GRAPH: end
GRAPH: begin(origin(1in, 0))
 COORD: transpose(dim(3, 4), rect(dim(1, 2)))
 ELEMENT: point(position(state/octile*popden*1*1), color.hue(popden))
 ELEMENT: line(position(state/octile*popden*1*1))
GRAPH: end
```

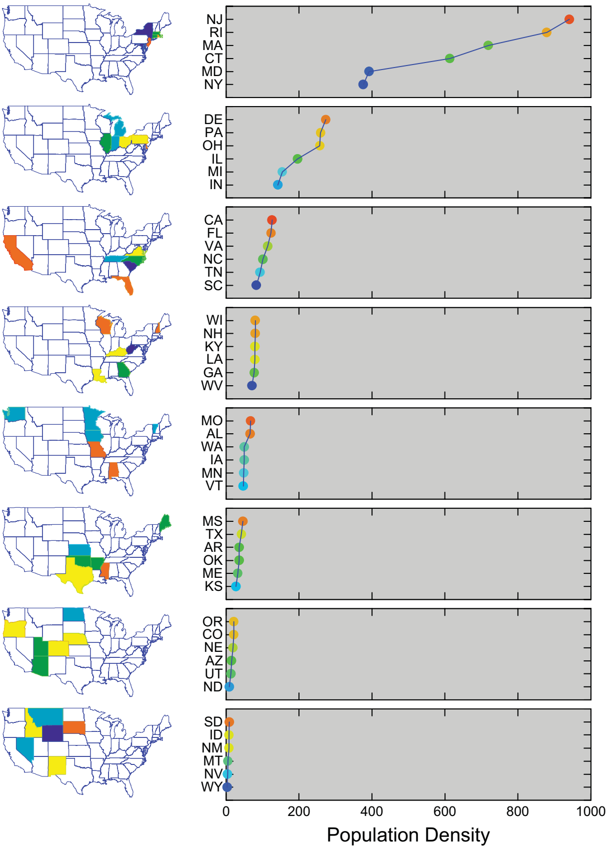


Figure 11.19 *Linked micromap plot of US population density*

The reason we need two frame models in this specification is that we cannot blend state and latitude. One is categorical and the other continuous, so the two cannot overlap on a common scale. Because we want to position the maps to align with the plot frames, we must make two separate graphics and align them using octile (the 8 fractiles of popden). Facets allow us to construct a tremendous variety of tabled graphics in a single frame specification, but there are limits. Some tabled graphics are really two or more graphics glued together. Because common variables are defined through a single data view, however, all such graphics are linked for the purposes of brushing, drill-down, and other operations.

We have taken one liberty with Carr's graphic (apart from minor differences in layout). Carr uses the same categorical color palette within each map block. This facilitates look-up and comparisons with the maps. We have instead represented each value with its appropriate color on a continuous scale within blocks. This reinforces the anchoring in the data values but makes look-up more difficult. Either method is preferable to using a single color scale across blocks. That would make the colors within most maps almost identical, particularly for a variable with a skewed distribution like popden's.

11.4 Sequel

So far, we have taken axes and legends for granted. The next chapter covers guides, such as axes and legends and titles, which help us to decode graphics.

12

Guides

Guides show us the way. They give us absolute directions in a relative world. We need guides in graphics to help us associate categories or quantities with aesthetics. Position, for example, allows us to compare magnitudes among graphics in a frame, but we need a positional guide to help us to relate magnitudes to non-graphical information. Color can help us discriminate categories, but we cannot associate colors with specific categories without a color guide.

Table 12.1 contains a list of standard graphical guides. Scale guides apply to aesthetics. These guides help us to decode continuous and categorical scales. Position guides are usually called *axes*. Other aesthetic guides are usually called *legends*. Annotation guides include *text* and *form*, and *picture*. These guides link a graphic to extrinsic metadata. Guide functions work like graphing functions (see Chapter 8). Their syntax is *type*(*dim*(), *args*). The arguments are aesthetic functions and formats.

Table 12.1 **Guides**

Scales	**Annotations**
axis	*text*
legend	*title*
size	*footnote*
shape	*form*
color	*line*
...	*rectangle*
	ellipse
	arrow
	tag
	image

In thinking about guides, we must keep in mind a built-in guide that all graphics share: the *label*() aesthetic attribute. This attribute associates a text string with each instance of a graphic. Because text can represent categorical or continuous values, the *label*() attribute can serve as a guide. It can be used to attach numerals to the tops of bar graphics, for example, or category names next to line graphics. Labeling of graphic elements is sometimes preferable to axes or separate legends because it allows local look-up without changing our focus and it can provide exact values without requiring comparative judgments. Kosslyn (1994) surveys some of the literature showing the superiority of labeling to legends. We also need to keep in mind that *label*() is an available attribute for guides themselves. It provides, in essence, a guide to a guide. Figure 1.1, for example, shows a label attached to a reference line guide. Labels are also used to annotate axes and legends.

12.1 Scale Guides

Chapter 6 covers continuous and categorical scales. Scale guides help us to decode specific scale values, translating them to numerical or string values that help us access metadata (in our memory or through hyper-links). Pinker (1990) and Simkin and Hastie (1987) discuss the perceptual issues and Cleveland (1993) addresses statistical issues in this process.

Scale guides are so ubiquitous and necessary that they should appear by default. Axes, for example, routinely define the position of a frame. Legends regularly appear when *size*(), *color*(), or other attributes are specified. It is easier to turn off guides than to require them in the specification every time an aesthetic attribute is used. To suppress the horizontal axis, for example, we add GUIDE: *axis*(*dim*(1), *null*) to the specification.

12.1.1 Legends

The *color*(), *size*(), and other non-*position* aesthetic guide functions produce a legend. Legends serve as guides for all aesthetics except *position* (although an axis, we shall see, is best thought of as a positional legend). Table 12.2 shows some examples of *size*, *shape*, *brightness*, and *hue* legends. A legend contains a **scale**, a **label**, and a **rule**. In the left panel of Table 12.2, the scale consists of the numbers 0 through 9. The label consists of the string "X" at the top of the legends. The rule consists of the linear arrangement of symbols or colors used to represent the attribute for the legend. The English word *rule* comes from the Latin *regula*, a straightedge. Rules help us measure lines, which is the role they play in an axis guide. Rules also give us formal methods for making associations, which is the role they play in a legend. On an axis, the rule spans the frame dimension referenced by the *position* attribute and, through the placement of tick marks, allows us to measure location and decode it through the adjacent scale. On a legend, the rule spans the dimension referenced by *size*, *shape*, and other attributes by instantiating reference values on that attribute. Tick marks, usually absent in legends because they are redundant, connect the points on the rule with the values on the adjacent scale so that we can decode the attribute.

Continuous legends have a continuous rule and categorical legends have a categorical rule. Some continuous legends blend hues to create a linear spectrum, marking the scale values with ticks. We have chosen to keep a slight space between the values, which obviates the need for tick marks and makes color attributes behave like other attributes such as *size* and *shape*. For categorical legends, we have chosen a wide spacing to differentiate the rule and scale from continuous legends. See Brewer *et al.* (1997) for further information on legending color.

There are many styles for legends. We have listed a few in Table 12.1. The legends in Table 12.2 are produced by the argument *vertical*(), the default. Horizontal legends, produced by the argument *horizontal*(), are useful when located underneath or above a frame graphic. Legends inside a frame graphic are produced by the argument *interior*(). These are used when there is available blank space inside a frame graphic.

Table 12.2 **Size, Shape, Brightness, and Hue Legends**

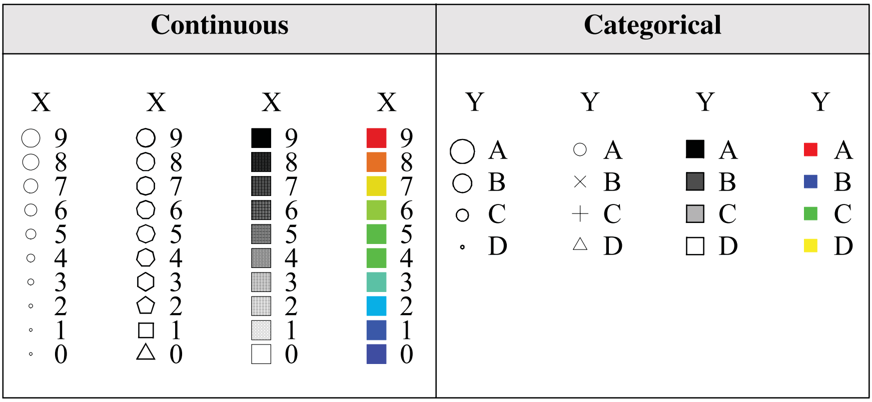


Figure 12.1 shows an example of a double legend. It is not generally a good practice to legend more than one attribute in a graphic, although it works in this example because similar symbols and colors are contiguous. Multiple legends raise configural/separable issues in perception and often make decoding difficult. See Chapter 10 for more information.

We have included the two legend functions in the specification, although these are automatically created by the use of the attributes in the *point*() function. Including them explicitly allows us to add a *label*() aesthetic attribute to each legend that we would customarily call a legend title. In this case, each label serves as a guide to a guide. We have also included the default axes, each with its own label.

```
ELEMENT: point(position(birth*death), shape(government),
            color(continent))
GUIDE: legend.color(dim(1), label("Continent"))
GUIDE: legend.shape(dim(1), label("Government"))
GUIDE: axis(dim(1), label("Birth Rate"))
GUIDE: axis(dim(2), label("Death Rate"))
```

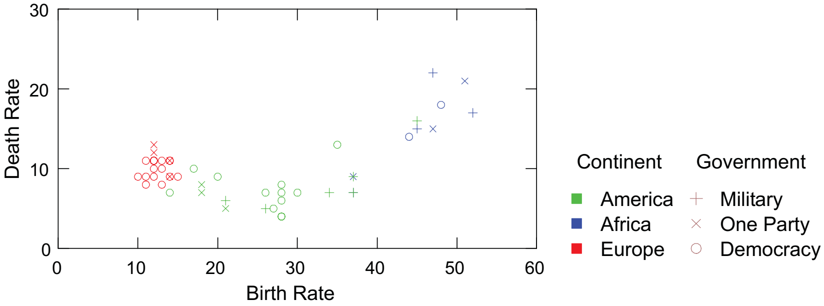


Figure 12.1 *Double legend*

12.1.2 Axes

The *axis*() guide function produces an axis. Axes are positional legends. As a legend, an axis contains a **scale**, a **label**, and a **rule**. The rule includes associated tick marks. As with legends, there are several styles for axes. The *cross*() argument crosses all axes in 2D or 3D at their zero values. The *single*() argument produces a single axis. The *double*() argument (the default) produces a single axis plus a parallel rule at another side of the frame. This default option produces squares in 2D and cubes in 3D.

The *format*() argument specifies the format of the scale associated with an axis. For example, a *format*() function for a time scale takes a picture format to specify dates. The symbol set "YMDhms" is used to denote year, month, day, hour, minute, second. When there is no ambiguity between month and minute, case may be ignored. All other characters in a format are used literally. Examples are "mmm dd, yyyy", "mm/dd/yy", "dd-mm-yy". When fewer than two characters are available for month, numerals are used. Examples of formatted dates are "Jan 1, 1956" or " 6/ 1/ 95". The *format*() function for a numerical scale specifies scientific or other notation.

The pains we take to devise uniform terminology for the components of axes and legends are not driven solely by semantics. We group components like scales, labels, and rules together so that the drawing methods to produce them are programmed only once. There is no reason to have one routine to draw a legend and another to draw an axis. Both should extend naturally from the architecture of a guide. Even if we are not programming, however, we can avoid confusion by learning why axes and legends are members of the same species.

There *is* one fundamental difference between an axis and a legend, however. Axes are transformable and legends are not. As numerous examples in this book show, especially in Chapter 9, axes can be bent all sorts of ways by coordinate transformations. Legends cannot, because the relative positions of their rules, scales and labels is governed by readability and not by frame coordinates. This is a simple distinction to implement. We simply transform the location of a legend into other coordinate systems, but not its components.

Figure 12.2 shows the super-smooth function $y = exp(-1 / x^2)$ plotted on crossed axes. Because the y-axis has its zero value at one end, the axes form an inverted “T” rather than a cross. This function, with its infinite number of derivatives, should be used in designing roller coasters for senior citizens.

```
DATA: x = mesh(min(–2), max(2))
TRANS: y =  exp(–1/ x^2)
ELEMENT: line(position(x*y))
GUIDE: axis(dim(1), cross(), label("Y"))
GUIDE: axis(dim(2), cross(), label("X"))
```

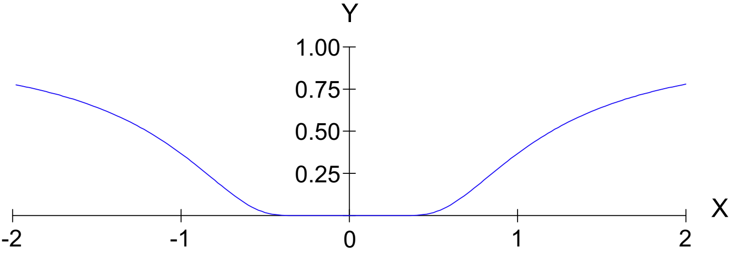


Figure 12.2 *Crossed axes*

12.1.3 Scale Breaks

In Section 9.1.8.1, we discussed the behavior of graphs when data exceed the bounds of frames. We indicated that nonlinear transformations can help this problem and simultaneously remedy some violations of statistical assumptions. We also questioned the wisdom of allowing data outside the bounds of a frame to determine the geometry of a graph inside a frame and claimed that this was a form of lying with graphics.

Sometimes graphic designers encounter the opposite problem. They have a graphic located far from the value of zero inside a frame spanning positive numbers. In a desire to include zero on the scale, these designers add a scale break that looks like this ——/\/—— or this —/ /—— to the axis. This device allows them to indicate zero without representing its value — to seem to use *position*() but not to invoke it. Curiously, this device appears to be favored by scientists, although it occasionally appears in business graphics as well.

If zero really is important, then it should be part of a scale. If it is not, then it should be omitted. Some would say scale breaks are confusing. If we think about what they are intended to do, we must conclude they are meaningless.

12.1.4 Double Axes

Some scientific and business graphics have double axes (dual scales). These usually appear as right and left vertical axes with different scales in 2D graphics. Double axes can serve two different purposes. The first is to represent two different scales for the same variable. For example, we might wish to present US dollars on the left vertical scale and the Euro on the right. The second purpose is to index different graphical elements inside the frame against different variables. For example, we might wish to align two different *line* graphics — one for educational spending and another for students' test scores — so that we can compare trends across time. Wainer (1997) comments on this questionable practice. Multiple axes with multiple graphics gives the designer extraordinary license to manipulate conclusions.

In either case, double axes imply double, superimposed frames. This is easy to implement with two graph specifications. Recognizing this geometrical fact should alert us to the danger of hiding it from the viewer. Double (or multiple) axes generally should be avoided.

12.2 Annotation Guides

Annotations become guides when they are driven by data. Otherwise, they are scribbles or doodles. Being driven by data means that the aesthetics of an annotation object are linked to variables in a dataset in a similar manner to that for a graphic. A text annotation, for example, might be positioned according to a value of a variable, or its color might be determined by a variable value, or its content might be determined by statistical parameters. There are many consequences to this data linking. For example, annotations that are anchored in a 3D frame rotate and stay frontally visible in 3D when the view is changed. Other linked annotation attributes, such as color, change when databases or other data sources are updated. This section offers a few examples to indicate how these graphical objects can function in a system.

12.2.1 Text

The *text*() guide function produces a text string. There are many types of strings that can be attached to graphics. We have included only two in Table 12.1: *title*() and *footnote*(). Ordinarily, we devise annotation from our own subject knowledge and embed it in a graphic as a simple string, for example, GUIDE: *text.title*(*label*("This is a title")). The other aesthetic attributes of text (*e.g., color* or *size*) can be determined by a constant or a variable.

Figure 12.3 shows an example of text determined by a statistical model. The data are taken from the car performance dataset used in Figure 8.2. We have fitted a quadratic regression predicting the acceleration of the cars from horsepower and vehicle weight. The title is determined by the parameters from the smooth through the *smooth.quadratic*() function that is passed to *label*(). This device assumes that statistical procedures have interfaces for reporting their parameter estimates as strings and that *label*() is written to know how to ask for such strings. The coefficients for WEIGHT*WEIGHT and HP*WEIGHT have been deleted from the string because they were zero (to printed precision). We have added *interval*(*dim(*3), *min*(0), *max*(25)) to the specification to avoid truncating the surface; this gives a larger range than used in Figure 8.2.

```
SCALE: interval(dim(3), min(0), max(25))
ELEMENT:surface(position(smooth.quadratic(weight*hp*quarter)))
GUIDE: axis(dim(1), single(), label("WEIGHT"))
GUIDE: axis(dim(2), single(), label("HP"))
GUIDE: axis(dim(3), single(), label("QUARTER"))
GUIDE: text.title(label(smooth.quadratic()))
```

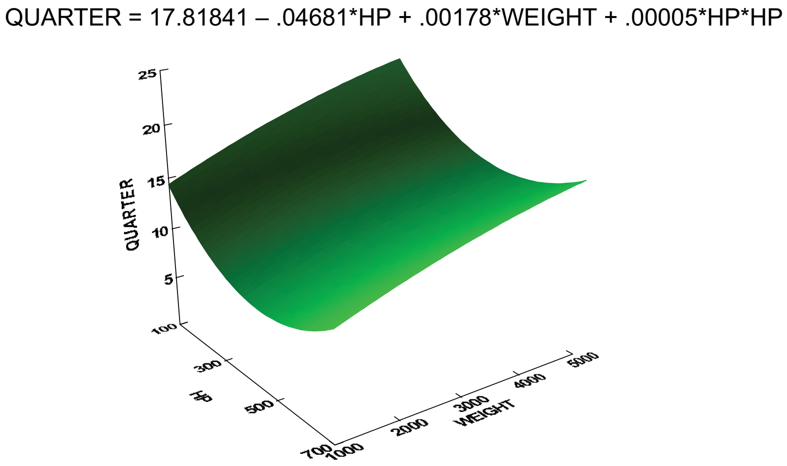


Figure 12.3 *Title text determined by parameters*

This example gives us the opportunity for a brief statistical digression. The model in Figure 12.3 is nonsensical. It implies that increasing horsepower above 500 retards quarter-mile acceleration. Dragsters are an existence proof against such a model. Nevertheless, the fit of this quadratic model to the car data is quite good. The squared multiple correlation is above .9 and residuals are reasonably distributed. Unfortunately, some researchers who use statistical methods pay more attention to goodness of fit than to the meaning of a model. It is not always convenient to remember that the right model for a population can fit a sample of data *worse* than a wrong model — even a wrong model with fewer parameters. We cannot rely on statistical diagnostics to save us, especially with small samples. We must think about what our models mean, regardless of fit, or we will promulgate nonsense.

12.2.2 Form

The *form*() guide function produces a geometric primitive form such as a line, rectangle, or ellipse. We have included only a few form methods in Table 12.1. A well-designed graphics system will have a set of drawing and text tools in its user interface. These tools will operate on the display surface in inch or centimeter coordinates referenced by a 2D or 3D grid system. Nevertheless, it is useful to include primitive objects that live in variable coordinate space so that we can drive them with data or direct value references.

Figure 12.4 shows an example of a *form.tag*() object. This allows us to annotate points or other elements in a frame by providing their coordinates. Through a constraint system, the software can determine where to place tags so that they do not collide with themselves or other graphic objects. Figure 12.4 annotates two unusual cases — a race car and a utility van — for the scatterplot introduced in Figure 8.2. Another example of a *form* object is the *form.line*() in Figure 1.1.

As we mentioned at the beginning of this chapter, the *label*() aesthetic attribute function can provide similar identification to the *form.tag*() and other annotations. The difference is that a *form* is a single object and must be specified repeatedly for more than one instance. Also, a *form* is not necessarily bound to a graphic such as a *point* cloud. We could attach a *form.tag*() to the edge of a frame, for example, or even to a title in a graphic. If we wished to label all points in a cloud, then using *form.tag*() would be inefficient.

```
ELEMENT: point(position(weight*hp*quarter))
GUIDE: axis(dim(1), single(), label("WEIGHT"))
GUIDE: axis(dim(2), single(), label("HP"))
GUIDE: axis(dim(3), single(), label("QUARTER"))
GUIDE: form.tag(position(4245, 109, 21.6), label("VW Van"))
GUIDE: form.tag(position(1950, 650, 11.3), label("Ferrari 333SP"))
```

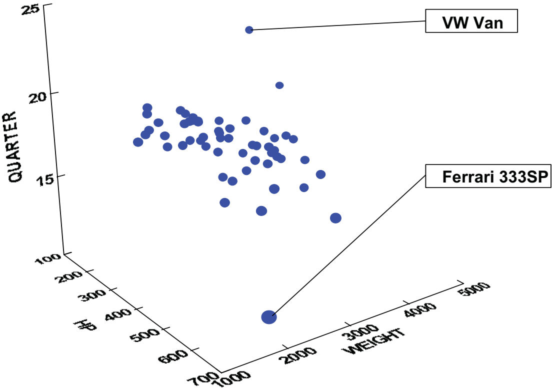


Figure 12.4 *Tag annotations*

12.2.3 Image

The *image*() guide function attaches an image to a frame. Figure 12.5 shows an example. This figure is adapted from a portion of a graphic in Shepard and Cooper (1982). The data are from a perceptual experiment in which subjects viewed pairs of objects differing only by rotational angle. Three different objects are represented in the panels of the figure. The rt variable is reaction time (delay in saying "same" for a pair). The depth variable marks the type of rotation used in the pair. Circles stand for picture-plane (2D) rotated pairs and squares stand for depth (3D) rotated pairs. Shepard's remarkable discovery in this and other experiments was that rotational angle is linearly related to reaction time. The February 19, 1971 cover of *Science* magazine displayed five of Shepard's computer-generated images under various rotations. This research has been replicated by psychologists and neuroscientists studying spatial processing in humans and other primates. Shepard received the National Medal of Science for this and other work in cognitive psychology.

As with *form.tag*(), the *image*() guide shares functionality with the *label*() aesthetic attribute function and even with legends. We could construct a legend, for example, by associating the three images in Shepard's graphic with the values of **object**. They could then appear as axis scale values (see Figure 11.8) or as legend scale values. We used the *image*() guide function in this example in order to place the images inside the frames where Shepard did.

This usage annotates the entire frame, but we can imagine other applications where a one-to-one relationship between an *image*() and a frame would not exist. For example, we could use *image*() to place images along various parts of a curve in a 2D plot to signify another dimension. Imagine, for example, pictures of cigarette packs along various sections of the path in Figure 8.10.

```
DATA: s1 = link("figure1")
DATA: s2 = link("figure2")
DATA: s3 = link("figure3")
COORD: rect(dim(3), rect(dim(1, 2)))
ELEMENT: point(position(angle*rt*object), shape(depth))
ELEMENT: line(position(smooth.linear(angle*rt*object)))
GUIDE: image(position(60, 6, 1), shape(s1))
GUIDE: image(position(60, 6, 2), shape(s2))
GUIDE: image(position(60, 6, 3), shape(s3))
GUIDE: shape(dim(1), label("Rotation"))
GUIDE: axis(dim(1), label("Angle of Rotation (degrees)"))
GUIDE: axis(dim(2),label("Mean Reaction Time (seconds)"))
```

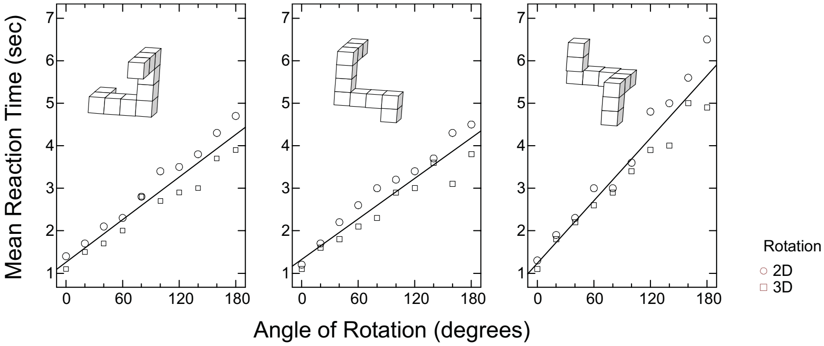


Figure 12.5 *An image annotation*

12.3 Sequel

We have completed our survey of the world of graphics. We move on to the second part, which covers the semantics, or meaning, of graphics. The remaining chapters demonstrate the consequences of our new world view. The next chapter covers the use of space in graphics.

Part 2

Semantics

Semantics is derived from the Greek $\sigma\eta\mu\alpha$, or sign. This same word underlies the title of Bertin's *Semiology of Graphics*. In linguistics, semantics involve questions of meaning. The second part of this book covers the theoretical aspects of representing data graphically. We omit grammar-of-graphics specifications in this section because we want to focus on content rather than structure. We also include in this part many more graphics generated by researchers, companies, and projects. In Part 1, it is important to show the capabilities of the grammar-of-graphics syntax for generating a diversity of charts. In Part 2, it is important to understand the concepts used in the broader fields of visualization and statistical graphics. The first edition of this book, and Part 1 of the current edition, explicitly cautioned that the grammar of graphics is not a visualization system. The second part of this book examines the field of visualization to identify common themes.

These themes are representing space, time, and uncertainty, linking statistical and data-mining analytics to graphic displays, user control of dynamic graphics systems, automation of production graphics through programming languages, and the theory of reading graphics to extract data and relations. We choose a theory-oriented (rather than applications-oriented) approach because we think it integrates results from fields outside the relatively small scientific visualization community. Geographers, psychologists, designers, statisticians, and political scientists all have a lot to contribute to the field of graphics. Consequently, our references in this part range more widely than they do in the first part.

We risk covering material in this part that is routine to some readers. Statisticians, for example, can skip the introduction to Bayesian methods and inference in Chapter 15. Engineers can skip the material on Fourier analysis and convolutions. We are surprised, nevertheless, to discover how little some visualization researchers in various fields know about the origins of many of the techniques that are routinely applied in visualization. We believe that research in visualization requires a basic understanding of statistics, experimental design, user interfaces, cognition, perception, and other fields. We hope this part provides a useful introduction to this material.

13

Space

The word **space** comes from the Latin *spatium*, which means a room or space. The Latin and English both carry as well the meaning of time. The space to do something implies both room and duration. Long before Einstein placed space and time in a common coordinate system, time was perceived to be spatial. We will discuss this relationship in the next chapter. In this chapter, we will discuss the role spaces play in organizing graphics. We begin with a review of terms.

A graphics **frame**, as defined in Section 2.1.10, is a set of tuples ranging over all possible values in the domain of a *p*-dimensional varset. A **graph**, as defined in Section 2.1.4, is a subset of these tuples. A **graphic**, as defined in Section 2.2.7, is a perceptual realization of a graph. Most of the time, we represent tuples using points of ink on a page or light on a computer screen. How do we decide where to locate these points in the viewing area in order to communicate the underlying graphic model? We use a space. Briefly stated, a space is a way of organizing tuples.

Actually, we use two spaces to make a graphic. The first is the **underlying space** in which we embed the tuples of our frame. Our graphic model (using algebra and statistics) organizes this space. This underlying mathematical space may have one, two, or many dimensions; it may be defined on the integers, real numbers, or {V, E} graphs; and it may be indexed by rectangular, polar, or other coordinate systems.

The second space is the 2D or 3D physical **display space** in which we embed points. This display space is Euclidean. How we map tuples in underlying space to points in the display space determines the interpretability of our graphic.

The following example illustrates this mechanism. Figure 13.1 shows a photograph of beadlet anemones (*Actinia equina*). Kooijman (1979) examined a colony of these creatures and measured their locations on the surface of a boulder near Quiberon Island off the Brittany coast. For more information on anemone anatomy, see McCloskey (2003).

Figure 13.1 Beadlet anemones (photo used by permission, UWPhoto ANS)

Figure 13.2 shows a graphic of Kooijman's data. Each of his (*x, y*) tuples consists of the relative location of an anemone on the surface of the rock. The underlying space in which Kooijman embedded his tuples is a 2D Euclidean representation of the rock surface. (Kooijman used a 2D space because the rock was relatively flat.) The display space is a different 2D Euclidean space resting on the plane of this page. Each point in the underlying space has been mapped to a point in the display space such that the Euclidean distances between points in the underlying space are proportional to the corresponding Euclidean distances in the display space.

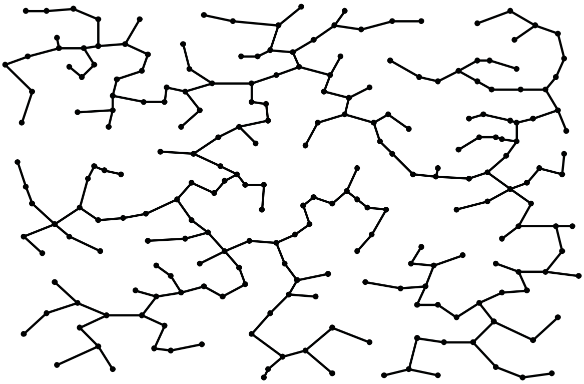


Figure 13.2 Beadlet anemone minimum spanning tree on a rock

There is a second element (in addition to the points) used in Figure 13.2. We computed a minimum spanning tree (MST) on Kooijman's tuples embedded in his underlying Euclidean space and mapped the edges of this $\{V, E\}$ graph to line segments in the display space. What information does this communicate? First, we can find the nearest neighbor to any anemone by looking for the shortest edge connected to that anemone. Second, we can find clusters of anemones by looking for connected subsets of relatively short edges. Third, we can find shortest paths between anemones by walking the tree. There are other inferences we can make from the MST embedding as well. The point is, we can make inferences about Kooijman's original measurements by looking at a display space because the tree has been mapped from the underlying space to the display space in a meaningful way.

Now, suppose our underlying space is different. Suppose Kooijman did not give us the (x, y) coordinates of the anemones. Suppose instead that he gave us a list of anemone names (Amie, Andrew, Angel, Anita, ...), a list of MST links between names (Amie–Angel, Andrew–Anita, Angel–Anita, ...), and a list of weights, one for each link, corresponding to the Euclidean distance between the linked anemones.

What underlying space could we use to organize these objects? One candidate would be a **graph-theoretic space**. This space organizes a set of vertices and a set of edges such that vertices are related to each other by being connected with an edge. This space also allows us to associate an edge weight with each edge, so that we can induce a weighted distance relation between vertices.

Figure 13.3 shows each vertex in this underlying graph-theoretic space mapped to a point in display space. And it shows each edge in the underlying graph-theoretic space mapped to a line segment in the display space such that the length of the line segment is proportional to the corresponding edge weight in the graph-theoretic space.

What information can we infer from this display? First, we can find the nearest neighbor to any anemone by looking for the shortest edge connected to that anemone. Second, we can find clusters of anemones by looking for connected subsets of relatively short edges. Third, we can find shortest paths between anemones by walking the tree. These are the same inferences we were able to make from Figure 13.3. Note, however, that we can *not* make strong inferences about the relative location of the anemones on the surface of the rock (although the MST graph allows us to make some weak inferences). Specifically, the Euclidean distances between the unlinked points in Figure 13.3 are not meaningful. The only distances we can calculate in the display space are along the edges. There is less information in this figure than in Figure 13.3.

How, then, did we map the objects in the underlying graph-theoretic space to objects in this display space? We used a layout algorithm that places points on the plane so as to minimize crossings between edges. This gives us a clean layout, but not a unique one. There are many other acceptable layouts we could have computed. We will discuss these methods in Section 16.3.

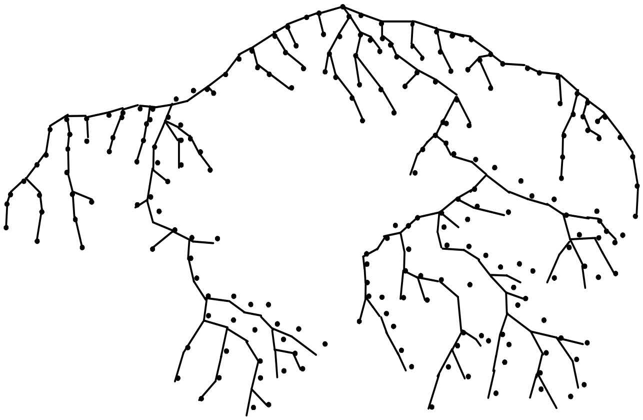


Figure 13.3 *Beadlet anemone minimum spanning tree on the plane*

The point of this example is to indicate that there is a mathematical space underlying a meaningful graphic. The display space gives us a view of that underlying space but it is not the underlying space itself. Therefore, if we have no information about the underlying space, we cannot understand a graphic. Figure 13.2 looks different from Figure 13.3, but we have no way of knowing whether these differences are ignorable or are an essential part of the structure of the graph.

This simple fact applies to pie charts and bar charts as much as it does to these more exotic charts. We take the underlying space of ordinary charts for granted because we often assume it has the same structure as the display space. That might not get us into trouble with Figure 13.2, because we can approximate Kooijman's point of view by imagining that the display space is an aerial photograph of the rock, with links superimposed by bits of string. The same assumption would get us into trouble with Figure 13.3, however.

More formally, graphics involve a mapping through the function $f : S \rightarrow P$, where *S* is the underlying mathematical space of the graph and *P* is the Euclidean physical display space represented on the page by a *position* aesthetic. To understand a graphic, we must know *f* and *S*. Because we omitted in these two figures some guides that might have helped the reader, there is no way to discern *f* and *S* by looking at either of the pictures.

We will now review some mathematical spaces that are frequently used in graphics. Then we will discuss the psychological spaces that underlie our perception and understanding of spatial layouts. Finally, we will illustrate through examples ways to make clear the mathematical space *S* underlying a graphic as well as the function f that maps *S* to *P*.

13.1 Mathematical Space

In making a graphic, we begin with a set of tuples. The elements comprising these tuples may be real numbers, integers, strings, images, or other objects. Depending on our measurements, these tuples may have various kinds of organization.

- Some values may lie on a continuum or otherwise be connected to each other.
- Other values may be categorical or otherwise be isolated from each other.
- The values may lie in a compact region.
- The values may lie in an unbounded region.

These kinds of organization are called **topological properties**. All the spaces we will consider in this chapter are topological. An important subclass of topological space is called a **metric space**. Most of the examples in this chapter involve metric spaces. We will first introduce topological space and later discuss various metric spaces. Munkres (1975) covers these topics in more detail.

13.1.1 Topological Space

A **topological space** S is a set X together with a collection T of subsets of X satisfying the following three axioms:

1) The empty set and X are in T.
2) The union of any number of sets in T is also in T.
3) The intersection of any pair of sets in T is also in T.

The elements of X are usually called **points**. The set T is called a **topology** on X. The elements of T are called the **open** sets of X. A subset of X is **closed** if its complement is open. A set B is called a **basis** for T if every member of T is a union of members of B.

The real line is an example of a topological space when we give it the following topology. We say that a set is open if every point in that set is contained in an open interval in that set. One can verify that this definition satisfies the three axioms. For example, the open interval $(0, 1)$ is an open set in the topology we have just defined. And the union of two open intervals $(0, 1) \cup (2, 3)$ is also an open set. There are also more complicated open sets in this topology, such as a Cantor set complement, which is the union of infinitely many open intervals. Examples of closed sets are closed intervals, unions of finitely many closed intervals, and more complicated sets such as Cantor sets. Some sets, such as the interval $[0, 1)$, are neither closed nor open. The topology we have just defined is the **standard topology** on the real line.

Note that a given set can have more than one topology on it. Another example of a topology on the real line is the topology where every set is open. In this topology, *T* equals the set of all subsets of *X* (the power set of *X*). This is called the **discrete topology** on the real numbers. Another topology at the opposite extreme has only two open sets: the empty set and the entire real line.

13.1.2 Connected and Disconnected Space

A topological space *S* is **connected** if it cannot be partitioned into two disjoint non-empty open sets on the topology *T*. A topological space is **totally disconnected** if it has no connected subset with more than one point. A **discrete topological space** is one in which every subset of *X* (including single points) is open. This condition implies that a discrete space is not connected.

In the grammar of graphics, a frame consisting of the product of categorical variables (whose range is equivalent to the integers) has tuples that we embed in a totally disconnected space. A frame consisting of the product of continuous variables (whose range is equivalent to the real numbers) has tuples we embed in a connected space. And a frame consisting of the product of continuous and categorical variables has tuples that we embed in a collection of connected spaces. These conditions lead us to look for empty regions in charts based on categorical variables. As we will see, these empty regions give us the opportunity to encode additional sources of variation using positional and other aesthetics.

A topology tells us when points in a space are related, namely, when they belong to the same open set. Usually, however, we want to add some conditions having to do with the *strength* of relations between tuples. These conditions have to do with metrics that formally define the concept of distance. We will now introduce metric spaces.

13.1.3 Metric Space

A **metric space** is a set *S* together with a distance function

$d\colon S \times S \to [0, \infty)$, satisfying the following three conditions

$$
\begin{aligned}
&1)\ d(x, y) = 0 \quad \Leftrightarrow \quad x = y \\
&2)\ d(x, y) = d(y, x) \\
&3)\ d(x, z) \le d(x, y) + d(y, z)
\end{aligned}
$$

The three metric axioms above are called the **identity**, **symmetry**, and **triangle inequality**.

A metric space is a topological space. We construct a basis for this metric space topology using open balls (a type of open set). A **ball** is a set containing all points within a specified distance of a given point. For a metric space S, an open ball centered on point p with radius r is defined as:

$$B_r(p) = \{x \in S : d(x, p) < r\} \ .$$

We use the words *ball* and *radius* in an abstract sense, since balls need not be spheres and the radius r is simply a distance value. A 1D ball is an interval. A 2D ball is a disk, square, diamond, or other convex region. A 3D ball is a sphere, ellipsoid, or other centrally-symmetric convex solid. We will examine these more closely in Section 13.1.9 below.

The most familiar example of a metric space is n-dimensional Euclidean space. Its basis consists of the balls

$$\sqrt{\sum_{i=1}^{n} x_i^2} < r$$

containing all points within radius r. Notice that when $n = 1$, the space is the real line and balls are open intervals. The metric topology here is the standard topology on the real line that we defined above.

The metric axioms seem so natural to us, perhaps because of our experience with Euclidean space, that we might be led to overlook how restrictive they are. The identity axiom means that we cannot use a metric space to represent comparisons of objects to themselves that yield less similarity than comparisons of different objects, as in some human judgments (Woody Allen's chameleon-like character Zelig seems to be less similar to himself than he is to Woody Allen). The symmetry axiom means that we cannot use a metric space to represent concepts such as commuting times (because traffic congestion is not symmetric). And the triangle inequality means we cannot use a metric space to represent contextual relations between objects (because introducing a third object might change the context of comparison). We will explore these issues further when we discuss using mathematical spaces to represent psychological spaces.

We must avoid the perception that the metric axioms are sacred or that nonmetric spaces are inferior to metric spaces. Spaces are simply structures on sets, so the suitability of a structure depends on what we are trying to represent in our data. If we need to forego metric assumptions in order to represent our data more effectively or parsimoniously, then so be it (this is what topologists do every day). Restrictiveness should not drive us to abandon the metric axioms too readily, however. Metric spaces should be our starting point for graphics and we should consider other representational methods only after identifying where the axioms are violated by our data. We will discuss this issue further in Section 13.3.

So far, we have examined features of an underlying space for representing relations among tuples. We now have to investigate how to map an underlying space to a display space. For that, we need to define a mapping.

13.1.4 Maps

A **map** $f: X \rightarrow Y$ is a function from one set to another.

- A map is **injective** if, for every element in *Y*, there is at most one element in *X* mapped to it. This definition allows for elements in *Y* that have no mapping from *X*, but it does not allow two different elements in *X* to map to the same element in *Y*. Injective maps are often called **one-to-one**.
- A map is **surjective** if every element in *Y* has at least one element in *X* mapped to it. This definition allows for two different elements in *X* to map to the same element in *Y*, but it does not allow an element in *Y* to lack a mapping from *X*. Surjective maps are often called **onto**.
- A map that is both injective and surjective is called **bijective**.
- A map $f: S \rightarrow P$ from a space *S* to a space *P* is **continuous** if points that are arbitrarily close in *S* (*i.e.*, in the same neighborhood) map to points that are arbitrarily close in *P*. For a continuous mapping, every open set in *P* is mapped from an open set in *S*. Examples of continuous maps are functions given by algebraic formulas such as $f(x) = x^2$.

Now we introduce an important class of maps.

13.1.5 Embeddings

A map $f: S \rightarrow P$ from one topological space to another is an **embedding** if *S* and its image *f*(*S*) in *P* are **homeomorphic**. A homeomorphism is a continuous one-to-one function whose inverse is also continuous. Roughly speaking, a homeomorphism between two shapes means we can stretch and bend (but not tear or glue) one shape to fit the other. Topological embeddings are thus both injective and continuous.

As geographers know, the sphere cannot be topologically embedded in the plane. Even though the surface of the sphere and plane are both intrinsically 2D, planar spherical projections must cut the sphere somewhere in order to flatten its surface or must form many-to-one mappings that blend two hemispheres (see the projections in Section 9.2.4).

We've come far enough now to sense that in the world of charts, where we must map mathematical space (*S*) to physical space (*P*) in order to display our data, it would be nice always to have a topological embedding. It would be nice for psychological reasons, if for none other. That is, a mapping in which points close to each other in mathematical space are far from each other in the chart (because of a tear) can be confusing. And a chart where entire regions are reduced to a single line or point can be confusing. Most map readers are accustomed not to fall off the edge of a map or into a hole, but we cannot count on this restraint among all chart readers looking at all kinds of charts.

Having said embeddings are nice for charts, however, we must add that embeddings are not necessary for making interpretable charts. Restricting ourselves to homeomorphic mappings would eliminate many useful charts, the most obvious being planar geographic maps.

Embedding points in a metric space allows us to make powerful inferences about relations among objects represented by those points in that space. The next section discusses how to invert this inference.

13.1.6 Multidimensional Scaling

Distances in a metric space are computed with a distance function. Given a finite configuration of points in a metric space, we can compute a symmetric matrix of pairwise distances on all pairs of points. By definition, the diagonal of this matrix is zero and the off-diagonal elements are positive.

Suppose we invert this condition and say that we are given a metric and a distance matrix **D** and wish to compute a matrix **X** containing the coordinates of points in the metric space that can be used to reproduce these distances. Young and Householder (1938) did this for a Euclidean space. Take any pair of points and apply the cosine rule for triangles:

$$d_{jk}^2 = d_{ij}^2 + d_{ik}^2 - 2d_{ij}d_{ik}\cos\theta_{jik}$$

So,

$$d_{ij}d_{ik}\cos\theta_{jik} = (d_{ij}^2 + d_{ik}^2 - d_{jk}^2)/2$$

The term left of the equality is the scalar product $\mathbf{x}_j'\mathbf{x}_k$, where the vector $\mathbf{x}_j$ contains the coordinates for point j relative to any point i. Renaming the right-hand quantity, we have

$$\mathbf{x}_j'\mathbf{x}_k = w_{jk}$$

Thus, we may take any point i represented in **D** and use its distances to the other points to create a scalar products matrix

$$\mathbf{W} = \mathbf{X}'\mathbf{X}$$

A singular value decomposition of **W** yields **X**, which contains the coordinates for the set of points. Point i will be at the centroid of the configuration.

Now suppose the distances contain random error. Torgerson (1952) devised a least-squares solution by centering **W** at the centroid of all the points (as opposed to an arbitrarily selected data point i). Torgerson coined the term **multidimensional scaling** (MDS) for this procedure, as an extension of the **unidimensional scaling** model used by psychophysicists for almost a century earlier.

Dissimilarity data often contain an **additive constant**, a condition that violates the identity axiom and produces bad fits with Torgerson’s model. Messick and Abelson (1956) offered a solution to the additive constant problem, allowing users to apply Torgerson's method to dissimilarity data of the form $d' = c + ad + e$. That is, a dissimilarity d' is assumed in this model to be a function of an additive constant c, a multiple of a distance d, plus random error e. After estimating and subtracting the additive constant, we can apply Torgerson’s method to these adjusted dissimilarities. Torgerson’s method with the additive constant adjustment is now called **classic** MDS.

Torgerson's MDS model is linear. Because psychological similarity or dissimilarity data often consist of ranks, psychologists were anxious to relax the linearity requirement. Roger Shepard solved this problem in the early 1960s (Shepard, 1962). Shepard's iterative algorithm was *sui generis*; it offered a simple computational method for achieving a feasible solution to the problem.

Shepard had developed his ideas while a graduate student at Yale working with his advisor Robert Abelson. He took a position at Bell Labs and met the mathematician Joseph Kruskal (brother of the statistician William Kruskal). Kruskal (1964) recast Shepard’s scheme as a least squares problem. Kruskal regressed distances between points in the spatial configuration onto the observed dissimilarities. He developed a loss function based on residuals from this regression and used a gradient method to find a solution with (locally) minimum loss. Kruskal’s linear regression version of this algorithm became known as **metric** MDS. Kruskal also included a monotonic regression procedure in his scheme. The monotonic version of his algorithm became known as **nonmetric** MDS. Kruskal’s algorithm had one other elegant aspect. He implemented the Minkowski metric function for computing distances between points and thus extended the model to non-Euclidean spaces.

Other significant contributors to the MDS literature were Louis Guttman, Forrest Young, Jan deLeeuw, and J. Douglas Carroll. Young and Hamer (1987), Cox and Cox (1994), and Borg and Groenen (1997) cover this history in more detail. The classic, metric, and nonmetric MDS models have been rediscovered, modified, and renamed many times over the last few decades (*e.g.*, Sammon, 1969), perhaps because it has so many applications in various fields.

MDS depends on computing distances among points in a metric space. It was developed for embedding points in Euclidean space, but the concept is more general. We can do MDS on points in other spaces by computing distances along geodesics.

13.1.7 Geodesics

A **geodesic** is a locally length-minimizing **path**. In a topological space S a path is a continuous mapping from the unit interval [0,1] into S. In graph-theoretic space, a path is a sequence of connected edges. The word *geodesic* comes from the Greek $\gamma\varepsilon\omega\delta\alpha\iota\sigma\acute{\iota}\alpha$, which means earth-dividing or surveying. In Euclidean space, geodesics are straight lines. On a sphere, geodesics are arcs of great circles; we speak of shortest global routes as great circles. In graph-theoretic space, geodesics are paths along edges.

Geodesics are associated with metrics. Since a geodesic requires an algorithm for computing the shortest distance between two points in a space, a geodesic depends on the definition of the spatial metric. Geodesics have several uses. Following a geodesic path through a graph or through a geographic space tells us about traversal time. Marking a geodesic locus around a point or node (by computing a set of geodesics of a fixed length) helps us to identify neighborhoods in space. We will identify geodesics for many of the spaces we discuss. Before we go further with this, however, we need to consider what is meant by the dimension of a space.

13.1.8 Dimensions

A naive definition of the **dimension** of a geometric object is the number of coordinates needed to represent an instance of the object in a Euclidean space. For example, a point is one-dimensional, a circle is two, a sphere is three, and so on.

Another definition of the dimension of a geometric object is the number of coordinates needed to specify a point on the object. For example, a square is two-dimensional and a cube is three-dimensional. A sphere is two-dimensional, however, because we can express any point on it with only two coordinates (we do that when we navigate the globe). Refining this definition leads to what mathematicians call **topological dimension**. Topological dimensions are integral. They characterize a large set of geometrical objects whose dimensionality is unchanged under a broad class of deforming transformations.

Fractal objects are not included in this class of objects, however. For them, a different mathematical definition of dimension is taken from the exponent in a power law for self-similar objects $n = s^d$, where n is a parameter for the number of elements in an object and s is a parameter for the scale of the elements. While the definition is quite technical, the exponent in this formula is an example of what is called the **Hausdorff dimension**, which can take fractional values. Edgar (1990) and Schroeder (1991) survey these ideas. We will discuss them briefly in Section 13.1.11.

Applied fields assign many different meanings to the word *dimension*. For example, vision scientists (*e.g.*, Marr, 1982), computer scientists (*e.g.*, Strothotte and Schlechtweg, 2002), and geographers (*e.g.*, MacEachren, 1995) use the term **2½D** to describe a space in which the half-dimension is used to contain geometric metadata such as texture and hue depth cues and shape rules that allow us to generate a 3D scene from the 2D information. Similarly, in graphics we talk about extra dimensions that exist in display space but are not used to represent data. Tallies, for example, arrange marks on a plane so that they can be counted individually or in clusters. The dimensions of the plane mean nothing except to define an area within which marks can be made.

13.1.9 Connected Spaces

We now examine some connected spaces used in graphing data. Most often these are metric spaces, but we will show some examples of quasi-metric spaces that have interesting uses.

13.1.9.1 Metric Connected Spaces

A well-known metric often used on connected spaces is called the **power metric** or **Minkowski metric**:

$$d(x_j, x_k) = \left(\sum_{i=1}^{n} |x_{ij} - x_{ik}|^p \right)^{1/p}$$

This formula is based on the ***L^p* norm** of the vector of coordinate differences in n dimensions. When $p = 2$ (the L^2 metric), this formula evaluates to the **Euclidean metric**. When p takes other values from 1 to infinity, the power metric corresponds to some interesting measurement models.

Figure 13.4 shows **isodistance contours** (level curves showing equal distance from a locus point) for selected values of p in a 2D space ($n = 2$). Distances from the center of each square run from red (near) to blue (far). When $p = 1$ (the L^1 metric), the contours resemble diamonds. This value of p corresponds to the so-called **city-block** or **taxicab** metric. City-block contours are computed as a sum of horizontal and vertical distance components. This type of function is used in least-absolute-values regression. When $p = 2$, the contours are circular. This is the familiar Euclidean metric. This type of function is used in ordinary least-squares regression. As p approaches infinity, the contours approach a square, because

$$\lim_{p \to \infty} \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p} = \max_i |x_i|$$

We call the L^p metric with large p a **dominance metric**, because one dimension dominates the distance computation. It is not difficult to imagine plausible psychological models with this metric. In such models, the most salient dimension dominates all others; these models fit the judgments of some fanatics and extremists. Markov models and neural networks where single nodes outvote all others in an ensemble may fit a dominance metric as well.

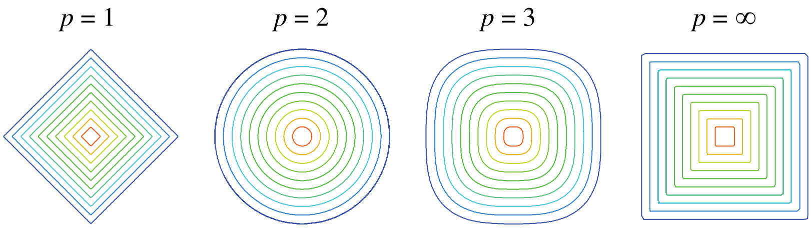


Figure 13.4 *Isodistance contours for power metrics in two dimensions*

13.1.9.2 Quasi-metric Connected Spaces

When a metric space fails to fit our data or our assumptions about the data, we might consider a **quasi-metric** and an associated **quasi-distance** model that violates one or more of the metric axioms. The word *violate* suggests mayhem, but it has its uses. We begin with the triangle inequality.

Nonconvex Distance Maps

One way to violate the triangle inequality axiom is to use a non-convex distance map. Figure 13.5 shows a space in which we use the power formula with $p < 1$ to compute distances among three points. In a psychological context, this type of distance function indicates that subjects judge distances by focusing on the *smaller* of the dimensional differences (the reverse of the dominance model). Notice that the distance from A to B and from B to C is roughly 5 contour units. A and C, by contrast, are separated by many more than 10 contour units. It is shorter to go from A to C (or C to A) by going through B than by going directly. This is not an implausible model for some psychological processes. Psychologists call this type of model, when applied to series of behaviors or a sequence of preferences, **chaining**.

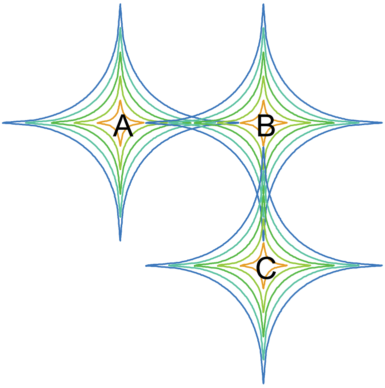


Figure 13.5 *Triangle inequality violation*

Asymmetric Distance Maps

It is easy to violate the symmetry axiom. One way is to use an asymmetric distance function. Figure 13.6 shows an example. In this case, distance is directional. Calculating the distance between two points involves a different direction for each point to the other. Can you see why this type of distance map would also violate the triangle inequality?

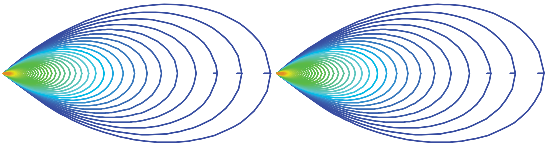


Figure 13.6 *Asymmetric isodistance contours at two points*

Another way to violate symmetry is to use two different distance maps, one for each point in a pair. Figure 13.7 shows an example using a separate kernel density estimate for each of two groups of data. The distance of any point to a group centroid is calculated by using the group's own distance map. This procedure is used routinely in nonparametric classification using Bayes' theorem. See Duda *et al.* (2001) or Hastie *et al.* (2001) for more examples.

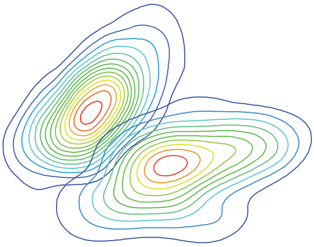


Figure 13.7 *Kernel density functions and their isodistance contours*

13.1.9.3 Geodesics on Connected Spaces

In Euclidean space, geodesics are unique straight lines. In 2D city-block space, geodesics are not unique. They are instead a set of zigzag paths laid out on a rectangular grid. Other exponents in Minkowski space produce curvilinear geodesics. On the complex plane under conformal mappings (see Section 9.1.5), geodesics are curvilinear as well.

In smooth Minkowski spaces ($p > 1$) geodesics can be drawn by using small straight-line plotting steps perpendicular to the tangent at the nearest point on the next contour (isolevel curve). This method (due to Gauss) works with other smooth spaces as well. If we embed the distance map in 3D space, where distances are represented by the height of a surface rather than a contour, then the projection of the geodesic onto the distance map delineates a **gradient**. A gradient is the path a ball would follow from a point near the top of a hill while rolling down the hill. The gradient, because it is a shortest path, figures in many maximization algorithms. Figure 13.8 shows a selected gradient (white curve) on one of the two densities shown in Figure 13.7.

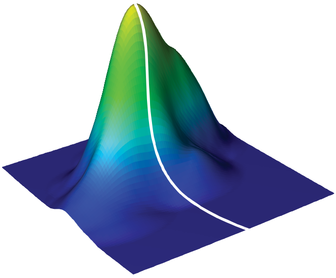


Figure 13.8 *Gradient on 3D kernel density*

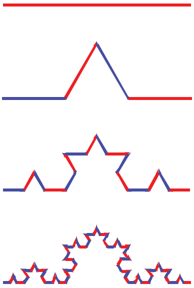
13.1.10 Fractals

A **fractal** is a shape made of parts similar to the whole. Being similar to the whole implies **self-similarity**. A set is self-similar if it can be cut into arbitrarily small subsets, each of which is similar to the entire set. A line of length 1, for example, can be broken into n lines, each of size $1/n$. Each of those separate lines can be broken into n lines, and so on. Self-similarity is not sufficient for defining a fractal, however. All fractals are self-similar, but not all self-similar objects are fractals. We need an additional requirement.

The second requirement is that an object's Hausdorff dimension be different from its topological dimension. Figure 13.9 contrasts our line example with the Koch curve, which is a fractal object. Both objects are constructed by an iterative scheme. The columns to the left of the graphics show the iteration (i), the number of segments (n), the length of the curve (l), and the scaling factor (s). For the line, $n = s$ because the length of the line is constant. For the Koch curve, however, the length of the curve increases by a factor of 4/3 on each iteration. This makes the function relating n and s the power function $n = s^d$. We solve for d in this equation by taking logs, $d = \log n / \log s$. At any iteration, we get the same result $d \sim 1.262$, which is log(4) / log(3).

For the line, the power function is equivalent to $n = s$ because $d = 1$. Thus, the topological and Hausdorff dimensions are the same. If we divide a *square* by the same iterative procedure, the number of sub-elements resulting from the partitioning increases quadratically relative to the scaling factor, so the power function yields $d = 2$. Again, the topological and Hausdorff dimensions are equal, so the recursively partitioned square is not a fractal object. For a *cube*, $d = 3$. And so on. Thus, fractal objects are different from simple iterative partitionings of geometric objects.

i	*n*	*l*	*s*
0	1	1	1
1	4	4/3	3
2	16	$(4/3)^2$	9
3	64	$(4/3)^3$	27



i	*n*	*l*	*s*
0	1	1	1
1	4	1	4
2	16	1	16
3	64	1	64

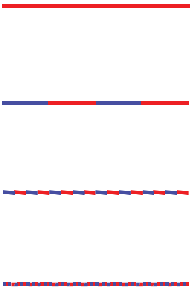


Figure 13.9 *Koch curve and partitioned line*

Figure 13.10 shows another fractal, the Sierpinski gasket. For the Sierpinski gasket, the Hausdorff dimension is log(3) / log(2), which is approximately 1.585. Figure 13.24, later in this chapter, contains an example of a data graphic based on a Sierpinski gasket iterative scheme.

i	*n*	*l*	*s*
0	1	1	1
1	3	3/2	2
2	9	$(3/2)^2$	4
3	27	$(3/2)^3$	8

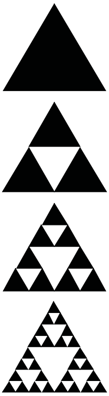


Figure 13.10 *Sierpinski gasket*

The Koch curve and Sierpinski gasket are self-repeating fractals. Self-repetition is a restriction of self-similarity to the condition in which every subset is a copy of the original. The Koch curve and Sierpinski gasket are self-repeating because the generator pattern (see the object at $i = 1$) is applied identically to every subset at each iteration. Self-similar fractals do not necessarily have this repetitive characteristic within each level of detail.

Figure 13.11 shows an example of one of the most famous self-similar fractal objects, the **Julia set**. This example was produced by iterating the complex function $z_i = z_{i-1}^2 + c$, where $c = .11 + .625\boldsymbol{i}$. (Review Section 9.1.5 for the basics of the complex plane.) The black areas of the graphic depict the set itself (within the resolution of the plot). The other areas are colored, using a scale running from blue to red, according to how quickly the iterated function goes to infinity (*i.e.*, far outside the unit circle on the complex plane). There are numerous interactive programs on the Web for exploring the family of Julia sets, popularly known as the Mandelbrot set.

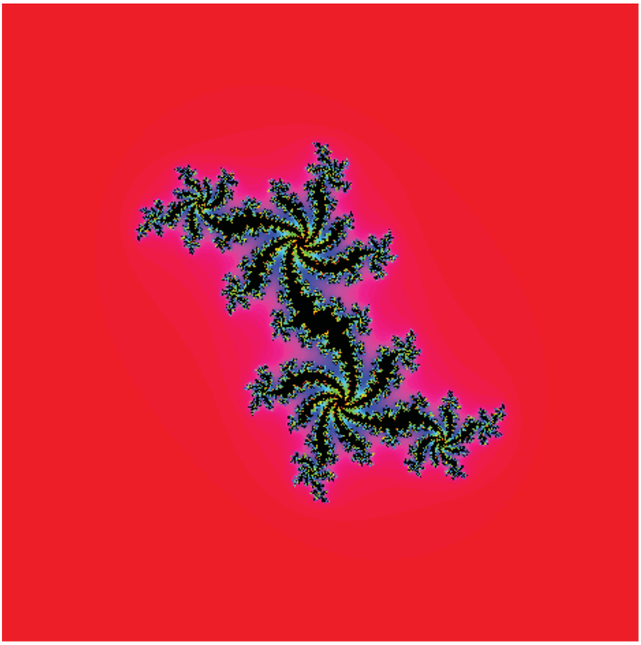
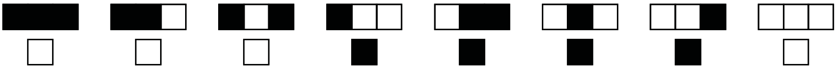


Figure 13.11 *A Julia set*

The question naturally arises whether fractal objects have a use in representing statistical data. The word *fractal* has been applied loosely to some recursive graphics (*e.g.*, Rabenhorst, 1993), but these are not fractal objects because their Hausdorff and topological dimensions are the same. Recursion and nesting are indeed useful graphical tools; we discussed them in Chapter 11. Fractals, though visually fascinating, are nevertheless specialized objects. The excitement fractals and chaos theory originally raised in the social and natural sciences has not been long lasting. The obvious applications of fractals to graphics have already been explored. It remains to be seen how they can be used to display data.

13.1.11 Discrete Metric Spaces

A discrete metric can be defined by letting the distance between any distinct points *x* and *y* be 1, and otherwise 0. Other definitions are possible. This section introduces a type of discrete space that has received a lot of attention recently. It is based on an iterative process for generating points. A **cellular automaton** is a recursive algorithm for generating a set of cells from another set of cells (Toffoli and Margolus, 1987). A one-dimensional automaton generates a row of child cells given the pattern in a row of parent cells. Here is a depiction of one such algorithm (parent cells above, child below):



The generating rule summarized in the two rows of this diagram specifies that a cell in a child row is to be black if its three adjacent parent cells have any of four patterns. The pattern of black and white cells arrayed in the second row of the figure represents *binary*(00011110) = *decimal*(30). Given eight possible binary patterns of three cells, there are 256 nearest-neighbor rules for generating child cells (2 to the eighth power) for this type of automaton. Figure 13.12 displays several hundred iterations of the Rule 30 automaton shown above. Wolfram (2000) shows all 256 patterns. There is a surprising variety of patterns in this one-dimensional inventory. Wolfram also discusses multidimensional cellular automata, as well as continuous and probabilistic automata.

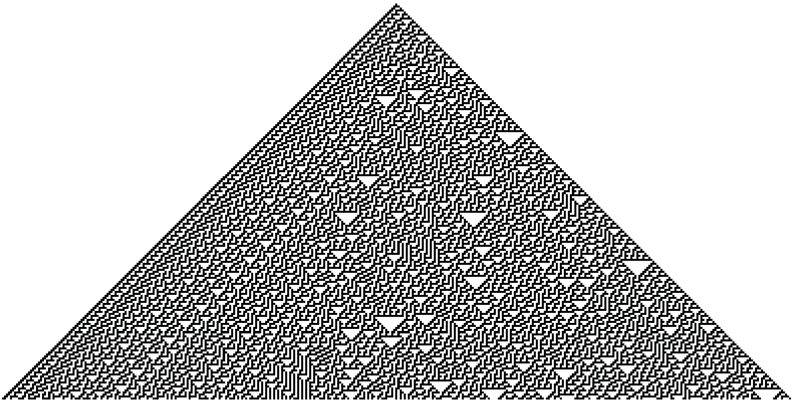


Figure 13.12 *Cellular automaton, Wolfram's Rule 30*

Wolfram is the comprehensive source for general and specific issues related to this class of algorithms. As a physicist, he focuses most intensely on the contrasts between continuous and discrete spatial theories of the cosmos. As with fractals, it remains to be seen how fruitful these models will be for graphical displays.

13.1.12 Graph-Theoretic Space

A **graph,** notated as $G = \{V, E\}$, consists of a finite set of **vertices** (V) and a finite set of **edges** (E). Vertices are sometimes called **nodes** and edges are sometimes called **arcs** or **links**. Each edge in a graph is an unordered pair of vertices (v_i, v_j). An edge with $v_i = v_j$ is a **self-loop**. A **simple graph** has no self-loops. A **weighted graph** assigns a non-negative real number to each edge. A **directed graph** is a graph whose edges consist of ordered pairs. A **complete graph** has an edge between every pair of vertices.

Vertices v_i and v_j are **adjacent** if they occur together in an edge. A **neighbor** of a vertex is an adjacent vertex. The **degree** of a vertex is the number of its neighbors. A graph can be represented by an **adjacency matrix**. The entries a_{ij} of this $n \times n$ matrix contain a 1 if vertices v_i and v_j are adjacent and a 0 otherwise. In a weighted adjacency matrix, the nonzero entries contain weights instead of 1's.

A **path** is an ordered set of vertices (v_1, v_2, ..., v_n) in which v_i is adjacent to v_{i+1} for $i = 1$ to $n - 1$. The **length** of a path is the count of the edges covering it (or the sum of the weights of the covering edges). A **cycle** (or **circuit)** is a path in which $v_1 = v_n$. An **acyclic graph** has no cycles. A graph is **connected** if there exists a path between each pair of vertices.

The dimensionality of a graph is sometimes defined in terms of the minimum dimensionality of a space into which it can be embedded. For example, a **planar graph** can be represented in 2D Euclidean space as a set of points connected by non-intersecting lines. The dimensionality of a planar graph in this context is 2.

13.1.12.1 Metrics and Geodesics on Graphs

A geodesic on a graph is the shortest path (not always unique) between two vertices. **Graph-theoretic distances** are the lengths of geodesics between pairs of vertices. Recall that a **metric space** is a set S together with a distance function on $S \times S$. If we associate the vertices of a graph with elements of S and let our distance function be the length of a geodesic path for each pair of vertices, then a graph is a metric space. You can verify that the three metric axioms are satisfied by this definition.

An adjacency matrix is a distance matrix for a complete graph. For connected incomplete graphs, we can calculate a distance matrix by powering the adjacency graph, but this is not computationally efficient. Skiena (1998) covers more efficient algorithms.

13.2 Psychological Space

The terms **cognitive map**, **cognitive landscape**, and **mental terrain** in popular science writing reflect a metaphor that has existed for centuries: the geography of the mind. As a template for cognitive science, mental maps derive from a time in the history of psychology when continuous spatial models were thought to be useful representations for both memory and cognition. It is an appealing psychological model. Pavlov, for example, theorized that the brain is an associative network in which evocation of a response to a stimulus is likely to activate spatially associated cortical events. This spreading activation model underlies many cognitive neural network theories, and brain imaging studies have often supported the notion that similar processes are located in contiguous areas of the cortex.

The history of spatial perception models has been closely linked to the history of mechanical models — from the *camera obscura* to the digital computer (Neumann, 1996), The camera model of vision includes a lens and/or aperture (lens and iris), film (the retina), and a stereoscopic arrangement of the eyes that is supposed to account for depth perception (see Section 9.2.2). This model has long been recognized as deficient (if only for the fact that two eyes are not needed for depth perception; cover one of your eyes and walk around). Even if the eye were a stereo camera, however, it would not mean that images are processed and stored as 2D pictures or 3D models.

Is there a mental landscape? This question has no simple answer. First, there is no necessary connection between cortical adjacency and mental associations. Showing that similar memories arise from stimulating adjacent areas of the cortex does not prove that mental events are organized spatially, and showing that adjacent areas of the cortex are activated by similar thoughts does not prove that the cortex is organized spatially. Second, we must distinguish the various perceptual processes before trying to answer this question. Most theories of perception postulate at least two stages of processing: 1) pre-attentive stages, in which we perceive a stimulus without effort, and 2) higher cognitive stages, in which we analyze aspects of the stimulus to make judgments. It is possible that color vision, for example, is representable by a metric space but judgment of risk is not.

13.2.1 Spatial Models of Pre-attentive Processes

Pre-attentive visual and auditory processes appear to be spatial because the prevailing model of both perceptual systems is a space-time component model. In audition, the time domain appears to be transformed and decomposed in the frequency domain. In vision, the spatial domain appears to be transformed and decomposed in the spatial frequency domain. Without getting into details, we can imagine feature detectors that work like filters. They look like the function on the right in Figure 13.13, which is a difference of Gaussians (DOG) shown on the left (red subtracted from blue). Functions like this are used in image processing and other applications to filter out unwanted frequencies. See the next chapter for more information on filters. Also, see Section 7.1.4 for information on smoothing kernels. These kernels are constructed from probability functions, which (unlike DOG) cannot be negative.

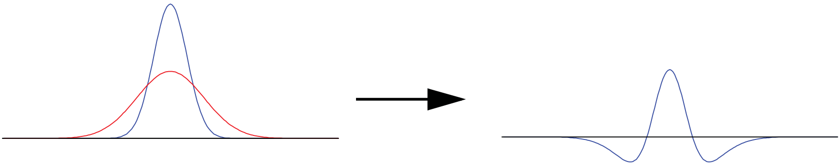


Figure 13.13 *Difference of Gaussians function*

These component models fit both psychological and physical evidence (Levine, 2000). Stimulation of cells in the retina and the visual cortex, for example, reveals a circular area of excitation centered on the site of stimulation, surrounded by a ring of inhibition, as suggested by a 2D polar DOG and similar functions. It is easy to imagine contrast, orientation, and other feature detectors based on this model, and again, the physiological evidence supports that conjecture.

The frequency-components-of-space/time model fits better locally than globally, however. The experimental evidence appears to suggest that pre-attentive processing is more similar to specialized video or audio filters used to process local areas of a video or audio stream (Kehrer and Meinecke, 1995). The perceptual system does not appear to process the whole image or stream with a single filter. The experimental evidence thus suggests that there is not a global Euclidean perceptual space, but other spatial models have done quite well.

Perceived color space, for example, appears to be metric (Indow, 1988), although not Euclidean. The left configuration in Figure 13.14 shows a sample of 12 spectral colors from the 1931 CIE chromaticity diagram used for color calibration. This is a 2D projection of a Euclidean three-component theoretical color space (see triangular coordinates in Section 9.2.3). The CIE space is used in calibrating computer monitors and other graphic displays. The right configuration shows an MDS solution for a matrix of similarities judged by a group of subjects on samples of the same spectral colors (Ekman, 1954). One is clearly transformable to the other, although not through simple isometry or similarity transformations. Indow has examined 3D perceptual color space more extensively and found that the metric axioms do apply. Question: Why is the bottom of the circle open? Hint: Try to imagine the colors that fit there.

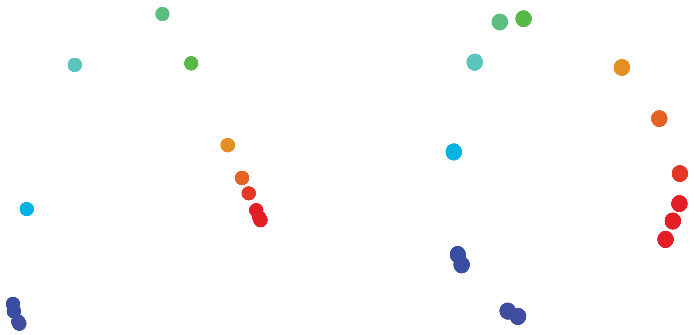


Figure 13.14 *1931 CIE (left), MDS of Ekman data (right)*

There is good evidence for a similarity model based on a metric space in the pre-attentive perception of form as well. The mental rotation research of Shepard, an example of which is shown in Figure 12.5, supports this model, as do other studies of Shepard (*e.g.*, Shepard and Cermak, 1973; Shepard and Chipman, 1970). Krumhansl (1978) reviews other evidence. Santini and Jain (1997) and Edelman (1999) have had success applying these similarity models to shape recognition and retrieval of spatial information from databases.

13.2.2 Spatial Models of Cognitive Processes

In an influential paper, Amos Tversky (1977) wrote that human judgment of similarity often does *not* follow the metric axioms. Tversky was well-known for carefully designed experiments with human subjects that revealed systematic violations of classic assumptions underlying psychological and economic models, such as violations of transitivity ($A > B > C > A$) and violations of the triangle inequality ($d_{ik} > d_{ij} + d_{jk}$). His work with his colleague Daniel Kahnemann, which showed people's decisions under risk do not follow expected utility theory, established the field of behavioral economics and resulted in a Nobel prize. Tversky's research showed, among other things, that higher cognitive processes often distort perceptual information and result in behavior that is not consistent with the metric axioms. Subsequent research on judgment of similarity has supported Tversky's claims (*e.g.*, Deregowski and McGeorge, 1998).

The significance of Tversky's research for graphics is that we cannot assume people will judge quantities and relationships the same way after a glance and after lengthy consideration. And we need to take into account the biases introduced by cognitive processes when people judge spatial material. Research by Barbara Tversky (Tversky and Schiano, 1989), for example, showed that adding a political boundary on a map will change the judgment of distances between towns in that map.

13.2.3 Spatial Cognition

What about the perception of space itself? Does a spatial model fit our perception of space? As we have indicated, the visual system is not a stereoscopic camera. Binocular depth cues are important in the perception of space, but so also are monocular depth cues such as linear perspective, size, lighting, texture, color, and kinetic cues.

The experimental and observational evidence indicates that the perceptual world integrated from these cues is not Euclidean. It is a space shaped by the surrounding environment. In the outdoor world it is probably a flattened spherical world, with the sky perceived as closer at the zenith than at the horizon (Koenderink *et al*., 2000). The moon illusion, in which the moon is perceived to be larger at the horizon than overhead, supports this hypothesis (Kaufman and Kaufman, 2000). In the indoor world, spatial perception is influenced by the structure of the room. The famous Ames room, whose walls are distorted in a way that causes people to become giants or dwarfs and balls to appear to roll uphill is one example of this influence (Levine, 2000). In flat world (a piece of paper or computer screen), perception is influenced by the monocular depth cues (texture, color, etc.) that govern our perception of 3D space; the 3D world, no matter how we try to flatten it, is inescapable. Wickens and Hollands (1999) is a good place to explore these issues further.

13.3 Graphing Space

Graphics live in physical space. Physical space is not physicists' space. It is the Euclidean physical space (a page, a screen, an immersive video environment) indexed by a simple 2D or 3D Cartesian coordinate system in which graphs are rendered through the use of aesthetics. When Bertin (1967) speaks of *position*, he means the placement of an object in a physical scene. As we have seen, graphs can be embedded in different mathematical spaces, but ultimately, we must render graphics of these mathematical graphs in a Euclidean physical space.

Thus, this section is about the issues involved in the mapping $S \rightarrow P$, where S is an underlying mathematical space, and P is 2D or 3D Euclidean space. We will first examine the cases where S is a connected space. Then we will explore the cases when S is discrete. Then we will look at examples when S is a $\{V, E\}$ graph. Finally, we will examine mappings where S is a collection of nested spaces.

13.3.1 Mapping Connected Space to Euclidean Space

We discuss mappings of non-Euclidean connected spaces to Euclidean space.

13.3.1.1 Mapping City-Block Space to Euclidean Space

Table 13.1 shows a distance matrix consisting of the number of blocks a pedestrian has to walk to get from one landmark in New York City to another. We assumed that walks through Central Park (*e.g.*, Lincoln Center to Hunter College) would follow the same grid as elsewhere and we disallowed any shortcuts or trips down Broadway.

Table 13.1 **New York City Blocks Between Landmarks**

	A Lincoln Center	**B** Hunter College	**C** Carnegie Hall	**D** Port Authority	**E** Rocke-feller	**F** Times Square	**G** Empire State	**H** Chrysler Building	**I** Grand Central
A	0								
B	9	0							
C	9	14	0						
D	22	32	16	0					
E	18	20	8	11	0				
F	22	29	12	4	7	0			
G	34	37	28	11	17	13	0		
H	28	26	18	6	10	5	11	0	
I	25	25	16	6	8	4	10	1	0

Figure 13.15 shows an MDS spatial layout of the New York City landmarks, using the city-block metric. The stress of this solution is almost zero. We set the SYSTAT MDS program to use the exponent $p = 1$ in the power metric. Because New York City blocks are approximately three times longer between avenues than between streets, we stretched the plot after the MDS computation to be three times taller than wide. This is equivalent to using a weighted city-block metric in the scaling.

The result is an almost perfect similarity between geographic and virtual MDS coordinates. This looks like an ordinary map of Manhattan. Why did this happen? The simple explanation is that New York's street grid is embedded in 2D Euclidean space. Actually, it's embedded in another 2D space the surface of a sphere — but the differences between plane and sphere are negligible at this resolution. Thus, there is a similarity mapping from city-block to Euclidean coordinates at this scale. In reading the map, therefore, we can assess relations between landmarks in both Euclidean and city-block space. This is not always the case. When we examine MDS scalings of dissimilarities not derived from physical space, we cannot interpret Euclidean distances between points. We must instead train our eye to follow city-block paths.

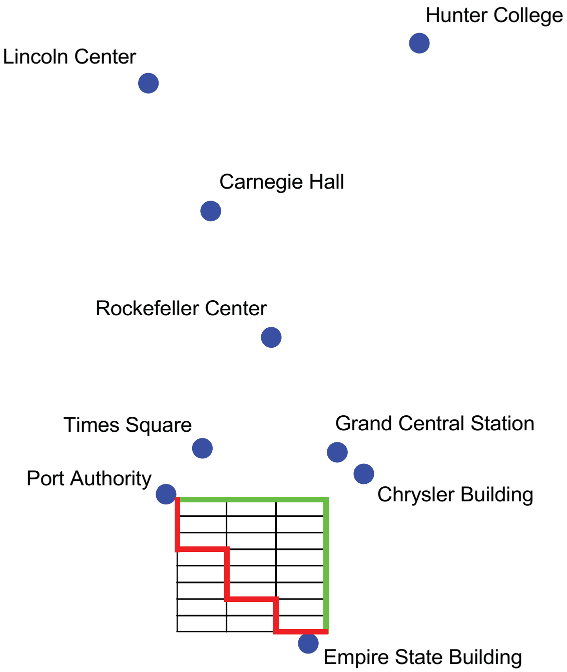


Figure 13.15 *City-block space and geodesics*

At the bottom of the figure, we show all the city-block metric geodesics between the Port Authority terminal and the Empire State Building (actually between the nearest street corners). Every geodesic lies on the black grid. We have highlighted two of these paths in red and green. There is no unique shortest geodesic in the city-block metric (unlike the Euclidean "crow-flies" metric on the plane, which in this case lies on a straight line between the two landmarks). Another way to understand the multiplicity of city-block geodesics is to realize that the green path and the red paths will take a pedestrian the same amount of time to walk (assuming no differences in stoplights or sidewalk obstructions). There is an interesting psychological artifact in this figure. Does it seem that the red path is shorter than the green? Pedestrians (and many drivers) experience this phenomenon when navigating through city space. They prefer zigzag paths through city streets, thinking they are shorter.

13.3.1.2 Mapping Affine Space to Euclidean Space

Recall from Chapter 9 that distances in a Euclidean space are invariant with respect to isometry transformations (including rotation). In the Euclidean metric, distance is not directional; the distance contours in Figure 13.4 are circular. That means geodesics of a fixed length all end on the circumference of a circle. We call a Euclidean space **isotropic** (from the Greek ἰσότροπος, which means *the same in any direction*).

Affine transformations introduce a dependency in the distance calculations, making distance directional. We call an affine space **anisotropic**. In some applications, we compute distances directionally because of gravitation, magnetism, temperature, or other biasing factors. It is easy to modify the Euclidean metric to be anisotropic. First apply different weights to the dimensions and then rotate. We modify the Euclidean formula as follows:

$$d(x_j, x_k) = \left(\sum_{i=1}^{n} w_i (x_{ij} - x_{ik})^2 \right)^{1/2}$$

This formula has the effect of stretching or shrinking each dimension separately. You can verify graphically that identity, symmetry, and the triangle inequality hold for this case. To create an anisotropic distance function, we rotate the weighted Euclidean distance function through a change of variables.

A well-known application of this metric to models in probability space is expressed through so-called **Mahalanobis distances** (named after P. C. Mahalanobis, founder of the Indian Statistical Institute). Let $(x_1, ..., x_p)$ be i.i.d. normals (independent, common mean and standard deviation). The joint density is the product of the marginal densities

$$\phi(x_1, \ldots, x_p) = \frac{1}{(2\pi)^{p/2}(\sigma_1 \sigma_2 \ldots \sigma_p)} \exp\left(-\frac{1}{2} \sum_{i=1}^{p} \left(\frac{x_i - \mu_i}{\sigma_i} \right)^2 \right)$$

In matrix form, this is

$$\phi(\mathrm{X}) = (2\pi)^{-p/2} |\Sigma|^{-1/2} \exp\left[-\frac{1}{2} (\mathrm{X} - \mu)' \Sigma^{-1} (\mathrm{X} - \mu) \right]$$

Set the exponent to a constant:

$$(\mathrm{X} - \mu)' \Sigma^{-1} (\mathrm{X} - \mu) = c$$

In two dimensions, this is the equation for a circle, assuming Σ is an identity matrix. If Σ is a diagonal matrix with different diagonal entries, this becomes the equation for an ellipse (weighted Euclidean metric). If Σ is a positive-definite symmetric matrix with a nonzero off-diagonal element, this becomes the equation for a rotated ellipse (anisotropic metric). Figure 13.16 shows an example of Mahalanobis isodistance contours.

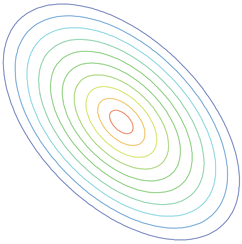


Figure 13.16 *Mahalanobic anisotropic isodistance contours*

The following example highlights the consequences of evaluating anisotropic distances. In Figure 13.17, we use a within-group bivariate normal distribution estimated from the Iris data to classify a specimen as one of two species. In the left panel, it appears that the blue star representing a new specimen is closer to the red distribution. By displaying a larger level curve for the density in the right panel, we see that it is (in the anisotropic metric) closer to the blue distribution. Graphically, the contours function as guides for distance calculations. Plotting more than one contour can assist us in evaluating the 2D space.

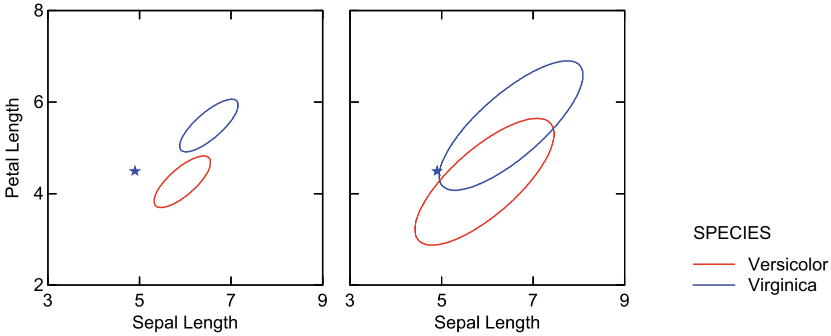


Figure 13.17 *Anisotropic classification*

13.3.1.3 Mapping the Sphere to 2D Euclidean Space

As we have seen, most geographic projections involve locally continuous mappings from the surface of the sphere to the plane. Tobler (1999) presents a number of unusual spherical-planar mappings. Figure 13.18 is a map projection based on the ocean routes between ports. In some geographic projections, great circle routes map to straight lines in Euclidean space. Tobler's projection in Figure 13.18 maps paths along shipping lanes between world ports to short curves in 2D Euclidean space. Tobler has turned land-world into water-world and has given us a sense of the space-time characteristics of ocean travel.

Is this mapping continuous. Is it a homeomorphism? Do the North and South Poles go to points in this mapping? Are the shortest paths between ports straight lines?

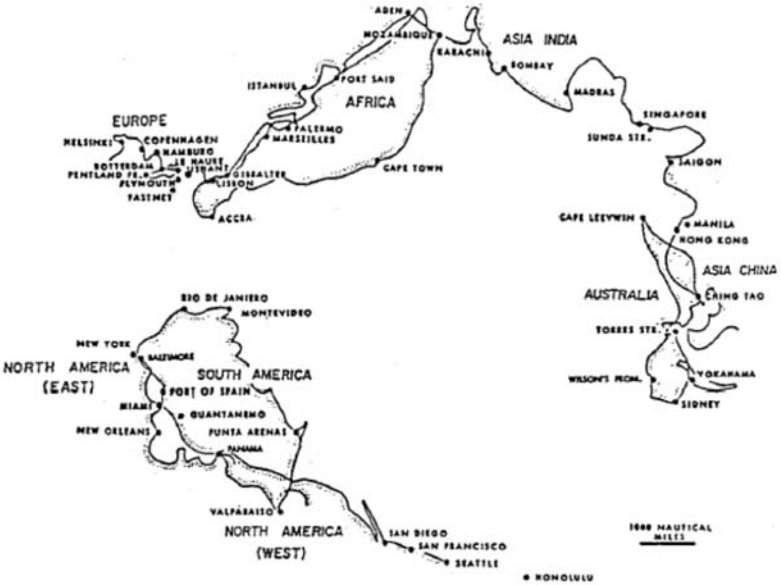


Figure 13.18 *Tobler's map of ocean distances (courtesy of Waldo Tobler)*

13.3.1.4 Sharing Space

Every scatterplot involves shared space. This is because symbols have nonzero sizes and thus occupy real space, often overlapping each other. We try to choose symbol sizes that minimize these overlaps. A popular chart called the **bubble plot** steals even more space, because symbols are sized according to an extrinsic variable. Figure 13.19 shows a simple example of this representation. We have sized the anemones according to an extra variable in Kooijman's dataset — the diameter of the anemones. This bubble plot helps us locate anemone bullies. Anemones can be territorial and aggressive toward each other (McCloskey, 2003). For this dataset, however, it is curious that there is almost no correlation ($r = .09$) between the size of the anemones and the area of their Voronoi polygons. Voronoi polygons are one estimate of the territory owned by each anemone (see the fiddler crab example in Section 7.2.5.3), so one might expect larger anemones to have larger territories if size gave them a competitive advantage.

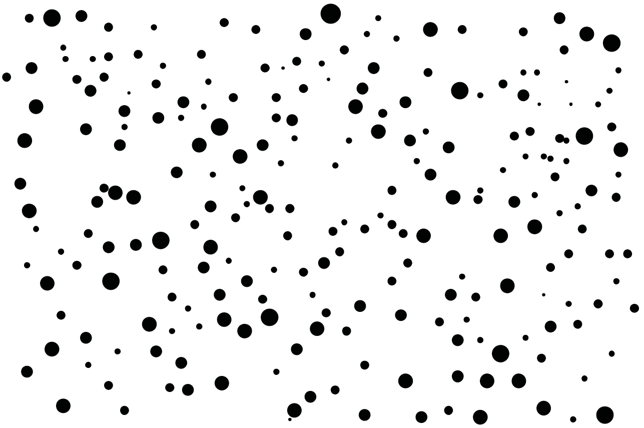


Figure 13.19 Beadlet anemone sizes

There is a problem with this example, however. If the size variable represents the physical sizes of the anemones, then we are not stealing space at all. In fact, Figure 13.19 can be considered more like a photograph of the scene. Two anemones cannot occupy the same space. Figure 13.20 shows a more typical example of a bubble plot. We have represented cancer rates aggregated within US states. The bubbles steal geographic space. Despite their visual similarity, these two plots are fundamentally different.

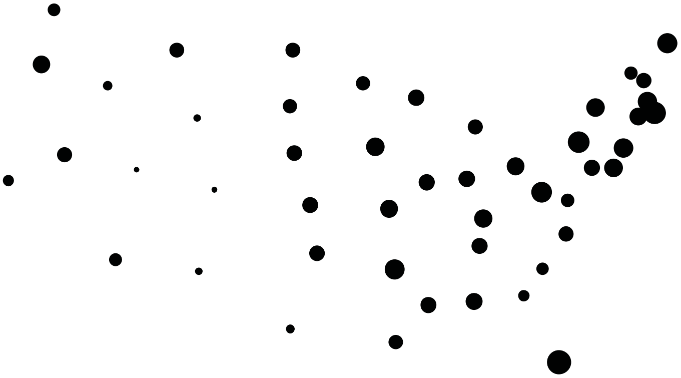


Figure 13.20 US cancer rates

13.3.1.5 Label Layout

Labels usually steal more space than do symbols. In a scatterplot, we can think of them as sitting on a second layer above the primary layer, with both layers representing the same space. How do we keep them from occluding the data? Figure 13.21 shows two scatterplots of the countries data shown in Figure 1.1. In the first, the labels collide. In the second, the labels have been moved slightly in order to avoid overlaps. The technology for doing this has been developed by geographers and computer scientists in the last few years. Although the problem complexity grows exponentially in its most general form, simple heuristics based on simulated annealing (a probabilistic iterative minimization scheme) do quite well. Christensen *et al.* (1995) provide a survey. The lower plot in Figure 13.21 makes use of an additional remedy. If the algorithm cannot find a clear place for a label, it hides the label.

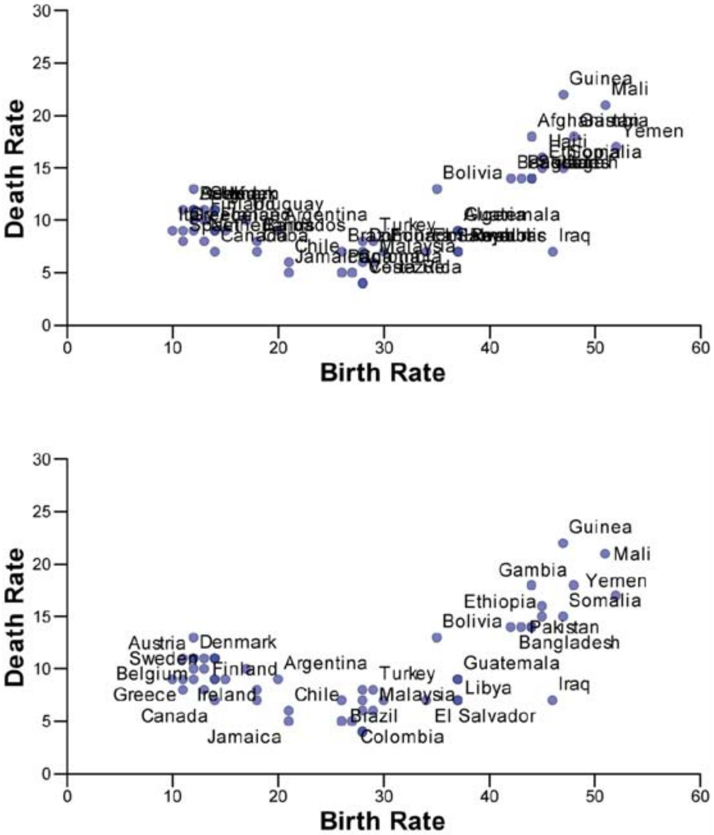


Figure 13.21 Country labels

13.3.2 Mapping Discrete Space to Euclidean

When we map discrete space to Euclidean space, there are areas inside the frame but outside the image of the mapping. That is, there is free space where no points go. We may use free space for shape and size aesthetics and for annotation. If we change rectangular shapes to anemone shapes for a bar graph, we need room to render the shapes without colliding with other elements. If we change the width of bars, we have a similar problem. This problem is different from the one we encounter when choosing the shape of a symbol in a scatterplot or finding a place to put a label. In that case, we *share* space with the layout of the points in a frame. This section covers how to employ free space.

13.3.2.1 Categorical Dimensions

Categorical variables are the most obvious creators of free space. A categorical dimension involves only the integers, so the space between tick marks is meaningless (has no measure). Figure 13.22 shows various charts on the same data from the EPA involving mileage of cars from several manufacturers. The charts in the middle row, popular in business graphics, disguise the lattice or mesh spacing on the horizontal axis. Because the lattice spacing is arbitrary, slopes or areas based on a categorical axis are subject to misinterpretation.

Slopes between categories are meaningful only if the categories are spaced on an interval scale. That is clearly not the case here, since the categories are sorted alphabetically — an arbitrary ordering relative to the data at hand. Accordingly, we have reordered and linearized the scale in the charts at the bottom of the figure. This ordering highlights the outstanding performance of VW and Honda. The bottom charts contain the same information as the ones above. They contain more perceivable information, however, because they make clearer the differences in scale. We have deformed the categorical space to reveal a simple pattern.

Rescaling doesn't solve all our problems, however. The areas between the bars or vertices of the line and area charts are free space — meaningless space. It makes no sense to connect them. We would argue that for applications such as this one, users stay with bar charts or dot plots to represent their data. Popular graphics software makes categorical line and area charts easy to do, but they are rarely appropriate.

Figure 13.23 shows how we can use free space (width of the bars) to represent an extra variable (sample size) with the *size* aesthetic. Bar sizes have one degree-of-freedom. That is, their lengths are determined by a *position* aesthetic, so their widths are free to vary with *size* in 2D. We can exploit this extra degree-of-freedom by attaching it to a variable. Because height and width of rectangles are configural dimensions, however, we are not fond of this practice. Try comparing the height of "Working full-time" with "Retired" and "Temp not working" with "Retired." The differences in width influence the comparative judgment of height. There are other reasons we do not like this graphic. Because the origin of the vertical axis is not zero, we question the usefulness of bars. It would be better, in our opinion, to use dot plots sized by sample size in this frame. See Cleveland (1995) for further discussion.

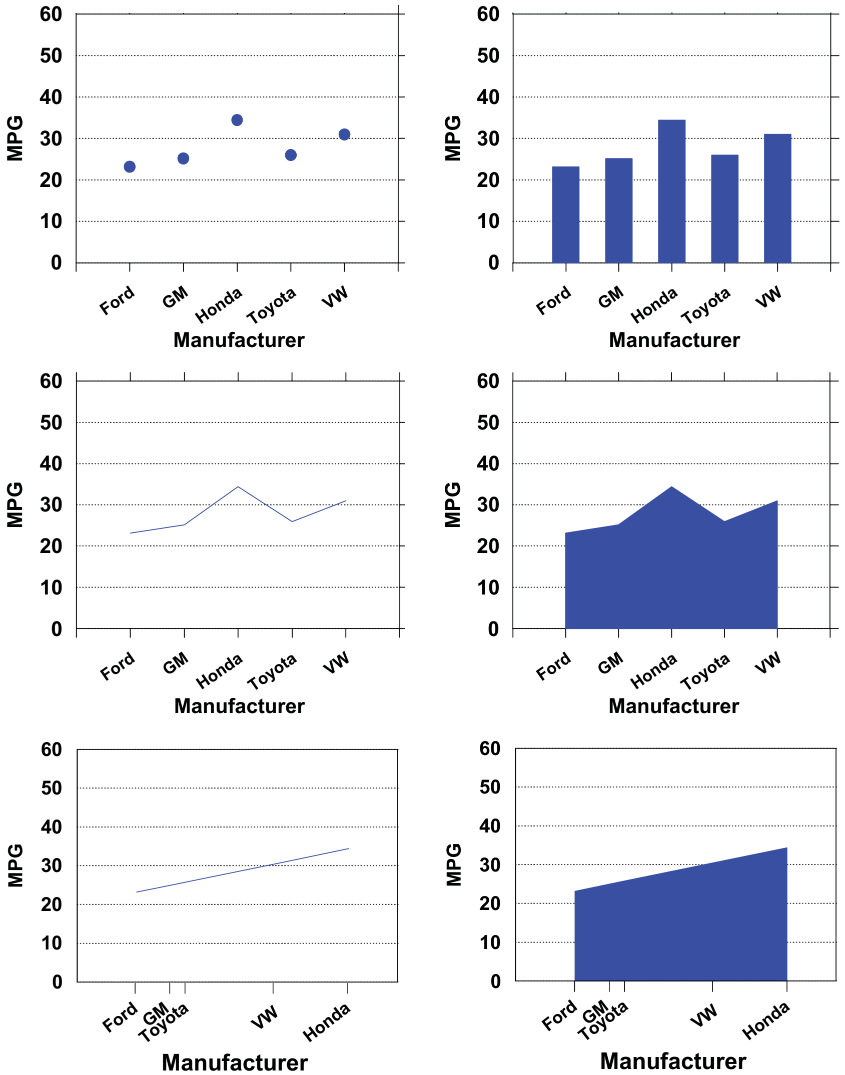


Figure 13.22 Bar, dot, line, and area charts of EPA data

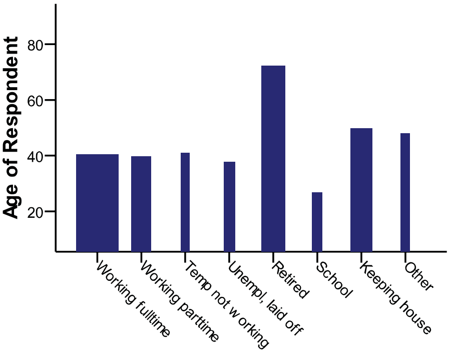


Figure 13.23 *Bar widths proportional to counts*

13.3.2.2 Fractals

Few data displays have employed self-similar discrete structures. An exception is Dan Carr's 3D Sierpinski gasket of biosequences (Carr, 2002). Figure 13.24 shows an example. This structure encodes sequences of the four letter base codes. One letter is situated at each of the four vertices of the tetrahedron. The example in the figure encodes six-letter sequences, so the corners represent the four uniform sequences AAAAAA, CCCCCC, GGGGGG, TTTTTT. Compound sequences, such as ACAAGT are represented as interior points. The right figure shows a rotation to the ACGT plane, where projections of subsequences can be observed. Coloring is used to identify subclassifications of sequences. The power of this chart resides in its ability to encode a huge number of sequences. This chart handles sequential patterns of length 6. Others following the same algorithm can handle many more. Amie Wilkinson has noted that base codes early in the sequence have more influence on location of points than later. Can you see why?

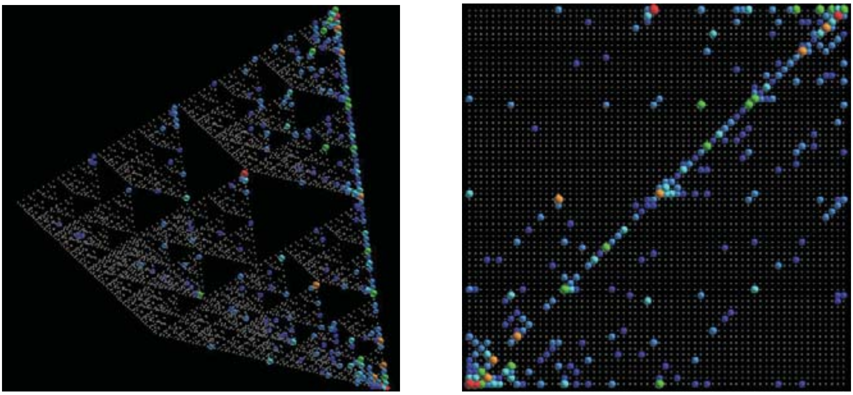


Figure 13.24 *Carr Sierpinski Gasket (courtesy Dan Carr)*

13.3.3 Mapping Graph-Theoretic Space to Euclidean

We will first discuss trees used to represent distances. Then we will discuss examples of layouts for other types of undirected and directed graphs.

13.3.3.1 Distance Trees

A tree is an acyclic connected graph. Any two nodes in a tree are connected by only one path. In this section, we will cover trees used to represent distances. In Section 16.3 we will discuss layouts for directed trees.

Suppose we wanted to represent the distances among all of our anemones with a tree instead of the Euclidean space used in Figure 13.2. This might seem like a peculiar desire, given that we already know all the distances exactly by computing Euclidean distances for each pair. But suppose that there is a network with fewer than $n(n-1)/2$ edges that could account for all the Euclidean distances quite well. Imagine, further, that this network revealed something about anemone society, such as anemone economies.

Spanning Trees

Our first candidate for such a network would be the tree in Figure 13.3. In a **spanning tree**, each object is represented by a node and each distance is computed along the edges connecting the nodes. Spanning trees have $n-1$ edges, quite a reduction in parameters from $n(n-1)/2$ possible edges. A **minimum spanning tree** (MST) has the shortest edge length of all possible spanning trees. The graph-theoretic distances among nodes on the minimum spanning tree in Figure 13.3 correlate .67 with the Euclidean distances — not a bad fit, but we can do better.

Is the MST the best graph of relatively small degree that we can construct to represent all the pairwise distances? If the anemones were arrayed on the rock near the vertices of an equilateral triangular or hexagonal grid (like anemone honey bees), it might be. If they were arrayed on a rectangular grid (like anemone urbanites), it definitely would not be (Holman, 1972). Notice in Figure 13.3 that some of the shortest paths between anemone nodes are rather circuitous. We need to examine some other distance-fitting trees.

Ultrametric Trees

A **hierarchical tree** has a set of **leaf** nodes (nodes of degree 1) representing a set of objects and a set of **parent** nodes representing relations among the objects. In a hierarchical tree, every node has exactly one parent, except for a **root node**, which has one or more children and no parent. A hierarchical tree can be represented by a one-dimensional lexical ordering through the use of parentheses. Figure 13.25 shows an example.

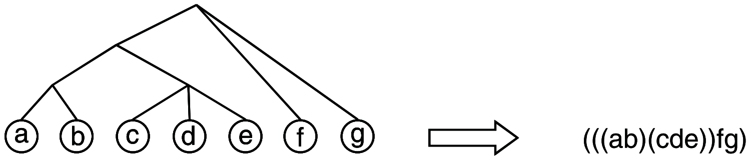


Figure 13.25 *Lexical ordering of a hierarchical tree*

While the linear layout is compact, a hierarchical tree is usually displayed along a distance dimension with the root node at one end (maximum distance) and the leaves at the other end (zero distance). In this layout, the distances between any two leaves is represented by the coordinate of their common ancestor on the distance dimension.

If the node distances are monotonically non-decreasing (*i.e.*, every parent is no closer to the leaves than its children are), then a hierarchical tree is **ultrametric**. An ultrametric is a metric with a more restricted form of the triangle inequality:

$$d(x, y) \le \max(d(x, z), d(y, z))$$

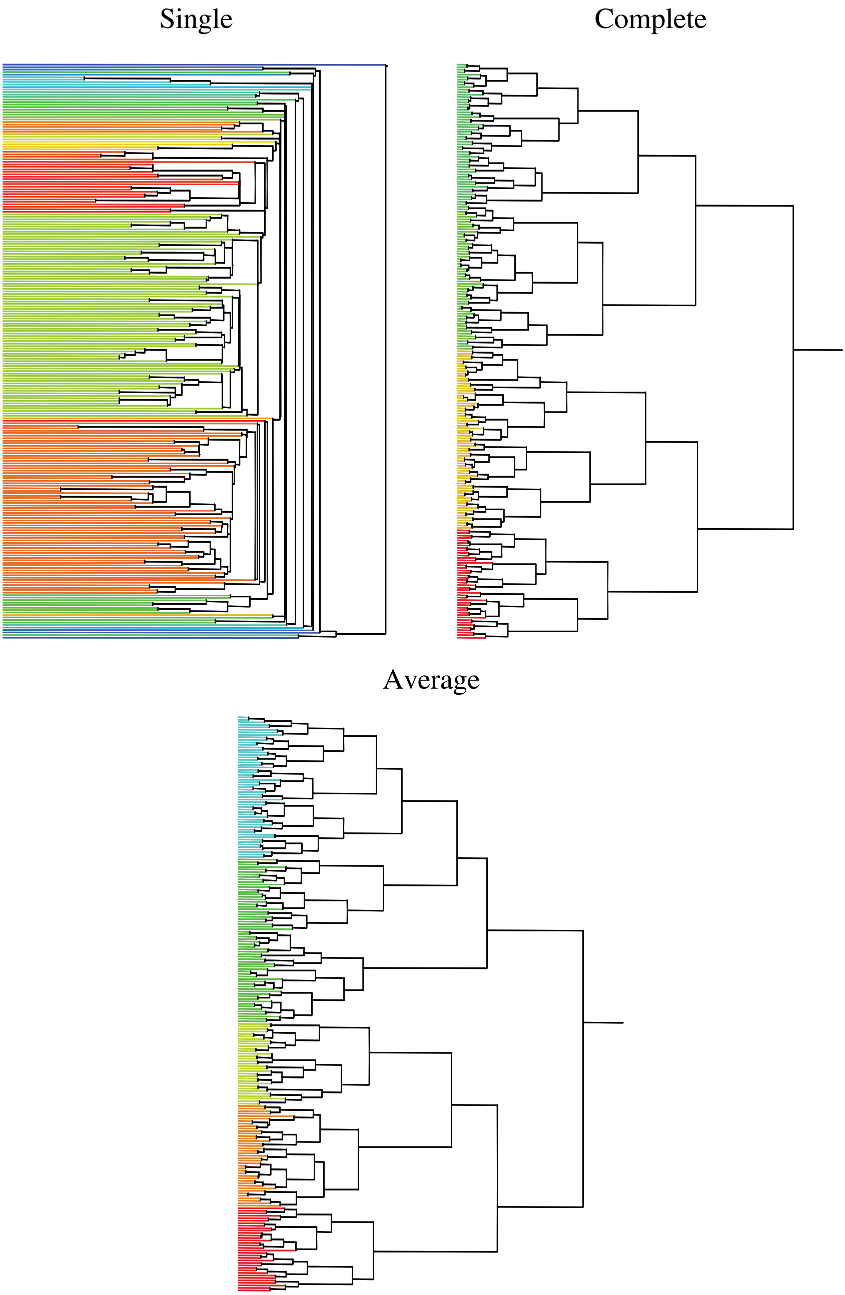
The graph-theoretic distances in an ultrametric tree take a maximum of $n - 1$ possible values, where n is the number of leaves. This is because of the ultrametric **three-point condition**, which says we can rename any *x, y, z* such that

$$d(x, y) \le d(x, z) = d(y, z) .$$

Another way to see this is to note that the distance between any two leaves is determined by the distance of either to the common ancestor.

The hierarchical trees in Figure 13.26 were produced by several popular hierarchical clustering methods. Hierarchical clustering outputs binary trees (parents having two-child families) from an input distance matrix. An hierarchical clustering algorithm joins the two closest objects among n objects, creating a new cluster object. At the next step, the algorithm joins the two closest objects among $n - 1$ objects, one of which is the previously joined cluster of size two. The algorithm continues joining until the last two objects are joined.

The trees in Figure 13.26 are displayed with the distance dimension oriented horizontally from zero distance on the left to maximum distance on the right. SYSTAT colors the tree branches to highlight the cluster structure.



Single

Complete

Average

__Figure 13.26__ Cluster trees on anemone data

Notice that the parent nodes are not spaced uniformly in the horizontal distance dimension. The horizontal spacing of the parent nodes in the three sub-figures is a consequence of the clustering algorithms used to join leaves. **Single linkage** assumes that the distance between two nodes is the *minimum* distance between any leaf belonging to the first node and any leaf belonging to the second; the correlation of this tree's geodesic distances with the input distances is .40. **Complete linkage** assumes that the distance between two nodes is the *maximum* distance between any leaf belonging to the first node and any leaf belonging to the second; the correlation of this tree's geodesic distances with the input distances is .66. **Average linkage** assumes that the distance between two nodes is the *average* of the distances between the leaves belonging to the first node and the leaves belonging to the second; the correlation of this tree's geodesic distances with the input distances is .57. These three (and other) linkage methods produce widely varying results on the same data. You can read further on the reasons for these differences in Hartigan (1975), Duda *et al.* (2001), or Hastie *et al.* (2001).

Of the widely used hierarchical clustering methods, only single and complete linkage always produce an ultrametric tree. In other words, these are the only popular cluster methods that allow us always to lay out a tree on a distance scale (with no branches crossing) and to compute graph-theoretic distances between leaves. Table 13.2 contains data that join with monotonically *decreasing* distances under the popular **median linkage** (median of the distances between the leaves belonging to the first node and the leaves belonging to the second) and **centroid linkage** (distance between the weighted center of the values of the leaves belonging to the first node and the weighted center of the values of the leaves belonging to the second). With centroid linkage, for example, B and D are joined at a distance of 303. Then C is joined with that cluster at distance 299.75. Then E is joined with that cluster at distance of 265.89. Finally, A is joined at a distance of 283.31. Try clustering these data with median or centroid linkage in your favorite statistical package to see how the cluster tree is displayed, if it is displayed at all. For these linkage methods on this distance matrix, SYSTAT evenly spaces the nodes in *join* order rather than *distance* order. For single and complete linkage, SYSTAT outputs the ultrametric tree.

Table 13.2 **Perverse Distance Matrix**

	A	**B**	**C**	**D**	**E**
A	0				
B	385	0			
C	449	384	0		
D	389	303	367	0	
E	461	462	310	377	0

The distance monotonicity condition led Fisher and Van Ness (1971) to call single and complete linkage **admissible** clustering procedures and to exclude other hierarchical methods from normal use. This categorization is extreme for at least three reasons. First, most methods produce ultrametric trees for the vast majority of real datasets, and when they don't, it is easy to verify. Second, hierarchical clustering is most often used to find clusters rather than to model distances. The cluster tree is used as a guide to identifying clusters, not a distance model. Third, single and complete linkage anchor a continuum of hierarchical clustering methods running from those that produce stringy clusters to those that produce compact clusters. Methods like average and median linkage, as well as a variety of other cluster algorithms, are often better at finding clusters with other shapes.

The monotonicity condition for cluster distances does mean that single and complete linkage are the only popular cluster methods that are invariant under monotonic transformations of the input distances (Johnson, 1967). This characteristic makes them especially suited for analyzing ranked dissimilarities, often in tandem with nonmetric multidimensional scaling. In these applications, we usually begin with a dissimilarity matrix that we believe is monotonically related to a set of distances. In this case, single and complete linkage and MDS give us useful tools for modeling the data.

Additive Trees

Let **D** be a symmetric n by n matrix of distances d_{ij}. Let T be a hierarchical tree with one leaf for each row/column of **D**. T is an **additive tree** for **D** if, for every pair of leaves (t_i, t_j), the graph theoretic distance between the leaves is equal to d_{ij}. Spanning trees are additive as well (as in Figure 13.3), but we will focus on additive hierarchical trees in this section.

The idea of approximating distances by additive trees originated among psychometricians working with similarity/dissimilarity data (Carroll and Chang, 1976). Sattath and Tversky (1977) introduced a method for fitting an additive tree to a matrix of dissimilarities. Since then, biologists and computer scientists have investigated additive trees extensively and reduced significantly the complexity of the fitting algorithms. Gusfield (1997) summarizes this more recent literature.

Figure 13.27 shows the Sattath-Tversky additive tree solution for the distance matrix derived from our anemones. The SYSTAT implementation of the algorithm for computing the fit was developed by Corter (1982). The correlation with the input distances is .91, which is a better fit than for any tree we have seen so far. The distances between any two nodes in this tree are computed by summing the vertical edges connecting the nodes. The horizontal edges are used simply to position the nodes conveniently for reading.

This additive hierarchical tree should not be confused with either a rooted spanning tree or a hierarchical tree display in some statistical software. The rooted spanning tree in Figure 13.3 is not hierarchical. And the trees displayed in some computer programs (*e.g.*, S and R) have shortened leaves that make them look similar to an additive tree. This truncation is *ad hoc*, however. Programs such as S truncate long edges leading to terminal nodes in a hierarchical tree to reduce the amount of ink in the display. While the S tree is less cluttered with long edges, particularly for single linkage trees, some might argue this aesthetic invites confused interpretations. Truncating edges at terminal nodes breaks the ultrametric properties of the tree.

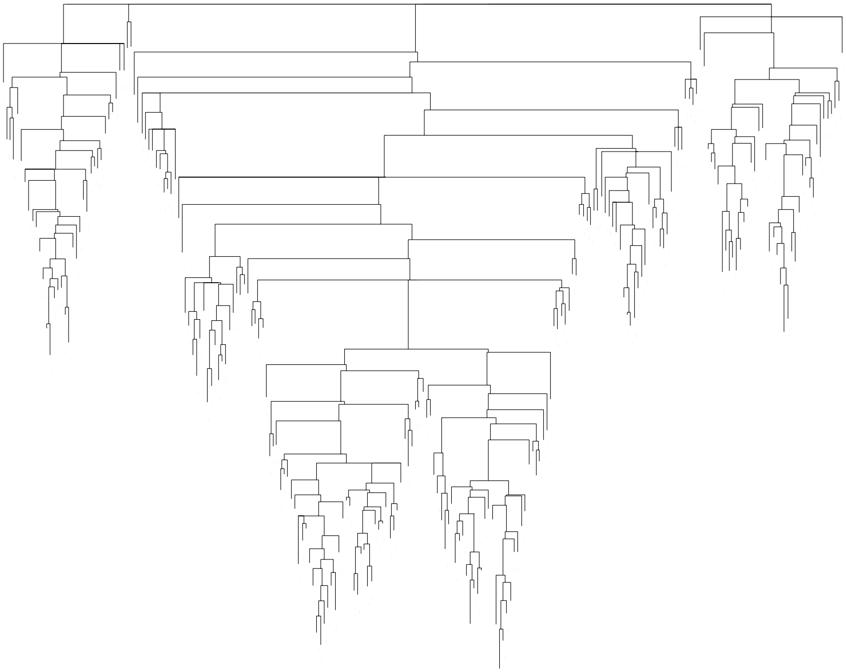


Figure 13.27 *Additive tree on anemone data*

Steiner Trees

There is a graph related to the additive tree called the **Steiner tree** (Skiena, 1998). This is the smallest tree connecting all the vertices of a graph. Unlike the minimum spanning tree, the Steiner tree is allowed to have intermediate connection points (pseudo-vertices) in order to reduce the cost (total length) of the tree. The Steiner tree of a planar graph is the shortest additive tree possible for that graph.

13.3.3.2 Spaces of Graphs

Figure 13.28 shows the nearest-neighbor graph of the anemones. For each vertex, we drew an edge to the closest neighbor vertex. This is a directed graph because the closest neighbor to a vertex may have some other vertex that is closer to that neighbor. If we ignore edge direction, however, we can see this graph is a subset of the MST. In any case, the resulting graph shows clusters of anemones that are closer to each other than to other clusters.

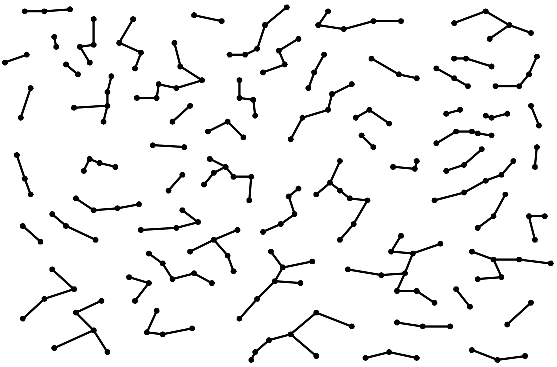


Figure 13.28 *Nearest neighbor graph for anemones*

Points in a space can represent sets. Consider a space based on nearest-neighbor clusters, for example. We can define distance functions on these clusters that follow the metric axioms.

A metric function of this sort is the nearest-neighbor cluster distance, which is the minimal distance between a point *p* in set *A* and a point *q* in set *B*. These distances are shown in green for selected pairs of clusters in Figure 13.29, If the nearest-neighbor distance between *A* and *B* is *d*, then at least one point of *A* must be within distance *d* of at least one point of *B*.

Another metric function is called the **Hausdorff distance**, which is the maximum of: (1) the maximal distance from a point *p* in *A* to its closest neighbor in *B* and (2) the maximal distance from a point *q* in *B* to its closest neighbor in *A*. Selected Hausdorff distances are shown in red in Figure 13.29. If the Hausdorff distance between *A* and *B* is *d*, then every point in *A* must be within distance *d* of some point in *B* and *vice versa*.

We use this example to illustrate that definitions of a space can involve levels of indirection. *C* programmers use *pointers* to refer to an object and they use *pointers* to *pointers* to add levels of indirection. Similarly, we have colored the graph in Figure 13.29 to show a distance that is based on the computation of another distance. Can you devise other graphical methods to make this relation clearer? Are there other examples where it makes sense to construct a graph of a graph?

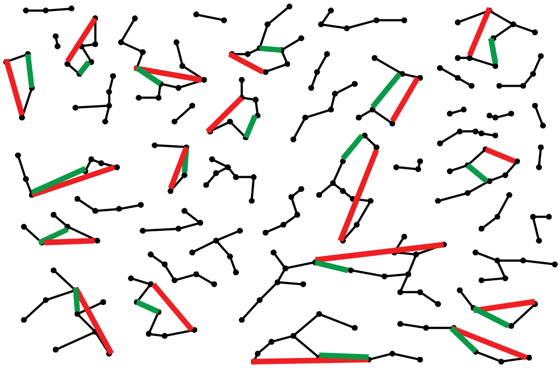


Figure 13.29 *Nearest-neighbor distances (green) and Hausdorff distances (red) between selected anemone nearest-neighbor clusters*

13.3.4 Mapping Nested Space to Euclidean

Nested space involves nesting of connected spaces. There have been many graphical incarnations of this idea. We will discuss two.

13.3.4.1 Treemaps

Treemaps (Johnson and Schneiderman, 1991; Schneiderman, 1992; Johnson, 1993) are hierarchical partitionings of 2D space. The partitioning algorithm is a fixed alternating sequence of vertical and horizontal cuts of nested blocks based on a hierarchical tree. We begin by cutting a rectangle vertically. We then cut each of the partitioned blocks horizontally. We then cut the blocks vertically, and so on. There is no requirement that the cuts be binary, although our examples are all based on binary trees.

Figure 13.30 shows an example using a simple binary tree. The first cut partitions (AB)(CDE). The second cut partitions (AB)((CD)(E)). The third cut partitions ((A)(B))((CD)(E)). The last cut partitions ((A)(B))(((C)(D))(E)). We have sized the rectangles according to the within-cluster distance taken from the ultrametric tree. Splits involving leaves are sized using their parents' distance. In most published treemaps, the blocks are resized according to the values of an extrinsic variable, such as stock market trading volume or market share.

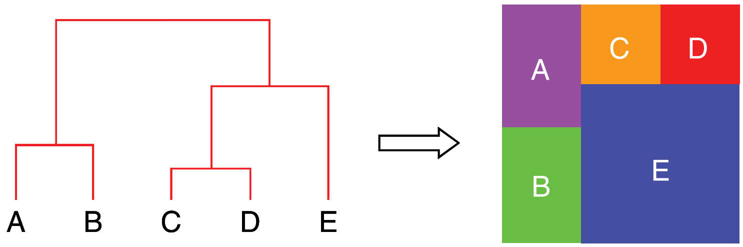


Figure 13.30 Mapping a tree to a treemap

The treemap is a dual of a hierarchical tree. This means that the layout of a treemap is best understood in terms of the known properties of hierarchical trees. Despite its appearance, the metric space of a treemap is *not* the 2D Euclidean plane. It is the graph-theoretic space of the tree that produced it.

What are the consequences of this? First of all, a treemap follows an ultrametric rather than a metric. Many distances derived from the treemap, like those derived from hierarchical clustering trees, are tied. Second, adjacency of blocks is not a distance measure. Two blocks may be adjacent (or near each other) and yet have different parents in the corresponding tree. Third, the dimensions (horizontal and vertical) of the treemap have no simple meaning in terms of the data on which they are based. While they may appear similar to the dimensions of a mosaic or other tiled plots, they have no intrinsic meaning related to the data because they can be reordered without changing the metric properties of the tree on which they are based.

A variety of algorithms have been suggested for constructing treemaps when the data are not already structured as a tree. Because of the dualism between treemaps and hierarchical trees, it is most fruitful to derive and evaluate these algorithms within the framework of the hierarchical clustering literature. To create a treemap, we simply (1) perform a hierarchical clustering, (2) apply the alternating recursive splitting rule, and (3) resize the tiles according to an extrinsic variable. This approach has several advantages. First, we are not limited to one variable or index for producing the splitting tree. We can, of course, cluster on a single variable. Or, we may choose a multivariate distance measure on several variables at once. Second, we may choose different linkage methods to produce different trees. Third, we may devise a tree-ordering algorithm to arrange blocks within each split (see Section 16.5.4).

Figure 13.31 shows three treemaps on the anemone dataset. We used single, complete, and average linkage to compute the trees and we sized the blocks according to the within-cluster distance. In order to make the blocks relatively distinguishable, the colors were randomly chosen, with the same random seed for all three treemaps.

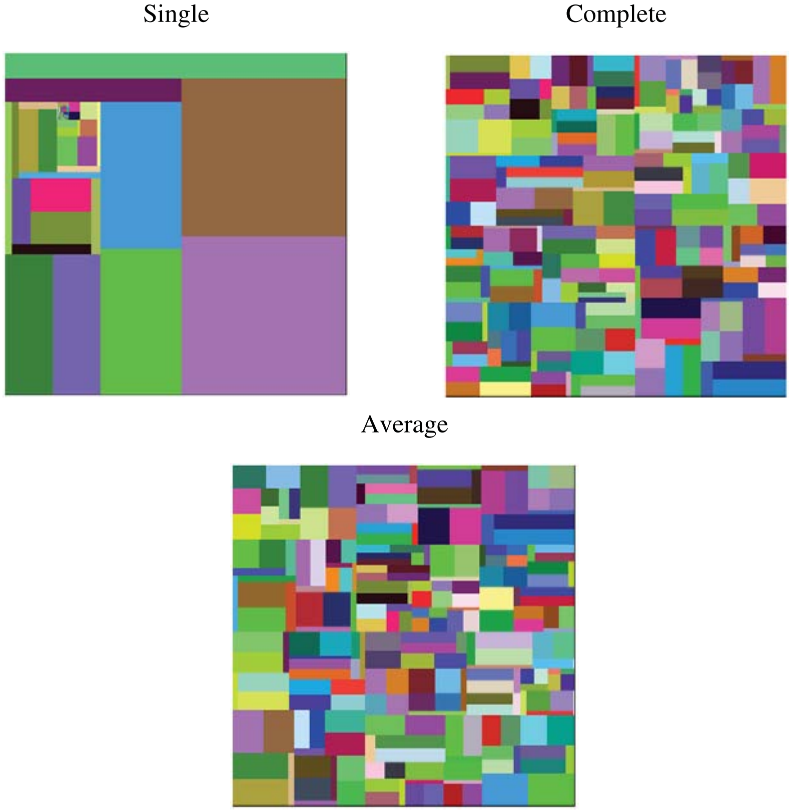


Figure 13.31 *Hierarchical clustering treemaps*

We have already pointed out that it is a mistake to interpret the embedding plane of the 2D treemap as a metric space. Nevertheless, if we pretended it were a partitioning of a Euclidean space, how good is the fit of the distances between the centers of blocks in the treemap to the original distances? For single linkage, the correlation is .14. For complete linkage, the correlation is .33. And for average linkage, it is .39. Not a good fit. A graph-theoretic distance model does much better.

This assessment misses the point of treemaps, however. They are best thought of as rectangular pie charts where the categories are nested within a 2D rectangle. Even though the global arrangement of the tiles is not driven directly by similarity, closely associated categories will nevertheless tend to be near each other.

Figure 13.32 shows another kind of treemap. The data are intrinsically hierarchical. They are the packages, classes, and methods of ViZml.

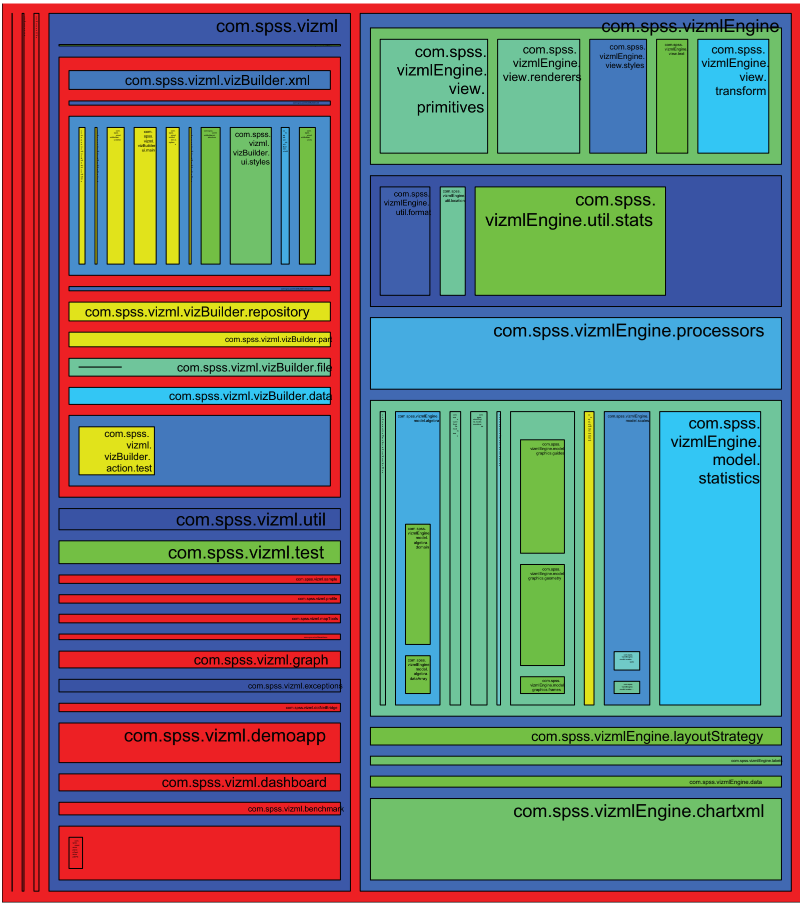


Figure 13.32 *Hierarchical data treemap*

If we think our data are best represented by a metric model, then multidimensional scaling can be used to represent the same information as a treemap. We simply plot the MDS solution and size the points according to the extrinsic variable. For example, if we had a measure of dissimilarity among anemones and if we had an extrinsic variable to represent size of the anemones, then we could use MDS to fit points in a Euclidean space and size them appropriately. The result would look like Figure 13.19. For intrinsically hierarchical data, as in Figure 13.32, treemaps are not a bad way to fit a large amount of information into a relatively small region.

Recursive partitioning trees also partition 2D (or higher-dimensional) space. What is the difference between a treemap and a tiling based on a recursive partitioning tree algorithm?

13.3.4.2 Temple MVV

Mihalisin *et al.* (1991) devised (in a program called TempleMVV) a nested disjoint partitioning that resembles treemaps. Unlike treemaps, however, this display is based on a nested set of metric spaces. Each block is a bounded space that is disjointly partitioned into similar subspaces based on the values of a categorical variable. The simplest incarnation of this idea is a bar graph within a bar graph within a bar graph . . .

Figure 13.33 shows a display of this sort for data representing proportions of survivors of the Titanic sinking, categorized by age, social class, and sex. The data are from Dawson (1995), who discusses various versions of these data and their history. Simonoff (1997) fits a logistic regression model to these data that predicts survival from class, sex, and the interaction of class by sex.

The top-level rectangle (green) includes all the passengers. The next level (blue) represents the proportion surviving within each of the four Class categories (Crew, First Class, Second Class, Third Class). The third level (red) represents the proportion surviving among two Age categories (Child, Adult). And the fourth level (yellow) represents the proportion surviving among two Gender categories (Male, Female). Each of the proportions is computed within the parent level of the hierarchy. Some of the bars (especially the yellow ones) cover their parent bars completely. This means that everyone in that subcategory survived.

Because the partitioning is hierarchical, this display does not provide the marginal information needed for most categorical analytic models: row, column, layer, and subtable margins. For this reason, there is no straightforward link between a nested partitioning and standard categorical techniques such as logistic regression or logit analysis. Mosaic displays, by contrast, were developed with categorical statistical models in mind. See Section 11.3.5.5 for a mosaic display of the same data. The mosaic reveals not only the proportion surviving (the color scale), but also (importantly) the relative sizes of each of the subgroups.

The layout underlying TempleMVV is based on a popular data structure called the **OLAP cube**. As Shoshani (1997) elucidates, the OLAP model is representable by a series of nested tables. The margins of these tables are organized in a **dimension hierarchy**. In graphics algebra, a dimension hierarchy is a nested facet: $(a(b(c)))$, for example, is the same as $c\ /\ b\ /\ a$. Both space and time lend themselves to representation as dimension hierarchies. An example of a spatial hierarchy is (*continent*(*country*(*state*(*county*)))) and an example of a temporal hierarchy is (*year*(*quarter*(*month*(*day*(*hour*(*minute*(*second*))))))). In Figure 13.33, the hierarchy is (*class*(*age*(*gender*))).

The motivation behind OLAP is efficiency. OLAPs can compress large datasets because they aggregate the data inside each subclassification in the dimension hierarchy. The aggregation usually depends on sums, but it can also employ other statistics, such as means or standard deviations, as long as they require only one pass over the data to compute.

The price paid for this efficiency is the loss of flexibility. We cannot examine relations not represented in the hierarchies. Navigation through OLAPs is quite difficult, which is why so many graphical models for representing them have been proposed. And it is no coincidence that programs such as TempleMVV have more controllers than graphics in their displays.

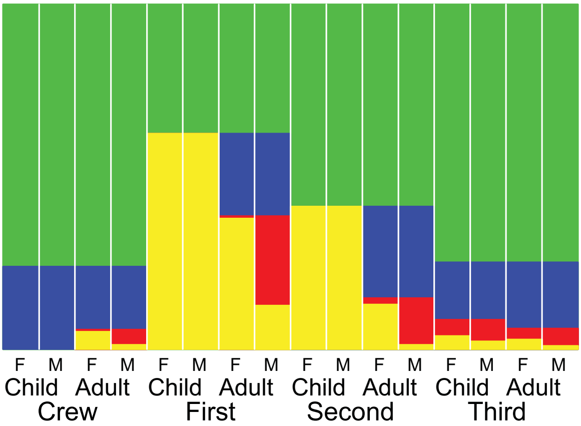


Figure 13.33 *Nested tiling of Titanic data*

What is the dimensionality of a dimension hierarchy? OLAP designers call a dimension hierarchy a single dimension (*e.g.*, *time*). Is the hierarchy (*class*(*age*(*gender*))) a single dimension? The display in Figure 13.33 suggests this by the shared ordering on a horizontal dimension. We would say a multidimensional contingency table of these three variables was three-dimensional, but do we produce a single dimension when we organize them in a hierarchy? One way to approach this question is to see that the nested sequence *a* / *b* / *c* expresses a partial order and the sequence *a***b***c* expresses a product set.

13.3.4.3 Region Trees

Like 3D bar graphs, nested tilings can occlude subregions. For example, the yellow M/F bars cover the red Child bars in Figure 13.33. One solution to this problem is to break dimension hierarchies into panels so we can view each separately. Putting each panel on a common scale allows us to make comparisons among survival rates within subcategory. And coloring each child by the hue of its parent category (with saturation used to distinguish subcategories) makes it easier to navigate the tree. We call this display a **region tree**, because it is a tree of regions (or rectangles in this case). Kleiner and Hartigan (1981), Dirschedl (1991), Lausen *et al.* (1994). Vach (1995), Urbanek and Unwin (2001), and Urbanek (2003) discuss this type of tree.

Figure 13.34 shows a region tree of the Titanic data. Compare this figure with Figure 13.33 and the figures in Section 11.3.5.5. We believe the region tree makes it easier to explore and understand factorial or nested data. As with other graphics, region trees are easily paneled or faceted on conditioning variables. Those looking for more detail may wish to add three vertical scales (running from 0 to 1) on the left side of the display.

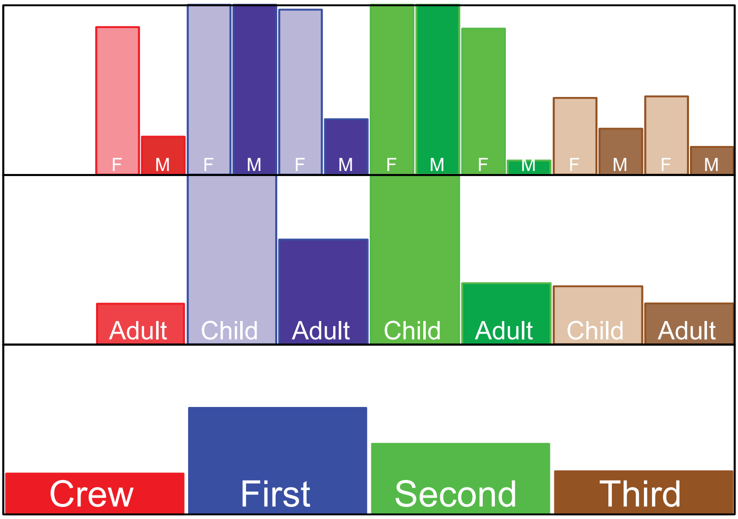


Figure 13.34 *Region tree of Titanic data*

13.4 Sequel

The next chapter covers the use of time in graphics. How do we represent time? How do we do graphics in real-time?

14

Time

The word **time** is closely related to the Latin *tempus*, which means time or season. The related word *tempo* means frequency or rate. Extent in time has long been viewed as similar to extent in space. In fact, the more closely we examine time as a variable, the more it resembles a *state* of space. If we think of cosmic space as a structure containing energy and matter, for example, then time is simply an ordering of different positional states. Past, present, and future are embedded in that ordering. Time is the fourth spatial dimension.

Time has a cultural aspect as well. Most cultures assume the following:

- Time is ordered. We assume one time point precedes another, is simultaneous with the other, or follows. The notion of time being ordered is closely bound to a notion of *causality.* A cause is linked to an effect only if it precedes the effect. Time ordering is also closely bound to cultural concepts of *knowledge*. Recalling events from the past is a routine skill. Predicting events in the future is an extraordinary (usually religious) gift.
- Time is continuous. Between any two times we can find another time. Some physicists consider time as discrete, but most people think of time as a flowing stream.
- Time intervals are additive. We speak of an interval of time without having to identify the location of that interval. When we say something took 24 hours, we do not need to know what day it was.
- Time is cyclical. Some properties of time are defined by natural cycles related to the Earth, Moon, and Sun, including day, month, and year. Many things we measure are directly affected by these natural cycles (such as outdoor temperature), and others are affected by patterns correlated with these cycles (such as airline ticket sales, commitments to mental hospitals, and escapes from prisons).
- Time is independent of location. Einstein's theory of relativity posits that time cannot be measured independently of the location of measurement (and its motion). But in practice, we typically ignore such issues. Time measurement has not always been independent of location. Medieval clocks divided daylight into a set of fixed intervals. These intervals varied in length by season and location, despite having common names.

- Dates are identified by month, day, and year and are synchronized with astronomical cycles by a series of *ad hoc* adjustments. The exact details of these adjustments have varied over time, so that the dates on different calendars and dates on the same calendar at different epochs do not necessarily correspond. For example, when the Gregorian calendar was instituted, the date October 15, 1582 would have corresponded to the date October 5, 1582 in the prior Julian system. Even though we think of time as being continuous, the calendar system in use may change.
- Superimposed across the astronomical periods are periods such as *quarter* (which varies in length so four quarters fit into the year) and *week* (which is fixed in length but overlaps yearly boundaries).
- Time is statutory. If a measurement is scheduled for 2 AM, we consider it to match another measurement scheduled for 2 AM, even if one was actually taken at 1:59:57 and the other at 2:00:01. For statutory purposes, time is treated as a partial rather than a complete order. That is, events within a time window may be treated as contemporaneous. Daylight Saving Time, said to have been invented by Ben Franklin, is a statutory adjustment of time to accommodate seasons.
- Time zones discretize solar time, so that within a range of longitudes all locations have the same time. In our indoor age, it is convenient to refer to the same moment with the same time, but if our movements were limited and the sun were of immediate interest, we might prefer local solar time. Time zones were developed for the railroads, when it became important to reference the same moment across a range of locations.

14.1 Mathematics of Time

Three models predominate in measurement of time: **deterministic**, **stochastic**, and **chaotic**. Deterministic models are based on linear or nonlinear functions of time itself. They usually, but not necessarily, include an error term. Stochastic models are based on recursive functions of time. They always incorporate random error in these functions. Chaotic models have no random component, but like stochastic models, they involve recursive functions.

14.1.1 Deterministic Models of Time

Deterministic time models can involve simple linear functions that are linear in the parameters (as in linear forecast models) or nonlinear in the parameters (as in quadratic models). They can also involve nonlinear functions (as in exponential growth models). And they often involve **periodic** functions. Periodic time series appear in graphics as waveforms, or sinusoidal patterns. We see these patterns when we examine graphics of monthly temperature, rainfall, or migration. We hear them when we listen to music. Because periodic models are peculiar to time series, we will focus on them in this section.

14.1.1.1 Orbits and Vibrations

Orbits and vibrations generate periodic time series. Figure 14.1 shows this graphically. The spokes on the wheel at the left of the top figure demarcate 32 equal-time intervals in a circular orbit of constant speed. From these spokes we derive 17 horizontal grid lines. The vertical gridlines on the right rectangle demarcate the same 32 equal-time intervals. Mapping from intervals equally spaced on the circumference of a circle to intervals equally spaced on a line produces a grid whose intersections fall on a sine wave, as shown in the figure. A cosine wave can be produced by rotating the circle by 90 degrees. This shifts the phase of the sine wave by $\pi/2$ radians.

The bottom figure shows a simple string of fixed mass and elasticity vibrating at a constant frequency. We assume the initial state of the string is represented by the topmost blue curve (not a triangular shape from plucking). And we assume that this is a free string (the vibration is not damped over time by friction or other forces). And we assume that the force responsible for the motion is always directed toward the equilibrium point in the center and is proportional to the displacement. At maximum displacement, the speed is zero and acceleration is maximum; at zero displacement, the speed is maximum and the acceleration zero. Under these assumptions, the red cosine function represents the displacement of the string from the center state at equal time intervals beginning at the top and returning to the top.

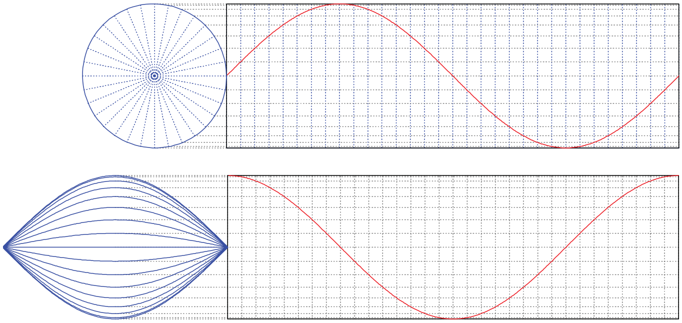


Figure 14.1 *Periodic functions*

14.1.1.2 Harmonics

In the sixth century BCE, Pythagoras observed that the most pleasing combinations of tones are produced by vibrating strings whose lengths are related through integer ratios. Pythagoras believed that these ratios applied to both instruments and natural phenomena, including the music of the spheres. The three consonant Greek intervals had the ratio of 2 / 1 (octave), 3 / 2 (fifth), and 4 / 3 (fourth). Overall, the preferred Greek intervals consisted of all fractions of the form $(n+1)/n$, with the numerator and denominator having only 2, 3, and 5 as prime factors.

Pythagoras' scale works in the range of an octave, but breaks down for larger ranges. For example, concatenating 12 fifths should yield 7 octaves (as happens on a modern keyboard), but $(3/2)^{12}$ is not equal to $(2/1)^7$. Consequently, the tempered scale used in modern Western music replaced the 16/15 diatonic semitone in the Greek scale with the interval $\sqrt[12]{2}$, which is just a few thousandths less in magnitude. Since scales are multiplicative in frequency (or additive in log frequency), concatenating 12 tempered semitones yields the ratio 2 / 1 on the tempered scale. Concatenations of other intervals on the tempered scale are transitive as well, so the instruments tuned to the tempered scale can transpose music without retuning. Recent musical theorists have investigated intervals generated by ratios of other small prime numbers. Playing this **microtonic** music requires a computer or a carefully tuned fretless stringed instrument. Papadopoulos (2002) discusses the mathematics of musical scales in more detail.

When music is played on several instruments, or several notes are played simultaneously on a single instrument, the waveforms blend. And, as Pythagoras determined, if the ratios of frequencies involve prime integers, they will sound pleasing. Moreover, most instruments (strings and winds) have pleasing tones because they produce **overtones** when they play a single note. Strings, for example, vibrate at several frequencies simultaneously depending on how they are plucked, hit or rubbed. The combination of these waves is what gives instruments their timbre or character. Each frequency component produces its own audible pitch called a **partial**. Figure 14.2 shows four standing waves whose frequencies follow a ratio of 2 to 1: red is twice black, green is twice red and blue is twice green. The partials of a string vibrating like this would be audible as four different octaves. The partials would be at lower volume than the black **fundamental**, but would be audible nonetheless. Partials usually occur at all integer multiples of a fundamental frequency — not just at doublings. On a modern piano, for example, hitting the lowest C key (C_1) produces a series of overtones that are integer multiples of the fundamental frequency. The single string produces the frequencies (in cycles per second, starting at the fundamental): 33, 66, 99, 132, 165, 198, ... These frequencies correspond to the notes C_1, C_2, G_2, C_3, E_3, G_3, ... A professional musician can hear at least the first several of these overtones. You can hear them more clearly by holding down several of those notes while striking C_1. They will resonate sympathetically. Hold down nearby notes and you will not hear them resonate.

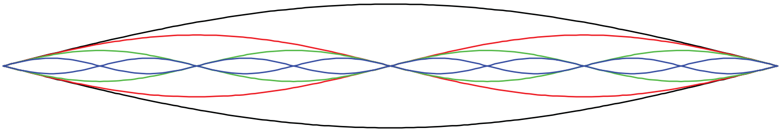


Figure 14.2 *Octave harmonics*

14.1.1.3 The Fourier Transform

Given a blended waveform produced by several simultaneous frequencies (like that from C_1 on the piano), can we recover those frequencies in separate waveforms? It had long been known that sinusoidal waveforms could be combined to produce smooth periodic functions, but the French mathematician Joseph Fourier surprised the French Academy in 1807 by claiming that sinusoidal waveforms could be combined to produce *any* periodic function — smooth or rough. Fourier's series consists of weighted sinusoidal functions at integer multiples of frequencies. If we take discrete measurements over time and if time periods are equally spaced (t, $2t$, $3t$, ...), then Fourier's function has a simple form. The discrete Fourier series is

$$F(t) = \frac{\beta_0}{\sqrt{2}} + \beta_1 \sin(t) + \beta_2 \cos(t) + \beta_3 \sin(2t) + \beta_4 \cos(2t) + \ldots$$

Figure 14.3 shows an example in the interval 0 to 2π, where $\beta_0 = 0$ and $\beta_{i>0} = 1$.

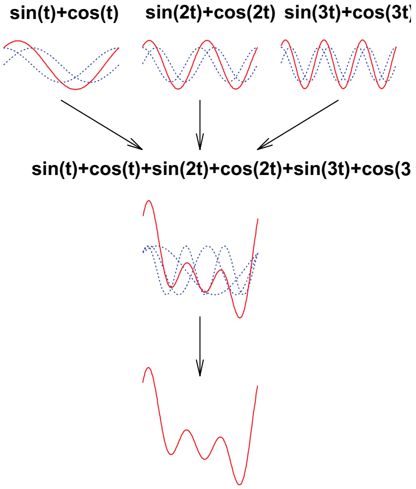


Figure 14.3 *Fourier components*

The discrete Fourier function can also be parameterized to map from the complex plane to the complex plane. If we use Euler's formula

$$e^{i\theta} = \cos\theta + i\sin\theta$$

then the discrete Fourier function becomes

$$F(t) = \sum_{t=0}^{n-1} z_t e^{-i\omega t}$$

In this form, ω represents frequency and z_t is a complex number whose real component represents a point in a data series and whose imaginary component is zero. Each point in the image on the complex plane represents a pair of coefficients for a frequency component, corresponding to the sine and cosine weights for that frequency.

14.1.1.4 The Periodogram

We call the quantity $m = \sqrt{R^2 + I^2}$, where R is the real (sine) coefficient and I is the imaginary (cosine) coefficient, **magnitude**, This quantity represents the relative contribution of each frequency component to the overall function. The plot of squared magnitude against frequency is called the **periodogram** or (if based on the theoretical model) the **spectrum** or **spectral density**. Figure 14.4 shows the periodogram for the composite waveform in Figure 14.3. Notice that only three frequencies have nonzero magnitudes.

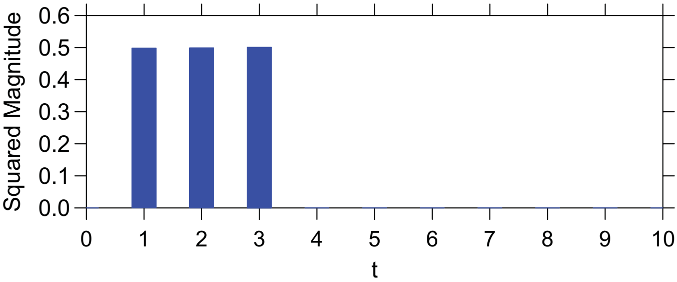


Figure 14.4 *Periodogram for series in Figure 14.3*

14.1.1.5 Smoothers, Convolutions, and Filters

The usefulness of the Fourier decomposition as a framework for smoothers is evident in applications that are fit well by only a few terms in the series. We show an example in Figure 14.16. The Fourier decomposition also provides a framework for applying the kernel smoothers discussed in Section 7.1.4. Kernel smoothing is often defined as a **convolution** of functions:

$$g \otimes h = \int_{-\infty}^{\infty} g(x)h(x-t)\, dt$$

In this formula, g is the function (or data density) to be convolved and h is a kernel. A convolution is the product of these two functions integrated over t (in this case, time). The kernel moves along x and at each point t on x, it operates on values of g wherever h overlaps g. Figure 14.5 shows a convolution of a discrete function g (represented by red spikes) and an Epanechnikov kernel h (represented by blue curves). To keep the display simple, we have drawn the kernels only at the locations of the spikes.

The convolution is the green function. At the right end of this function, the kernel functions overlap exactly with the convolution because they cover only one spike in that neighborhood. Notice that the convolution goes to zero between these last two spikes because the moving kernel covers neither spike in that interval.

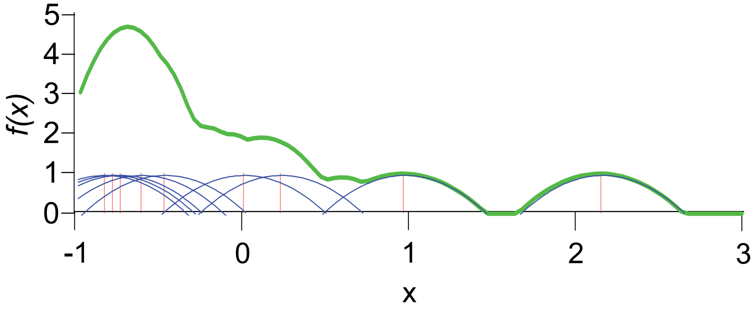


Figure 14.5 *Convolution*

The Fourier decomposition is widely used for developing kernels to smooth time series and spatial images because of a dualism inherent in the **Fourier theorem**. This theorem states that

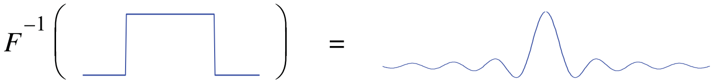
$$g \otimes h = F^{-1}(F(g) \otimes F(h)) ,$$

where F is the Fourier function and F^{-1} is its inverse. Thus, we may kernel smooth in the time domain or smooth in the frequency domain and inverse-transform back to the time domain. We will illustrate this method in Section 14.3.1.2.

The Fourier theorem is used by engineers to develop digital linear filters. A typical low-pass filter, for example, has a frequency magnitude response of 1 from 0 to 50 Hertz (cycles per second) and zero elsewhere. The inverse Fourier transform of this rectangular function is the **sine cardinal** function:

$$\operatorname{sinc}(x) = \begin{cases} 1 \quad , & x = 0 \\ \dfrac{\sin(x)}{x}, & otherwise \end{cases}$$

In pictorial form,



Thus, we may filter frequency components in the frequency domain and inverse transform the result, or we may use the *sinc* function to smooth in the time domain directly. If we use a truncated *sinc* function (as shown here), then the results will differ to a small extent (because the *sinc* function is wavy everywhere on its infinite domain). Many kernels, such as the Epanechnikov (see Section 7.1.4), are shaped similarly enough to the *sinc* function that they produce similar results as well.

14.1.2 Stochastic Models of Time

As we have seen, deterministic time models take the form $x = f(t) + \varepsilon$, where the ε_i are independent, identically distributed random errors. Examples of these models are the linear regression model $x = \beta_0 + \beta_1 t + \varepsilon$, or the sinusoidal regression model $x = \beta_0 + \beta_1 \sin(\alpha + \omega t) + \varepsilon$, where α is a phase parameter and ω is a frequency parameter. We fit these models by ordinary least squares or maximum likelihood.

Independence of errors rarely happens in time series data, however. Time series data are usually serially correlated — measurements at one time period are correlated with measurements at nearby time periods. And if we try to use deterministic models on time series with serially correlated errors, our fits will be biased. Furthermore, we may end up fitting models to aspects of the data that are simply random. Time series data can fool us into thinking there is order where there is none.

Stochastic time models take the form of the recursive equation $x_t = f(x_{t-d}) + \varepsilon_t$, or $x_t = f(\varepsilon_{t-d}) + \varepsilon_t$ where d is a time delay (lag). The word *stochastic* comes from the Greek *στοχαστικός*, which means able to hit a mark or to guess. The Greek sense of target practice — hitting something with an arrow — makes sense in the modern usage, since stochastic equations are especially suited to describing such things as the track of machine gun bullets across a wall.

Figure 14.6 presents several illustrative stochastic series. The first and simplest is **white noise**, shown in the top row. This is generated by the equation $x_t = \varepsilon_t$, where the errors are independent Gaussians. White noise sounds like the hissing noise between stations on the radio or the waterfall sound used in relaxation audiotapes. The periodogram for this series shows a fairly uniform density, albeit with considerable variability in the magnitude of frequencies. The theoretical spectral density for white noise is uniform. The **autocorrelation function** plot (ACF) in the middle column of the white noise row is also uniform. This plot shows the correlations between the series itself and the same series shifted to the right by one time period, by two, and so on. The red bands in this plot are approximate confidence intervals on the correlations. For white noise, all the autocorrelations are near zero. Since everything in this chapter seems to be related to everything else in the chapter through the Fourier transformation, we note that the periodogram is the Fourier transform of the ACF.

The next three rows show different forms of noise whose names are based on a pun. **Brown noise** (third row) is named after Brownian motion, the random jittering of particles first observed in the early 19th century by the Scottish botanist Robert Brown. This series is a 2D **random walk**, in which the location of a point is given by the location of the previous point plus random error. **Pink noise** is so named because it lies between white and (reddish) brown. **Black noise** is named for being darker than brown. The spectral densities of these series follow a simple functional progression. The spectrum of pink noise is proportional to $1/f$, where f stands for frequency. The spectrum of brown noise is proportional to $1/f^2$. And the spectrum of black noise is proportional to $1/f^p$, where $p > 2$. We see increasing **coherence** (smoothness) in these series, mirrored by increasing dependence in the ACF plot.

One person's noise is another's music (depending on one's age). If we use the vertical axis in a time series to represent pitch rather than amplitude, then interesting patterns appear in melodic music. Voss and Clarke (1978) found a $1/f$ spectrum (pink noise) in a variety of music. Brillinger and Irizarry (1998) found $1/f$ in samples of Baroque, Classical, Romantic, Atonal, Spanish Guitar, Jazz, Latin, Rock & Roll, and Hip Hop music. You can listen to Irizarry's random series generated to produce white, pink, and brown music at his Web site (Google his name).

The next series in Figure 14.6 was generated by the stochastic equation $x_t = \phi x_{t-1} + \varepsilon_t$, where $\phi = 0.95$ and ε represents standard-normal random error. (In these recursive equations we follow the convention that x is measured at unitary time points.) This series is a first-order **autoregressive**, or **AR(1)** process. Do you notice the similarity between this series and brown noise? That's not coincidental. Brownian motion is a first-order autoregressive process, but $\phi = 1$. Unlike the AR(1), a random walk diverges infinitely.

The next to the last row of Figure 14.6 contains a first-order **moving average** model, or **MA(1)**, where $x_t = \theta\varepsilon_{t-1} + \varepsilon_t$ and $\theta = 0.8$. Notice the large first spike in the ACF plot. This reflects the single MA coefficient. The last row contains the same model with $\theta = -0.8$. The spike in the ACF plot is negative. Notice the difference between the two periodograms. The spectral densities are almost complementary. The positive coefficient enhances the low frequencies (more coherence in the series). The negative coefficient enhances high frequencies.

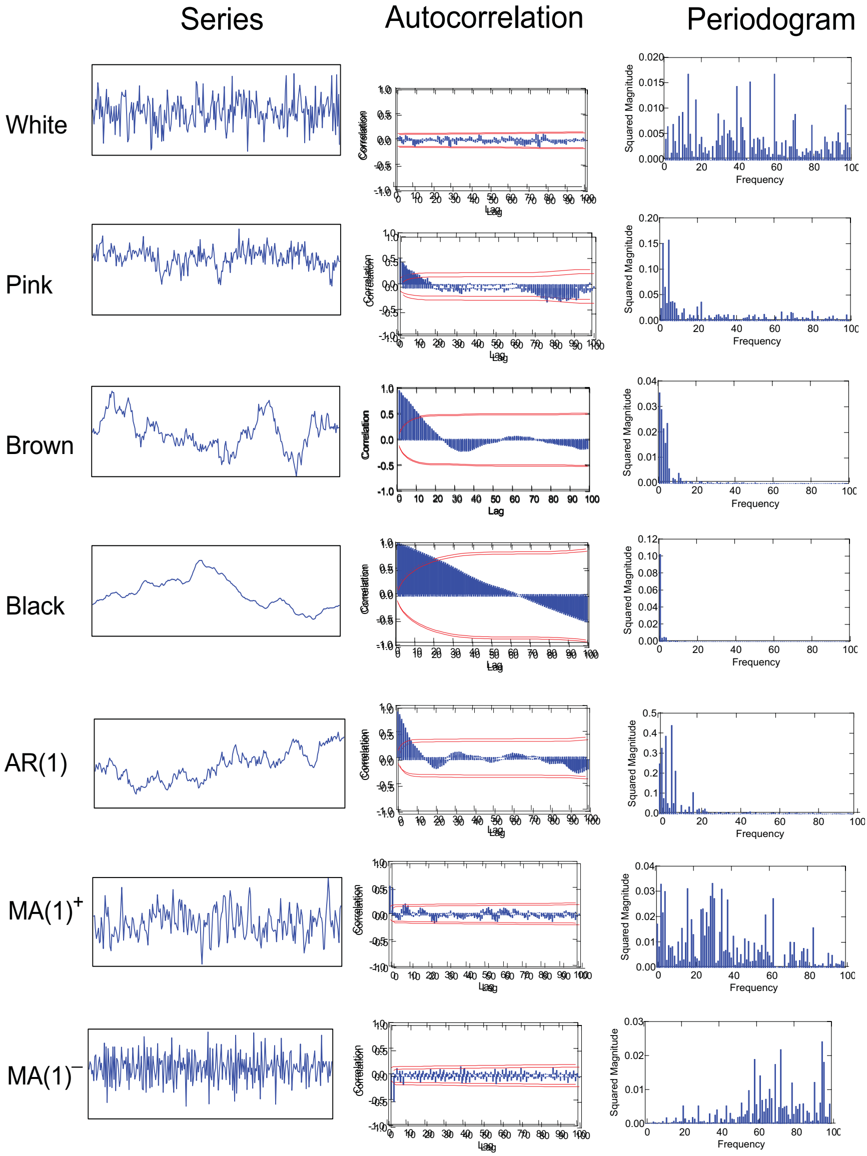


Figure 14.6 Stochastic series

There are many other stochastic models, including the **seasonal autoregressive** model, where $x_t = \phi x_{t-s} + \varepsilon_t$ (s is a seasonal period), and the **seasonal moving average** model, where $x_t = \theta \varepsilon_{t-s} + \varepsilon_t$. We will present a seasonal model on real data in Section 14.3.1.3. There are also compositions of autoregressive and moving average models. Box and Jenkins (1976) discuss these models in more detail. All of these stochastic models share the essential component of correlated errors across time. They differ in the structure of those errors, which is to say they differ in the way errors are incorporated in the model. And without visual tools such as the ACF plot and the periodogram, it is often difficult to distinguish them by looking at the raw series

14.1.2.1 Stationarity

The left panel of Figure 14.7 shows a scatterplot matrix of a random walk series and the first nine lags (shifts) of the series. The first row/column of this matrix contains the scatterplots corresponding to the first 10 autocorrelations in an ACF plot. There is a discernible pattern in the SPLOM; as we move away from the diagonal, the correlations decrease uniformly. We discuss this pattern, typical of autoregressive processes, in Section 16.5.2.1.

The correlations between adjacent points in the series are an obvious consequence of the generating model. Why are the correlations between *non-adjacent* points nonzero? Because the lags are related by a sequence of backward shifts in the model. If we remove the first-order dependency by differencing the series once, however, the resulting series is white noise. The right panel of Figure 14.7 shows the result.

Our differencing has also made our series **stationary**. Stationarity means (weakly, not strictly) that the mean, variance and autocorrelation structure of a process are defined and do not change over time. A stationary process eventually drifts back toward its mean (and variance) from extreme values. In the case of a random walk, the series does not drift back to some mean value. It keeps wandering farther afield. You don't want to bet against a random walk. Unless you have infinite reserves, it will eventually lead to your ruin.

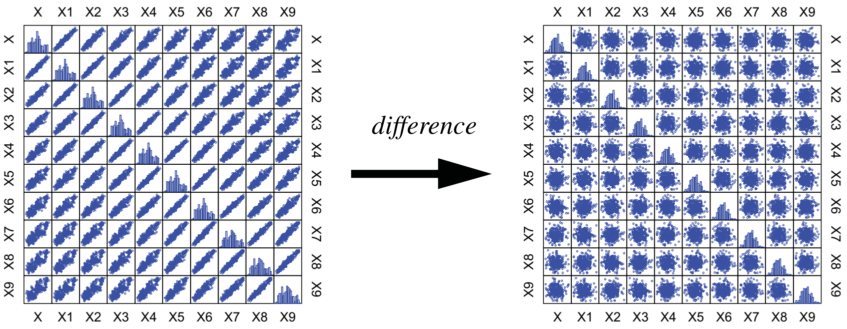


Figure 14.7 *SPLOM of random walk series (9 lags) and differenced series*

14.1.3 Chaotic Models of Time

Chaotic models of time are related to *deterministic* models of time in the sense that both incorporate deterministic equations. Chaotic models of time are related to *stochastic* models of time in the sense that both are based on recursive equations. Chaotic models do not have a random component, however. The apparently random behavior of series produced by chaotic models is a consequence of the behavior of certain nonlinear equations under iteration.

Chaotic models are instances of **dynamical systems**. A dynamical system models a set of ordered states. The evolution of states over the ordering corresponds to a trajectory in a space of all possible states of that system. Consider the following three-parameter recursive system (originally due to Lorenz, 1963), where $\phi_1 = 10$, $\phi_2 = 28$, and $\phi_3 = 8/3$:

$$\begin{aligned} x_t &= x_{t-1} + \phi_1(y_{t-1} - x_{t-1}) \\ y_t &= y_{t-1} + \phi_2 x_{t-1} - y_{t-1} - x_{t-1} z_{t-1} \\ z_t &= z_{t-1} + x_{t-1} y_{t-1} - \phi_3 z_{t-1} \end{aligned}$$

Figure 14.8 shows two different 3D views of 10,000 points generated by this system, plotted in what is called a **phase space**. A phase space contains all possible states of a dynamical system. Each point in this plot corresponds to a value of *x*, *y*, and *z* at a given point in time. Each time point is colored to run from blue (early) to red (late). Following Lorenz, we have rescaled *t* in the legend to run from 0 to 50 instead of 0 to 10,000.

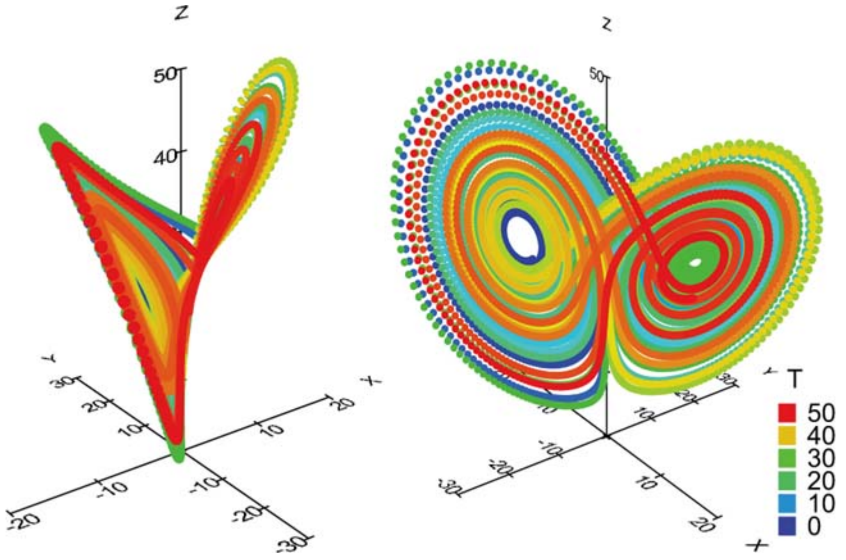


Figure 14.8 *Lorenz butterfly*

The striking feature of this configuration is its bifurcation into two **orbits**. This configuration of orbits is called a **strange attractor**. The term suggests that the orbits are attracted to (lie on) a manifold that is embedded in the phase space. For the Lorenz attractor, this manifold has a distinctive structure shaped like a butterfly. What does the shape imply? It indicates that the Lorenz system generates two types of states and, if we examine the graphic in more detail, we see that the system jumps between these two states at apparently random points in time. The wings of the butterfly contain a mixture of colors.

Not every deterministic system based on recursive equations generates strange attractors. Consider the following recursive system:

$$
\begin{aligned}
x_t &= x_{t-1} + \left|171(x_{t-1}\%177) - \ 2(x_{t-1}/177)\right| \\
y_t &= y_{t-1} + \left|172(y_{t-1}\%176) - 35(y_{t-1}/176)\right| \\
z_t &= z_{t-1} + \left|170(z_{t-1}\%178) - 63(z_{t-1}/178)\right|
\end{aligned}
$$

The percentage sign (%) represents a modulo (remainder) operator. As with the Lorenz equations, there is no random component in this system. Paradoxically, however, this system is frequently used to generate random numbers. More precisely, this is the generating system that underlies a **mixed-triple multiplicative linear congruential pseudorandom number generator** (Wichman and Hill, 1982; L'Ecuyer, 1988). Despite its deterministic origins, it passes the standard statistical tests for randomness.

Figure 14.9 shows 10,000 points generated by this system. Compare this configuration of points filling the interior of a cube with the Lorenz butterfly in Figure 14.8. No coherent attractor is visible.

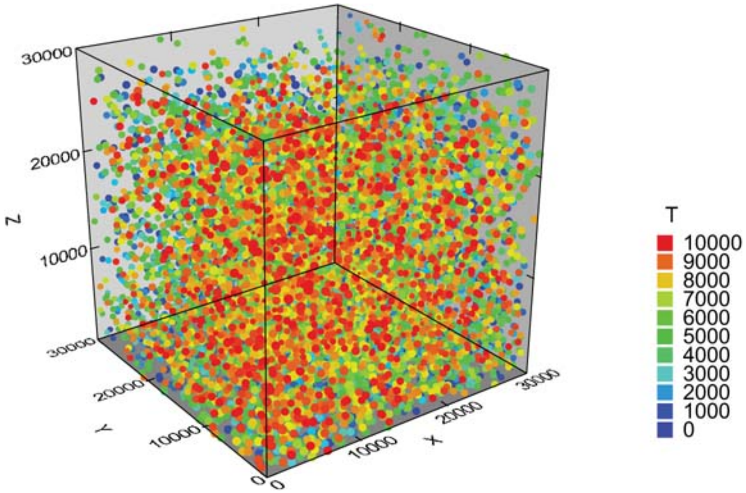


Figure 14.9 *Plot of pseudorandom system*

Let’s examine the behavior of the Lorenz system over time in more detail. In Figure 14.10 we plot x against t. Suppose we are given this plot and do not know the generating model. Is there a way to discern whether this series is based on a chaotic system rather than a stochastic or deterministic system with a random component?

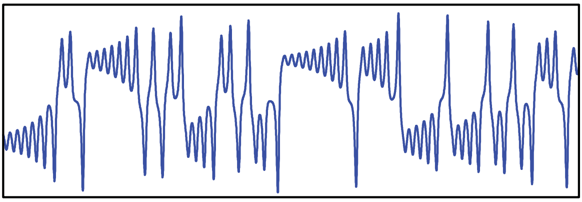


Figure 14.10 *Lorenz series (x component over time)*

Obviously, we cannot inspect the series itself to determine that it is generated by a three-parameter system or to know that its attractor is two-dimensional. And despite the local regularities (funnel-shaped envelope surrounding a periodic oscillation over time), we can’t tell if the global behavior is random or chaotic, As with stochastic models, a visual inspection of a raw time series reveals little about its generating model.

The standard time series graphical tools we have examined so far will not be of much use to us either. Figure 14.11 shows both the ACF and periodogram for this series. The ACF is smooth, but so is the ACF for black noise. And the spectrum closely resembles a spectrum for 1/*f* noise. The chaotic behavior is not evident in these plots.

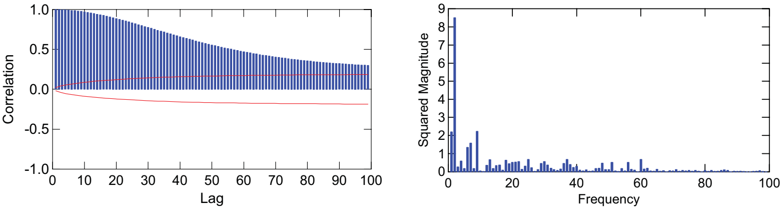


Figure 14.11 *Lorenz butterfly series autocorrelation plot and periodogram*

There *is* a phase space derivable from a time series that is remarkably useful for this purpose, however. We will illustrate this with a simple example. Consider the series at the top of Figure 14.12. It consists of three subseries: $\sin(x)$, white noise, and a squarish wave produced by the function $\operatorname{atan}(\cos(20x))$. The graphic at the bottom of Figure 14.13 plots dx/dt against x. To construct the graph, we numerically difference the original series by computing $(x_t - x_{t-1})$ and then plot this differenced series against x.

This **phase plane plot** picks up the two types of subseries in the series at the top of the figure. The points on the sine wave subseries appear on the narrow green ellipse at the center. The points on the square wave appear on the red rectangular trajectory. The other points are distributed randomly throughout the plot.

What makes this phase plane plot useful is that trajectories in phase space that represent similar subseries in the time domain are located near each other regardless of how far the subseries are separated in time *and* regardless of whether the subseries are in phase.

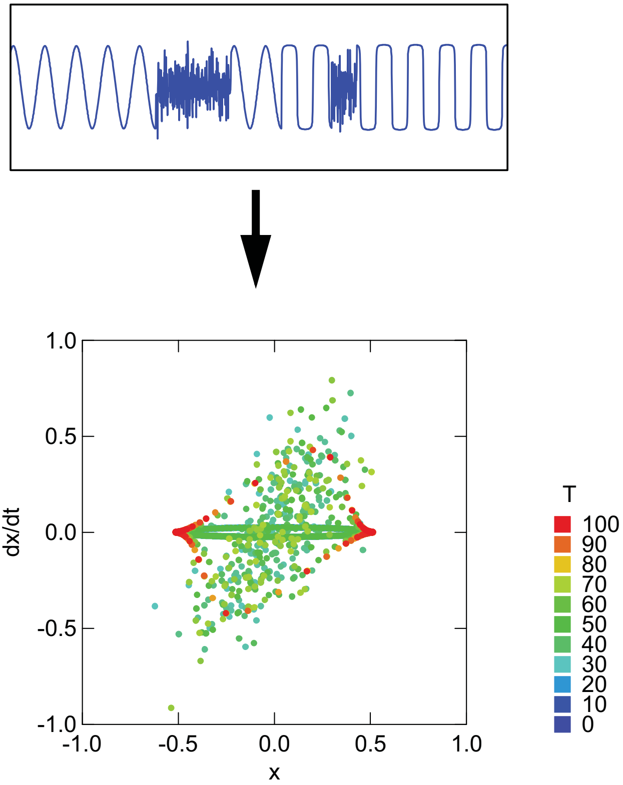


Figure 14.12 *Phase plane plot of composite series*

How does this particular phase plane plot work for the Lorenz data? Figure 14.13 contains a phase plane plot for that system. The essential features of the butterfly have been captured. This plot looks much like the projection shown in the right plot in Figure 14.8.

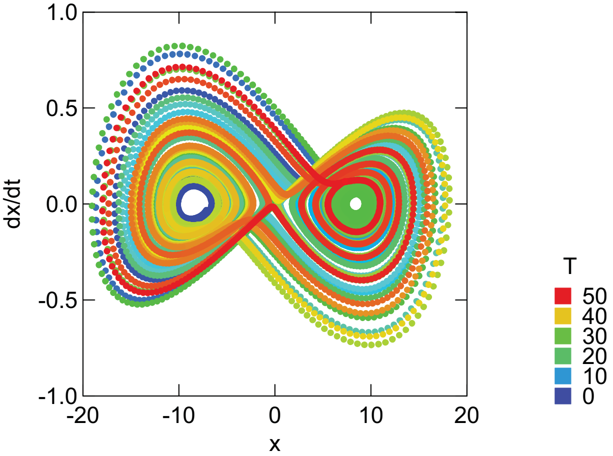


***Figure 14.13** Phase plane plot of Lorenz system*

The phase plane plot reveals attractors if they are low-dimensional and if they involve primarily first-differences. For higher-dimensional attractors, we can use **Poincare sections** to examine slices of the attractors. A Poincare section is a (usually 2D) slice of phase space that captures one state (or a small neighborhood of one state) of a trajectory. We can put several sections together to infer the topology of an attractor. We will not examine these plots in more detail, partly because we want to include a particularly interesting multidimensional plot that can be used to distinguish chaotic from random behavior.

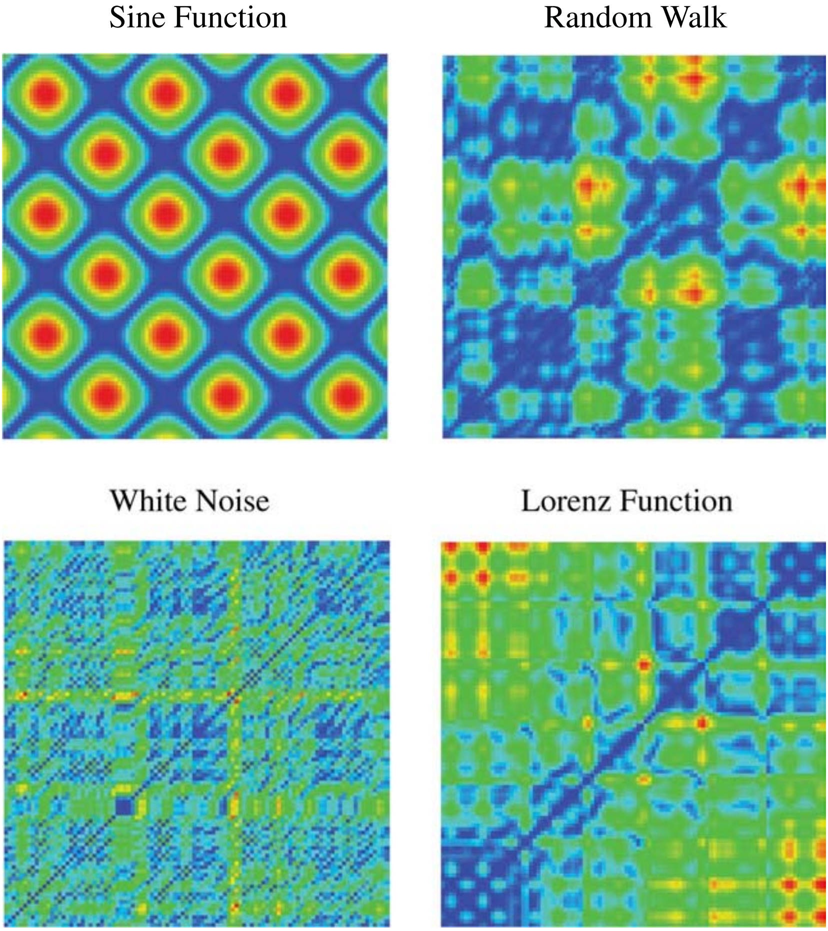
The **recurrence plot** displays repeating patterns at multiple lags. The simplest form of this plot is to graph $|f(t_1) - f(t_2)|$ in the $t_1 \times t_2$ plane. The upper left corner of Figure 14.14 shows this plot for the sine function. We graph $\sin(t_1)$ against $\sin(t_2)$ for $-10 < t_1, t_2 < 10$. The periodicity of the sine function is evident in the repeating pattern.

For diagnosing chaos, we generalize this plot (Eckmann *et al*, 1987; Casdagli, 1997; Gao and Cai, 2000). For each time point in a given dataset, we compute the vector

$$x_t^m = (x_t, x_{t+d}, x_{t+2d}, \ldots, x_{t+(m-1)d})$$

where m is an embedding dimension and d is a time delay. Then we compute a matrix of all possible distances between these vectors. The distances are used to color the plot. We use city-block distances and $d = 1$ and $m = 3$ for the examples in Figure 14.14. The plots are relatively resilient to the choice of distance metrics and d and m, although the above references offer guidelines for good choices (similar to the choice of kernels in smoothing).

The result is a symmetric plot (around a diagonal running from southwest to northeast) whose colors represent the distance between subsections of the series. In a phase plane plot, times in the neighborhood of similar subseries will overlap. In a recurrence plot, times in the neighborhood of similar subseries will have the same color denoting small distances (blue in the examples in Figure 14.14).


Sine Function

Random Walk

White Noise

Lorenz Function

Figure 14.14 *Recurrence plots*

The lower-left plot in the figure shows what we can expect for white noise. There is a faint plaid pattern to the colors, but it is not systematic. The upper-right plot shows a recurrence plot for a random walk. Why is this plot patterned if the series is random? Remember, the random walk has large autocorrelations and is not stationary (see the Brown series in Figure 14.6). Nearby values are likely to be similar and relatively distant subseries are likely to have different mean values. This leads to smaller distances (blue values) near the diagonal and larger distances (redder values) elsewhere in the plot (see also Figure 14.7). This condition should alert us to make sure a series is stationary before doing a recurrence plot. Otherwise random behavior will look systematic.

Finally, the recurrence plot for the Lorenz series in the lower-right corner shows distinctive patterning. How do we infer the shape of the attractor from the pattern in the recurrence plot? This problem is similar to that of inferring the shape of high-dimensional densities from parallel coordinates plots (see Wegman, 1990). More research needs to be done to make 2D multivariate plots useful for discerning topology. Nevertheless, recurrence plots can be helpful in distinguishing chaotic from random behavior.

14.2 Psychology of Time

The psychological literature on time is vast, long-standing, and diverse. Consequently, we will present only a few theories and findings that apply to the development and viewing of time graphics. First, we will ask what it means to sense time. Second, we will discuss aspects of the perception of duration and production of estimates of duration. Third, we will discuss the perception of motion. For a recent review of psychological research on time, see Grondin (2001).

14.2.1 Sensation

Although we speak of sensing time, we do not perceive it the same way we do other stimuli such as light or temperature. Identifying a time stimulus is problematic, since time is a dimension rather than an aspect of matter or energy. Furthermore, perceiving temporal relations requires us to employ memory, so a sense of time cannot be said to be immediate. For example, we can perceive binary spatial relations (left, right, above, below) by attending to two points in space at one point in time, but we cannot perceive binary temporal relations without storing events in memory and comparing them. Even our experience of the present is derived from memory, since the psychological present is a window on a continuous stream of sensations across time. In light of these problems, psychologists have taken a variety of approaches to temporal sensation. Two contrasting models illustrate the diversity of these approaches.

On the one hand, despite common sense, time may not be sensed at all. Instead, perceived time might be viewed as constructed logically from short-term and long-term remembered events in space-time (Gibson, 1975). We summarized Gibson's realist perspective with regard to other stimuli in Section 10.4. If, as Gibson argues, the perception of duration and motion can be constructed from perceived sequences of events (states of space), it may not be necessary to invoke a sixth sense to account for time perception.

On the other hand, perceived time can be regarded as a response to an internal clock or pacemaker that might be modified or reset by other stimuli such as light or temperature (Killeen and Fetterman, 1988; Triesman, 1993). Researchers have found evidence of a neurological clock in both human and animal experiments. The period of such a clock is debated; there may be several different periods in a single clock or even multiple clocks. A clock with a daily period is called **circadian**, from the Latin *circa* (around) and *dies* (day). The synchronization of internal clocks with external or ecological stimuli is called **entrainment**. There is a lot of folklore and pseudo-science concerning entrainment (especially in popular psychotherapies), but the phenomenon has been widely observed.

Neither approach implies that the sensation of time is a uniquely human attribute. Animal studies investigating phenomena such as migration, rhythmic behaviors, and predation all show responsiveness to time. Nevertheless, there is considerable experimental support for the idea that animals are stuck in time (Roberts, 2002). Despite their ability to learn and react to different schedules of reinforcement and periodic stimuli, animals do not appear to be able to anticipate long-term future events or to remember sequences of events in the past.

If we are to use changes in visual or acoustic signals over time to represent variation in data, we must ask how duration is perceived and how we interpret motion through space. The following sections summarize these issues.

14.2.2 Duration

The energy of a stimulus is the product of its duration and intensity. Two stimuli of equal energy hitting the same sensor will be perceived as different if their durations are different or if their intensities are different. There is a **critical duration**, however, within which this is not true. If two focal stimuli are separated by a time interval less than a critical duration, then they will be perceived as the same if their energy is the same. From this perspective, the critical duration can be thought of as the shortest perceivable duration. The critical duration is the temporal equivalent of the **just noticeable difference** (JND) between simultaneously presented stimuli.

For vision, the best theoretical and empirical estimate of critical duration appears to be around 34 milliseconds (Reeves, 1996). Two equal-energy stimuli presented to the same retinal location within this time interval cannot be distinguished. For hearing, the best estimate is approximately 100 milliseconds for spatially separated stimuli (Andreeva and Vartanyan, 2004).

Not surprisingly, the 24 frames-per-second Hollywood movie frame rate is within the same neighborhood as the critical duration for foveal vision. This is not to say that perceivers could not detect frame-to-frame changes in a movie that ran at a faster frame rate. Keep in mind that the visual critical duration refers to stimuli at the same foveal location. Changes over different areas of the visual field will be detected at higher frame rates. To adjust for this, commercial movies enhance the perception of smooth motion by blurring parts of scenes that are changing rapidly. Without this blurring, we would perceive a **strobe effect** (as if a strobe light were illuminating the scene). This use of motion blur is a form of **temporal anti-aliasing**.

Durations less than a second are processed differently than those of more than a second, for both vision and hearing (Grimm *et al.,* 2004). Judgment of shorter durations is pre-attentive, or at least pre-cognitive. Judgment of longer durations involves working memory and higher cognitive processes. For short intervals, the Stevens exponent for duration is approximately 1.1 (see Figure 10.2); this makes the perceptual curve for duration concave, while that for area, volume, brightness, and loudness is convex. For longer intervals, the results are mixed (Grondin, 2001) and for extended intervals, social and cultural effects influence duration judgments (Flaherty, 1999). And, not surprisingly, there are temporal illusions that functionally resemble many of the well-known visual illusions (Hellström, 1985).

Given the complexities and contextual dependencies of these results, there is nevertheless experimental evidence that temporal sonifications and visualizations can result in judgments that are no less reliable than those based on static visual stimuli such as areas or lengths of lines (Walker, 2002). At the least, sonifications and animations can be used to supplement static graphics and reinforce pattern detection; at the most, they can potentially reveal patterns not evident in static displays.

14.2.3 Motion

Statistical graphics usually involve more than one dimension. Temporal statistical graphics map data dimensions to location, pitch, tempo, and other modalities. Changes in these modalities over time produce what we call the sensation of motion. That sensation can be produced across both visual and auditory fields.

The Gestalt principles we introduced in Chapter 10 — proximity, similarity, good continuation, closure — apply to the perception of motion as well as to shapes in static space. Deutsch (1996) summarizes their role in audition. The serial aspect of temporal sensation, however, has certain consequences. For example, we recognize rotated, reflected, and translated objects in static visual presentations. We do not generally recognize reflected temporal patterns, however. Have you ever heard a tune played backwards? It is unlikely you recognized it. We *do* recognize transposed (in pitch) musical tunes and we recognize the same tunes when moderately dilated or compressed. Thus, we cannot expect sonifications or animations to represent coherent patterns in the same ways that static visual representations do.

14.3 Graphing Time

We will discuss three types of time graphics in this section: **static graphics**, **dynamic graphics**, and **real-time graphics**. Static graphics, by definition, include all the figures in this book. None of the graphics in this book move. They represent either an instant in time, or (through the use of aesthetics such as position or color) an interval of time. Dynamic graphics use motion to represent time or another variable. Each time instant is represented by a frame; graphics are displayed by playing frames like a movie. We may play a hundred years of a country's economic productivity in five seconds or animate a DNA sequence in a minute. Real-time graphics, sometimes called **streaming graphics**, are visually indistinguishable from dynamic graphics, but they include the present as well as the past. We play one second as one second, one minute as one minute. Structurally, real-time graphics involve peculiar computing issues that we will discuss in a separate subsection.

14.3.1 Static Graphics

Time series are usually plotted with point, bar, line or area elements. In this section, we will present several examples in which the choice of element is guided by the model of the process that generated the series.

14.3.1.1 Fourier Decomposition

We begin with the data from Figure 3.5. These data consist of instantaneous firing rate measurements of a cat retinal ganglion cell in a 7-second interval. We noted in Chapter 3 that there is a low-frequency component in the series due to respiration. Oxygen uptake causes the firing rate to increase. Figure 14.15 shows the periodogram for this series. We see two conspicuous spikes. The first is around a half-cycle per second and the second is around two cycles per second. The half-cycle spike corresponds to the respiration component.

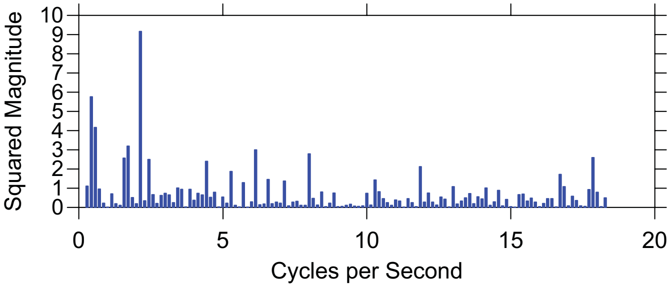


Figure 14.15 *Cat respiration periodogram*

To display the respiration component in data space, we compute a Fourier transform of the series and smooth the spectrum by retaining the first four Fourier components. Figure 14.16 shows the inverse Fourier transform of the first four components displayed as a red line. The respiration cycle is clearly discernable.

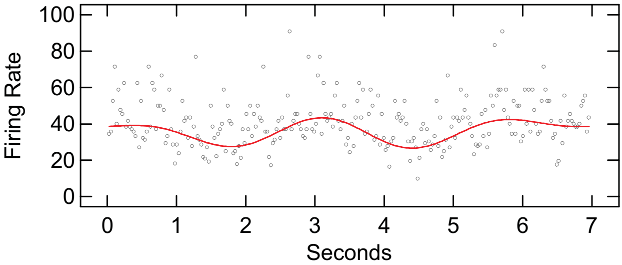


Figure 14.16 *Cat respiration revealed in neural firing rate by smoother based on first four Fourier components*

The respiration component is an artifact from the point of view of the study itself (Levine *et al*., 1987). Of focal interest was a higher-frequency component involving the light stimulus. To investigate this response, we rerun the analysis and retain the first 15 Fourier components. Figure 14.17 shows the result. A regular series of cycles, approximately two per second, is visible on top of the fundamental respiration component.

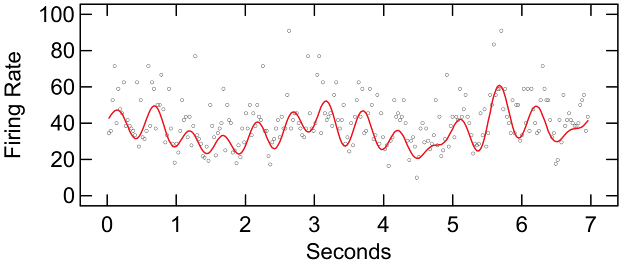


Figure 14.17 *Multi-component Fourier smooth*

Figure 14.18 shows an ACF plot of the residuals from this Fourier smooth. The red lines are approximate 95 percent confidence intervals on the autocorrelations. The residuals show no significant autocorrelations remaining after the smooth.

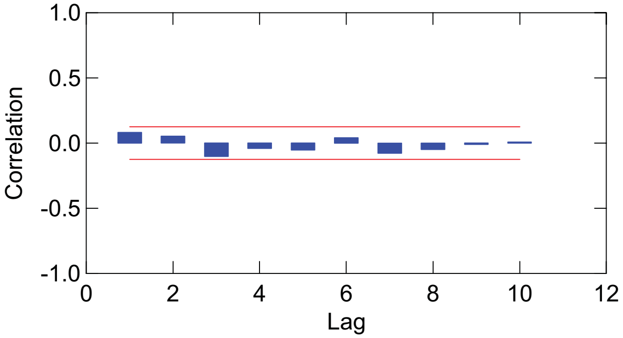


Figure 14.18 *ACF plot of residuals from Fourier smooth*

Figure 14.19 shows the Fourier coefficients on the complex plane. The full set of coefficients (except for the constant) are on the left and the coefficients we retained are on the right. Notice that we retained some frequencies with rather small magnitudes (coefficients near the center of the plots). If we didn't mind fitting just a few sine and cosine waves, we could throw these out. In that case the smooth would be *very* regular, but fairly biased.

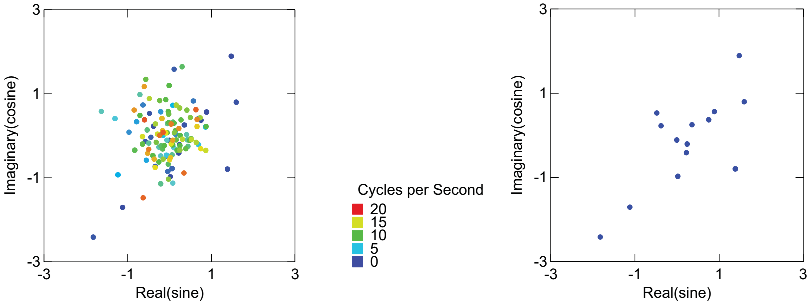


Figure 14.19 *Fourier coefficients on the complex plane (all coefficients in left panel, retained coefficients in right panel)*

14.3.1.2 Kernel Smoothing

We mentioned in Section 14.1.1 that smoothing in the frequency domain and back-transforming is equivalent to smoothing in the time domain directly with a kernel that is the inverse of the smoother used in the frequency domain. Figure 14.20 shows an Epanechnikov smooth of the cat data with a window width of .2 seconds. As we noted earlier, the center of the Epanechnikov kernel $f(x) = 1 - x^2$ is shaped like the center of the sinc function $f(x) = \sin(x)/x$

near $x = 0$. Thus, the result is similar to that in Figure 14.17. Note, however, that there is slightly less regularity in the kernel smooth. The Epanechnikov filter allows a few more high-frequency components into the smooth.

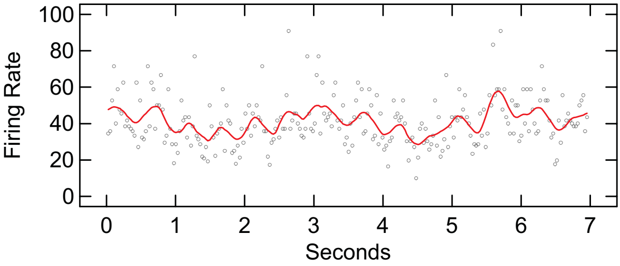


Figure 14.20 *Epanechnikov kernel smooth of cat series*

Thoughtlessly applying smoothers to stochastic processes can lead to mistaken conclusions. Perhaps the most egregious example is using ordinary linear regression to fit the model $y = \alpha + \beta t + \varepsilon$ to a stochastic process. The residuals will be correlated. We are less likely to pay attention to this problem when applying kernel smoothers, however, because we can choose a kernel that makes the residuals practically uncorrelated. Achieving independence in the residuals should not be our only goal, however. There is a trade-off between bias and variance.

Figure 14.21 shows a **moving average** fit to a random walk series. The moving average was calculated by averaging values within a time interval using a rectangular kernel. This method is commonly used on financial time series. We chose a window width of five time points, which made the autocorrelations of the residuals relatively small.

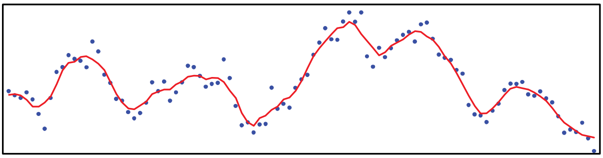


Figure 14.21 *Moving average of random walk*

If we consider what we have done, however, we realize that we have modeled the random fluctuations in the process. Our smoother fits the data *too* well. In short, kernel or local polynomial smoothers can track the series more closely than simple global parametric smoothers, and the residuals can appear to be independent, but this is no guarantee that the smoothers are picking up nonrandom features in the data. Once again, there is no substitute for knowing the model that generated the data. Absent that, large autocorrelations should point us toward stochastic models. We should be careful about using any deterministic model, including locally parametric ones, on stochastic data.

14.3.1.3 Stochastic Modeling

What if the Great Depression had never happened? Specifically, can we show that US Patent applications would have followed a different trajectory if the Depression had not occurred? We want to change the course of time and see what would have happened.

This goal is dubious for several reasons. The first problem is that the model we propose may be implausible even though it fits the data well. The second problem is that, like all analyses of historical data, **postdicting** (predicting historical events from previous historical events) usually involves **counterfacts**. Asking, "What would have happened if the Great Depression had not occurred?" is like trying to describe what a man's attitude about his wife would be if he were a woman (but otherwise the same person). The best we can do is to make sure that the assumptions on which our conclusions are based are plausible in most respects. Specifically, if we can imagine an experiment in which we could intervene economically to see the effect on patent applications, then we are in a better position to justify our conclusions. Rubin (1974, 1976) and Holland (1986) discuss this approach in more detail.

Nevertheless, we will forge ahead to illustrate the graphical display of forecasts. The data consist of US patent applications for new inventions since 1880. These data were compiled from several US government sources, including *Historical Statistics, Colonial Times to 1970*, and the *U.S. Statistical Abstract* for years following 1970. We have normalized patent applications in each year by the Census population in hundred thousands.

Figure 14.22 shows this series plotted in blue. The red series is a forecast of patent applications per 100,000 starting at 1931. The model we fit for the forecast is $x_t = \theta\varepsilon_{t-8} + \varepsilon_t$, where $\theta = 0.64$. This is a seasonal moving average model, where we presumed a cycle of eight years. Some would call this a trend-cycle model because the cyclic period is longer than a year, but the model is mathematically the same in either case. We settled on an eight–year cycle by inspecting the ACF for the entire series. And we took first differences and seasonal differences before fitting the model (making it an integrated seasonal moving average model). Notice that the forecast series reflects this periodic cycle as well as the upward trend from the pre-Depression era. The hills and valleys in the forecast mirror most of the ones in the post-Depression series. Regardless of quibbles over the choice of model or *ceteris-paribus* assumptions (not the least of which is World War II), it is not implausible that the Depression harmed US innovation in the long term. And this example illustrates that a good graphic is one of the best ways to communicate the results of a time series model.

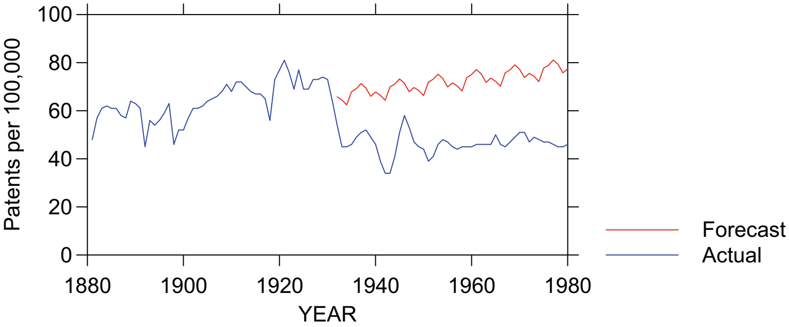


Figure 14.22 *Patent applications per 100,000 US population*

14.3.1.4 "Causal" Modeling

Do increases in patent applications cause economic upturns? This question raises almost as many problems as the speculation in the previous section. To avoid some of them, we will modify our statement to be, "Are patent applications leading indicators of economic upturns?" We worded the question in its original form to highlight the common desire to inspect joint series and make causal statements when the series appear to be congruent. This is a risky enterprise. As we shall see, despite its widespread use, visual inspection of raw series is almost of no use in estimating correlation, much less causation.

Figure 14.24 plots annual total US patent applications against time and GNP against time in the same frame. GNP is the US gross national product adjusted for 1970 dollars. We have colored the scales to match the series. The correlation between the two series is .76, yet they do not appear to be strongly associated visually. Why is that? Despite the lack of parallelism, the upward trend in both series substantially influences the correlation. Any two series trending up or down over their length will be strongly correlated.

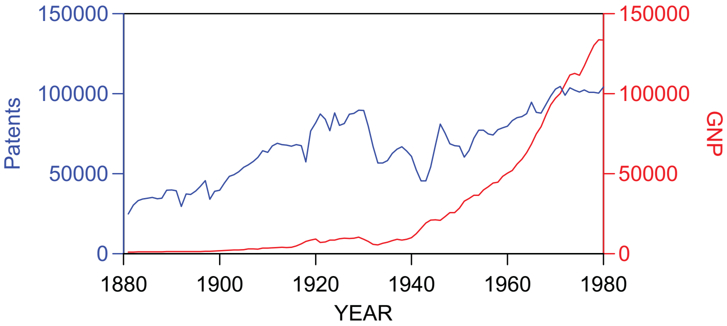


Figure 14.23 *Raw GNP and patent series*

We can enhance the correlation even more by transforming the series. In Figure 14.24 we log the series and adjust the scales to make the series roughly parallel. Now the correlation is .85 and it is easier to see local features shared by both series.

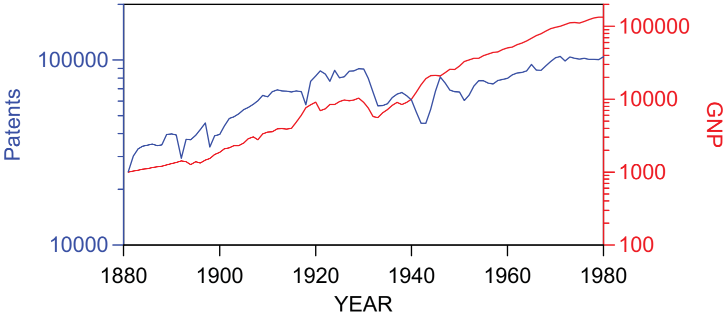


Figure 14.24 *Logged GNP and patent series*

Comparing series visually can be misleading, however. Local variation is hidden when scaling the trends. We first need to make the series stationary (removing trend and/or seasonal components and/or differences in variability) and *then* compare changes over time. To do this, we log the series (to equalize variability) and difference each of them by subtracting last year's value from this year's value. Figure 14.25 shows the result. We are now in a position to ask, "Do year-to-year fluctuations in patent applications follow year-to-year fluctuations in GNP?"

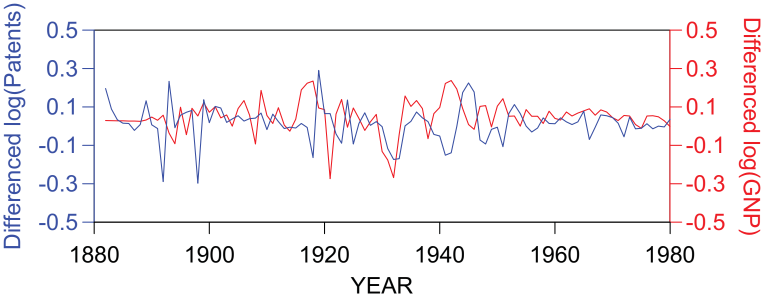


Figure 14.25 *Logged and differenced GNP and patent series*

Closely inspecting Figure 14.25, we can see that GNP changes appear to precede patent changes by a few years, especially in the second half of the series. Our eyes can fool us, however, so we resort to another graphic to reveal this structure more formally. The **cross-correlation function** plot is similar to the ACF plot. It lags both series forward and backward (the ACF lags only backward). Then the correlations between lagged series are displayed with bars. Figure 14.26 shows the result. There appears to be a significant correlation at forward lag 3, indicating that changes in patent applications trail those in GNP by 3 years. The plot is not symmetric, indicating that the correlations do not work in the other direction. Notice that the eight–year seasonal variation remains in the CCF plot. We could go on to deseasonalize the series and see whether we are left with white noise.

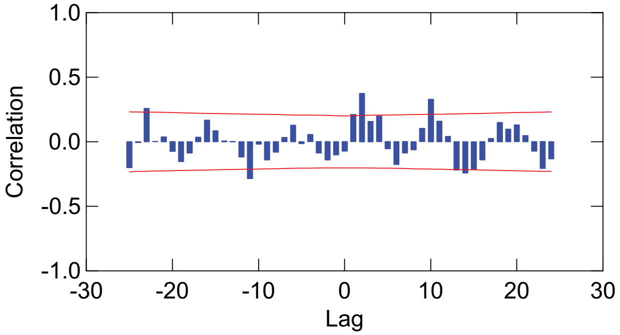


Figure 14.26 *CCF of differenced GNP and patent series*

Smookler (1966) came to similar conclusions with similar data, Smookler found that invention follows economic trends rather than scientific developments:

> When time series of investment (or capital goods output) and the number of capital goods inventions are compared for a single industry, both the long-term trend and the long swings exhibit great similarities, with the notable difference that lower turning points in major cycles or long swings generally occur in capital goods sales before they do in capital goods patents. (page 205)

Smookler concluded that

> (1) invention is largely an economic activity which, like other economic activities, is pursued for gain; (2) expected gain varies with expected sales of goods embodying the invention; and (3) expected sales of improved capital goods are largely determined by present capital goods sales. (page 206)

It does *not* follow from these conclusions that investment in scientific research is unrelated to future economic prosperity, however. Our finding of a leading indicator of patent activity is only part of the picture.

14.3.2 Dynamic Graphics

If we render a graph on a video display or analog projector (as opposed to paper), we are not restricted to static methods; we can change the display over time. Such a display can be used to represent data that are themselves changing over time or space or some other continuum. Motion is an aesthetic, since it is something we perceive and construct an internal representation for, in order to process dynamic information.

One of the more intriguing uses of motion is to animate immersive scenes. We can do this in a virtual-reality environment such as the Cave (Symanzik *et al.*, 1997), or we can do it on a perspective wall or large computer display of a 3D scene (Wegman and Symanzik, 2002). Eddy and Oue (1995) present an example that is now well-known to pilots around the world.

The data were taken from an FAA tape containing a data stream of all commercial flights in the US during a 24–hour period. Much of Eddy and Oue's work involved sifting through the dataset to recover instantaneous latitude, longitude, and altitude coordinates for thousands of flights and merging these coordinates with geographic metadata. They then rendered these flights in two ways. The first involved a "satellite" view of all flights moving across the country, with 24 hours compressed into a minute's time frame. The second involved following a single flight from takeoff in Newark, NJ, to landing in Denver, CO.

Figure 14.27 shows selected frames from the Newark–Denver flight movie. The black squares represent the airports. We begin with an introductory zoom from satellite view into Newark. The movie ends with a return to satellite view after the landing. Two interesting aspects of the flight are apparent. The first is a course adjustment near Chicago. At 6:20, the jet takes a relatively sharp turn north and then readjusts its course about 10 minutes later. This adjustment appears to be related to another flight crossing its path (viewable in the upper right of the ninth panel (6:26). The intruding flight passes quickly through in the next panel and the jet returns to its original path. The second pattern most visible in the film is the lining up of flights during the landing (visible in the 13th panel, 7:43) like a string of pearls.

The striking aspect of the Eddy and Oue animation is the simplicity in the choice of rendering and representation elements. Airliners are represented as pearls with size as a distance cue, geography is represented by a US state boundary map, and airports are represented as black squares. Additional realism would do little to improve the communication of temporal/spatial information in the animation; more likely, more would be less.

There are other interesting applications of dynamic graphics involving variables animated over time. We discuss these in Chapter 17.

.

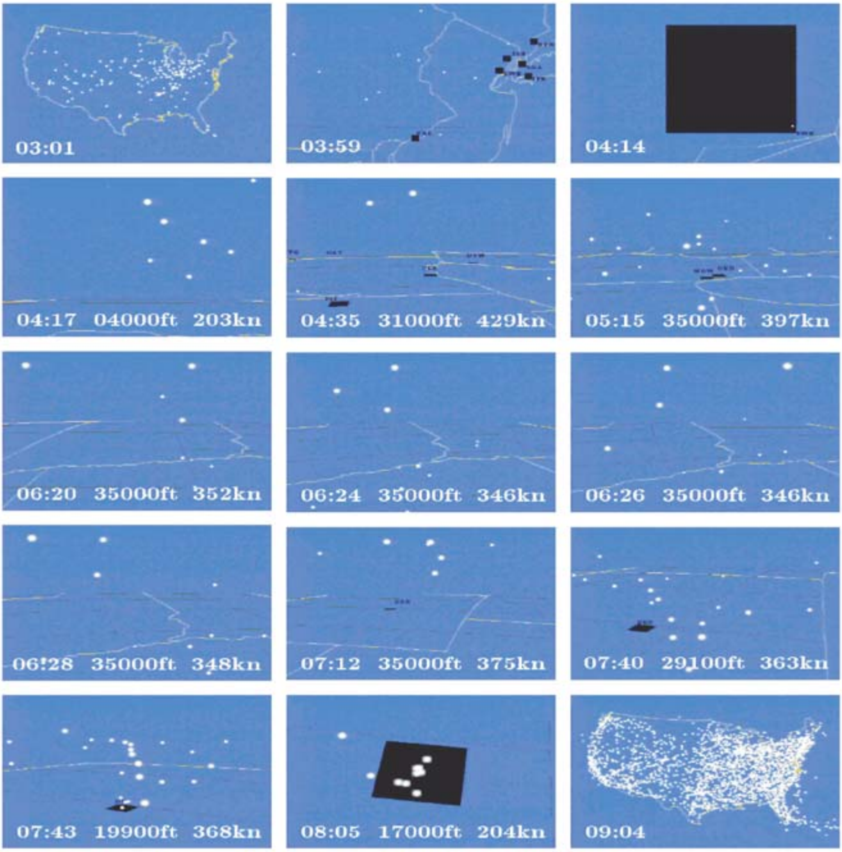


Figure 14.27 *Eddy and Oue airline flight movie*

14.3.3 Real-Time Graphics

Real-time graphics are not new. Sundials, thermometers, barometers, clocks, speedometers, oscilloscopes, lie detectors, and heart monitors are all real-time graphics devices, What is new is the technology to record massive data streams, sample from them, analyze them in real time, and display intricate graphics that incorporate these analytics.

We call this recent phenomenon **streaming graphics**, that is, graphics based on streaming data (Norton, Rubin, and Wilkinson, 2001). The term *streaming graphics* evokes the term **streaming media** (*e.g.*, Feamster, 2001). While similar in outward appearance, streaming graphics are fundamentally different from streaming media, however. Streaming media systems are generally concerned with delivering sound and video information in real time. In streaming media, the data structures are defined by the aspects of the display. In streaming graphics, the display is defined by aspects of the data. Applications of streaming graphics involve many different environments and data structures, including real-time monitoring of manufacturing processes, personal health indicators, financial series, telephony, Web, and sensor networks.

Streaming graphics displays also look similar to video animations. They are fundamentally different, however. Rendering and animating virtual reality frames is similar to making computer movies. These frames require minutes or hours to render in order to achieve a high degree of realism. Streaming graphics systems cannot afford this rendering luxury. They must render scenes up to 30 times a second in order to keep up with real time.

Preparing a streaming graph is similar to preparing a static graph. The data values are read, statistically summarized, converted to aesthetic values, and rendered on the screen. But unlike a static graph, the display must be updated if the data change. The most efficient way to do this is to process only the ramifications of the changes, rather than to recreate the entire graph on each change.

14.3.3.1 Data

Streaming data consist of an indexed set $\{x_1, x_2, ...\}$, where each x_i is a state space. Alternatively, a data stream may be viewed as a set of ordered tuples $\{(x_1, t_1), (x_2, t_2), ... \}$, where t_i is calendar time or elapsed time. The simplest state space is a binary digit (bit). More complex state spaces are text strings, images, data tables, or geographic spaces. We assume streaming data sources are ordered in time, but in practice they may not always arrive at a queue or gate in a complete time ordering, because physical sensors and transmission systems are subject to bias (asynchronicity) and error (noise).

This definition subsumes all the data structures considered so far in this chapter, but streaming data involve certain restrictions. First, data streams are typically massive in extent. In fact, they are *infinite* in extent when they have no known beginning or end. Second, data streams are often massive in packet size, particularly when sensors are multiplexed into a single feed. Both these characteristics mean that data streams usually cannot be stored or archived. Thus, algorithms designed to handle streaming data cannot allocate enough memory to allow multiple passes over an entire stream. And we cannot solve this problem by random sampling, because a probability sample from an infinite stream is infinite in size.

Two approaches to processing streaming data have received the most attention by computer scientists (Henzinger *et al.*, 1998; Feigenbaum *et al.*, 2004). The first is called the **streaming** model. In this approach, we process each data packet sequentially and update a model (sum, median, spectrum, subgraph, etc.) as we proceed. By definition, we are allowed only one pass through the data. In statistics, this approach has been called the **sequential sampling** model (*e.g.*, DeGroot, 1970). Statisticians have tended to focus on estimation (when is an estimate good enough to be able to stop sampling?) and computer scientists have focused on efficiency (what are the most efficient algorithms with respect to time/space dimensions?).

The second approach has been called the **sliding-window** model. In this approach, we choose a window (interval) of fixed time width or fixed number of data points. We analyze tuples in the data stream falling within that interval. In statistics, this approach has been called the **kernel** or **moving-window** model, and has been used for forecasting, smoothing, and density estimation. See Section 7.1.1 in the Statistics chapter and Section 14.3.1.2 in this chapter for examples.

The sliding-window model may have more applications in streaming graphics because we are most often interested in viewing recent history (only *n* prior packets) or the present (only one packet). There are important applications of the streaming model, however. Imagine, for example, a sequential polling scenario in which the standard errors of estimates shrink in real time. Observing the rate of convergence can help viewers, particularly those not trained in statistics, to understand the behavior of estimators with respect to sample size.

Updates

In order to update a graph, the first step is to identify what has changed since the graph was last updated. If we have access to the process by which the data are changed, we can leverage the process to update our graph. For example, we might install a proxy that will intercept the updates, running additional code after the data have been changed. If we are using a database, the database may support **triggers**, which are bits of code that are run whenever a value of a particular variable is changed.

The added code could directly update the graph. But a more general solution is to follow the **observer** pattern (Gamma *et al.*, 1995). A list is maintained of all objects that want to be notified when a change occurs. Whenever a change occurs, each of the observing objects is sent an event object identifying the location and type of change. Each observer can then take appropriate action, such as updating a graph or table.

Because each event is represented as an object, the events can be stored, classified, replayed, and otherwise manipulated. In particular, repeated update events for the same table cell can be coalesced within a specified time period into a single event.

Distributed processing

The Internet and distributed computing technology make it possible to collect live information from around the world. When the events are coming from another computer or sensor, it is important to minimize communications cost and delay. A productive approach is to make use of agents (that is, information about the client) on the server side to collect and filter events. Rather than sending irrelevant information to the client, this approach processes the events on the server side on behalf of each client.

For example, if the client is only interested in certain variables or certain cases, the server needs only to send events pertaining to those cases and variables. By assembling the requirements of all clients, the server can restrict the monitoring that it needs to do. Many clients use events periodically (for example, to update a graph image) even if the updates are more frequent. The number of transmissions can be reduced by coalescing events on the server and sending them to the client on request, so that all information required to render a graph is obtained at once. An enhancement is to keep track of the last notifications sent to each listener, and to send updates only if the current value differs significantly from the value held by the listener. Minor changes might not be worth the cost of updating the graph.

A further refinement, called **dead reckoning**, can be used when variables change at a nearly constant velocity. In this approach, the client keeps track of both the current position and the current velocity, and assumes that changes continue at the same velocity. The server compares the client estimate to the actual position, and sends only those updates that diverge beyond some specified tolerance. When an event is sent, it includes both updates to current position as well as the current velocity vector.

New data values

In a streaming data context, we may encounter new data values that have not previously been in the data. How do we extend a graph to accommodate these new values? The first line of defense is through domain declarations. The frame is not defined by the transient contents of the data, but by declarations. For each variable, an administrator declares the valid contents of that variable: perhaps a range, perhaps a list of valid values. Thus, it does not matter if some rare values are present sometimes but not others, because the domains, aesthetic functions, legends, and axes will remain constant.

In many cases, we will not know all of the data values until we encounter them in the data. Domain definitions can handle previously declared rare values, but they are of no use for unanticipated values. The solution requires domains, aesthetic functions, and legends to be extensible. If a domain is designed to be extensible, a new value can be added if the domain considers it valid. It then notifies all aesthetic functions dependent on the domain. Furthermore, if we extend the domain in round numbers (*e.g.* extending the maximum salary in increments of $50,000), we will not have to extend the domain for each new value.

When an aesthetic function is notified of an extension to a data domain, it must modify itself to handle the new data values. For example, the new data value might be assigned to the next available value from a pool of values. Ideally, the extension can be accomplished without changing the mapping of existing values, but this might not be possible.

Invalid values are those which the domain will not accept, because of type incompatibility, impossible values, missing values, or server faults. A typical approach for a static graph is to produce an error message instead of a graph or to render the graph without the invalid values. In a streaming graphics context, the graph already may be active when an invalid value is encountered. We need to anticipate this possibility and provide recovery methods when an exception is encountered.

14.3.3.2 Statistics

Statistical algorithms for streaming data require special attention. In the streaming model, we need to minimize the number of update computations in order to keep up with the velocity of the data. In the sliding-window model, we need to downdate and update efficiently. Rectangular kernel methods are trivial. We simply drop the last case and add the next from the stream to update the model. Non-rectangular kernels require reweighting of each point in the window. Step-function kernels can be designed to reduce the number of new multiplications required at each increment.

Figure 14.28 shows an example of this approach. The data are a daily series of SPSS and Oracle stock prices. An exponential smoothing model was fit to the data in order to produce a forecast over subsequent days. (This forecast models customer behavior more than market performance of the stocks.) Because the computations in the exponential smoothing are programmed efficiently, the series can be animated to cover 10 days per second on a typical laptop computer. The vertical scales of the frames are log–transformed in order to control variance in the process. In the animation, one can sense the convergence and divergence of the series over the entire period.

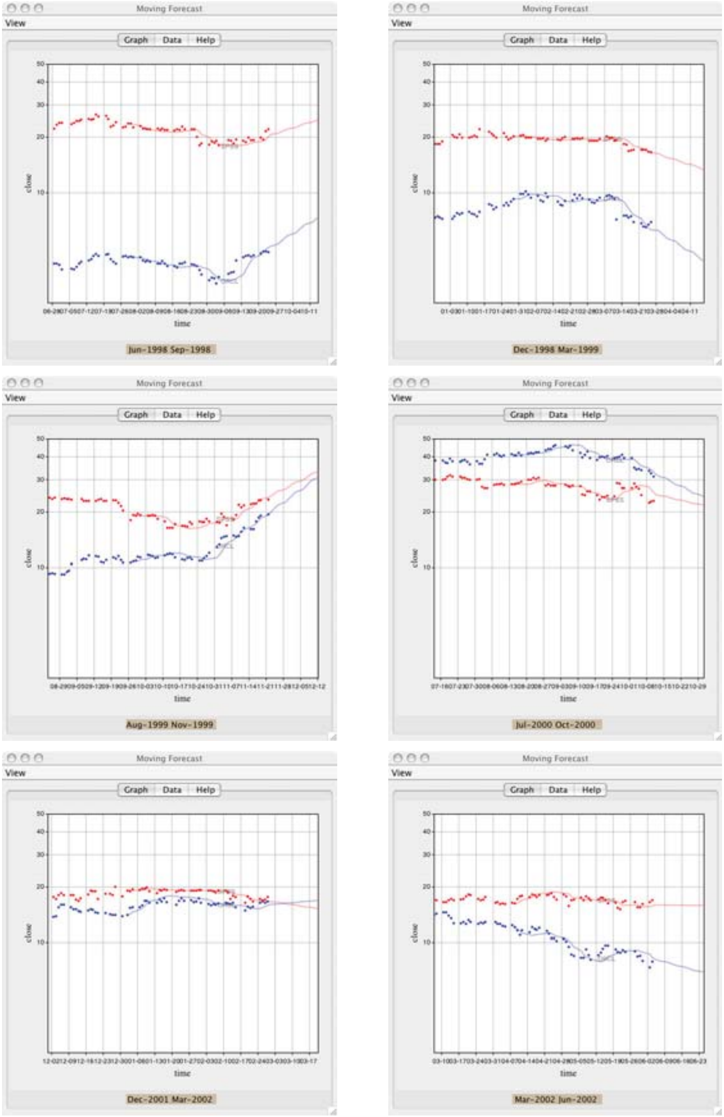


Figure 14.28 *Moving forecast*

Figure 14.29 shows an example of a surface fitted with a *loess* smoother in real time. This data model is different from the stock example. In this case, we have a finite number of points whose values are changing in real time. Each packet consists of a set of tuples (x, y, z) at a given time point. The *loess* calculations must be computed for each time period in order to render the moving surface. If the number of points is not large, the calculations can be accelerated by using observers to identify the points that change coordinates, leaving other unchanging points out of the update.

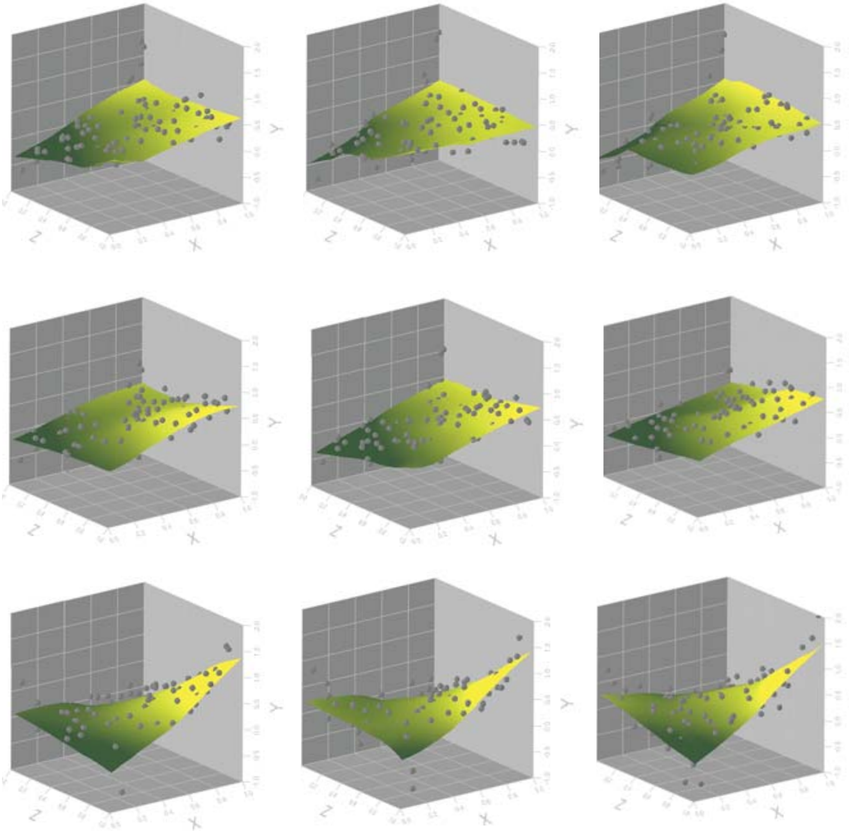


Figure 14.29 *Moving loess surface*

14.3.3.3 Frames

A frame is a region that is a subspace of the domain-range product, as defined in Chapter 2. In streaming graphics, frames are displayed up to 25 times a second in order to give the illusion of movement. It is not necessary to update the screen every time a data value is updated, however. Rendering 500 frames per second is not ten times better than 50 frames per second, because computing resources are being used to produce a result that no human being can perceive. The frame rate can be controlled through a timer that updates the screen when there have been changes, but no faster than a specified minimum delay between frames.

In general, there is no guarantee in a streaming graphics system that frames will be rendered at an even rate. Frame rendering is competing with other processes for computing resources. Consequently, we need mechanisms to handle data change notifications that may arrive at any time relative to rendering. Recall that event notification is generated through the process of monitoring the data. Monitoring must not interfere with rendering, so we use multiple **threads** to handle monitoring and rendering.

Data values may be updated far more rapidly than the frame rate. There is no advantage to processing updates that will never make it to the screen, however. Consequently, we maintain a set of **dirty** values (ones that have changed since the last frame was rendered). The rendering thread reads the data for each of the dirty values, and updates the graph.

While it is possible to render each frame from scratch, it is more efficient to preserve as much information as possible from the preceding frame. Data that are expensive to obtain (due to remote access or extensive computations) may be cached for future use. When new data arrive, we discard obsolete data values, downdate their contributions to any accumulated statistics, and update the statistics from the new contribution. This process can accumulate rounding error, so we need to recalculate occasionally from all values. Among those working on such issues are Guha *et al.* (2000) and Datar *et al.* (2002).

Interpolation

The eye is very sensitive to fluctuations in the speed of movement. For a glyph to appear to move at a constant rate, the change of position between frames must correspond to the actual time lag between rendered frames. If rendering of a frame is delayed, the glyph must move a greater distance. In order to preserve the illusion of smooth movement, we need to coordinate frame rendering and glyph positioning.

Even if frames are rendered at a constant rate, there is no guarantee that a data value will be available at frame display time. For example, if measurements are made infrequently, the same outdated value will be used for several frames, and then there will be a jerky movement as the accumulated change is applied in a single frame. One solution to this problem is to interpolate or extrapolate data values to yield estimates for several intermediate frames.

Inconsistencies within a frame

Inconsistencies within frames arise when the measurement time varies between values in the same frame. Random inconsistencies within frames are usually a minor concern because they are corrected when new data arrive, but there could be a problem if the renderer is not prepared to deal with inconsistent data. Estimation techniques can be used to combine data measured at different times.

More serious problems arise if the inconsistencies are systematic from frame to frame. Typically this occurs when iterating over every geometric element to gather current measurements. If we always iterate in the same order, the later glyphs will always have later measurements. The resulting graph may display systematic biases such as shortening or lengthening of distances.

One solution is to adjust for the differing measurement times through estimation. The problem can also be alleviated by obtaining all data values as quickly as possible (and not rendering until after collocation), or by randomizing the order of iteration. Such randomizing is similar to **frameless rendering**, in which pixels are updated in a random order (Bishop *et al.*, 1994).

Scale of motion

The minimum acceptable frame rate depends upon the physical speed of movement of the glyph. A frame rate of only once per second is adequate if a dot only moves a slight distance (*e.g.*, ½ the diameter of the dot or less). But if a dot moves too far on each refresh, it may be perceived as two dots flashing rather than one dot moving. We cannot help the speed of a dot moving — it comes from the data. But we *can* reduce the absolute distance the dot moves between frames, either by increasing the range of the axes or physically reducing the size of the display. The more we zoom in on a moving graph, the more movement there is, and the more likely we are to lose the illusion of motion. So we can successfully zoom in closer if movement is slower, or if the frame rate is faster.

Instant replay

The advantages of animation (pausing, random access, varying speed) can be applied to data that arrive in a live data stream. If we save recent data (or rendered frames) in a buffer, we can access recent events at will. When the user invokes instant replay (such as by pulling a current time slider backwards), animation begins from the buffer, rewinding and replaying from the selected point. We continue adding data to the buffer during replay, so the replay can continue beyond the original stopping point.

Buffering data allows us to transform time as well as replay time. For example, we can log or power past time and replay frames on a nonlinear time scale (Eames and Eames, 1996). Or we can reverse time to deconstruct a process. Auditory feedback can be used to cue the user to speed changes.

Time axes

When time is used as an axis in an animated graph, the gridlines change over time. This usually causes no confusion to the user because the motion of gridlines is readily visible and comprehensible. Such graphs display new information at the leading edge, and show older information for context. Graphs with a time axis may be differentiated by the animation of that axis: neither end moving (filling in the contents), the top end moving (compressing as it goes), or both ends moving (a sliding window). If the time axis is fixed at the low end and the high end changes over time, then the distance allocated to each unit of time gradually becomes smaller. Such a graph always shows all available data using all of the available space. The disadvantage is that slopes change over time. In a sliding-window axis, the low and high ends change together so that the total distance of the axis remains fixed. Slopes and distances remain constant.

Figure 14.30 shows a sliding window of stock trades. Three symbols are used in the right panel: a downward-oriented triangle to indicate an offer to sell, an upward-oriented triangle to indicate an offer to buy, and a square to indicate a cross trade. The symbols are colored by trader. The top panel shows trades shortly after the exchange opening. By the middle of the day, trades become more dense. And by the end of the day they thin. The traded stock (Microsoft) has sufficient volume to allow tick marks at each second. Anomalous offers stand out from the series and can be recognized within a second of the event. The bar graphs on the left are linked to the trading data. Together with various controls at the top of the window, these graphs can be used to get more detailed information on the traders.

Alerts

Alerts are trigger conditions that identify when something interesting has happened, such as a value crossing a threshold. By having a computer monitor the conditions, we can avoid the tedium of visually monitoring a process. When an alert does occur, appropriate action will often include saving the data (and the context around it) for later review. By maintaining a buffer of recent events, we can store both events before and after the alert. If multiple alerts have occurred, they can be matched and compared using time series analytics.

Zooming

Zooming is the adjusting of aesthetic functions to clearly distinguish the current data. It is necessary because data change over time, and a frame that serves well for today's data may not be so effective for tomorrow's data. Zooming differs from merely excluding certain cases from the graph, because the aesthetic functions are modified to increase the contrast between the cases of interest. The term **zooming** comes from the traditional use of the technique: reducing the range of an axis to spread the points out across the available positional area.

.

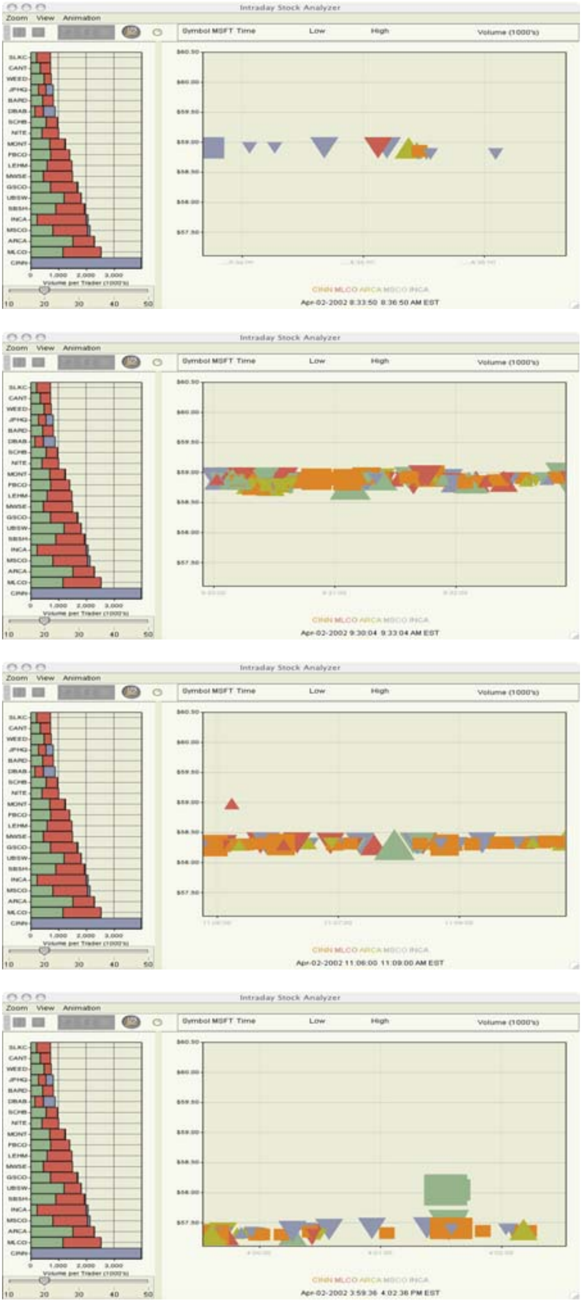


Figure 14.30 *Stock trades*

We can zoom for a continuous aesthetic by specifying the maximum and minimum data values to be used. For example, instead of mapping 0 to black and 100 to red, we may map 70 to black and 90 to red, more clearly differentiating between values in the 70-90 range. Values falling outside of the specified range may be mapped to the extreme aesthetic values (clamping) or made invisible (clipping). In the case of positional aesthetics, we usually clip values outside of the selected range. But in the color example, we might choose to represent all values under 70 as black and all values over 90 as red.

We can also zoom for categorical aesthetics (*e.g.*, shape). Even if we have a great number of categories in simultaneous use, it is still possible to use a limited number of codes, provided that some categories are not represented or share a code. For example, we could select only the top *n* categories based upon some criterion, allowing the user to distinguish between those categories while ignoring the others (excluded or collapsed into an "other" category). This ranking approach is easiest to follow if the ranking is itself represented in the graph, so we can see categories come into the graph and fall out of the graph.

We can control zooming automatically, based on the values currently within the data. But how do we orient the user so that changes in the frame are not confused with actual movement? There are two contrasting approaches: (1) make the frame changes rare and obvious, or (2) make the frame changes continuous and predictable. If changes are made suddenly, the shifting of aesthetics, legends, and gridlines indicates to the user that a change has occurred. It is easier for the user to follow changes in position alone, with scale held constant. By providing some leeway beyond the current data range, the changes can be kept to a minimum to avoid distracting the user.

The alternative is to make changes at a constant rate, such as advancing a sliding window at a specified speed. For animated data, regression can be used to determine a suitable speed that will keep up with the data. Another approach is to designate a given glyph of interest and always keep that glyph at the center of the graph.

14.3.3.4 Rendering

The frame provides context. Once users become familiar with the coordinates and legends, they only have to refer to the frame occasionally. Typically, the frame either changes in a regular way or does not change.

The elements, by contrast, change constantly. Elements represent the data itself, and thus are the focus of the user's attention and monitoring. Comprehending the elements requires perceptual tasks that may be somewhat difficult, including locating and tracking a glyph within a collection of glyphs, identifying which glyphs are changing, and distinguishing trends from noise.

Tracking glyphs

In order to perceive a glyph as moving, the user needs to recognize it as being the same glyph. This can be difficult if there are many glyphs on the screen or if position changes are large.

Confusion occurs when position changes at the same time as other aesthetics. It helps if some aesthetic is distinctive (*e.g.* a unique color, label, or icon), or if the aesthetics (*e.g.* color or size) are changed gradually and consistently. A useful aid to glyph tracking is to label or highlight the points of interest. The label might only made visible when the user clicks on a point (to avoid clutter from other points not of interest). Or the user might select all points within a given region and watch them move as a flock (or diverge).

Another aid to glyph tracking is to control the rendering layer, so that objects of interest are rendered on top, where they can clearly be seen moving across other objects. Larger or slower-moving objects can be rendered on the bottom, where they will be less likely to obscure other objects.

Velocity can be explicitly represented on the screen by another aesthetic, such as vector arrows (showing current direction and speed), coloring glyphs by speed, or hiding slow moving objects. We need not restrict ourselves to representing velocity. Graphing acceleration or percentage change may help us spot interesting changes earlier. If a path is displayed showing positions within a fixed window of time, the distance between points indicates the speed. Faster moving paths are longer.

Gradual change

It is crucial to avoid movement or change that does not convey information. A good streaming graphics design will direct the user's attention toward changes of interest. One way to do this is to represent changes gradually over time. Abrupt changes are difficult to comprehend, because by the time you have noticed a change it has already completed, and you have to remember what used to be there.

Events have zero duration by definition. But by extending the representation of an event over time, the change attracts the eye. In the computer animation world, this is called **anticipation**. When introducing or removing a new glyph, the shapes can fade in and out, or grow and shrink in size. When changing values, colors or shapes can gradually mutate from one value to the next.

One possible objection is that we are no longer literally representing the data. We are representing events before they take place or after they have completed, and we are showing intermediate values on the way from one state to another. But sometimes, this is actually what happened, and we just don't have all of the intermediate measurements.

14.4 Sequel

The next chapter covers issues involved in representing uncertainty in graphics. Uncertainty occurs in many forms: missing data, measurement error, hidden bias. We discuss how these forms affect interpretation and how we can represent uncertainty graphically in order to facilitate reasonable decision making that depends on displays.

15

Uncertainty

The word *uncertainty* derives from the Latin adjective *certus* (determined, fixed, settled), which itself is derived from the verb *cernere* (to discern or perceive, usually with the eyes). This etymology is relevant to our perspective on graphics and aesthetics because we encounter uncertainty when we are unable to perceive without doubt (*sine dubio*).

If we must choose a single word to characterize the focus of modern statistics, it would be *uncertainty* (Stigler, 1983). It is tempting to ignore uncertainty. Statisticians have shown, however, that including terms for bias and error in models improves predictions. If we allow for error, we are less likely to find rules in randomness. This fact was often ignored by early data miners, many of whom lacked statistical training, until Glymour *et al.* (1996), Hastie *et al.* (2001), and others brought it to their attention.

This chapter is about graphics that guide, qualify, or soften our judgments of uncertain data. Popular summary graphics (*e.g.*, simple pie charts, bar charts) omit uncertainty. When used on data containing error, these graphics can be deceptive because they conceal variation. In this chapter, we will discuss ways to add uncertain aspects to these and other graphics so that perceivers can temper their conclusions. We will also discuss graphics that can be used to guide decisions based on uncertain evidence.

15.1 Mathematics of Uncertainty

This section will cover some of the basics of probability and other measures of uncertainty. It is not intended to be an introduction to statistical inference, although statisticians and readers familiar with probability theory can skip this section without loss. The primary purpose of the section is to provide a common notation and set of definitions so that we can discuss the graphics of uncertainty beyond the obvious applications.

15.1.1 Defining Uncertainty

First, a few definitions. Many words are associated with uncertainty (or its opposite). The following list runs roughly from primitive to composite and from negative to positive.

- *Variability* is non-constancy. A set of data is variable if it has two or more elements and if any two of its values differ. Data variability leads to uncertainty only if we do not know how it occurs.
- *Noise* is variability produced by a stationary stochastic process (see Section 14.1.2). Perhaps the most obvious example is Gaussian white noise in a linear system. Noise produces uncertainty because it is random.
- *Incompleteness* is the presence of missing data. Missing values can be produced by breakdowns in equipment, refusal to answer questions, confidentiality restrictions, and other factors. Missing values produce uncertainty when we cannot reliably impute them.
- *Indeterminacy* is the existence of more than one set of parameter values satisfying the conditions of a model and its associated data.
- *Bias* is systematic discrepancy from a standard, as in a biased opinion. Measurement bias is systematic discrepancy between a true value that we attempt to measure and an observed measurement of the value.
- *Error* is random discrepancy between a measured and true value. Unlike bias, error varies to the left or right of the truth with equal likelihood.
- *Accuracy* is relative lack of bias and error. If a measurement is representable by the equation *measurement* = *truth* + *bias* + *error*, then a high level of accuracy reflects a high level of truth in the measurement.
- *Precision* is relative lack of error. A highly precise measurement can be biased. We associate *significant digits* (the number of digits in a measurement that are not affected by error) with precision of a measurement.
- *Reliability* is the repeatability of a measurement over time. The smaller the variance in a series of measurements, the higher their reliability.
- *Validity* is the association of a measurement with the true process generating what is measured. A valid measurement need not measure a variable itself. It simply needs to measure something perfectly correlated with what is measured under all relevant measuring conditions.
- *Quality* is a combination of completeness, reliability, and validity.
- *Integrity* is the presence of information that allows a judge to establish quality. This usually involves an audit trail or lineage for a set of measurements and the context in which they were made.

Further definitions are available in the *Guide to the Expression of Uncertainty in Measurement* (GUM), published by the ISO in Geneva, Switzerland and in the *NIST Technical Note 1297*, available from the NIST website. We will now begin with the theory of probability, on which rests the concept of randomness, the concept of error, and much of the concept of uncertainty.

15.1.2 Defining Probability

Let S be a countable set of values, *e.g.*, {true, false}, {heads, tails}, {1, 2, 3}, { ♦, ♥, ♣, ♠}. We call S a **sample space**. An **event** A, $A \subseteq S$, is a subset of S. The set of all possible events based on S is the power set of S:

$$E = 2^S = \{\text{all subsets of } S, \text{ including the null set and } S \text{ itself}\}.$$

The function $P(.)$ on the domain E is a **probability function** if

$$P(A) \geq 0 \quad \text{for all} \quad A \in E \ ,$$

$$P(S) = 1 \ , \text{and}$$

$$P(A \cup B) = P(A) + P(B) \quad \text{when} \quad A \cap B = \varnothing$$

These three axioms are known as **Kolmogorov's Laws**. From these laws, we can deduce that the range of $P(.)$ is the interval [0, 1]. We can also deduce that

$$P(\varnothing) = 0 \ ,$$

$$P(\bar{A}) = 1 - P(A) \ , \text{where } \bar{A} = \{x : x \notin A\} \ , \text{and}$$

$$P(A \cup B) = P(A) + P(B) - P(A \cap B)$$

If A and B are events in S, we define the **conditional probability** of A given B (assuming $P(B) > 0$) as

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

Notice that

$$P(A|A) = P(B|B) = 1$$

Notice also that if A and B are disjoint, *i.e.* $P(A \cap B) = 0$, then

$$P(A|B) = P(B|A) = 0$$

And if $P(A|B) = P(A)$, then A and B are **independent**. In this case,

$$P(A \cap B) = P(A)P(B)$$

Now we present the famous rule for computing conditional probabilities.

15.1.3 Bayes' Theorem

We illustrate Bayes' theorem with a simple example (French, 1984; Lee, 1989). Many introductions to Bayes' theorem employ a two-by-two table or cross-tab, but there is no necessary relation between the theorem and a cross-tab. This example, which we have modified to fit actual distributions, better illustrates the generality of the theorem.

Approximately three percent of human births involve twins. Approximately one-fourth of twin births are identical or monozygotic (MZ) and approximately three-fourths are fraternal or dizygotic (DZ). Of MZ twins, roughly half are male and half female. Of DZ pairs, approximately one-fourth are both males, one-fourth are both females, and one-half are mixed gender.

Figure 15.1 shows a mosaic plot representing these proportions. The sample space corresponding to the plot is all possible combinations of cohort gender and zygosity. We assume an equivalence between the tiles and the combinations. For example, the area of the pink tile in the upper left corner corresponds to $P(M \cap G) = .125$, assuming M stands for monozygotic twins and G stands for twin girls.

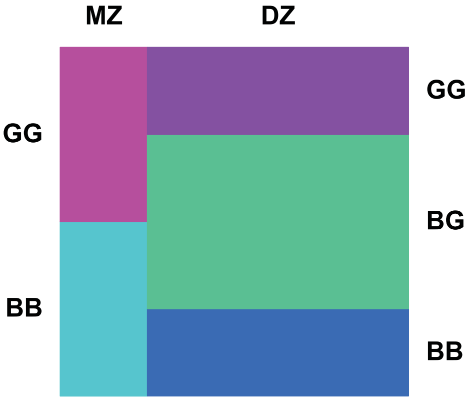


Figure 15.1 *Cohort gender of twins*

Suppose we are told that a particular set of twins is boy-boy (BB). Using Figure 15.1 as our reference, what would we say is the probability that this is an identical set of twins (MZ)? Looking at the areas in Figure 15.1 and using M to represent MZ and B to represent BB, we find

$$P(M|B) = \frac{P(M \cap B)}{P(B)} = \frac{\text{light blue}}{\text{light blue + dark blue}} = \frac{.125}{.125 + .1875} = .4$$

Bayes' formula can be derived from our definitions and this setwise formula as follows:

$$\begin{aligned} P(M|B) &= \frac{P(M \cap B)}{P(B)} \\ &= \frac{P(M \cap B)}{P(M \cap B) + P(\overline{M} \cap B)} \\ &= \frac{P(B|M)P(M)}{P(B|M)P(M) + P(B|\overline{M})P(\overline{M})} \end{aligned}$$

Using this latter formula, we of course get the same result as before:

$$P(M|B) = \frac{.5 \times .25}{.5 \times .25 + .25 \times .75} = .4$$

We now have expressed the conditional probability $P(M|B)$ (usually called the **posterior probability**) as the ratio of inverted conditional and unconditional probabilities. The value $P(B|M)$ is usually called the **likelihood** of B and the value $P(M)$ is usually called the **prior probability** of M. The numerator of Bayes' formula tells us that posterior probability is proportional to the product of a likelihood and a prior probability:

$$P(M|B) \propto P(B|M)P(M)$$

In other words, the chance of twins being monozygotic if they are boys is proportional to the likelihood that they are boys if they are monozygic times the chance that twins in general are monozygotic.

We specify Bayes' formula more generally as follows. Let $A_1, A_2, ..., A_n$ be any partition of S and let B be any event such that $P(B) > 0$. Then

$$P(A_i|B) = \frac{P(B|A_i)P(A_i)}{\sum_{j=1}^{n} P(B|A_j)P(A_j)}$$

Applications of Bayes' formula can lead to surprising results. Consider, for example, an airline passenger screening procedure. Let's assume this procedure has a .999 probability of alarming when a terrorist is encountered. Let's also assume the procedure has only a .001 probability of alarming when an innocent citizen is encountered. And let's assume the prior probability of encountering a terrorist at the test site is .00001 (we are alarmists). Under these assumptions, the chance of there being a terrorist if the procedure alarms is only one in a hundred, according to Bayes' theorem.

15.1.3.1 Continuous Distributions

Although we began this tutorial with discrete probability functions, Bayes' theorem applies to continuous probability functions as well. A **random variable** is a function defined on *S*. Let *X* be a random variable whose range is a set of real values $\{x: x \in S\}$, called a **realization** of *X*. We assume a probability density function $f(x) = P(X = x)$ is continuous over the sample space *S*. Then, as with the discrete formulation, we have

$$P(A) = \sum_{x \in A} f(x)$$

For a continuous density based on a parameter θ, Bayes' formula is

$$f(\theta|x) = \frac{l(x|\theta)f(\theta)}{\int l(x|\theta)f(\theta)d\theta}$$

where $f(\theta|x)$ is the posterior probability function, $l(x|\theta)$ is a likelihood function, and $f(\theta)$ is a prior probability function. The continuous version of Bayes' formula is similar to the discrete. Continuous probability density functions replace discrete probability functions and the integral replaces the summation in the denominator.

Figure 15.2 shows two examples of how the formula works. In the upper panel, we compose a normal prior (green) and normal likelihood function (blue) to derive a normal posterior density function (red). In the lower panel, we compose an arcsine or beta prior (green) and binomial likelihood (blue) to derive a beta posterior (red). Bayesian updating works like a convolution (see Figure 14.5). Each red ordinate is proportional to the product (*not* the sum) of a blue and green ordinate.

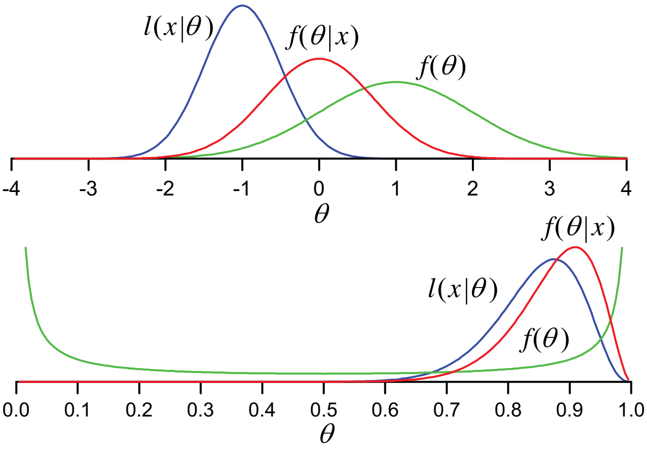


Figure 15.2 *Normal (top) and binomial n=24 (bottom)*

To use Bayes' formula, we need to assume a probability function. The next section summarizes the theorem on which much statistical inference and graphical display of error rests.

15.1.4 The Central Limit Theorem

Choosing a discrete or continuous probability distribution for representing a process depends on prior knowledge (however scanty) of the physical system that we assume underlies the process. We have many choices available, of course, and there is not room to discuss them here. However, the widespread use of the **normal** distribution in statistical inference is due to the fact that the sampling distribution of means tends to be normal. More precisely, the **central limit theorem** states that as the number of independent, identically distributed random variables with finite variance σ^2 increases, the distribution of their mean becomes increasingly normal. Furthermore, the variance of the mean decreases proportionally to n ($\sigma_\mu^2 = \sigma^2/n$). The square root of the variance of the mean (σ_μ) is called the **standard error** of the mean.

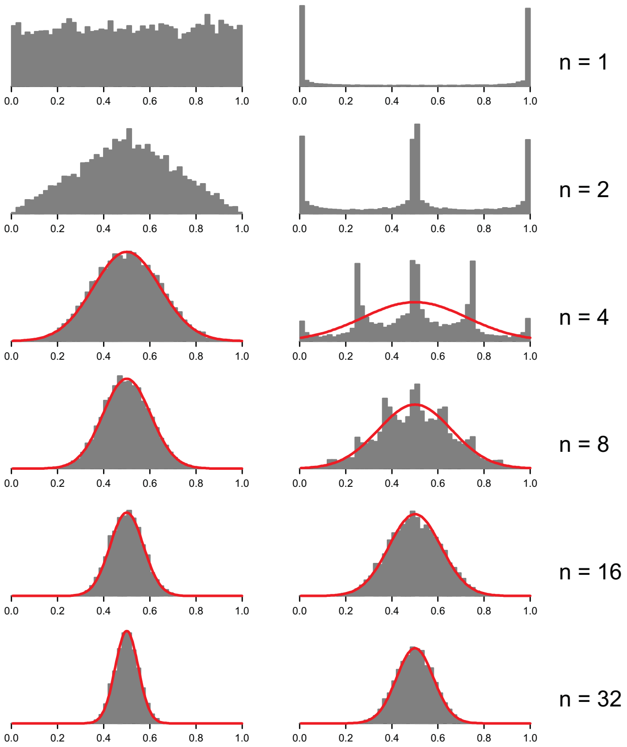


Figure 15.3 *Central limit theorem behavior, uniform left, beta(.1, .1) right*

Even if variables are not normally distributed, means based on them become more normally distributed as n increases. Figure 15.3 illustrates this convergence. We chose two distributions that are far from normal — a *uniform*(0, 1) and a *beta*(0.1, 0.1). We generated 10,000 cases on n variates from these distributions ($1 \geq n \geq 32$). We produced the histograms by averaging n of these variates in each row of the figure. We have superimposed normal curves for $n \geq 4$. Notice that the spread of these distributions decreases as the square root of n.

If you see a symmetric error bar on the mean of a sample of data represented in a chart, it is most likely based on the percentage points of a normal distribution (or a t distribution if n is small). The justification for this usage is the central limit theorem. This practice is not always secure, however. Means based on asymmetric and/or bounded distributions converge more slowly to the normal. And interactions of multiple skewed variables can get nasty.

15.1.5 Interpreting Probability

So far, we have not related the probability function $P(.)$ to supposedly random events in our world except by analogy. Kolmogorov's axioms and the axioms of set theory give us a foundation for computing functions of probabilities, but relating our results to real-world events leads to interpretive problems. Without getting into philosophical and mathematical thickets, we will nevertheless try to summarize three prevailing approaches to interpreting probabilities. There are other interpretations and there are many nuances, but we will restrict ourselves to short summaries in order to save space. The important thing is that these approaches lead to different interpretations of uncertain data and, importantly for our purposes, they lead to different ways of representing events and their associated probabilities in graphics.

15.1.5.1 Combinatoric

The combinatoric interpretation is often called **classical**, probably because some form of it was employed in Classical and Enlightenment analyses of games of chance. In this view, we define S to contain n elements, each equally likely. The probability associated with an element of S is defined to be $1/n$. We use set-theoretic operations to combine these primitive probabilities and calculate the probability of an event in E. Computing a probability of an event is thus a matter of enumerating subsets of the sample space. For example, we define a fair die as a cube with one of six different values on each face. The probability associated with each value is 1/6.

One criticism of this approach focuses on the equal-likelihood premise. In short, the definition is circular, because it uses the concept of probability in its premise. There have been refinements that ameliorate this problem, but they have flaws as well. Nevertheless, statisticians, gamblers, and card players routinely use enumeration to calculate probabilities in real-world scenarios.

15.1.5.2 Frequentist

The frequentist interpretation is based on the long-term behavior of a process. In this view, probability is the ratio of the number of times an event occurs in a series to the total number of trials in the series. Computing a probability precisely would therefore involve an indefinitely repeated experiment.

One criticism of this approach rests on the ambiguity of the repeatability assumption. This assumption is similar to, and shares flaws with, the equal likelihood assumption of the classical view. In practice, we can never repeat *all* the circumstances involved in, say, flipping a coin. Moreover, even if we could repeat them, frequentists cannot make statements such as "the probability that this coin will be a head on the next toss is ½" because such statements refer to a particular event rather than a process. For similar reasons, frequentists would say that statements like "the probability that it will rain tomorrow is extremely small" are meaningless.

15.1.5.3 Subjective

The subjective interpretation of probability avoids these problems through a radical definition. Subjectivists say probability is a measure of the degree of belief that an event will occur. Thus, probability is the primitive operand of the functions that a rational decision-maker uses when placing bets or assessing risks. Statisticians who employ some variant of this definition are usually called Bayesians (although not all Bayesians employ a purely subjective interpretation of probability).

The essence of this definition constitutes its principal weakness, according to Bayesians' opponents (usually frequentists). They say subjective probability is not objective (!). In other words, there is no assurance that subjective probabilities will obey Kolmogorov's axioms the way classical or frequentist probabilities do. Bayesians have responded to these objections with several arguments. First, subjective doesn't mean psychological. Subjective probability estimates may be produced by a person, an animal, or an automaton. Second, subjective doesn't mean arbitrary. In fact, rules for the behavior of a rational decision-maker can be defined (without circularity) so that the Kolmogorov axioms are satisfied. A decision-maker following such rules is said to be **coherent**. Third, applications of Bayes' theorem can be made remarkably insensitive to variations in prior probability distributions, at least in simple cases (although not necessarily for all univariate or, especially, multivariate probability distributions). To deal with these exceptions, Bayesians sometimes recommend testing the sensitivity of posteriors to variations in subjective priors before promoting models.

15.1.6 Uncertainty Intervals

We now have the basic framework for computing errors in order to represent uncertainty graphically. In this section we will show how this is done for the most common application — constructing uncertainty intervals from the normal distribution of means. We begin with the frequentist approach.

15.1.6.1 Confidence Intervals

Frequentists construct a **confidence interval** to express the precision of a parameter estimate. To construct a confidence interval, the frequentist assumes that observed data $\{x_1, ..., x_n\}$ are sampled from a continuous theoretical probability function $f(.|\theta)$ having one or more unknown fixed parameters θ, $\theta \in S$. To approximate θ, the frequentist seeks a function $\hat{\theta}$ of the data, called an **estimator**, that ideally should have the following properties:

- **Sufficiency**. Given the estimator, there is no other estimator that provides additional information about θ.
- **Consistency**. As the sample size increases, the estimator should converge in probability to θ.
- **Efficiency**. For a given sample size, the estimator should have the smallest variance among all unbiased estimators.
- **Unbiasedness**. The expected value of the estimator is θ.

For the normal distribution, $f(.|\theta) = f(x|\{\mu, \sigma\})$. Several theorems regarding the above properties lead to the choice of the sample mean ($\bar{x}$) as an estimator of μ and the sample standard deviation (s) as an estimator of σ.

To construct a confidence interval on μ, the frequentist assembles the interval $[\bar{x} - s/\sqrt{n}, \bar{x} + s/\sqrt{n}]$ from the parameter estimates. For convenience, this interval is centered on $\bar{x}$, which makes it the shortest interval bounding approximately 68 percent of the area under the normal distribution. To change the width of the interval (to, say, 95 percent bounds), the frequentist uses some multiple of the estimate of the standard error of the mean.

The frequentist says something like the following about this interval: "Approximately 68 percent of a large number of confidence intervals constructed from replications of this procedure on samples from the same distribution will cover the unknown parameter μ." Statements about the probability of the parameter falling inside a given interval, while frequently made by frequentist practitioners, are meaningless. The parameter is not a random variable and a given interval is an instance, not a process.

Figure 15.4 shows an example of this procedure repeated 100 times on samples of size 100 taken from the standard normal distribution. We have drawn a red horizontal line to highlight the fact that 30 intervals, including the first, fail to cover μ. This result is not far from our expectation of 32.

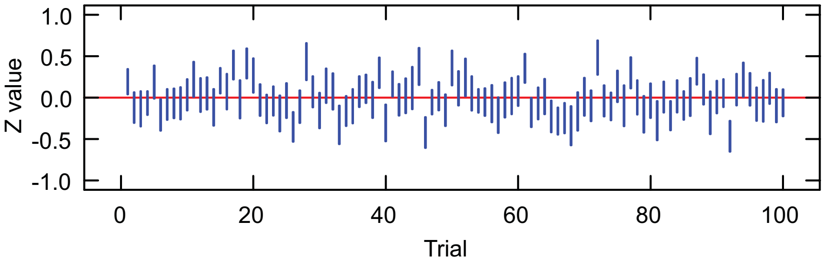


Figure 15.4 *Confidence intervals*

15.1.6.2 Credible Intervals

Bayesians construct a **credible interval** to express the range of their belief in an outcome. In contrast to frequentists, who assume θ is a fixed parameter associated with an estimator, Bayesians assume θ is a random variable with probability density $f(\theta)$. The prior density $f(\theta)$ usually has relatively large variance, unless a Bayesian has information from previous experiments to reduce the uncertainty.

Bayesians compute the likelihood $l(x|\theta)$ from the sample data. Sometimes they use the same sufficient estimators employed by frequentists, but they often use other functions of the data in order to get a likelihood. For a Bayesian, the data values *themselves* are sufficient statistics. Using a computed likelihood, Bayesians revise the prior belief expressed in $f(\theta)$ by computing the posterior density $f(\theta|x)$. Bayesians construct the credible interval $[f(\theta|x)_{\text{lower}}, f(\theta|x)_{\text{upper}}]$ directly from the percentage points of the posterior distribution.

Figure 15.5 shows an example of this procedure applied to the same data used in Figure 15.4. Because we used a standardized normal likelihood and a uniform prior for the first interval, we get the same result as the frequentist. The remaining intervals are quite different from the corresponding confidence intervals, however. They were calculated using the posterior distribution from the previous interval calculation as the prior distribution for the current.

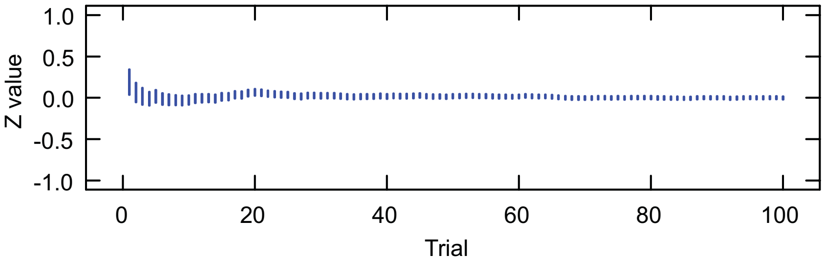


Figure 15.5 *Credible intervals*

Each interval constructed in the sequential procedure shown in Figure 15.5 relies on an **informative** or **sharp** prior distribution derived from the posteriors of previous experiments. This accounts for the sharpening of the estimates over time. Depending on the situation, however, Bayesians may employ other types of prior distributions when constructing credible intervals. If there is little experimental or theoretical information available for deriving priors (as in frontier physics experiments), then Bayesians may employ an **uninformative** or **diffuse** prior. Uninformative priors have the following features: (1) the prior density is **proper** (it integrates to unity); (2) within the region covering the data, the prior density is relatively uniform; and (3) most of the posterior density is concentrated inside the region covering the data. These conditions allow the information in the data to dominate prior information (see the arcsine prior in the lower panel of Figure 15.2). Alternatively, Bayesians may use a **reference prior** that serves as a non-controversial standard in an application domain. In physics, for example, certain parameters are strongly constrained by theory to lie in relatively small regions.

Each of these specialized classes of prior is evidence of Bayesians' desire to remove arbitrariness and sensitivity from the choice of a prior. In real applications, these attempts are more or less successful. In any case, the sequence shown in Figure 15.5 illustrates the trial-by-trial convergence that one might expect when scientists apply Bayesian methods to replicated experiments. See Figure 15.22 for an example taken from applied physics.

By contrast, frequentists have developed a field called **meta-analysis** (Hedges and Olkin, 1985) to evaluate the credibility of the results of repeated frequentist-analyzed experiments. Meta-analysts pool statistical results over separate experiments, often using the same algorithms (such as inverse-variance weighting) that Bayesians employ. Meta-analysts use these pooled estimates to construct confidence intervals on the aggregated outcome. Although there are disagreements over this issue, some would say that more routine application of Bayesian methods by scientists would obviate the need for this type of retrospective analysis. An alternative to this view is a Bayesian approach to meta-analysis of existing studies (DuMouchel, 1990).

For a non-technical introduction to Bayesian methods see Edwards, Lindman, and Savage (1963) or Mosteller and Tukey (1968). For statistical texts, see Box and Tiao (1973), Hartigan (1983), or Gelman *et al.* (2003). Because we do not identify completely with either the frequentist or Bayesian camps, we are skeptical of some claims made by a few radical proponents in either group. It is fair to say that Bayesian methods rest on a firmer philosophical foundation and that frequentist methods are more objective. But is unfair to say that priors are necessarily arbitrary or that frequentist methods are always noninformative. There are many statisticians who use a Bayesian approach to some problems and a frequentist approach to others. And there are some, such as Berger (1985, 1997) who have gone a long way toward developing a synthesis.

15.1.7 Model Error

We have seen how statisticians represent parameter error. How do they represent model error? The left panel of Figure 15.6 shows an example of a fixed regression or analysis of variance model $Y = f(X) + \varepsilon$, where $\mathrm{E}(\varepsilon) = 0$ and $\mathrm{Var}(\varepsilon) = \sigma^2$. The right panel of Figure 15.6 shows an example of a joint bivariate model $\boldsymbol{X} = \boldsymbol{\mu} + \boldsymbol{\varepsilon}$, where $\boldsymbol{X} = (X_1, X_2)$ and $\boldsymbol{\mu} = (\mu_1, \mu_2)$ and $\boldsymbol{\varepsilon} = (\varepsilon_1, \varepsilon_2)$ and $\mathrm{E}(\boldsymbol{\varepsilon}) = (0, 0)$ and $\mathrm{Var}(\boldsymbol{\varepsilon}) = \Sigma$, where Σ is a covariance matrix. These two models, or simple linear and nonlinear extensions of them, are common and long-standing in the scientific literature. Of course, there are numerous other models suited to either Bayesian or frequentist analysis, including probabilistic graphs, stochastic systems, tree classifiers, and so on.

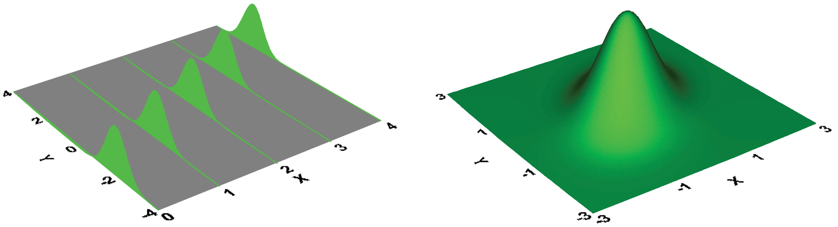


Figure 15.6 *Conditional and joint models*

For Bayesians, model error is represented in the spread of the posterior distribution $f(\theta|\boldsymbol{X})$, where $\boldsymbol{X}$ represents the data tuples and θ represents the parameters in the model. For frequentists, error is represented in the spread of the random variable ε. In either case, the differences between the observed values and the modeled values, called **residuals**, help us diagnose our model by revealing whether the error distribution posited for the model is plausible for the data.

Following standard practice leads to models that fit a sample of data in some optimal sense (least squares, maximum likelihood, minimum risk, ...) based on aspects of the data and assumptions about the process that generated the data. We call the error associated with this fit **in-sample error**. In contrast, we call the error associated with applying this fit to a *new* sample from the same population **out-of-sample error**. A common example involves the error of a forecast computed on a time series (in-sample) and the error from applying the forecast model to data from a non-overlapping or subsequent interval of the series (out-of-sample). Another example involves the error of a regression predicting grades for a randomly selected group of college students (in-sample) and the error resulting from applying the same regression equation to a different randomly selected group (out-of-sample). In-sample error does not provide an estimate of how well a model will do in a new sample. Estimating out-of-sample error requires a new sample (to test the model) or, in certain circumstances, a specialized statistical function of in-sample error. Hastie *et al.* (2001) cover these issues in more detail.

15.1.8 *Resampling*

Several circumstances can thwart our use of parametric models to estimate error. Traditionally, researchers resorted to nonparametric methods in these circumstances, although computers now make several alternatives more attractive. We will describe two.

On the one hand, we might know the distribution that generated the data but be unable to find a closed-form expression for a posterior distribution. In this case, we can resort to **Monte Carlo** methods. We generate pseudo-random samples from the hypothesized distribution and use these samples to estimate functions of the distribution. As an example, Wilkinson and Dallal (1981) used Monte Carlo sampling to analyze the null distribution of the squared multiple correlation coefficient under stepwise regression. In a major recent development in statistics, Bayesians have begun to employ Markov Chain Monte Carlo methods (MCMC or MC^2) for models that are not analytically tractable. Robert and Casella (2004) discuss these methods.

On the other hand, we may not know the distribution that generated the data. In this case we can employ a procedure called the **bootstrap** (Efron, 1979). Given a set of (possibly multivariate) data $\boldsymbol{X} = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_n\}$, we randomly draw *with replacement* a sample dataset $\boldsymbol{X}^*$ of size n. We do this m times (m is usually in the neighborhood of a few hundred). Then we fit our model to each of the bootstrap datasets and derive our error estimate from the bootstrap residuals. While there are qualifications, Efron and Tibshirani (1993) show that the following equation (derived from the usual sample variance estimator)

$$\hat{\mathrm{Var}}\,[f(\boldsymbol{X})] = \frac{1}{m-1}\sum_{i=1}^{m}\left(f(\boldsymbol{X}_i^*) - \frac{1}{m}\sum_{j=1}^{m} f(\boldsymbol{X}_j^*)\right)$$

is in general an estimate of the variance of a function of the data under the empirical distribution for the data $\boldsymbol{X}$. Fortunately for graphical use, the qualifications are usually negligible.

The bootstrap's generality underlies its usefulness. For linear models with independent-normal error assumptions, the bootstrap results mirror the standard theoretical results. For many nonlinear functions, however, we can construct error estimates that are otherwise intractable. We can do this for smoothers, kernel density estimates, and other unusual statistics. Examples can be seen in Figure 15.25 and Figure 15.26 below.

15.1.9 *Missing Data*

The methods we have discussed so far require a complete dataset. Missing values are not defined in textbook statistical formulas. And in computational systems they are usually assigned a unique non-numeric value (called **Not a Number** or **NaN**; see Stevenson, 1981). This value produces a non-numeric result when used in arithmetic operations, so it is not ignorable. Furthermore, statistics packages tend to deal with missing values in an ad-hoc manner. Thus, we need well-defined methods for accommodating missing values.

Missing values occur in datasets for many reasons. In general, the processes that generate missing values can be grouped under systematic, random, or a mixture of systematic and random effects. Estimating or imputing missing values requires an understanding of the process that produced them. Graphics can help in this understanding, but there is no substitute for prior knowledge. And, as Little and Rubin (2002) make clear, to think about missing values is to think about models.

Systematic effects include structural components that produce missingness by design. For example, a sampling scheme or fractional factorial experimental design may cover only a subset of possible category combinations. Analyzing data based on these schemes requires structural models that incorporate the scheme itself. Systematic effects also include fixed response biases, such as responses unavailable to subsets of subjects or units measured. This happens with conditional responses (*e.g.,* "If you have lived here longer than 3 years ..."), as well as conditional attributes (*e.g*., options lists for cars). Conditional statistical models are needed to analyze data from these sources.

Random effects include factors such as random dropouts, equipment failures, and interference with data collection. If values are missing at random (MAR), then we can use the **expectation maximization** (EM) algorithm to estimate them (Dempster, Laird, and Rubin, 1977). Roughly speaking, the algorithm consists of iterative cycles: (1) replace missing values by estimated values, (2) estimate parameters of the model using current estimated values, and (3) estimate missing values using current parameter estimates. Steps 1 through 3 are repeated until estimates converge. A typical model might be based on a joint multivariate normal distribution, though the algorithm is general enough to accommodate a large variety of models. Little and Rubin (2002) discuss these methods in more detail and relate EM to maximum likelihood and Bayesian estimation.

Estimating the missing values in a dataset solves one problem — imputing reasonable values that have well-defined statistical properties. It fails to solve another, however — drawing inferences about parameters in a model fit to the estimated data. Treating imputed values as if they were known (like the rest of the observed data) causes confidence intervals to be too narrow and tends to bias other estimates that depend on the variability of the imputed values (such as correlations). Consequently, Rubin (1978) developed a procedure, called **multiple imputation**, that works like the bootstrap. We estimate *m* sets of missing values (using *m* different models and/or sampling methods). We then analyze each set of data as if it were complete (known). Finally, we combine the estimates to construct confidence intervals in a manner similar to that used for the bootstrap. Rubin (1987) discusses this in more detail. We will illustrate graphics that are related to these methods in Section 15.3.6.

15.2 Psychology of Uncertainty

Given a prior opinion concerning the probability of an outcome and new evidence updating the information on which that opinion is based, Bayes' theorem is the optimal way to revise that opinion. Given the exalted anthropology found in the writings of the Psalmist ("little less than a god ..."), Shakespeare ("in apprehension, how like a god!"), and Darwin ("his god-like intellect ..."), one might expect that man (*sic*) is by nature a Bayesian.

No. Human decision-making in the face of uncertainty is not only prone to error, it is also biased against Bayesian principles. We are not *randomly* suboptimal in our decisions. We are *systematically* suboptimal. Until psychologists, led by Amos Tversky and Daniel Kahneman, began to study decision-making experimentally, social scientists thought that suboptimality in human decision-making was due to either fallibility or ignorance. Numerous studies of decision-making now make it clear that the mistakes novices *and* experts make are due mainly to innate biases. We will summarize these biases in the following subsections. More detailed reviews are available in Kahneman, Slovic, and Tversky (1982) and Hastie and Dawes (2003). See also Hogarth (1975) and Tversky (2003).

15.2.0.1 Misuse of Priors

Decision-makers process priors incorrectly in several ways. First, people tend to assess probability from the representativeness of an outcome rather than from its frequency. When supporting information is added to make an outcome more coherent and congruent with a representative mental image, people tend to judge the outcome more probable, even though the added qualifications and constraints by definition make it less probable. Tversky and Kahneman (1983) called this **conjunction probability error**.

Second, humans often judge relative probability of outcomes by assessing similarity rather than frequency. For example, when asked to rank occupations by similarity and by probability, subjects produced the same rankings in both cases; the rankings were substantially wrong in the case of probability (Kahneman and Tversky, 1973).

Third, when given worthless evidence in a Bayesian framework, people tend to ignore prior probabilities and use the worthless evidence (Kahneman and Tversky, 1973). The existence of superstition based on artifactual correlations is well known; what is surprising is that people would favor uninformative evidence over informative priors. Moreover, people often ignore prior probabilities even when they are explicitly and authoritatively given them. For example, when told two classes have different base rates, people ignore this information and assume they have similar rates if the classes share enough common features.

Fourth, people often judge the frequency of a class on the basis of availability of typical instances. For example, Tversky and Kahneman (1973) found that most subjects think there are more English words beginning with the letter *R* than words having *R* in the third position (not true). This bias is probably due to the ease of recalling a word beginning with *R* and the difficulty of recalling one having *R* in the third position. Tversky and Kahneman have conducted numerous other experiments showing similar biases due to the availability in memory of prototypical instances of a class.

15.2.0.2 Misuse of Likelihoods

People often ignore priors, but at least as often, they ignore or discount evidence. Edwards (1968) has denoted the systematic underestimation of the impact of evidence **conservatism**. People often hold on to prior beliefs in the face of conflicting evidence.

Although Edwards and most statisticians and economists have believed people behave approximately (if conservatively) like Bayesians, the experimental evidence is more consistent with our not being Bayesians at all. In fact, people (novices and experts alike) often confuse likelihoods with posteriors. For example, many well-trained medical doctors have been found to assume that the probability of cancer given a positive X-ray is equivalent to the probability of a positive X-ray given cancer (Eddy, 1982).

15.2.0.3 Insensitivity to Sample Size

People usually pay attention to sample proportions but often ignore sample sizes on which those proportions are based. This has been called belief in the **law of small numbers** (Tversky and Kahneman, 1971). Examples are scientists who overestimate the power of small-sample experiments to demonstrate experimental effects, who are overconfident in the replicability of results found in small samples, and who rarely attribute deviations from expectations to sampling variability. This bias is so powerful that it is often resistant to training. Amos Tversky was famous for devising simple judgment scenarios that frequently fooled statisticians at colloquia.

15.2.0.4 Nonlinearity

We discussed nonlinear psychophysical functions involving aesthetic stimuli in Chapter 10. Not surprisingly, the function relating actual frequencies to judged frequencies for a variety of events is nonlinear as well (Slovic, Fischhoff, and Lichtenstein, 1982). In addition, there are regression effects in these judgments — infrequent events are judged relatively more frequent and frequent events are judged relatively less frequent. We cannot assume that subjective probability is proportional to observed frequency, even in the most controlled and objective circumstances.

15.2.0.5 Implications for Graphics

The overwhelming experimental evidence that we are not innate Bayesians does not discredit the use of Bayes' theorem in practice. We believe it leads to the opposite conclusion: decision-makers need statistical tools to formalize the scenarios they encounter and they need graphical aids to keep them from making irrational decisions. The analogy with visual illusions is apt. We need to design aircraft or automotive displays to counteract visual illusions and we need to design statistical graphics to counteract probability illusions.

What graphical aids are available? We used a simple one in Figure 15.1 to illustrate Bayes' theorem — the mosaic plot. We present additional ones in the next section. Others have yet to be discovered, because the use of graphics for decision-making under uncertainty is a relatively recent field. Underlying all of this, of course, is our belief in the importance of representing error in statistical graphics. We need to go beyond the use of error bars to incorporate other aesthetics in the representation of error. And we need research to assess the effectiveness of decision-making based on these graphics using a Bayesian yardstick.

15.3 Graphing Uncertainty

We will begin with a section on the use of aesthetics to display error. Then we will show examples of specialized uncertainty displays.

15.3.1 Aesthetics

It is probably the scientific and geographic visualization communities that have given the most attention to devising and evaluating aesthetics for representing uncertainty. MacEachren (1992) added blur and transparency to the Bertin visual variables for this purpose. Other geographers have experimented with size, shape, color, texture, and other visual variables. Pang *et al.* (1997), Buttenfield (2000), Djurcilov (2001), Pang (2001), and Johnson and Sanderson (2003) cover recent developments in the visualization community. We will introduce some examples in this section.

15.3.1.1 Position

Perhaps the simplest representation of error is to plot an error estimate against some other variable. Instead of plotting means, we can plot standard deviations. Nearly as simple is to plot intervals such as standard errors or confidence intervals. We will show several examples of these types of plots. First, however, we will use position on the plane to represent confusability directly.

We featured data from Rothkopf (1957) in Figure 9.8. In the Rothkopf study, 598 subjects were presented with the sound of a Morse code letter or digit, followed by the sound of another coded letter or digit. Subjects answered "same" or "different" to each possible pair. The data matrix consists of one row for each leading code and one column for each trailing code. The data entries of this asymmetric square matrix consist of the percentage of "same" answers for each ordered pair.

Figure 15.7 shows a nonmetric multidimensional scaling of the Rothkopf data after symmetrizing (averaging complementary off-diagonal elements). We have omitted the digits for simplicity. The plot reveals that letters represented by few primitives (like E and T) are confused less often with letters represented by more (like X and Z). Also, letters represented by dots (like S and H) tend to be confused less with letters represented by dashes (like O and M).

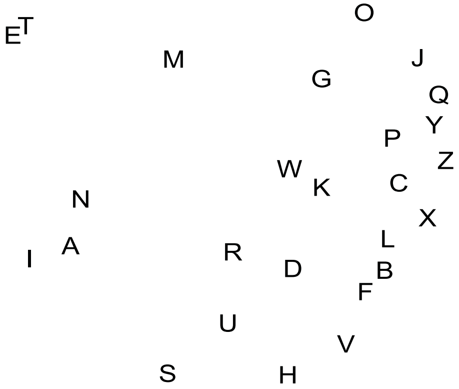


Figure 15.7 *Nonmetric MDS of Rothkopf data*

We can get a better idea of the structure of this configuration by looking at the codes themselves. Figure 15.8 shows this plot. Shorter codes are to the left and dashes are toward the top.

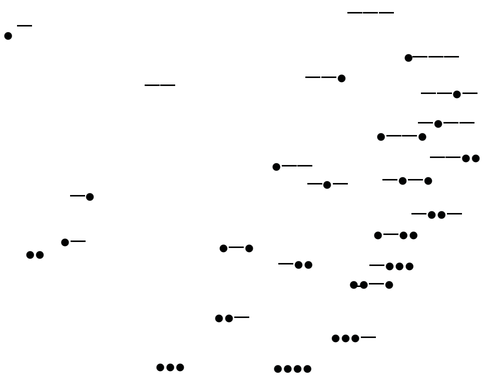


Figure 15.8 *Nonmetric MDS of Rothkopf data plotted with codes*

We discussed in Chapter 13 how Tversky (1977) explained the inappropriateness of the Euclidean metric for representing asymmetric similarity data. Nevertheless, if the asymmetry ($|s_{ij} - s_{ji}|, \forall i, j$) is not severe, an MDS plot can be quite useful for getting a first look at confusability and other quasi-measures of similarity. Let's now look at the ordered data.

To construct Figure 15.9, we fit a log-normal distribution to the set of confusions for each leading letter. The intervals are mildly asymmetric about their modes because of our using the log-normal distribution. We could do the same plot conditioned on the second letter in each pair, but we will not show it here.

In the lower panel, we have sorted the confusions by magnitude. In devising his code, Samuel Morse counted the number of slugs in each compartment of a typecase and assigned the simplest codes to the letters with the highest counts. He took the counts to be indicative of frequency-of-occurrence of letters in the English language. This procedure would tend to make typical messages shorter, but not necessarily less confusable. The Spearman correlation between Morse's letter frequency rankings and the confusion ranks in the Rothkopf data is only .32.

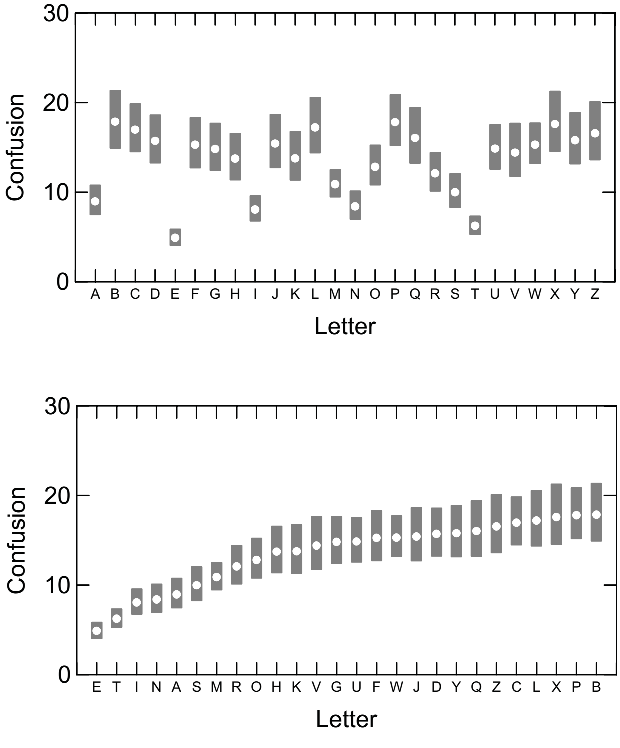


Figure 15.9 *Confusions of Morse Code sounds*

Notice how the sorting improves interpretability if we are interested in confusability, although it is probably best to display both orderings together to facilitate different lookup strategies for different decision tasks. There is a problem with the interpretation of both displays, however. We must keep in mind that the letter confusions are not statistically independent. Each letter involves a reference to one of the other letters and the correlations among the letters are not uniform. We will have more to say about this in Section 15.3.3.

We fit a log-normal distribution in Figure 15.9 because the data were positively skewed and the log transformation yielded approximate normality. This gives us the occasion to present another valuable plot that displays variation in data through position of an estimate of variation. Tukey (1977) devised a **spread-level** plot to help with transformation decisions where a power transformation can yield constant spread across location. To accomplish this goal Tukey plotted a measure of spread against a measure of location. An obvious case would be to plot the standard deviation of subgroups against their means, although Tukey preferred hinge spreads (interquartile ranges) against medians because the influence of outliers is reduced.

Figure 15.10 shows a spread-level plot of the brain weight variable from the sleep dataset. We divided the data into quintiles and computed the median and H-spread within quintile. Section 6.2.4.2 discusses the theory behind Tukey's ladder of power transformations and the related Box-Cox transformation. If we log both axes in the spread-level plot and fit a linear regression to the points, then 1 – slope of the regression line is a good choice for the exponent in the power transformation. The slope of the regression line in Figure 15.10 is approximately 1, indicating that a log transformation would be appropriate. Figure 6.10 supports this choice for these data.

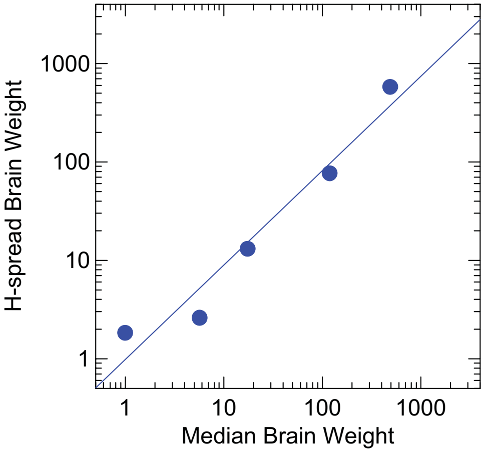


Figure 15.10 *Spread-level plot*

Some might ask whether it is better to proceed directly to the machinery of the Box-Cox transformation and estimate the exponent directly. As with most comparisons that pit statistics against graphics, this begs the question. Neither approach is to be preferred. They are complementary. To support the Box-Cox model, we need to know that the regression of spread on level is linear with good residuals. We need graphics to see that. Also, as Tukey pointed out in his discussion of the original Box-Cox paper, the maximum-likelihood estimation procedure is sensitive to asymmetric outliers. Inspecting spread-level plots on medians and H-spreads is a robust check on the model.

15.3.1.2 Size

We are not finished mining Rothkopf's dataset for examples. We will illustrate the use of size to represent confusability. In Figure 15.11, we construct an eye chart using the confusion values from Figure 15.9. It is a simple chart, but it conveys graphically the distinctiveness of letters when represented by Morse Code. Some of the letters (B, P, X) almost disappear because they are frequently confused with other letters.

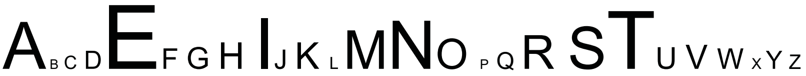


Figure 15.11 *Morse code confusability*

15.3.1.3 Color

Color can be useful for representing confusions or uncertainties. Figure 15.12 shows a color representation of the entire Rothkopf dataset. We have logged the brightness scale to reveal small variations. This display gives us a chance to examine the asymmetries directly. P followed by J is confused more often than J followed by P, for example. Two other pairs (DK and OG) stand out as well. Using hue instead of lightness does little for this chart. Constructing linear color scales is problematic (although see the matrix examples in Chapter 16).

The data mining and classification communities use the term **confusion matrix** to refer to a slightly different kind of dataset. Classification confusion matrices show observed classes against predicted classes. Using color (lightness) to represent the values in these matrices can improve readability when there are many classes. It helps readability to sort rows and columns, as we have done with the Rothkopf matrix.

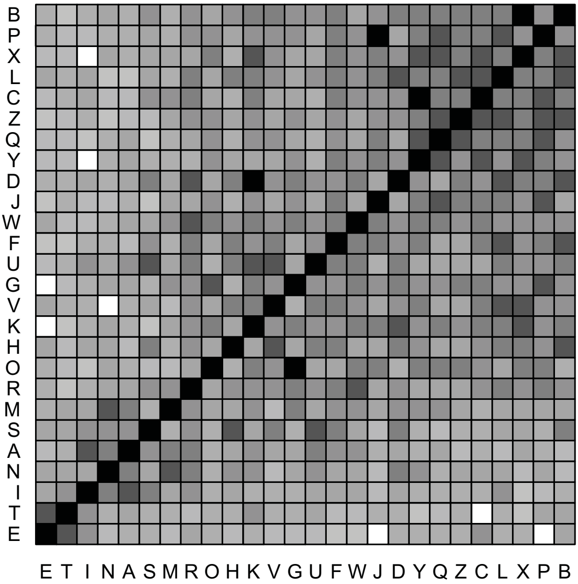


Figure 15.12 *Rothkopf data represented by lightness*

15.3.1.4 Shape

The shapes of icons and glyphs can be modified to incorporate uncertainty features. Figure 15.13 shows an application of vector glyphs modified to represent uncertainty by fanning them (Wittenbrink *et al.*, 1996). The data are measured surface wind vectors from buoys and meteorological stations, interpolated and resampled over the Monterey Bay region on the coast of California.

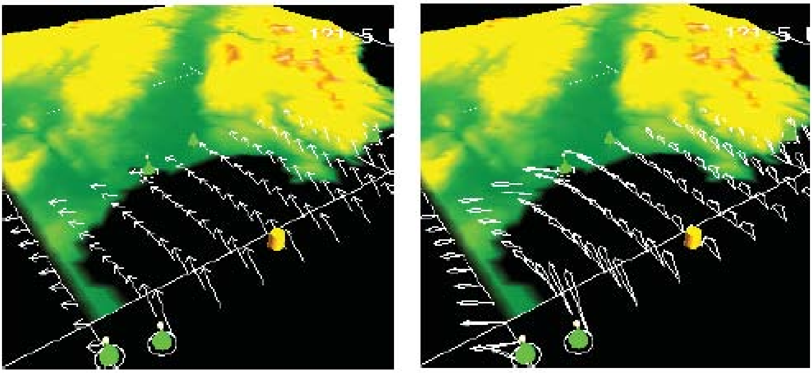


Figure 15.13 *Glyphs (© 2005 IEEE, used with permission)*

15.3.1.5 Blur

Figure 15.14 contains a semicircle or ring chart, similar to a pie chart. The chart is based on the GSS dataset used in Chapter 10. The ring chart represents the number-of-reported-sex-partners variable. Figure 15.15 shows the same chart with blurring based on an arc-sine transformation of the proportions underlying each sector. The edges of the blur correspond to the 95 percent confidence intervals on these proportions.

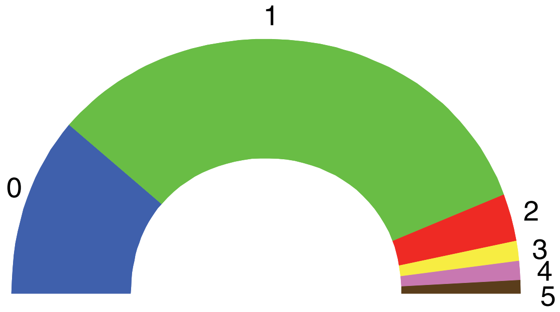


Figure 15.14 *Number of sex partners in last year*

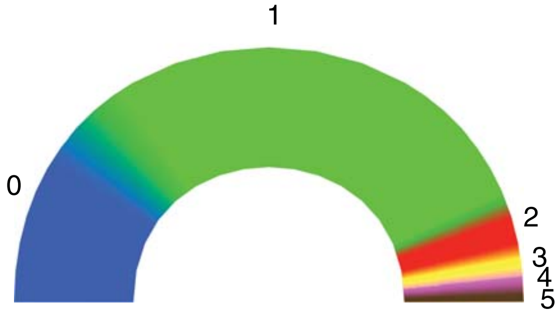


Figure 15.15 *Number of sex partners in last year with blur for error*

As MacEachren (1992) pointed out, blurring can be an especially useful method for representing uncertainty on geographic fields because the eye is prevented from focusing in uncertain areas. Figure 15.16 shows an example from Jimenez *et al.* (2003). This virtual picture uses blur to represent an estimated density front at the edge of a freshwater plume.

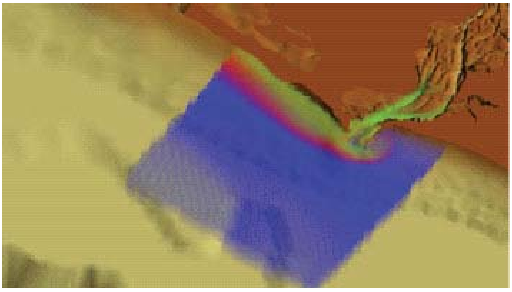


Figure 15.16 *Detail of fresh water plume during ebb tide (green denotes fresh water, blue, salt water, and red, density front) (© 2005 IEEE, used with permission)*

Figure 15.17 shows the use of blur in a model of Crambin, a small protein molecule (Lee and Varshney, 2002). The crisp version on the left is based on a smooth molecular surface model, which is defined as the surface which an external probe sphere touches when it is rolled over the spherical atoms of a molecule. Because atoms vibrate, however, there is statistical uncertainty associated with each atom visible on the surface. Lee and Varshney developed a Gaussian model for this uncertainty and applied it to the surface in the right panel of Figure 15.17. Grigoryan and Rheingans (2004) have followed a similar approach using pixel blurring on tumor surfaces.

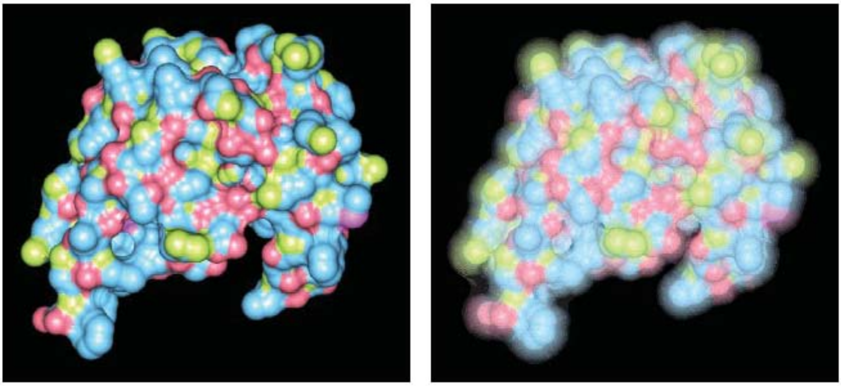
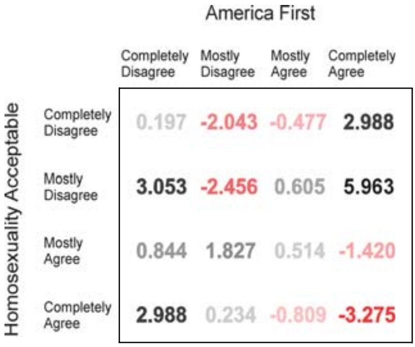


Figure 15.17 *Crisp and uncertain visualizations of Crambin molecule (© 2005 SPIE, used with permission)*

15.3.1.6 Transparency

Figure 15.18 uses transparency to represent the size of residuals. The data are from a national survey on religion and values (Bowman, 2005). The rows of the table represent responses to the question, “Homosexuality should be considered an acceptable alternative lifestyle.” The columns represent responses to the question, “America should help Americans first and the rest of the world later.” The standardized residuals in the table are computed from an independence model, making them contributions to the chi-square test of interaction. Bowman colored the negative residuals red and the positive black, with larger residuals opaque. We see a negative association between the two questions.



America First

Homosexuality Acceptable	Completely Disagree	Mostly Disagree	Mostly Agree	Completely Agree
Completely Disagree	0.197	-2.043	-0.477	2.988
Mostly Disagree	3.053	-2.456	0.605	5.963
Mostly Agree	0.844	1.827	0.514	-1.420
Completely Agree	2.988	0.234	-0.809	-3.275

Figure 15.18 *Transparency used to represent residual values*

Figure 15.19 shows confidence intervals on a multiple regression. We regressed life expectancy on the log of health and military expenditures from the countries dataset. We included an interaction term, which makes the surface nonplanar. The pink surface is the upper 95 percent bound, and the green is the lower 95 percent bound, for the conditional values of the predicted life expectancy. We are able to see both surfaces plus the data points themselves because of the choice of approximately 50 percent transparency for both surfaces.

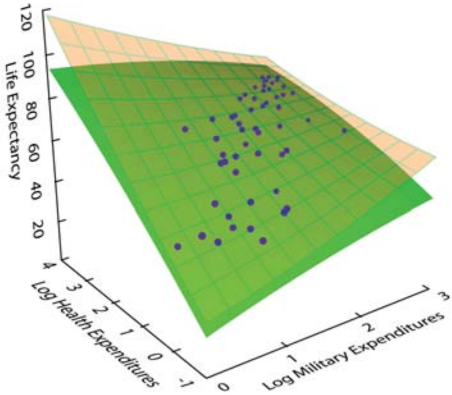


Figure 15.19 *Transparency used to represent uncertainty isosurfaces*

Figure 15.20 shows a 3D kernel density of three variables from the EPA car data (horsepower, carbon monoxide, and carbon dioxide) produced by the *JMP* statistical package. Transparency is used to reveal an inner contour for the data. Wegman and Luo (2002) present other examples.

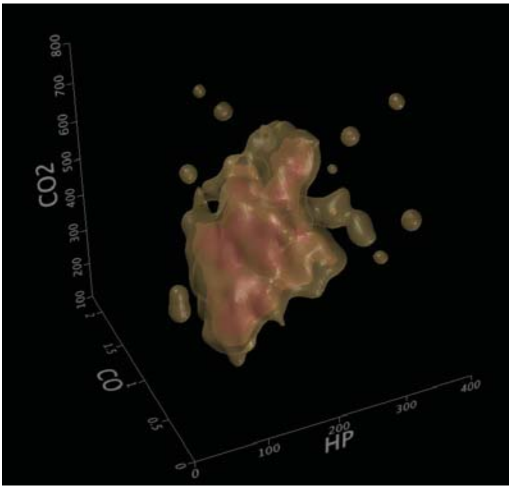


Figure 15.20 *3D kernel density of EPA car data (JMP statistical software, courtesy of John Sall)*

15.3.2 Uncertainty Intervals

We discussed frequentist *vs*. Bayesian approaches to constructing uncertainty intervals in Section 15.1.6. In this section, we will show several examples involving both approaches.

15.3.2.1 Confidence Intervals and Sample Size

A confidence interval can be wide because of small sample size or because of large error. Figure 15.21 illustrates this fact using some variables from the GSS dataset. For this application, we assume that feelings about the Bible can be regarded as continuous (biblical absolutism vs. relativism?) even though the responses are integers. We have computed 95 percent confidence intervals on these reported feelings separately for groups reporting 0 or more sex partners in the last year.

The upper panel represents the square root of sample size with saturation. The lower panel uses size (width). There are advantages and disadvantages to both approaches. The important thing to note is that the intervals increase in size as the number of sex partners increase not because of heteroscedasticity but because the sample sizes decrease.

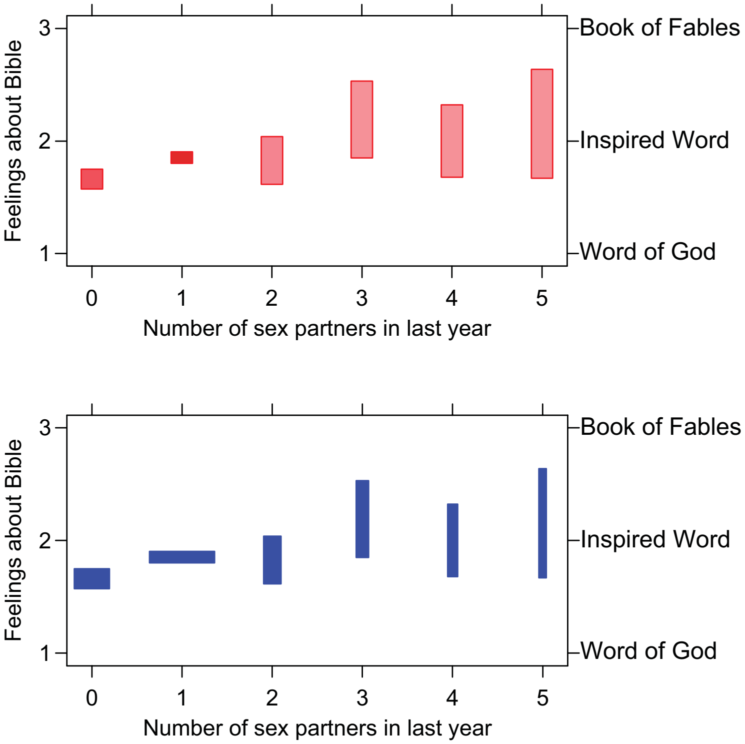


Figure 15.21 *Confidence intervals with saturation (top) and size (bottom)*

15.3.2.2 Confidence Versus Credible Intervals

In Section 15.1.6 we contrasted Bayesian and frequentist approaches to uncertainty intervals. Here we present a real example based on a historical dataset. Gillies (1997) discusses the reasons why the estimation of Newton's gravitational constant has varied considerably over more than two centuries of experiments. We have assembled data from Gillies and from Setterfield and Norman (1987). The latter is a creationist document whose conclusions are based on a statistical artifact, but the authors appear to have diligently compiled data from primary and secondary sources (according to our random checks for accuracy).

In the upper panel of Figure 15.22 we have graphed the 68% individual confidence intervals based on these estimates. In the lower panel, we have graphed 68% credible intervals based on the same data.

As Gillies notes, much of the variation in location and spread can be accounted for by the design of instruments. Thus, there is most likely substantial bias and error in the series. Nevertheless, if we construct credible intervals by inverse variance weighting, then we find that the data swamp the priors by the beginning of the twentieth century.

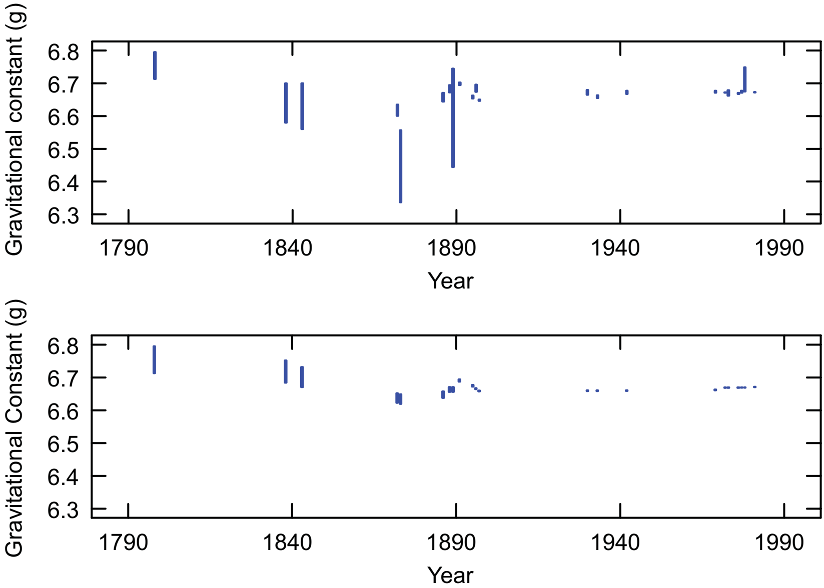


Figure 15.22 *Measurements of gravitational constant (upper panel shows s68% confidence intervals, lower panel shows 68% credible intervals)*

15.3.3 Multiple Comparisons

Cleveland (1984) sampled a variety of articles from 1980–81 scientific journals and computed the percentage of page area devoted to graphics in each article. Figure 15.23 shows these area percentages grouped by discipline for 47 of Cleveland's articles. We have computed 95 percent *t*-distribution confidence intervals for these percents, based on inverted arc-sine transformed proportions (see Section 6.2.5.1).

Analysts often want to make **multiple comparisons** of groups in examples like this one. They compare all possible pairs of means to see if they significantly differ. We can do this informally for Figure 15.23 by seeing whether two confidence intervals overlap. We see that the error bars for the percentage areas of Physical and Social articles do not overlap, for example, so we conclude they are significantly different.

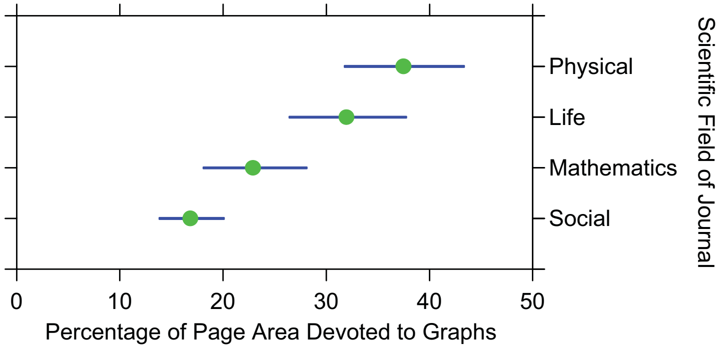


Figure 15.23 *95 percent confidence intervals*

There are problems with this idea. First, using separate confidence intervals for multiple comparisons does not correct for the number of comparisons being made. Let

$$A = \textit{false result on test } A$$

$$B = \textit{false result on test } B$$

Set $P(A) = P(B)$. For multiple comparisons, we define

$P(A) = P(B) =$ the **comparisonwise error rate**

$P(A \cup B) =$ the **experimentwise error rate**

If A and B are independent, then

$$P(A \cup B) = 1 - P(\bar{A})P(\bar{B})$$

and if not (which is more likely on real data), then

$$P(A \cup B) = P(A) + P(B) - P(A \cap B)$$

Either way, the comparisonwise and experimentwise error rates are not the same. The definitions above extend straightforwardly to k comparisons. We can use the **Bonferroni inequality**

$$P(A \cup B) \le P(A) + P(B)$$

to set an upper bound on experimentwise error rate by dividing our desired experimentwise error rate by 2 (or k) for each comparison. The problem with this approach, however, is that the tests become more conservative as k increases.

To remedy this, we can use a simultaneous test method such as Tukey's HSD test (Tukey, 1953). If the counts in the groups differ, we need to adjust the Tukey bounds with the Tukey-Kramer formula (Kramer, 1956). This formula involves a factor of $\sqrt{1/n_i + 1/n_j}$ for each comparison. However, each pairwise comparison requires a different interval in this case, and there is no way to compute lengths for the error bars on the separate means that always satisfy every comparison.

Hsu and Perrugia (1994) addressed these problems in a new display. Figure 15.24 shows a version of their plot for the Cleveland data. Four pairs of orthogonal dotted gray gridlines are drawn so their central intersections align on the vertical axis at the mean values for each of the four groups (P, L, M, S). The other intersections of these gridlines demarcate the *differences* between means on the horizontal scale. Because of the constraint that the grid be a square rotated 45 degrees, the horizontal axis must be scaled accordingly. The lengths of the horizontal colored lines are computed from the Tukey-Kramer formula. We have colored red those intervals that do not contain a contrast or difference value of zero.

The Hsu-Perrugia-Tukey plot is easy to inspect and difficult to understand. It is easy to inspect because each pairwise contrast can be examined separately on the horizontal scale and significant contrasts are readily located. It is difficult to understand because the display involves the Pythagorean theorem (the lengths of the horizontal intervals are proportional to the hypotenuse of triangles in the 45 degree rotated metric of the gridlines) and because the display has dual scales for means and differences. Complex theory does not diminish the usefulness of a plot, however. This graphic is designed for a single purpose and may be the best available display for this application. For alternative multiple comparison displays and further reading on this topic, see Sall (1992), Heiberger and Holland (2004), and Benjamini and Braun (2002).

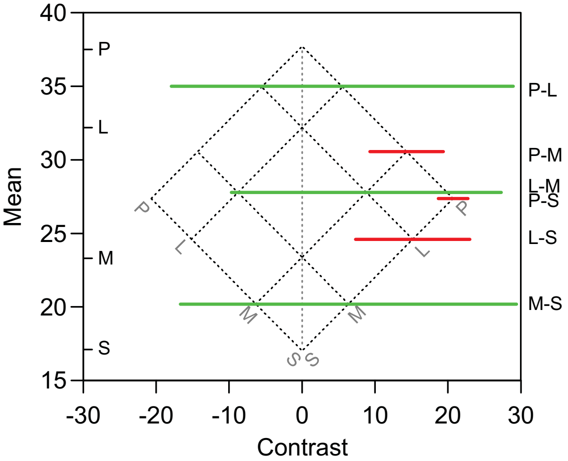


Figure 15.24 *Hsu-Peruggia-Tukey multiple comparison chart*

15.3.4 Resampling

We discussed bootstrapping in Section 15.1.8. As an illustration of an application to graphics, Figure 15.25 shows the bone alkaline phosphatase (BAP) data we used in Figure 7.7. The original authors modeled these data with a single linear regression of BAP on age, noting that BAP increased with age, especially among women. If we do a piecewise regression of BAP on age for women, however, we find that the increase is in level only, not slope. It occurs at the onset of menopause.

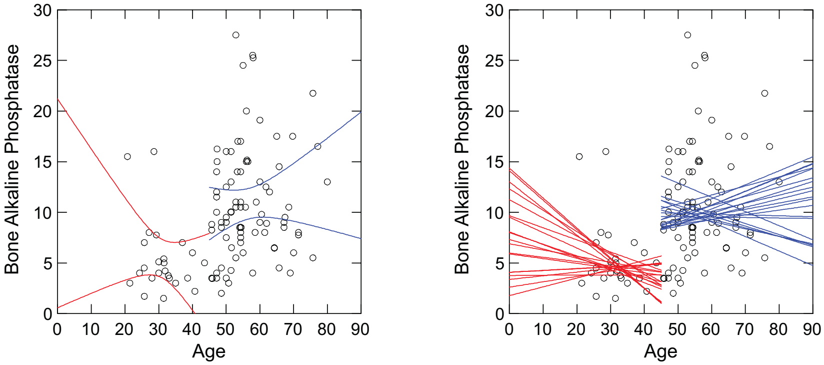


Figure 15.25 *Confidence intervals (left) bootstrap (right) for BAP in women*

The left panel of Figure 15.25 shows the standard 95 percent confidence intervals for the piecewise regression. The right panel shows piecewise regressions for 20 bootstrap samples from the dataset. We would expect to see at most one of these outside the theoretical intervals on the left. The coverage, as expected, is quite good.

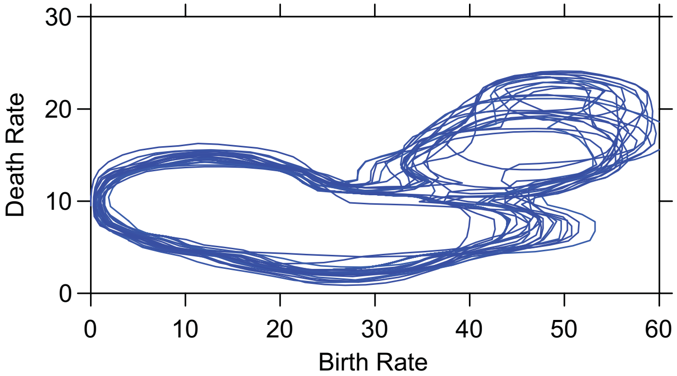


Figure 15.26 *Bootstrapped kernel density contours*

Bootstrapping is more valuable on data and models where analytic estimates are problematic. As an example, Figure 15.26 shows kernel density contours with integrated 95 percent coverage for 20 samples of the birth and death rate variables from the countries dataset. Compare this graphic to Figure 1.1 and to Figure 1.3. We see clearly two clusters of Third World and developed countries.

15.3.5 Indeterminacy

Suppose we have a random vector $\boldsymbol{Y}$ made up of p random variables Y_j, each a linear function of a single random variable X plus random error:

$$Y_j = \beta_j X + \varepsilon_j , \quad j = 1, \ldots, p$$

We assume $\boldsymbol{Y}$ is multinormal with covariance matrix $\Sigma_{yy} = \boldsymbol{\beta}\boldsymbol{\beta}' + \Psi$, where Ψ is diagonal. We also assume X has a standard normal distribution. So far, we have an instance of the multivariate normal linear regression model. However, we will add the assumption that X is *unobservable*. Adding this assumption transforms the multivariate linear model into the **common factor model** (Spearman, 1904).

Given a sample covariance matrix $\mathbf{S}_{yy}$, we can estimate $\boldsymbol{\beta}$ via maximum likelihood. Our estimates will have several sources of uncertainty. One source, of course, is **sampling uncertainty**. The smaller our sample, the less precise our estimate of $\boldsymbol{\beta}$ will be. Another is **rotational uncertainty**. If $\boldsymbol{\beta}$ is multivariate (as in the multiple factor model), we may apply an orthogonal rotation to our estimate of $\boldsymbol{\beta}$ without affecting loss. The same type of uncertainty applies to other orthogonal decompositions such as principal components, and we often use it to our advantage in order to improve the interpretability of exploratory decompositions. Another source is **scaling uncertainty**. We may rescale our estimates (a dilatation) without affecting loss. Factor analysts commonly assume variables are standardized to unit variances in order to avoid this problem. A fourth type of uncertainty is **indeterminacy**. The common factor model contains a mathematical indeterminacy that is of a different kind from the statistical uncertainties we have encountered in this chapter. We will illustrate this graphically.

Figure 15.27 shows a fit of a single common factor model to a correlation matrix of the combat data in Figure 3.2. The vectors are plotted in what we call **score space**. Instead of plotting n observations as a set of points in p dimensions (as in a scatterplot), we plot p variables as a set of vectors in n dimensions. Figure 15.27 shows a 3-dimensional subspace of this 93-dimensional score space. The orientation of each symptom vector is determined by the estimate of $\boldsymbol{\beta}$ (the relative depth is cued by the size of the typeface). Because the data are standardized, the angle between any two vectors in this space is an estimate of the observed correlation between the corresponding variables.

We have plotted all the Y variables in the figure. How do we plot X? One approach advocated by most factor analysts is to plug our estimate of $\boldsymbol{\beta}$ into the model and then solve for X as an inverse regression problem. The result is the red vector running through the center of all the vectors — the average of the regressions of the Y variables on X. This is not an estimate of the unobserved X, however. It does not take into account the unobserved error associated with X. If we include the unobserved error, we find there are infinitely many vectors, all lying on the surface of the green cone (or more generally, hypercone) that satisfy the model and its assumptions. The factor scores cannot be uniquely determined from the specification of the model. Wilson (1928) and Guttman (1955) explain this problem in more detail.

If the X variable were observed or were a function of observed variables, then the green cone would collapse to a single vector. An example would be the principal component model. In the common factor model, however, any factor score vector on the surface of the cone will fit the data equally well. This nondeterminacy is not caused by sampling error; it will not go away if we let n go to infinity. It will decrease if the number of observed variables increases (all else being equal), but slowly.

Steiger and Schönemann (1977) recount the history of the factor score controversy. Factor analysts have argued over this problem for over 80 years, often in esoteric and heated language. One can get a flavor of these disputes in the volume 31 (1996) issue of *Multivariate Behavioral Research*. Economists have named a variant of this the **model identification** problem. Aitkin and Aitkin (2004) have resolved the controversy, we believe, in a lucid Bayesian analysis that compares the frequentist and Bayesian interpretations of the problem.

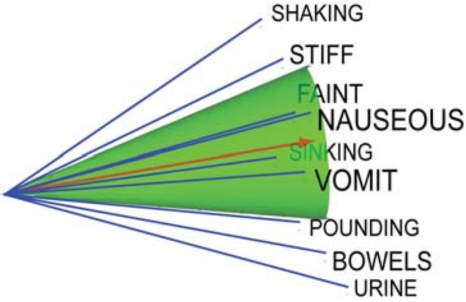


Figure 15.27 *Common factor model*

Incidentally, we transformed the dataset before doing the maximum-likelihood factor analysis. We sorted the rows according to the row sums and differenced each tied block of values across the rows. We then computed the weighted correlation matrix of the first differences. This resulted in a single-factor model. Otherwise, we would have required multiple factors to represent the data. Can you see why? See Section 16.5.2.1 for clues.

Finally, notice that the cone represents indeterminacy in this example. Cones can also be used to represent confidence regions on vectors. Examine Figure 15.13. Would the use of small 3D cones instead of white outlines improve the representation? Is there a trade-off between realism and readability?

15.3.6 Missing Values

Graphics help us to identify missing values on a variable and patterns of missing values in a dataset. Neither task is simple. First of all, computer statistics packages and databases code missing values in a bewildering variety of schemes. Early relational databases had no code for missing because of restrictions in the relational algebra. More recent databases allow a miscellany of approaches. Early statistical packages that were capable of dealing with missing values took one of two approaches. On the one hand, packages such as Data-Text and SAS implemented a unique value designed to propagate itself in mathematical calculations (including three-valued logic). On the other hand, packages such as SPSS used integer numerical values associated with value labels to indicate different types of missingness (No Answer, Not Applicable, . . .). The latter approach fit the needs of survey analysts who had to encode different types of nonresponse.

Numerically coded missing values have the potential for doing great damage when ignored or miscoded because they are usually designed to look like outliers. Codes such as 99, or –999 can profoundly affect numerical calculations, even when sparse in the data. Jasso (1985), for example, analyzed the frequency of marital coitus in the prior month reported by a sample of 2,361 married couples. In her paper she computed a *ceteris paribus* effect of over 100 acts of coitus per month for a prototypical 45-year-old married woman. Reanalyzing her data, Kahn and Udry (1986) discovered four cases with highly unusual values of 88; no other coded value was greater than 63 and almost all were under 40. Since the missing value code for this variable was 99, Kahn and Udry concluded that the peculiar values were keypunch errors. While Jasso (1986) offered a rejoinder, it is most likely that inspecting histograms or tables prior to modeling would have triggered an alarm on Berkson's **intraocular traumatic test** (Edwards *et al.*, 1963).

15.3.6.1 Representing Missing Values

Figure 15.28 shows how we might display such histograms. The 1993 GSS survey contains the question, "How many sex partners have you had in the last 12 months?" Figure 15.28 shows a bar graph of the responses coded numerically and another bar graph of the responses displayed with their value labels. The missing value codes and labels we used for this item were 99 = No Reply (no answer) and <blank> = No Apply (not applicable). Jointly inspecting values *and* labels helps identify incorrect values *or* labels.

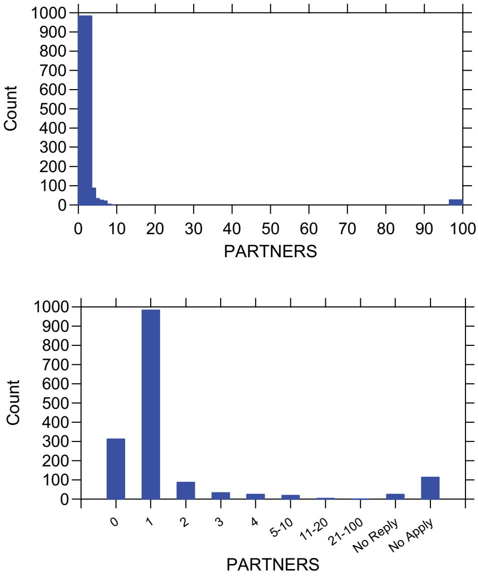


Figure 15.28 *Responses to "How many sex partners have you had in the last 12 months?" on 1993 GSS*

Some software fails to display labels if no value is assigned or fails to display values if no label is assigned, Notice that the upper panel of Figure 15.28 omits the "No Apply" cases because they were coded as blanks. To protect ourselves, we should inspect a sorted table containing *all* values and labels. Table 15.1 shows an example for these data.

Table 15.1 **Reported Sex Partners**

Count	CumCount	Percent	CumPercent	Value	Label
313	313	19.5	19.5	0	0
983	1,296	61.2	80.7	1	1
87	1,383	5.4	86.1	2	2
33	1,416	2.1	88.2	3	3
25	1,441	1.6	89.7	4	4
20	1,461	1.2	91.0	5	5–10
4	1,465	0.2	91.2	6	11–20
1	1,466	0.1	91.3	7	21–100
0	1,466	0.0	91.3	8	>100
0	1,466	0.0	91.3	9	1+
0	1,466	0.0	91.3	95	Several
26	1,492	1.6	92.9	99	No Reply
114	1,606	7.1	100.0	blank	No Apply

15.3.6.2 Displaying Patterns of Missing Data

To analyze patterns of missing values across variables and cases, we need multivariate plots. Unfortunately, placing variables on common scales can be difficult, especially when they involve disparate missing value codes.

Figure 15.29 illustrates one approach. The Gender variable was coded 1 = Male, 2 = Female. The Partners variable was coded as in Figure 15.28, with values greater than 7 changed to missing. Five additional variables were derived from five follow-up questions to the Partners question. Matesex, for example, was derived from the question, “Was one of the partners your husband or wife or regular sexual partner?” Each of the sex variables was coded 1 = Yes, 2 = No, with values greater than 2 changed to missing.

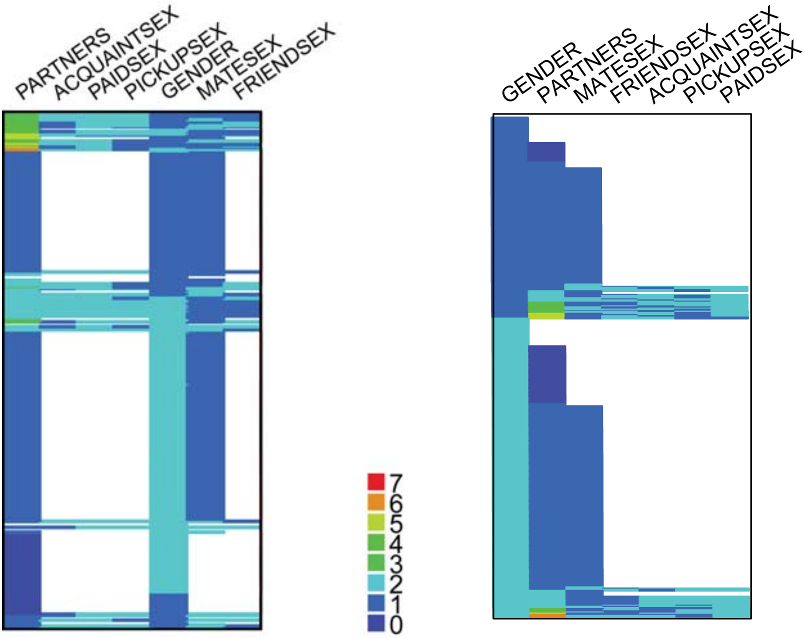


Figure 15.29 *Missing data in 1993 GSS for selected sex variables*

The left panel of Figure 15.29 was produced by a two-way complete linkage cluster analysis (see Section 16.5.4 for further examples). The resulting structure was clean enough to suggest another strategy. We reordered the columns to reflect an *a priori* sexual behavior scale (marital . . . prostitution) and reordered the rows in a nested (lexical) sort. That is, we sorted Gender, then sorted Partners within the Gender blocks, and so on.

A simple structure is evident. Respondents not reporting a partner or reporting zero partners also reported no additional encounters. There do not appear to be any illogical combinations such as reporting more sex partners in the type of partner columns than are shown in the number of partners column.

15.4 Sequel

The next chapter covers analytic methods. Fitting models to data reveals patterns in graphics. Model fitting also helps to simply complex displays. Analysis involves filtering data before display.

16

Analysis

The word *analysis* derives from the Greek ἀνάλυσις, which means to loosen. To analyze means to untangle. Even when we "let the data speak for themselves," we need to untangle some aspect of the data before displaying things in a graphic. The more analytics we can include in the process of displaying graphics, the more flexibility our tools will have.

In graphics grammar, statistics and analytics are different. They occur at different locations in the graphics pipeline. The Statistics component of the pipeline outlined in Chapter 2 is designed to provide flexibility whenever a statistical function can operate on a few variables. Statistics functions such as summaries and smoothers operate on the variables that constitute a graphics frame. Because they are embedded in the data flow, these functions automatically update graphics when data change. Furthermore, they can be interactively modified by controllers to provide different views of the same data.

Sometimes we wish to analyze models based on variables that do not appear in graphics, however. To accomplish this, we insert analytics at the beginning of the pipeline. These analytics receive a varset whose variables correspond to columns of the data source; they output another varset whose variables are used to construct a frame. By putting analytic functions in the same pipeline as statistical functions, we maintain automatic updating and interactive control of our model.

This chapter will present several analytics that depend on graphics for their usefulness. In contrast to conventional graphic displays of parameters or posterior distributions, these analytics are tightly bound to specific types of displays and are usually highly multivariate. We have already seen some examples of this type of analytic in the preceding chapters, such as multidimensional scalings and network diagrams. Here, we will focus on displays not covered earlier or discussed only briefly. We do not intend to cover the field of data mining or statistical analytics here. Each method covered in this chapter is defined by graphic display, inspired by graphic display, or enhanced by graphic display.

16.1 Variance Analysis

The **analysis of variance** was developed by Fisher (1925, 1935) for decomposing the effects of agricultural experiments. Suppose we wish to analyze the effects on plant growth of two different types of fertilizer. We suspect these effects might differ across three crop species, so we fertilize each crop with each fertilizer — six treatments in all. These six treatments comprise all combinations of the levels of two **factors** — species and fertilizer. Fisher demonstrated that conducting the experiment jointly this way provides more information than examining factors alone in separate experiments. Specifically, it gives us the opportunity to assess the **interaction** of the factors. We are able to answer the question of whether the effects of fertilizer type differ among species.

We stated in Section 5.3.1 that chart algebra is related to the algebra used in the design of factorial experiments. In this section, we show more specifically how chart algebra corresponds to the chief operators used in the design of experiments. Introductions to the design notation used for statistical models are Heiberger (1989) and Kutner *et al.* (2003). In the following subsections, we assume a functional model $Z = f(X, Y)$, where Z is a (possibly multivariate) variable. In statistical terms, we sometimes call Z a **dependent variable** and X and Y **independent variables** or **treatments**. In this section, we ignore Z and focus on expressions involving X and Y. These expressions are used to construct statistical models that help to estimate Z. Each example shows the chart algebra notation on the left of each red equivalence expression and the statistical model notation on the right. Note that some symbols (*e.g.*, +) are common to both notations but have different meanings.

16.1.1 Cross.

$$\mathsf{X} * \mathsf{Y} \sim C + X + Y + XY$$

The cross operator corresponds to a fully factorial experimental design specification. This design employs a product set that includes every combination of levels of a set of experimental factors or treatments. The terms on the right of the similarity represent the linear model for fitting fully factorial designs. The terms in the model are

C : constant term (grand mean)
X : levels of factor X (X main effect)
Y : levels of factor Y (Y main effect)
XY : product of factors X and Y (interactions)

An example of a two-way factorial design would be the basis for a study of how teaching method and class size affect the job satisfaction of teachers. In such a design, each teaching method (factor X) is paired with each class size (factor Y) and teachers and students in a school are randomly assigned to the combinations.

16.1.2 6.2.2 Nest.

$\mathsf{X} / \mathsf{Y} \sim C + Y + X(Y)$

The nest operator corresponds to a nested experimental design specification. The term $X(Y)$ represents the series $X \mid (Y = Y_1) + X \mid (Y = Y_2) + \ldots$ The terms on the right of the similarity are

C : constant term
Y : levels of factor Y (Y main effect)
$X(Y)$: X levels nested within levels of Y

Notice that there is no interaction term involving X and Y because X is nested within Y. In a nested design, not all combinations of the levels of X and Y are necessarily defined. An example of a nested design would be the basis for a study of the effectiveness of different teachers and schools in raising reading scores. Teachers are nested within schools when no teacher in the study can teach at more than one school. With nesting, two teachers with the same name in different schools are different people. With crossing, two teachers with the same name in different schools may be the same person.

16.1.3 Blend.

$\mathsf{X} + \mathsf{Y} \sim C + F_{XY}$

The blend operator corresponds to a repeated measures experimental design specification. The terms on the right of the similarity are

C : constant term
F_{XY} : function of X and Y (*e.g.*, $X - Y$)

In a repeated measures design, we predict using functions of a time series. The simplest case is a prediction based on first differences of a series. Time is not the only possible dimension for ordering variables, of course. Other multivariate functional models can be used to analyze the results of blends (Ramsay and Silverman, 1997). An example of a repeated measures design would be the basis for a study of improvement in reading scores under different curricula. Students are randomly assigned to curriculum and the comparison of interest involves differences between pre-test and post-test reading scores.

16.1.4 Smoothing by Design

Smoothing data reveals systematic structure. Tukey (1977) used the word in a specific sense, by pairing the two equations

data = *fit* + *residual*
data = *smooth* + *rough*

Tukey's use of the word is different from other mathematical meanings, such as smooth functions having many derivatives. We smooth data in graphics to highlight patterns in order to make inferences. Graphics for displaying results of factorial analysis of variance have traditionally involved plotting means and standard deviations in paneled or overlaid plots (see Figure 11.4). The factorial structure of most designs can produce rather complex models, however. We need to consider strategies for selecting subset models that are adequate fits to the data and that simplify a display of results. We will discuss one simple approach in this section. This approach involves eliminating interactions (products of factors) in factorial designs.

Interactions are often regarded as nuisances because they are difficult to interpret. Comprehending interactions requires thinking about partial derivatives. A three-way interaction *XYZ*, for example, means that the relation between *X* and *Y* depends on the level of *Z*. And the relation between *X* and *Z* depends on the level of *Y*. And the relation between *Y* and *Z* depends on the level of *X*. Without any interaction, we can speak about these simple relations unconditionally.

One strategy for fitting useful subset models is to search for plausible subset models with as few interactions as possible. In this search, we require that any variables in an interaction be present as a main effect in the model. This restriction reduces the search space for plausible submodels. By using branch-and-bound methods, we can reduce the search even further. Mosteller and Parunak (1985) and Linhart and Zucchini (1986) cover this area in more detail. Figure 16.1 shows a subset model tree for a three-factor design. The further we go down the branches of this tree to select a model, the smoother our fit will be. If we go too far, of course, our fit will be biased and not a good representation of the data. Finding the "right" node is best done with Bayesian or sequential methods.

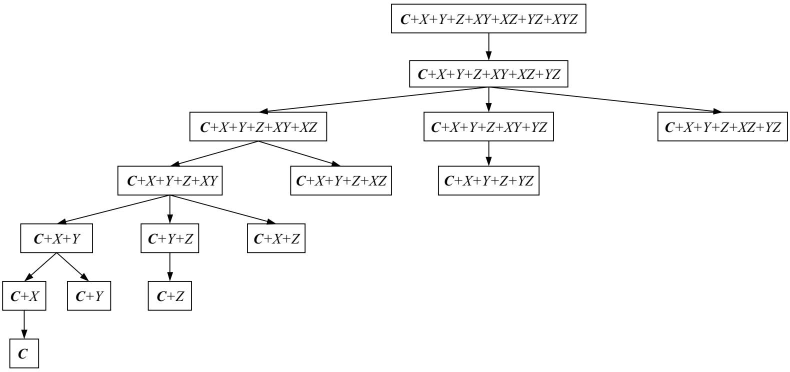


Figure 16.1 *Tree of subset models*

16.1.5 An Example

Figure 16.2 shows NHTSA crash test results for selected vehicles tested before 1999. The dependent variable shown on the horizontal axis of the chart is the Head Injury Index computed by the agency. The full model is generated by the chart algebra H*T / (M*V) * O.

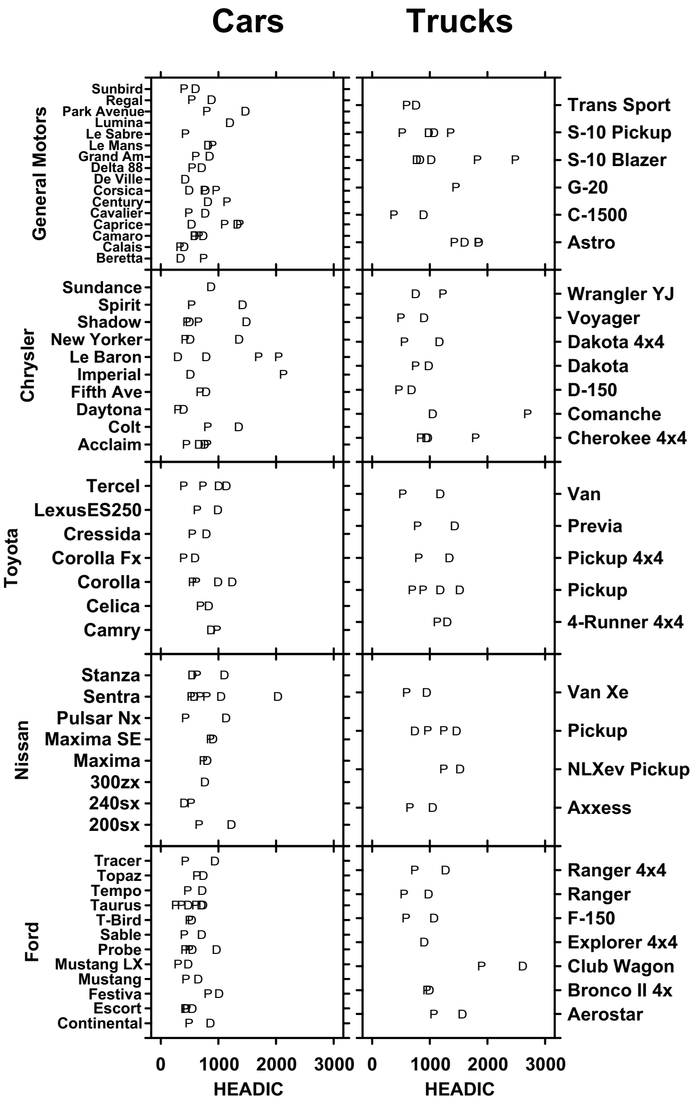


Figure 16.2 *Estimated crash head injury criterion (P=passenger, D= driver)*

The full algebra expression corresponds to the model

$$H = C + M + V + O + T(MV) + MV + MO + VO + OT(MV) + MVO$$

where the symbols are

H : Head Injury Index
C : constant term (grand mean)
M : Manufacturer
V : Vehicle (car/truck)
O : Occupant (driver/passenger)
T : Model

The display in Figure 16.2 is difficult to interpret. We need to fit a model and order the display to find patterns in the results.

Figure 16.3 charts fitted values from the smallest plausible subset model:

$$H = C + V + O + T(MV)$$

Figure 16.3 has several notable features. First, the models are sorted according to the estimated Head Injury Index. This makes it easier to compare different cells. Second, some values have been estimated for vehicles with missing values (e.g., GM G-20 driver data). Third, the trends are smoother than the raw data. This is the result of fitting a subset model. We conclude that drivers should expect more head injuries than passengers, occupants of trucks and SUVs should expect more head injuries than occupants of cars, and occupants of some models should expect more injuries than occupants of others.

We must remember this is a smoothing of the raw data based on the most plausible statistical model for all the data. There are exceptions to the general pattern (*e.g.*, Chevrolet Beretta) that we can detect by plotting residuals to the model. Viewers of smoothed graphs need to be informed in the annotation that the results are smoothed or modeled rather than raw.

The main factor underlying greater head injuries in certain trucks and SUVs is the rigid frame rail that is part of body-on-frame truck design. The frame transmits more kinetic energy to the passengers than does a crushable engine compartment. Popular discussions of the danger of these vehicles have focused on rollovers, but frame rail truck design remains a serious problem for trucks used as passenger vehicles. Some manufacturers have ameliorated this problem by using unibody construction for their passenger trucks and SUVs.

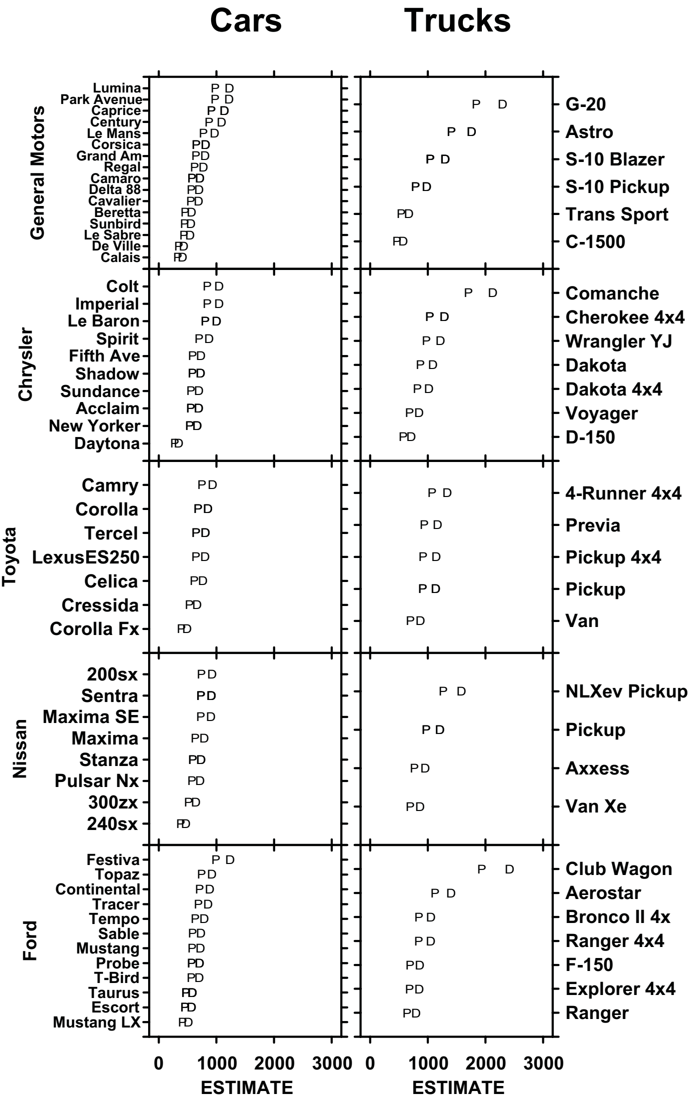


Figure 16.3 *Subset model for crash data sorted by estimate*

16.2 Shape Analysis

Shape analysis brings to mind the field of **morphometrics**, where we attempt to distill 2D or 3D shapes into a few analyzable components or manifolds (Bookstein, 1991; Kendall *et al.*, 1999). Morphometrics arose in biology as a mathematical method for recognizing physical growth paths within species and similarity of form between species (Thompson, 1928). See Section 9.1.9.1 for an example.

In this section, we will focus on an important but little-known application of the analysis of shape to statistical data analysis called **scagnostics** (Tukey and Tukey, 1985; Tukey, 1993). The term is a neologism (a John Tukey hallmark) derived from the words *scatterplot diagnostics*. The Tukeys used scatterplot matrices to discern unusual structures in high-dimensional data.

Before we discuss the machinery used by Tukey and Tukey, however, we will demonstrate the usefulness of SPLOMs for revealing unusual structures in low-dimensional datasets. Figure 16.4 shows a lower-triangular symmetric SPLOM. The data are from Chartrand (1997). This was a national survey of attitudes regarding psychological counseling. There were 3,035 respondents to the survey. The variables in the SPLOM are age, gender, income, number of children, and number of years together with one's partner in a current relationship. The dot plots on the diagonal are especially handy for SPLOM applications. They serve as ordinary histograms for marginal continuous distributions and as discrete histograms for marginal categorical distributions.

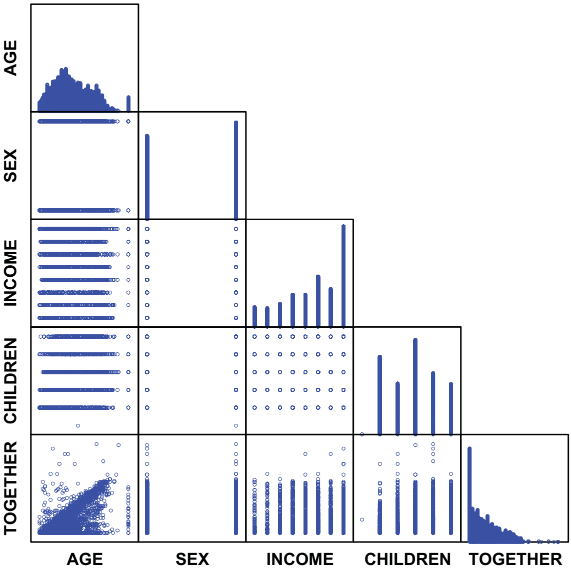


Figure 16.4 *Triangular SPLOM of Chartrand survey*

There are several anomalies in this SPLOM, including the single low value on the children variable and the multiple extreme values at the high end of the age scale. One anomaly is striking, however; the graphic leads us to it. The age–together plot at the lower left corner has an unusual shape. A regular triangular distribution like this suggests an **implication**. If x is greater than y for nearly every pair (x, y) then it may be due to a *necessary* relation between x and y rather than a *stochastic* relation. The necessary relation, in this case, would be that the duration of a relationship must be less than the age of the respondent. There appear to be some respondents represented above the main diagonal in this panel who either believe in reincarnation or have made a serious error in judging their age or the duration of their relationship (assuming both scales are the same). This curiosity was not detected in the preliminary data cleaning performed by the polling organization. Sometimes only graphical methods will reveal anomalies like this.

We can use Chartrand's data to examine this further. Another item in the survey asked respondents to choose the type of work that they liked the best. Respondents were asked to make all six pairwise comparisons between the descriptive words IDEAS, DATA, PEOPLE, and THINGS. Summing these comparisons, we created four new variables to indicate the strength of these preferences. We also created a fifth variable, namely the difference between respondents' reported years in the relationship and their reported age. We call this a TOGETHER–AGE discrepancy (TA). Figure 16.5 shows a rectangular SPLOM of this variable against the others. The SPLOM reveals that those more interested in data are less likely to make errors on the TOGETHER question and those more interested in people are more likely to make such errors.

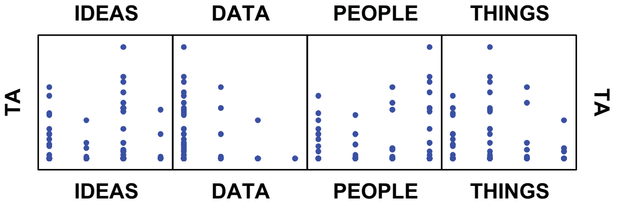


Figure 16.5 *Rectangular SPLOM*

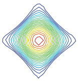
We discovered these anomalies by examining pairwise scatterplots. Can we develop tools to regularize this process? Imagine a p-dimensional space (where $p >> 25$) containing a cloud of points. We seek unusual 2D views of this cloud. Standard methods for doing this involve projections, such as **projection pursuit** (Friedman, 1987) or the **grand tour** (Buja and Asimov, 1986). Tukey and Tukey (fifth cousins) devised a different attack on this problem. They focused on the $p(p - 1)/2$ marginal subspaces of the joint p-dimensional space of observations, to look for unusual bivariate distributions. The Tukeys employed a variety of measures to detect unusual distributions: area of closed

2D kernel density contours, perimeter length of these contours, convexity of these contours, multimodality of the 2D densities, nonlinearity of principal curves fitted to the bivariate distributions, average nearest–neighbor distance between points, and so on. They then displayed these derived measures in a scatterplot matrix (SPLOM) for user exploration. Each point in the frames of this scagnostic SPLOM represented one 2D scatterplot. By linking the scagnostic SPLOM to the data SPLOM as well as to separate scatterplots, the Tukeys assembled a system that is capable of interactive exploration in extremely high-dimensional spaces. It is one of the few graphics systems to overcome the curse of dimensionality.

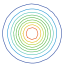
We will show a simplified example. The lower panel of Figure 16.6 contains a SPLOM of the 1997 EPA emissions data for cars sold in the US. The variables are horsepower (HP), gallons-per-mile (GPM), hydrocarbons (HC), carbon monoxide (CO), and carbon dioxide (CO2). The emissions of pollutants are measured in per-mile weight. To highlight the shape information used in scagnostics, we have used filled kernel density contours to represent the point clouds. See Figure 11.10 for the raw SPLOM of these data.

The upper panel of the figure shows the scagnostic SPLOM. We have selected five scagnostic variables to keep the plot simple. We have highlighted one outlier in red and highlighted the corresponding panel in the lower SPLOM to show how the linking works. The high correlation between fuel consumption and carbon dioxide emissions is flagged by the slope, perimeter, and area statistics. Had we used miles-per-gallon instead of gallons-per-mile, then the curvature statistic would have flagged it as well (see Figure 11.10). There are other outliers in the scagnostic SPLOM we might examine. The plot HP-CO2, for example, is an outlier on nearest neighbor distances. This is because it has a number of outliers in the raw scatterplot, evident in Figure 11.10.

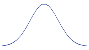
There are limitations to what scagnostics can uncover. First, we must remember that marginal 2D distributions do not give us full information about the joint p-dimensional distribution. For example, this



and this



both have the same x and y margins that look like this



Second, scagnostics are only as good as the measures chosen. Curvature, density, and volume are obvious choices, but there may be other shape indicators that we may need to uncover subregions in our data. Third, scagnostics depend on a well-designed interface. Interactive controllers such as brushes and lassos are essential to exploiting the power of the method.

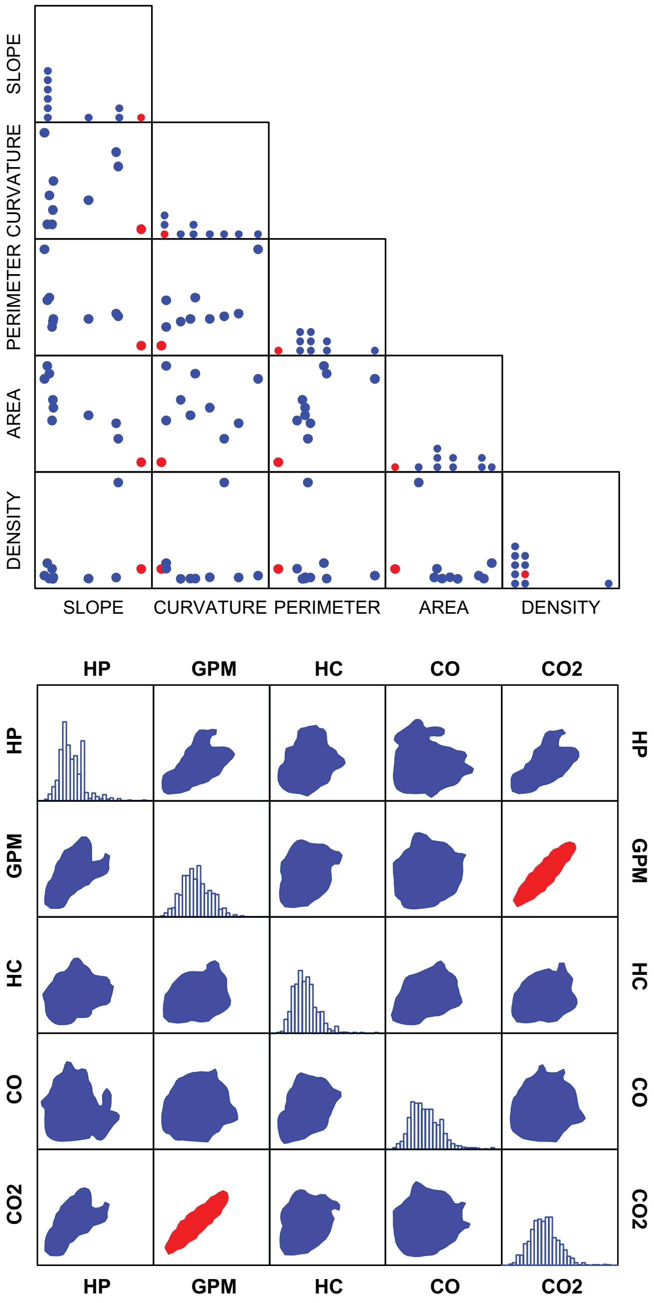


Figure 16.6 Scagnostics

Recently, Wilkinson *et al.* (2005) employed graph theory to develop measures of shape, trend, density, and other aspects that the Tukeys assessed. The advantages of the graph-theoretic approach are (1) improvement in computational complexity; graph algorithms have been intensely investigated recently and are now $O(n \log n)$ or better in performance, and (2) relaxing of assumptions; measures on graphs can be developed for continuous and categorical variables without making assumptions of smoothness or continuity.

16.3 Graph Drawing.

A graph $\{V, E\}$ is **planar** if it is embeddable on the plane such that no two line segments representing edges cross each other. The **graph drawing** (or **graph layout**) problem is as follows. Given a planar graph, how do we produce an embedding such that no two edges cross? And if a graph is not planar, how do we produce an embedding that minimizes edge crossings? The problem has generalizations (layouts on the surface of a sphere, 3D layouts, etc.), but it is usually restricted to 2D Euclidean space.

Kruja *et al.* (2001) survey the history of graph drawing, especially the period before computer-assisted methods. Di Battista *et al.* (1999) survey the more recent literature in computer-assisted graph drawing. The use of computers to lay out graphs got its start in the period when multidimensional scaling was being developed, probably because the algorithms are closely related. Kruskal and Seery (1980) were among the first to demonstrate an automated graph layout. They adapted the Kruskal iterative MDS algorithm to graph-theoretic distances.

Different types of graphs require different algorithms for clean layouts. We begin with simple network graphs. Then we discus laying out trees. We conclude with directed graphs. In all cases, the input data are non-numeric. They consist of an unordered list of vertices (node labels) and an unordered list of edges (pairs of node labels).

16.3.1 Networks

A network is a simple graph. It makes sense that we might be able to lay out a network nicely if we approximate graph-theoretic distance with Euclidean distance. This should tend to place adjacent vertices close together and push vertices separated by many edges far apart. The most popular algorithm for doing this is a variant of multidimensional scaling called the **springs** algorithm (Fruchterman and Reingold, 1989; Kamada and Kawai, 1989). It uses a physical analogy (springs under tension represent edges) to derive a loss function representing total energy in the system (similar to MDS stress). Iterations employ steepest descent to reduce that energy. The basic springs algorithm has a flaw, however. It is vulnerable to local minima because it starts with a random layout. The original nonmetric MDS programs had the same flaw; more recent

MDS programs begin with a metric stage. Consequently, we recommend a two-stage procedure: (1) compute a geodesic distance matrix, break tied distances randomly, and do a scalar products metric MDS to get an initial layout for the nodes; and (2) compute a few iterations with the springs algorithm to reduce crossings. This approach works well for layouts of up to a few thousand nodes.

We offer a simple example. Following Henley (1969), we asked 37 people to speak the names of the first ten or so ordinary animals that came to mind. Our definition of an edge was adjacency in a list. We included in the layout all animals adjacent to each other in at least 3 of the 37 lists. This subset constituted a graph with 17 nodes and 20 edges. Figure 16.7 shows a planar layout of the animal data graph. Consistent with earlier published results, we can identify clusters of barnyard animals, zoo animals, and domestic animals.

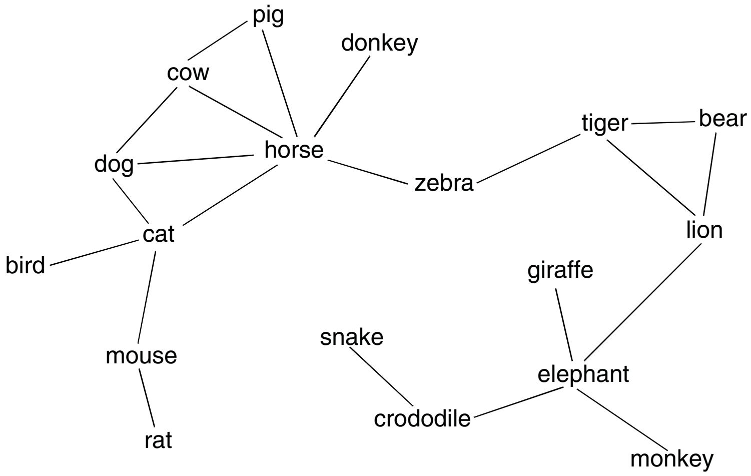
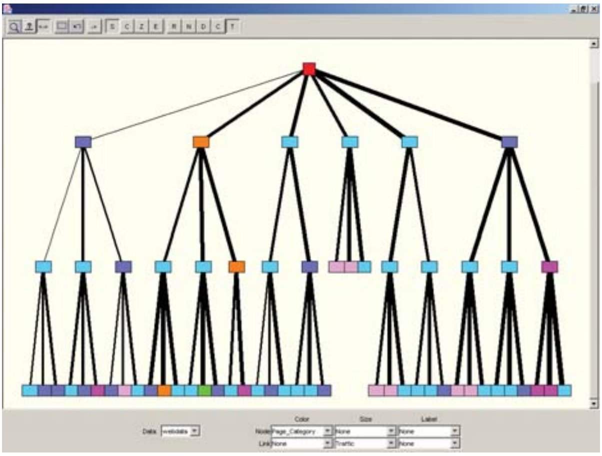


Figure 16.7 *Network layout of animal naming data*

16.3.2 Trees

A tree is an acyclic graph. Rooted trees are usually arranged vertically with the root at top or bottom. Spanning (unrooted) trees are usually arranged to cover a region somewhat uniformly. Laying out rooted trees on the plane is relatively simple. We assign layers for each node by finding the longest path to a leaf. By this calculation, the root is at layer *n* and the leaves are at layer 0. Then we begin with the leaves, group daughters by parent, and align parents above the middle daughter(s) in each group. After this sweep, we can move leaves up the hierarchy to make shorter branches, as we did in the following example. Di Battista *et al*. (1999) present other algorithms for laying out rooted trees.

Figure 16.8 shows a small Web site represented as a hierarchical tree. This application is designed to allow interactive exploration of the Web site graph. We used the simple layout method and made the thickness of the branches proportional to the number of visitors traversing nodes. The colors of the nodes represent page categories. The home page is at the top.

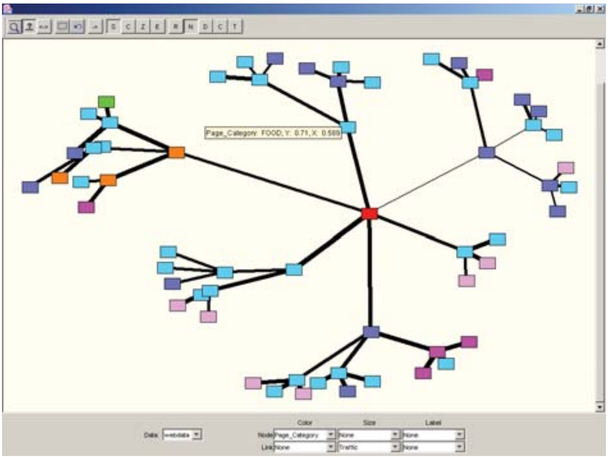


Data: webdata

	Color	Size	Label
Node	Page_Category	None	None
Link	None	Traffic	None

Figure 16.8 *Web site as rooted tree*

Unrooted trees have no implicit hierarchy. Consequently, the springs algorithm works nicely for unrooted trees. In Figure 16.9 we lay out the same Web data as a spanning tree. Notice the popup that reveals data about a selected node. Because the layout makes greater use of the frame, the spanning tree makes better use of the display space than the rooted tree layout.



Page_Category: FOOD, Y: 0.71, X: 0.589

Data: webdata

	Color	Size	Label
Node	Page_Category	None	None
Link	None	Traffic	None

Figure 16.9 *Web site as spanning tree*

Figure 16.10 illustrates a different type of graph layout using a display from Wills (1999). Wills arranges trees and other graphs on a hexagonal grid, using a variety of algorithms from simulated annealing to steepest descent. Using the hexagonal grid regularizes the structure and facilitates navigation. The coloring of the nodes in this figure reflects the Web resource type (pages, images, or scripts).

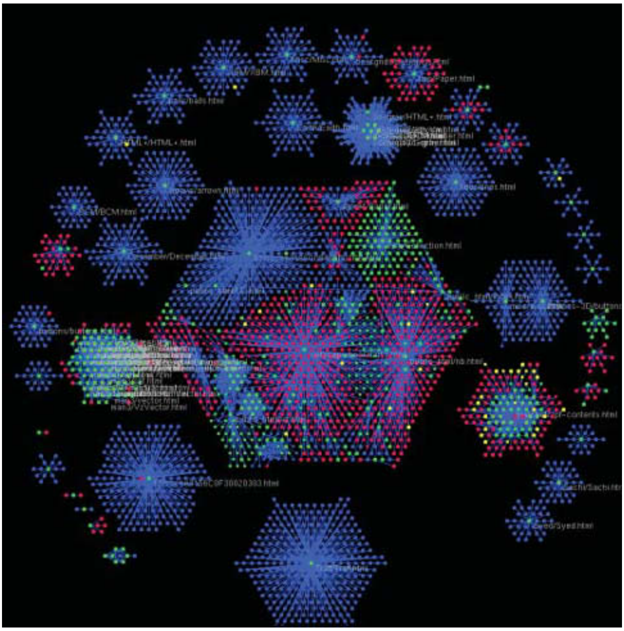


Figure 16.10 *Wills Web data*

16.3.3 Directed Graphs

Directed graphs, like rooted trees, are usually arranged in a vertical partial ordering with source node(s) at top and sink node(s) at bottom. Nicely laying out a directed graph requires a **topological sort** (Skiena, 1998). We temporarily invert cyclical edges to convert the graph to a directed acyclic graph (DAG) so that the paths-to-sink can be identified. Then we do a topological sort to produce a linear ordering of the DAG such that for each edge (*u, v*), vertex *u* is above vertex *v*. After sorting, we iteratively arrange vertices with tied sort order so as to minimize the number of edge crossings. Of course, we may flip or rotate the configuration when we are done.

Let's compare a network layout with a directed layout computed this way. Figure 16.11 shows an example of a network layout of a graph from Figure 10.12 in Di Battista *et al.* (1999). The layout in Figure 16.11 was produced by the springs algorithm. The resulting graphic has two crossings and the positions of the nodes do not reveal the hierarchy. Di Battista *et al*. illustrate a variant of the springs algorithm on this graph that results in five crossings.

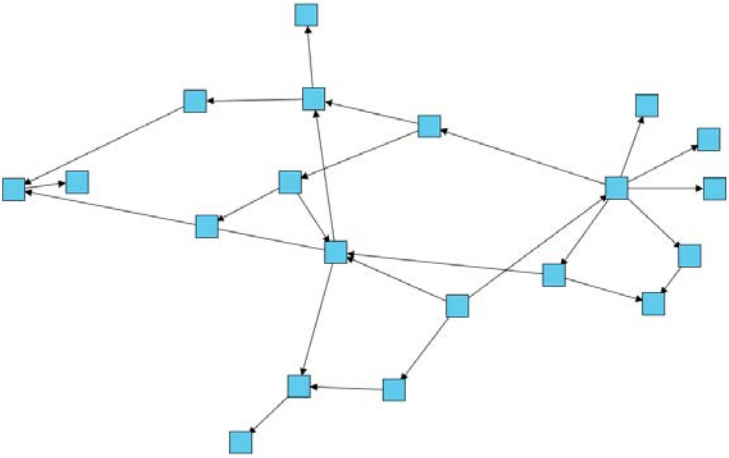


Figure 16.11 *Network graph drawing*

The graph in Figure 16.12, by contrast, was produced by the layered method for a directed graph, resulting in no crossings. Notice that the horizontal layout is rather intricate. Several iterations were needed to avoid collisions between nodes. The solution can be improved aesthetically by dragging a few nodes horizontally to eliminate gaps and make edges more vertical. Most graph layout software allows manual adjustment after iterating.

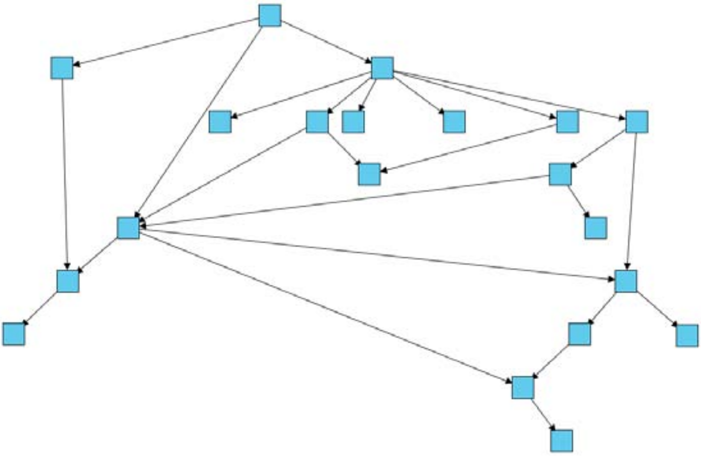


Figure 16.12 *Layered graph drawing*

16.4 Sequence Analysis

A sequence is a list of objects, *e.g.*, $\langle x, y, z, \ldots\rangle$. The ordering of the list is given by an order relation. Let R be a binary relation defined on a set A. For R to establish a sequence, $x\ R\ y$ must imply $\langle x, y\rangle$ for all x, y in A. Let's examine this for several types of order relation.

We begin with a **partial order relation**. A relation R is a partial order if and only if it is

reflexive ($x\ R\ x$ for all x in A),

transitive ($x\ R\ y$ and $y\ R\ z$ implies $x\ R\ z$ for all x, y, z in A), and

antisymmetric (if $x\ R\ y$ and $y\ R\ x$, then $x = y$ for all x, y in A).

Under these conditions, we call A given R a **partially ordered set**, or **poset**. The elements of a poset are only partially ordered because not all elements are related through R. We say elements x and y are **comparable** if either $x\ R\ y$ or $y\ R\ x$. If neither $x\ R\ y$ nor $y\ R\ x$, then x and y are **noncomparable**. For example, $\{x, z\}$ and $\{x, y, z\}$ are comparable under the relation $\subseteq$, but $\{x, z\}$ and $\{x, y\}$ are noncomparable under the same relation.

We may represent a poset with a directed graph (digraph). Vertices in this graph are elements of A and edges are the relation R. The left panel of Figure 16.13 shows a digraph for a three-element set partially ordered by the subset ($\subseteq$) relation. We see, for example, that $\{x, y\}$ is a subset of $\{x, y, z\}$ because there is a path from the $\{x, y\}$ vertex to the $\{x, y, z\}$ vertex. Note that there is no path from $\{x, y\}$ to $\{x, z\}$ because they are noncomparable. Also, note that every vertex has an edge looped to itself because the relation is reflexive.

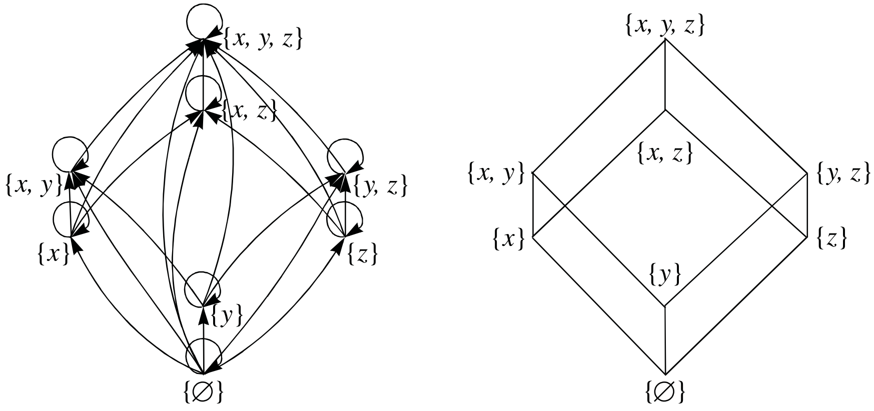


***Figure 16.13** Directed graph for a poset and its Hasse diagram*

Digraphs for even simple posets are fairly messy. The right panel of Figure 16.14 shows a cleaner alternative called a **Hasse diagram**. It preserves all the information in the digraph without the clutter. The rules for recovering the digraph from the Hasse diagram are simple: (1) make all edges directionally upward, (2) add self-referent loops to every vertex, and (3) add upward edges for every pair of connected vertices if they are missing. To go the other way, reverse these rules.

Any path in a digraph of a poset delineates a sequence based on the relation *R*. We call such a path a **chain**. All elements of a chain are comparable, which makes the elements in a chain a **totally ordered set**. Thus, to generate a set of sequences that follow a partial order relation *R*, we search for totally ordered subsets of a partially ordered set based on *R*. Figure 16.14 shows an example of the longest nonrepetitive sequences that are contained in a three-element set partially ordered by the subset ($\subseteq$) relation. We can trace these chains in the poset by starting at the bottom of the Hasse diagram and following all paths to the top. Note that there is an infinite number of additional sequences that follow *R*. They are all possible subsequences of the ones in Figure 16.14, including infinite repetitions of any vertex.

$$\{\varnothing\} \subseteq \{x\} \subseteq \{x, y\} \subseteq \{x, y, z\}$$

$$\{\varnothing\} \subseteq \{x\} \subseteq \{x, z\} \subseteq \{x, y, z\}$$

$$\{\varnothing\} \subseteq \{y\} \subseteq \{x, y\} \subseteq \{x, y, z\}$$

$$\{\varnothing\} \subseteq \{y\} \subseteq \{y, z\} \subseteq \{x, y, z\}$$

$$\{\varnothing\} \subseteq \{z\} \subseteq \{x, z\} \subseteq \{x, y, z\}$$

$$\{\varnothing\} \subseteq \{z\} \subseteq \{y, z\} \subseteq \{x, y, z\}$$

Figure 16.14 *Sequences generated by the subset relation*

Now suppose that a relation *R* is **irreflexive** (*i.e.*, $x\ R\ x$ is undefined) and **transitive** and **antisymmetric**. Such a relation (which we call a **quasi order relation**) induces a sequence that has no repetitions. For example, $x < y < z$ implies the sequence $\langle x, y, z \rangle$. From this definition, we can see that a quasi order is a restriction of a partial order. It eliminates the loops from the corresponding digraph.

We have introduced order relations that generate sequences. Given a sequence, is it possible to deduce an order relation? Not generally. But usually we can rule out some relations and we can identify candidate relations that are not inconsistent with the observed sequences. In any case, it helps to have long sequences to make these inferences less speculative. We will now examine a few algorithms for analyzing observed sequences.

16.4.1 *Identifying Numeric Sequences*

Identifying a numeric sequence is a nontrivial exercise. The subject is vast, so we will restrict ourselves to sequences of integers (Sloane and Plouffe, 1995). That narrows the range to many thousands of regular sequences and a huge amount of associated mathematics. To narrow further, we will focus on a few integer sequences involved in computing and graphics.

Suppose we are given the integer sequence $\langle 1, 2, 4, 11, 33, 166, \ldots\rangle$. How do we provide the next integer in the sequence? If we know this sequence represents the number of graphs with n nodes, for example, we can enumerate to determine the answer (the numbers in this case get large very quickly, however). If we do not know what it represents, we can apply a number of heuristic strategies. In the end, we will identify a sequence by finding either (1) an implicit recursive algorithm or (2) an explicit generating function.

16.4.1.1 Recursions

An implicit recursive algorithm expresses an element f_n in a sequence as a function of prior elements f_{n-p} (where $p > 0$) in the sequence. We may sometimes identify implicit recursive sequence algorithms by induction. We look for (1) a **recurrence relation** and (2) an **initial condition** that both fit the numbers in the sequence. Recurrence relations take several forms.

A **linear recurrence relation** has the form

$$f_n = a_0 + a_1 f_{n-1} + a_2 f_{n-2} + \ldots + a_p f_{n-p}$$

- If $a_0 = 0$, we call the relation **homogeneous**.
- If $a_1 = a_2 = \ldots = a_p$, we say the relation has **constant coefficients**.
- If $p = 1$, we say the relation is **first order**. If $p = 2$, we say the relation is **second order**, and so on.
- If we have a second–order linear homogeneous recurrence relation with constant coefficients and initial condition $f_1 = f_2 = 1$, we have the famous **Fibonacci sequence** $\langle 1, 1, 2, 3, 5, 8, 13, 21, 34, \ldots\rangle$.
- If $p = 1$, $a_0 \neq 0$, and $a_1 = 1$, we have a simple **arithmetic sequence**, such as $\langle 0, 2, 4, 6, 8, 10, \ldots\rangle$. If $a_0 < 0$, the sequence decreases. If $a_0 > 0$, the sequence increases.
- If $p = 1$, $a_0 = 0$, and $a_1 \neq 0$, we have a simple **geometric sequence**, such as $\langle 1, 2, 4, 8, 16, 32, \ldots\rangle$. If $a_1 < 1$, the sequence decreases. If $a_1 > 1$, the sequence increases.

A **nonlinear recurrence relation** has a nonlinear form, such as

$$f_n = af_{n-1}(1-f_{n-1})$$

We saw this type of recurrence relation used in Chapter 13. We will not explore it further here.

16.4.1.2 Generating Functions

An explicit generating function expresses an element f_n in a sequence as a function of n. Explicit generating functions for a given sequence are often hard to find and sometimes nonexistent. Wilf (1994) presents ingenious methods for guiding this search. A few standard methods follow.

If a sequence is based on a second–order linear homogeneous recurrence relation with distinct characteristic roots, then we can construct an explicit formula from the roots. The Fibonacci sequence, for example, has the generating function

$$F_n = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n$$

based on the roots

$$\left(\frac{1+\sqrt{5}}{2}\right) = 1.618033989 \text{ and } \left(\frac{1-\sqrt{5}}{2}\right) = -0.618033989$$

The positive root is called the **golden number** (ϕ). The **golden rectangle** has ϕ for its aspect ratio. Conway and Guy (1996) and Koshy (2001) give many fascinating contexts in which this number appears. There is a huge amount of lore concerning ϕ in nature, especially on the Internet. Much of the latter is nonsense. Many popular examples in biology (seashells, plant growth), art (Classical, Renaissance), psychology (form and music perception), and finance (Fibonacci calculators) are only approximately true and sometimes plainly false. They are most subjective when they involve numerological or theological application of mathematics to physical phenomena. The golden rectangle has been suggested by some as the ideal reference frame for charts, for example. There is little psychological evidence (and less theory) to support this idea. In fact, the experimental evidence points elsewhere (Cleveland *et al.*, 1988).

If a sequence is based on a polynomial function, then we can use finite differences to derive the polynomial equation. The finite backward difference of a numerical sequence is of the form $\nabla f_n = f_n - f_{n-1}$. The finite forward difference is of the form $\Delta f_n = f_{n+1} - f_n$. If the polynomial generating function has degree p, then p successive forward differences of the sequence will result in a constant series. This result stems from the fact that the finite difference is the discrete analog of the derivative. We can get coefficients by a small amount of additional work.

Finite differences can tell us something even if a sequence is not polynomial. Differencing the Fibonacci sequence, for example, yields a Fibonacci sequence (shifted by one). Why? We can work out an answer in terms of the recurrence formula, but let's look at the generating function. Note that for large *n*, the second term in the Fibonacci generating function approaches zero, so we have $F_n = \phi^n/\sqrt{5}$. (This works even for small *n* if we round the function to an integer.) Thus, our differencing the Fibonacci sequence is comparable to differentiating an exponential function. Finding a derivative equal or proportional to the function itself points to an exponential form.

If all else fails, Neil Sloane maintains an online database of more than 80,000 sequences. You can type in a sequence and the database yields known generating functions.

16.4.1.3 Graphing Algorithm Performance

The link between simple recurrence relations and generating functions underlies an important application: the analysis of algorithms. Generating functions allow us to make global assessments of algorithm performance. If we can break down an algorithm into a simple set of primitive operations that each take one unit of time, and if we can specify the size of a problem as an integer (*n*), then we can count the number of computation steps for given values of *n* (sometimes in theory, sometimes by running a program) and construct an integer sequence over *n*. If, for example, an algorithm requires visiting every leaf of a complete binary tree whose depth is *n*, the number of visits is representable by a geometric sequence with a multiplicative factor of 2.

Figure 16.15 shows three different integer sequences and their associated continuous functions. We have already encountered the Fibonacci sequence. The powers-of-two sequence is $\langle 1, 4, 8, 16, 32, 64, 128, 256, 512, \ldots \rangle$. This is a geometric sequence with $a = 2$. The perfect squares sequence (a nonlinear recurrence) is $\langle 1, 4, 9, 16, 25, 36, 49, 64, 81, \ldots \rangle$. The right panel contains the continuous versions of their generating functions. If the range of these functions is time, then over the long run, some do better than others.

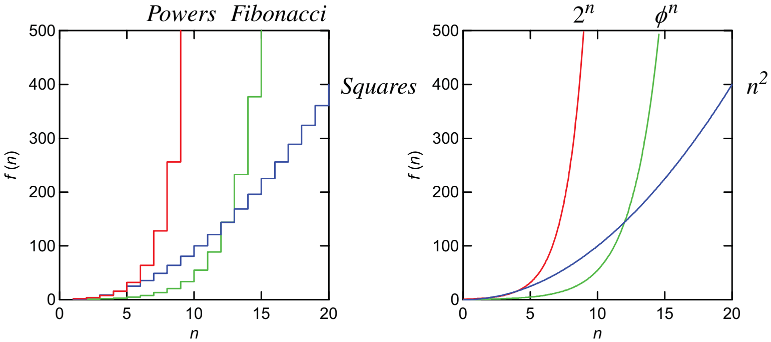


Figure 16.15 *Integer sequences and their associated continuous functions*

Figure 16.16 shows some of the most common time complexity functions encountered in algorithms. Clearly, we should prefer those functions at the cooler (blue) end of the scale. Some programmers wonder, however, why algorithm designers don't care a lot about the specific values of parameters associated with these classes of functions and focus instead on worst-case performance. In other words, why do algorithm designers use an order-of-magnitude function instead of a complexity function based on, say, average performance? First, a definition:

> If $f(n)$ and $g(n)$ are functions of n, then we say that f is $O(g)$ if $kg(n)$ is an upper bound on $f(n)$ for sufficiently large n.

We say "sufficiently large" because we don't care about small n. The $O(.)$ function gives us a bound on the worst performance of an algorithm. Suppose, for example, we plotted the function $f(n) = n^9$ instead of $f(n) = n^2$ (the green curve). In the frame for Figure 16.16, that would place the curve to the left of the factorial function $f(n) = n!$ (seemingly worse performance). Nevertheless, the factorial crosses $f(n) = n^9$ at $n = 14$. In other words, there is a big payoff for jumping to a different $O(.)$ class of algorithm and little payoff for fine-tuning within a class. If a computational unit required 1 nanosecond, for example, then 30! = 8.4 x 10^{15} years. As Illinois Senator Everett Dirksen once said, "A billion here. A billion there. Pretty soon we're talking real money."

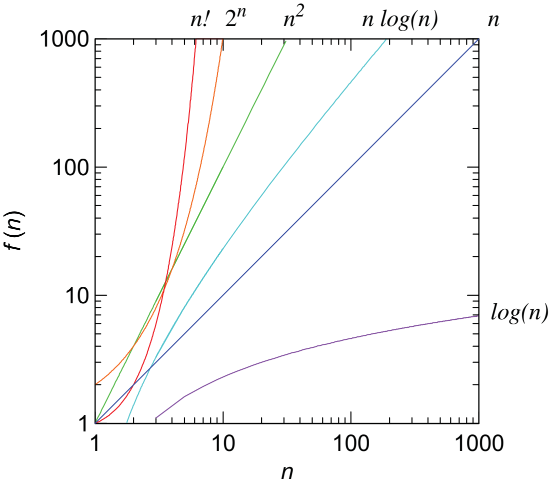


Figure 16.16 *Complexity functions*

Can we look at code and get some idea of its time complexity? Eick *et al.* (1992) developed a system for viewing large software code files called *SeeSoft*. This simple but ingenious idea involved reducing the font size enough to represent thousands of lines of code on a computer screen. Lensing tools allow zooming to view specific sections of code. By looking for deeply nested loops, one can identify segments of the code with polynomial execution time.

Code profilers automate this process by recording execution time and displaying bar charts of time for different segments of a running program. Software developers use these routinely to optimize code. Algorithm animation can also reveal interesting patterns in executing programs.

Figure 16.17 shows a different approach. Instead of displaying code itself or histograms of execution time, we assume a constant-time operation and represent each operation with a bar. The bars for nested operations have half the height of their parent. Recursions raise the level of the bars altogether. We have chosen $n = 3$ for this example. The linear time algorithm executes 3 operations. The quadratic time algorithm executes 9. The cubic, 27. The factorial time algorithm has six recursions (3!) of the triply nested operation. Finally, we show the execution profile of a Cholesky decomposition of a size 3 symmetric matrix, which has an order of magnitude of n^3. This graph is somewhat like a horizontal bar graph. We can measure performance by the length of the "bars" and get an idea of computational profiles by their shape. Deep recursions will severely skew the bars, a sign that we might encounter stack overflows when running our program.

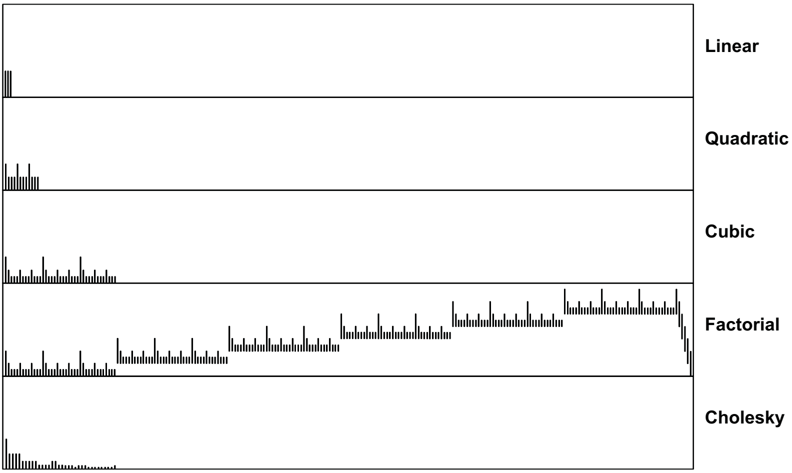


Figure 16.17 *Complexity graphs*

16.4.2 Finding Sequences in Strings

In many applications of sequence analysis, objects are represented by tokens and sequences are represented by strings of tokens. In biosequencing, for example, the letters A, C, T, G are used to represent the four bases in a DNA strand. Suppose we are given a length n string of tokens and want to find the most frequently occurring substrings of length m in the string ($m << n$). A simple (not especially fast) algorithm to do this involves generating candidate substrings and testing them against the target string. We begin with strings of length 1, each comprised of a different token. Then we build candidate subsequences of length 2. We count the frequency of each of these subsequences in the target string. Using any of these length 2 subsequences with a count greater than zero, we build candidate subsequences of length 3. We continue the generate-and-test process until we have tested the candidates of length *m* or until all counts are zero. This stepwise procedure traverses a subset of the branches of the tree of all possible subsequences so we do not have as many tests to perform.

Embedding a sequence analysis in a graph layout often gives us a simple way to visualize these subsequences. The layout may be based on known coordinates (as in geographic problems) or on an empirical layout using adjacency in the sequence list as edge information. Figure 16.18 shows sequences computed this way, superimposed on the animal names graph in Figure 16.7. We have included all sequences of length three or greater that were found in at least two persons' lists. Because the task allowed no repetitions (we supposed a quasi order), we did not have as large a set of candidate sequences to generate.

Notice that *dog* and *cat* have the same number of input and output edges between them, so neither appears to dominate the other (thinking of a dog apparently leads to thinking of a cat as much as the reverse). Other nodes, such as *elephant*, appear to be more a source than sink among this subset of popular animals. And nodes such as *zebra* are sinks. Of the 18 mentions of *zebra* in the lists, only two involved the first occurrence of a zoo animal.

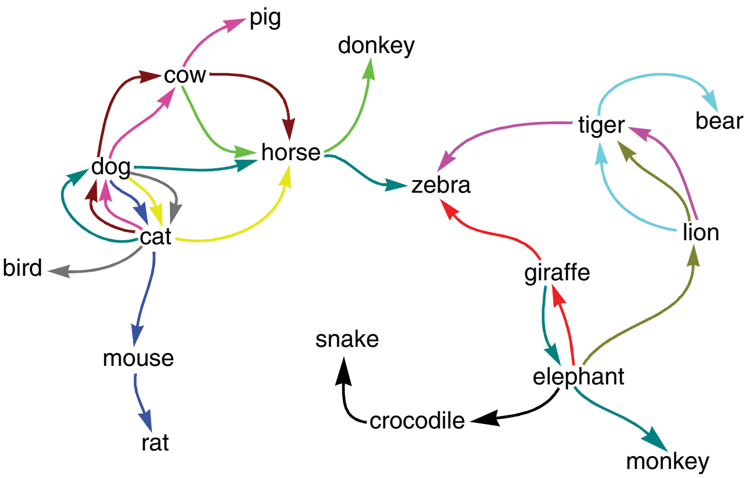


Figure 16.18 *Sequences of recalled animal names*

When repetitions are allowed, we have a larger set of candidates to test. Figure 16.19 shows a musical example. Our data come from the scores for the six Gregorian chant settings of the creed found in the *Liber Usualis* (1953). We transcribed the notes in the scores into six extended sequences of the letters A through G (we did not differentiate octaves). We restricted our search to sequences found in at least four of the chants.

Instead of annotating a graph with sequences, we decided to present the results in musical notation. Chant is written in **neumes**, which are notes sung on a single syllable of the text. The single staff consists of only four lines, instead of the five used in modern musical notation. The Do clef (𝄡) marks where Do or C is on the staff. The notation, of course, can be produced in the grammar by using points with musical note shapes in a time series frame represented by five grid lines and a scale anchored at the clef.

If you sing or play these notes you will immediately recognize the Gregorian tonalities. Chant has had a renaissance (sorry) among busy, secular moderns. Its original upward gaze has been turned inward by many, from worship toward meditation. The next step in this sequence will be for someone to market these clips as "Chant-lite for busy people on the go."

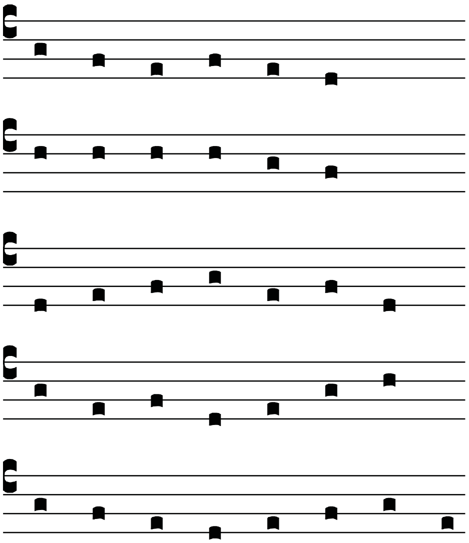


Figure 16.19 *All sequences of six or more notes occurring in at least four of the Gregorian chants for the Credo (I – VI) (Liber Usualis, 1953)*

For further reading on sequencing, especially in bioinformatics, see Gusfield (1997). Agrawal and Srikant (1995) developed a general sequencing algorithm for market basket analysis. Wong *et al.* (2002) developed a visualization tool for Agrawal's algorithm.

16.4.3 Comparing Sequences

Suppose we have two sequences of characters or objects and we wish to compare them. If the sequences are of length *n*, we can construct an *n* by *n* table of zeros and place a 1 in a diagonal cell if the value in each sequence at the corresponding position is the same. We would have an identity matrix if both sequences were identical.

With real data, however, we are more likely to encounter matching runs of subsequences that occur in different locations in each sequence. Moreover, we often encounter missing or ignorable values in subsections of sequences, especially in sequences of DNA base letters. In these cases, our table is more likely to look like the one in Figure 16.20, adapted from Altschul *et al.* (2001). This graph reveals matching subsequences beginning at the positions indicated by the circles at the top left of the red paths. For multiple strings, we could construct a casement plot of pairs of strings.

The plot in Figure 16.20 bears a close resemblance to the recurrence plot shown in Figure 14.14. In fact, the plots are equivalent. Can you see why? There are differences, of course. In Figure 14.14, the orientation runs from southwest to northeast, while here the orientation runs from northwest to southeast. And in Figure 14.14, the cell values are computed by a real–valued distance function, while here they are computed by a binary–valued indicator function. Nevertheless, both have the same lag structure. To assess overall strength of agreement in either type of plot, we should look for the number, length, and regularity of similar diagonal paths.

Since a sequence is mathematically equivalent to a time series, it should not be surprising to find similar plots based on similar models arising in different literatures. Models are the mothers of invention.

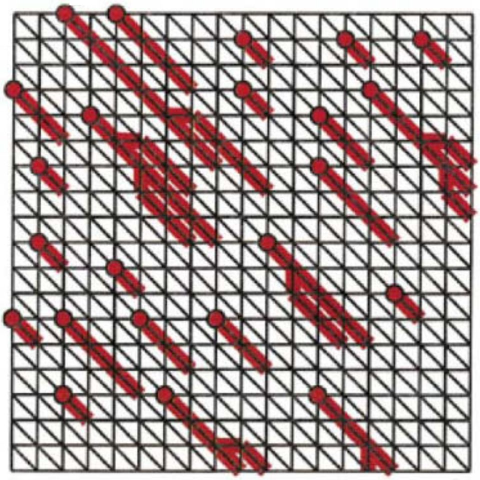


Figure 16.20 *Comparing two sequences (courtesy of Steven Altschul)*

16.4.4 Critical Paths

Suppose we have a directed acyclic graph (DAG) where the vertices represent tasks and an edge (*u*, *v*) implies that task *u* must be completed before task *v*. How do we schedule tasks to minimize overall time to completion? This **job scheduling** problem has many variants. One customary variant is to weight the edges by the time it takes to complete tasks. We will mention two aspects of the problem that involve graphing.

First, how do we layout a graph of the project? We use the layout for a directed graph described in Section 16.3.3 above and flip the graph to a horizontal orientation. The result of our efforts is called a CPM (critical path method) graph. We have added a constraint to our layout by forcing the critical path to be linear in the layout. Di Battista *et al.* (1999) discuss how to do this.

Second, how do we identify and color the **critical path**? Identifying the critical path is easy if the edges are not weighted. We simply do a breadth-first search of the DAG and keep a running tally of the path length. Finding the shortest path through a weighted graph requires dynamic programming. Skiena (1998) discusses standard methods.

Figure 16.21 shows an example of a CPM layout for a simple visualization software project. Each task is represented by a rectangle. The lengths of the edges are often made proportional to the weights (months to complete the tasks in this case). We have chosen instead to keep a sum (left to right) of the weights inside the rectangles. Each number in the rectangle is the sum of the largest sum from a precedent node plus the corresponding input edge weight. Can you derive the edge weights from these numbers?

The source node for the project is the decision to commit to the project. We have given this a weight of zero, although some huge software companies seem to spend more time on this decision than on doing the project itself. The design tasks (Model, View, UI) are prerequisites to coding, although some software companies have been known to do design (including the SOR, or statement of requirements) after the code is written. Web tools and templates are prerequisites to beta testing, although some companies seem to shorten the beta test period to a duration of zero. Finally, promotion and marketing materials and documentation tasks are prerequisites to shipping the product, except in those instances where software companies can get users to pay for beta product.

The critical path represents tasks that cannot slip without affecting the ship deadline. Thus, identifying the critical path (which we have colored in red) is a matter of accumulating edge weights at each node and identifying the precursor with the maximum cumulative weight. Despite our levity, we recognize the value in a CPM analysis because the critical path highlights the dependencies within a project. Long critical paths (like the one in Figure 16.21) reflect a poorly designed project. By reducing dependencies, we reduce risk of delay.

We reduce dependencies in a software project by modularizing and shortening tasks. The **Extreme Programming** methodology (Beck, 2000) is one approach to this problem. XP, as it is called, places coding at the center of project development. (Notice how code is the backbone of the critical path in our example?) By paying close attention to client needs, carving goals into small pieces, focusing on "the simplest thing that could possibly work" for a given goal, and testing while the code is being written, XP attempts to eliminate the bottlenecks that paralyze large projects.

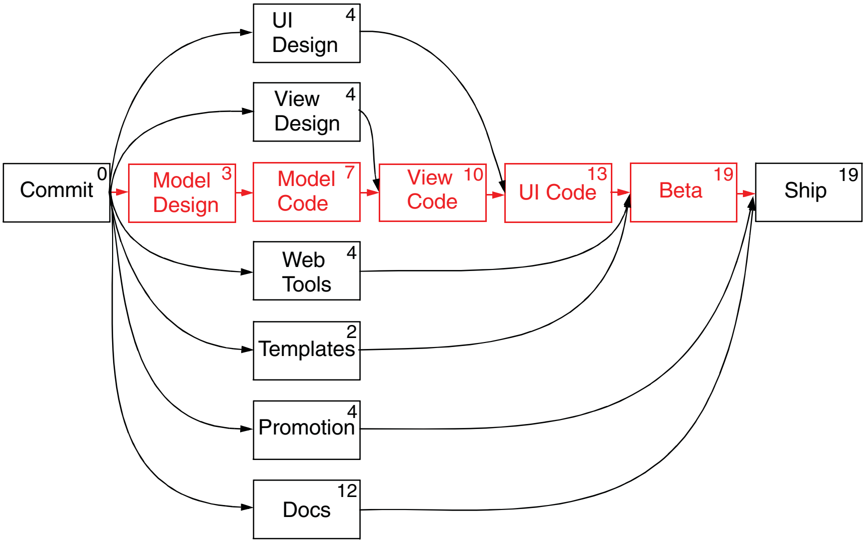


Figure 16.21 *CPM graph*

Graph layouts of large projects can become messy. Even without edge crossings, a large CPM graph can be difficult to interpret. Figure 16.22 shows an alternative called a **Gantt chart**. The horizontal axis measures time. The length of a bar represents the duration of a task. The vertical axis separates the tasks. The coloring categorizes tasks.

The original form of the chart (Gantt, 1903) did not have the benefit of the graph theory behind CPM, but modern incarnations have blended the bars of the Gantt chart with the information on the critical path. Most computer project management packages compute the critical path with graph-theoretic algorithms and display the results in some variety of the Gantt chart.

We have colored the critical path elements red in Figure 16.22 and placed a bar delineating the length of the entire project at the top of the frame. The remaining non-critical tasks are colored purple. Because of the coloring, there is less need to place the critical path tasks adjacent to each other. The groupings can be organized by length, teams, task type, or other variables.

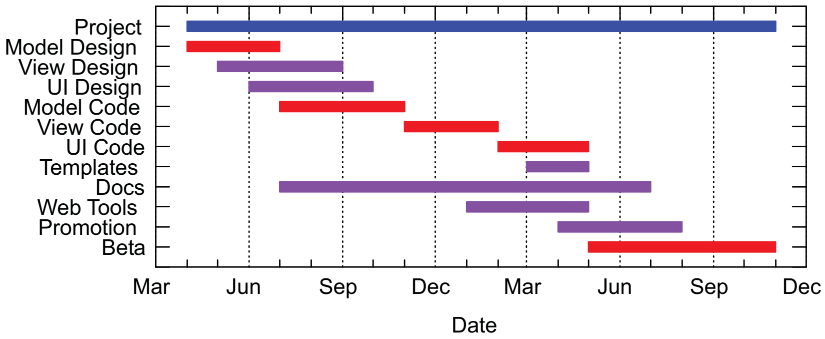


Figure 16.22 *Gantt chart*

16.5 Pattern Analysis

Pattern analysis frequently refers to the search for patterns and structures in pictures and bitmaps. This section goes one step further. Suppose we scramble the pixels in a bitmap. Can we unscramble them to find patterns? More specifically, this section deals with a specific kind of display called a **heatmap**. This display is not new. Sneath (1957) suggested ordering the rows/columns of a similarity matrix via hierarchical clustering and shading the values according to similarity. Bertin (1967) used light and dark values to highlight a Guttman scale pattern in data matrices. Ling (1973) used shading to highlight numeric structure in correlation and data matrices. BMDP and SYSTAT have included heatmap matrix displays in their clustering modules since the early 1980s. Gower and Digby (1981) present a variety of permuted matrix displays.

16.5.1 Matrix Permutation

Suppose we have a data matrix

$$\mathbf{X} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \\ j & k & l \end{bmatrix}$$

whose rows and columns we wish to permute. To do this, we define row and column permutation matrices, *e.g.*,

$$\mathbf{P}_R = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \qquad \text{and} \qquad \mathbf{P}_C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

and apply the product

$$\mathbf{P}_R \mathbf{X} \mathbf{P}_C = \begin{bmatrix} j & l & k \\ d & f & e \\ a & c & b \\ g & i & h \end{bmatrix}$$

Note that **P** is composable:

$$\mathbf{P}_{R1}\mathbf{P}_{R2}\mathbf{P}_{R3}\ldots\mathbf{X}\ \mathbf{P}_{C1}\mathbf{P}_{C2}\mathbf{P}_{C3}\ldots$$

so we may perform the permutation as a sequence of transformations.

To permute data matrices, we require some objective function for determining the row and column permutation matrices. A general requirement might be to permute so that similar values in the data matrix are near each other and there are as few regions of similar values as possible. Unfortunately, this requirement is so general that we would have an NP-hard problem achieving it. We would have to examine every possible permutation and look for the $\mathbf{P}_R$ and $\mathbf{P}_C$ that produce the best result. Consequently, most approaches to this problem have involved separately permuting rows and columns using cluster analysis, singular value decomposition, multidimensional scaling, or some other multivariate analytic method. In the following sections we will generate some canonical data patterns and examine the performance of a variety of methods for permuting the generated data.

16.5.2 Canonical Data Patterns

The top of Figure 16.23 shows heatmaps of five different covariance matrices. Below these are heatmaps of the datasets that produced them. Because we scaled the columns of the datasets to have zero mean and unit standard deviation, the covariance matrices are simple quadratic forms, namely,

$$\mathbf{S} = \frac{1}{n}\mathbf{X}'\mathbf{X}$$

where **X** is the $n \times p$ data matrix and **S** the $p \times p$ covariance matrix. Because of this scaling, the covariance matrices are equivalent to correlation matrices, with 1’s on the diagonal and entries between –1 and 1 elsewhere. We used a bipolar color scale (red for negative, black for zero, green for positive) to represent the correlations. We chose a perceptually linearized monopolar color scale from Levkowitz (1997) to represent the data values in the lower matrices.

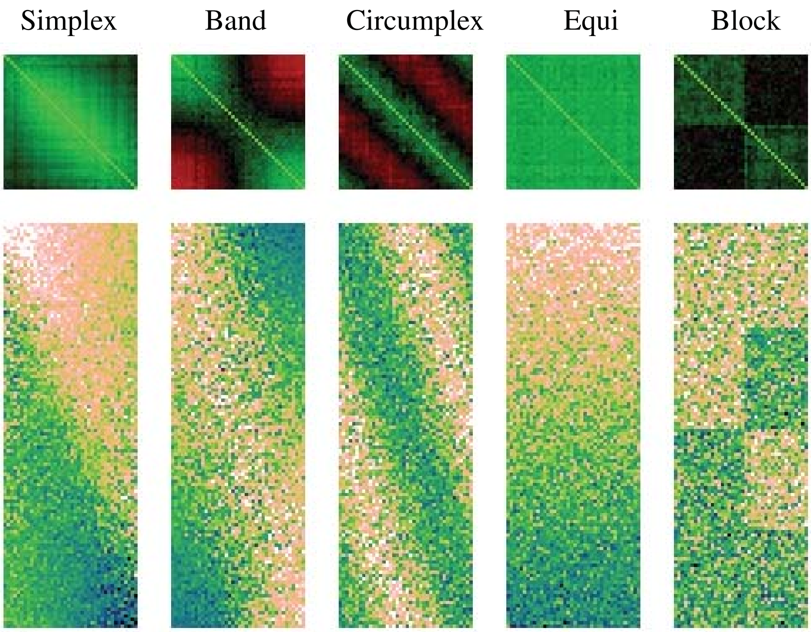


Figure 16.23 Data structures (simplex, band, circumplex, equi, block)

We call these data patterns canonical not because they are exhaustive or strictly typical of real datasets. Instead, they represent different multivariate distributions of data that form a basis for generating a variety of interesting patterns presumably discoverable by heatmaps. More complex patterns can be generated from the product of these patterns. We will cite examples of real data for each of these patterns in the sections below.

Most of these patterns differ from the one presumed in simple statistical models: n independent and identically distributed (i.i.d.) cases sampled from a p-variate distribution. As such, they comprise a good set for testing the effectiveness of various permutation schemes where there is a structure on both rows and columns. They extend the universe of permutation problems in other ways as well. Because most matrix permutation programs use some form of cluster analysis, we are accustomed to view this as a method to be applied when data are segmented or clustered. Our analysis does not make that assumption.

16.5.2.1 Simplex

The data values in the first dataset (Simplex) were generated from the following formula (ignoring rescaling the columns to unit standard deviations):

$$x_{ij} = e^t/(1+e^t) + u_{ij},$$

where

$$i = 1, \ldots, n, \quad j = 1, \ldots, p,$$

$$t = (s_j - r_i)/b,$$

$$s_j = j/p, \text{ and}$$

$$r_i = i/n.$$

The positive, nonzero parameter *b* determines the slope of the logistic function generated by the exponentials, and thus the sharpness of the boundary between the blue and pink regions. The random error u_{ij} is based on a weighted standard normal random variable *Z*:

$$u_{ij} = wZ$$

where

$$w = ke^{-t^2}$$

The parameter *k* determines the relative amount of random error overall in the data. This dataset is named after the term coined by Louis Guttman (Guttman, 1954). If *b* is near zero and *k* is zero, we have the data structure Guttman called *simplex*, As Guttman noted for simplex-type data structures, the correlations are all positive; correlations between near columns are high and the correlations between distant columns are low. Guttman's simplex correlation pattern is a specific case of a **Toeplitz matrix**, which is a square matrix in which all the elements are the same along any diagonal that slopes from northwest to southeast.

The simplex correlation structure is produced by a number of data models and arises in numerous applications. If the columns of X can be assumed to consist of measurements ordered in time, for example, then a first-order autoregressive model based on the indices of these columns produces a **Markov process** whose correlation matrix has a simplex structure (Morrison, 1976). In social measurements, the **latent structure model** (Lazarsfeld and Henry, 1968) and **ordered multinomial model** (Goodman, 1978) involve a simplex correlation structure. In educational measurement, the **latent trait model** (Rasch, 1960; Lord and Novick, 1968; Bock, 1975) is a generalization of the simplex. In the latent trait model, the columns of X represent an item difficulty dimension (one column per item) and the rows of X represent subjects (*e.g.*, students). Each row of data contains a set of item scores (usually binary) for a given subject. The estimate of a subject's ability is based on a logistic or normal cumulative distribution fit to the profile of test scores. In physics, Brownian motion follows a **Wiener stochastic process**. A Wiener process is a continuous-time stochastic process $W(t)$ for $t \geq 0$, with $W(0) = 0$ and $W(s) - W(r) \sim N(0, s - r)$. The differences are normally distributed and the covariance matrix is a simplex. In archaeology, seriation problems are defined in terms of a Robinson matrix (Robinson, 1951), which is a Toeplitz dissimilarity matrix.

16.5.2.2 Band

The data values in the second dataset (Band) were generated from the formula

$$x_{ij} = e^{-t^2} + u_{ij},$$

where the parameters are defined similarly to the simplex model. Unlike the simplex, correlations are both positive and negative; correlations between near columns are positive and correlations between distant columns are negative.

The band correlation structure can be found in a variety of fields. Thurstone (1927) proposed a **law of comparative judgment** that involved a one-dimensional probabilistic, non-monotonic magnitude item scale. Coombs and Avrunin (1977) presented a similar model for preferences. If the columns of X index a set of ordered objects and the rows represent a set of subjects, then a subject's preference for a set of objects can be represented by a (usually single-peaked, symmetric) probability density function centered on or near the most preferred object. Coombs, Dawes, and Tversky (1970) describe several other varieties of this model. In some physical systems (acoustics, optics), single-peaked multivariate spectra exhibit a banded correlation structure when ordered along a common frequency dimension according to spectral sensitivity. In information retrieval, citation patterns sometimes follow a band structure (Packer, 1989). Related texts cite each other but unrelated texts do not. More general citation and hyperlink data often involve a band structure.

16.5.2.3 Circumplex

The data values in the third dataset (Circumplex) were generated from the same formula used for Band, except the variate has been circularized:

$$x_{ij} = e^{-t^2} + u_{ij}$$

where

$$t = \cos(\pi(s_j - r_i))$$

This dataset is named after a term coined by Louis Guttman (1954) to describe a circular correlation structure. In a circumplex correlation matrix, correlations between near columns are positive and correlations between distant columns are negative. As with Guttman's simplex, the circumplex correlation matrix follows a Toeplitz pattern. With the circumplex, however distance is measured on the circumference of a circle. That is, the series $j = 1, \ldots, p$ is mapped to points on a circle between $-\pi$ and π.

The circumplex structure arises in a variety of contexts. In psychology, examples occur in personality, social psychology, and perception (Plutchik and Conte, 1997). LaForge and Suczek (1955) found this structure in student's ratings of other's personalties. The psychopharmic (pharmopsychic?) Timothy Leary popularized a variant of this scale as the Leary Interpersonal Circle (Leary, 1957). Other psychological phenomena have been found to be best fit by a circumplex model: emotions (Ekman and Friesen, 1978; Benjamin, 1979; Russell, 1980), interpersonal traits (Schaefer, 1959; Wiggins, 1979, 1982, 1996), color perception (Ekman, 1954), and pitch perception (Shepard, 1964). Browne (1977, 1992) and Shepard (1978) describe other circumplex models in psychology. In time and spatial models, the **circular serial correlation coefficient** measures circular dependence. Olkin and Press (1969) discuss a circular moving average model.

16.5.2.4 Equi-correlation

The data values in the fourth dataset (Equi) were generated from the formula

$$x_{ij} = t + u_{ij}$$

where

$$t = bi/n$$

and u_{ij} is sampled from a standard normal random variable. The correlations in this dataset are relatively large and positive ($\bar{r} = 0.65, \quad sd_r = 0.05$).

The equi-correlation pattern is found in single-factor datasets. That is, data depending on a single factor have an equi-correlation matrix. Single-factor datasets occur when there are multiple measurements of a single trait, all with common reliability. Perhaps the most famous single-factor model is the **general intelligence** factor called g, originally proposed by Spearman (1904). Spearman analyzed tests involving cognitive abilities and found that all the correlations among these tests were positive. Spearman mistakenly concluded that this **positive manifold** of correlations was evidence of a single underlying intelligence factor. It was a mistaken conclusion because there are ways to generate equi-correlations other than a single factor. Even intercorrelations above .9 do not prove that a set of variates measure (or are caused by) the same thing.

16.5.2.5 Block

Finally, the data values in the fifth dataset (Block) were generated by computing the row-wise k-bit pattern

$$\langle 00, 01, 10, 11 \rangle$$

for blocks of size $n/2^k$ by p/k and adding error. In this example, $k = 2$. For $k = 3$, we would use the bit pattern

$$\langle 000, 001, 010, 011, 100, 101, 110, 111 \rangle$$

The resulting dataset is a set of points near the vertices of a p-dimensional hypercube. Of these vertices, $1/k$ have few points near them.

The ordering of the blocks and the ordering of the rows/columns within blocks is arbitrary. Instead of generating the sorted binary sequence given above, we could use another ordering called the **Gray bits** sequence, for example. A Gray bits sequence is an ordering of 2^n binary numbers such that only one bit changes from one entry to the next. A Gray ordering for $n = 3$ is

$$\langle 000, 001, 011, 010, 110, 111, 101, 100 \rangle$$

These sequences are useful in encoding because they minimize bit changes across tokens, improving the reliability of a physical encoder. They were used by the French engineer Émile Baudot (from whose name we get the computer modem unit *baud*) in 1878 and were patented for use in telecommunications by Frank Gray, a Bell Labs researcher, in 1953.

The block data pattern corresponds to a block-diagonal covariance matrix. Correlations between blocks are near zero (although if some row patterns are missing, these correlations will be larger). And correlations within blocks are near one. Block patterns are common in real data. Experimental designs are an obvious example. The **multimethod-multitrait matrix** defined by Campbell and Fiske (1959) is another. Its high-correlation blocks consist of multiple measurements of a single trait and its low-correlation blocks consist of measurements of different traits. The Thurstone (1945) **simple structure** pattern for describing a simple multiple factor solution is another. From a clustering perspective, we expect to see a block pattern when clusters are convex and distributed on the vertices of a simplex. Some structures in microarray studies in biology appear to be block patterns (Liu *et al.*, 2003).

16.5.3 Permuting Randomly

We begin our analysis by scrambling the data. Figure 16.24 shows our five datasets with rows and columns randomly permuted. Only the Simplex and Equi data matrices appear to have a systematic pattern under this random permutation. While they look similar in the raw data matrix, the scrambled covariance matrix shows them to be different.

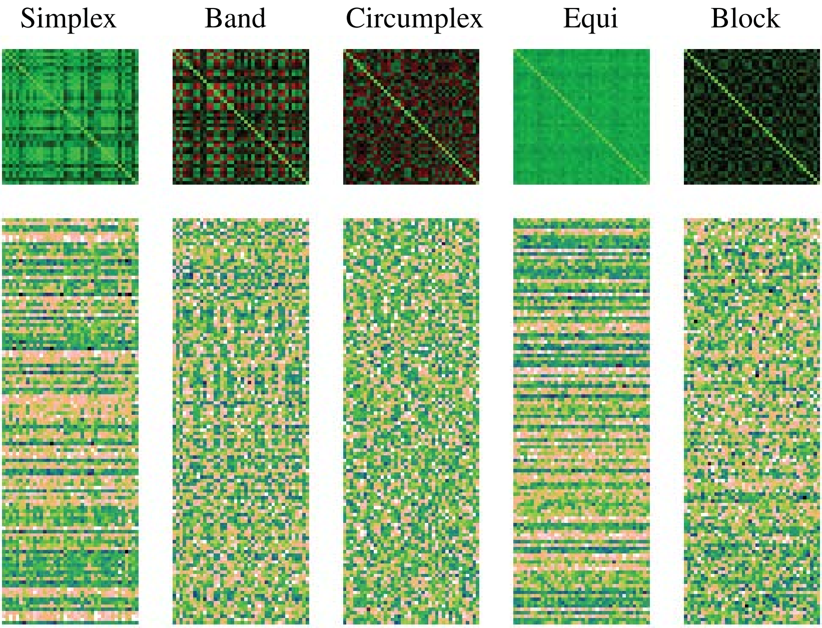


Figure 16.24 *Randomized data structures*

The visual similarity between the Simplex and Equi data matrices is not surprising. Researchers have long been concerned that Guttman scaling programs that permute data to a simplex structure are surprisingly effective on single-factor data. An algorithm tailored to shuffling rows and columns to a simplex structure will push dark rows to the bottom and light columns to the right. We see the other side of the coin in the random shuffling of these two matrices. Dark and light rows are interspersed randomly in both matrices. Because of the column permutations, the half-dark/half-light rows in the Simplex dataset are hidden in the interstices.

16.5.4 Permuting Systematically

We now ask whether there exists an algorithm for permuting the rows and columns of the randomized matrices so as to restore the structures shown in Figure 16.23? Many have investigated this topic in many different fields.

If we define the problem as seeking a row/column permutation that uncovers (possibly non-convex) clusters of similar values in the data matrix when they exist, as well as continuous fields or flows (such as simplex) when they exist, then the problem becomes **NP hard**. To evaluate the relative goodness of a solution, we would have to compare it to another solution, and to verify that a solution is global, we would have to compare it to every other possible solution. And since the total number of permutations grows factorially with matrix size, the solution time would be worse than polynomial.

Consequently, researchers have tended to focus on multivariate statistical methods designed to recover specific multivariate structures. They usually have applied these methods to **XX**' and to **X**'**X** separately, although some have devised algorithms to work on rows and columns jointly. Many of these methods have been rediscovered and given peculiar names, particularly in different fields. In this research summary, we cite earlier sources rather than some of the recent incarnations and rediscoveries.

Stouffer *et al.* (1950), Bertin (1967) and others seriated matrices by hand-sorting to a Guttman simplex structure. When computer card sorters became generally available in the 1960's, many researchers used them to sort data card decks to a Guttman scale pattern. Wilkinson (1979) developed an efficient computing method for doing this particular seriation on large matrices.

Hartigan (1972) applied cluster analysis to the matrix permutation problem. He devised a procedure that clustered rows and columns jointly. His algorithm was implemented in the Block Clustering routine of the BMDP statistical package. Slagel *et al*. (1975), Defays (1978), and many others used row/column clustering to permute a matrix.

McQuitty (1968) observed the convergence (to a binary matrix) of the sequence $\mathbf{R}^{(1)}$, $\mathbf{R}^{(2)}$, . . ., $\mathbf{R}^{(n)}$, where $\mathbf{R}^{(i)}$ is the correlation matrix of $\mathbf{R}^{(i\text{-}1)}$. Breiger, Boorman and Arabie (1975) applied this method to seriation.

Others have placed matrix permutation in a class of graph-theoretic sorting problems, and have used dynamic programming methods to attack them (McCormick *et al*., 1972; Lenstra,1974; Hubert, 1974a, b, 1976; Hubert and Golledge, 1981). Matrix reordering has relevance to many graph partitioning and sparse matrix problems, but this application is less relevant to our situation, since we expect to encounter more general patterns.

Many have recommended principal components (PC), singular value decomposition (SVD), correspondence analysis (CA), and multidimensional scaling (MDS) for the matrix permutation problem. Computationally, PC (on **XX**' and **X**'**X**), SVD, and CA are all based on the same matrix decomposition, so they are equivalent (apart from minor rescaling issues). We will have more to say about these methods later.

Finally, Wilkinson (2004) and Niermann (2005) explored pixel–based methods for permuting. These methods attempt to minimize a local measure of entropy, roughness, or stress computed over all the pixels of the matrix, Because the problem is NP hard, Wilkinson proposed simulated annealing and Niermann proposed genetic algorithms as heuristic solutions.

16.5.4.1 Permuting by Cluster Analysis

Perhaps the simplest and most popular permutation method is to apply hierarchical clustering separately to rows and columns. For permuting the columns, we use the Euclidean distance matrix of the columns and for permuting the rows, we use the Euclidean distance matrix of the rows. If the data are standardized, as in our example, then the column Euclidean distances and correlations are a simple function of each other, through the formula $d_{ij} = 1 - r_{ij}$.

To produce Figure 16.25, we computed a hierarchical clustering using **average linkage** to seriate our data. Average linkage assumes the distance between two clusters is equivalent to the average of all the distances between members of one cluster and members of the other. Average linkage is a middle-ground between two extremes (single and complete linkage). Over all *i* in cluster *A* and *j* in cluster *B*, we define

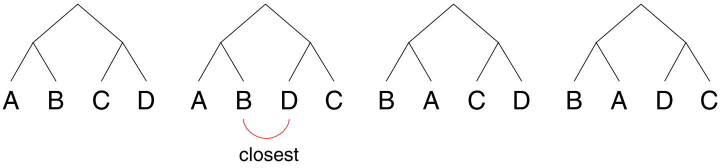
$$\text{single linkage:} \quad d_{AB} = min(d_{ij})$$

$$\text{average linkage:} \quad d_{AB} = avg(d_{ij})$$

$$\text{complete linkage:} \quad d_{AB} = max(d_{ij})$$

Complete linkage and single linkage are at the extremes of a continuum that includes most other hierarchical methods (Mirkin, 1996). Complete linkage tends to produce spherical clusters and single linkage tends to produce snaky clusters.

Although hierarchical clustering is a popular method for permuting data in heatmaps, it is not sufficient for producing a seriation of the sort we are seeking. Because a rooted tree is invariant under transposition of its branches, we must find an ordering of the leaves that is unique and desirable for our application. Consequently, we follow a method in Gruvaeus and Wainer (1972) for producing a unique ordering of a binary hierarchical clustering tree (assuming no tied distances). After each join step, we compare all four possible edge permutations of leaves as follows.



We pick the permutation that has the closest data distance between the two middle leaves. For example, if $d_{BD} < d_{BC} < d_{AC} < d_{AD}$, we pick the ABDC ordering for this join step, as shown in this figure. This is the default method for displaying hierarchical clustering trees in SYSTAT. Figure 16.25 shows the result. Hierarchical clustering rearranges Simplex and Band into blocks, although they are ordered sufficiently to reveal the diagonal and banded structures. The Circumplex and Equi-correlation data are rearranged quite well. Clustering uncovers the blocks in the Block dataset, but the rows are not sorted to the original pattern. Notice that the bottom block of values in the original matrix is split in two, a consequence of the local behavior of most clustering algorithms.

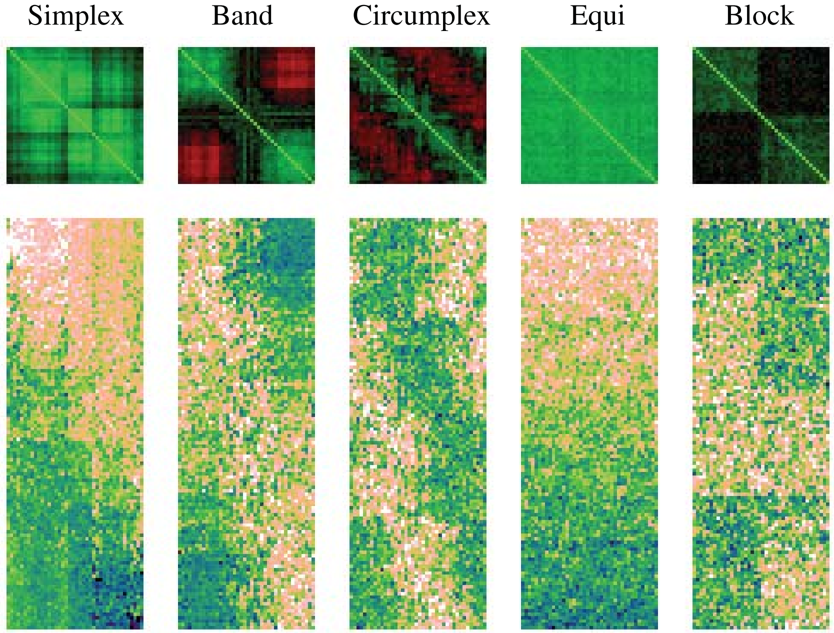


Figure 16.25 *Matrices permuted by average-linkage clustering*

16.5.4.2 Permuting by Eigen Decomposition

Figure 16.26 shows the SVD permutation of our datasets. SVD does well on Simplex, Band, and Equi. It degenerates the middle of Circumplex, however, because it cannot find the circular dimension. We will discuss this further below. The Block pattern is clear in the columns, but the rows are less distinct.

The Simplex, Band, Circumplex, and Equi datasets have intrinsic dimensionality of 1. That is, the deterministic component of each depends on a single parameter (*i,* or the row index). Thus, we should expect that any effective permutation method for these matrices should identify this unidimensionality. If we evaluate SVD on this criterion, it fails. Let’s examine this further.

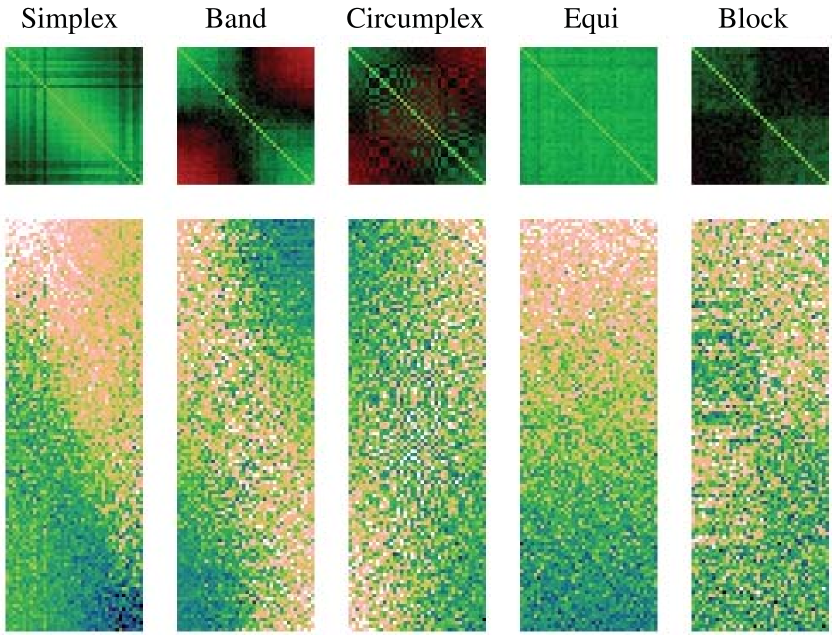


Figure 16.26 *Matrices permuted by singular value decomposition (SVD)*

Figure 16.27 shows the first two principal components, colored by the original column indices. Simplex and Band are forced into two dimensions because the model computes similarity on the basis of cosines between pairs of vectors. Since the vectors are constrained to have unit length, the projection requires two dimensions. The circumplex is recovered in the first two components. Unfortunately, we cannot detect its unidimensionality unless we convert to polar coordinates, and we cannot know that in advance. Finally, SVD recovers the single factor underlying Equi and the two factors in Block.

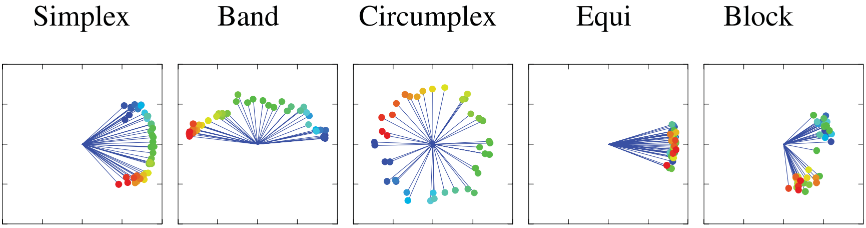


Figure 16.27 *Two-dimensional factor solutions*

16.5.4.3 Permuting by Multidimensional Scaling

Because it works well in recovering nonlinear data structures, iterative nonmetric MDS has been suggested as a seriation alternative. Figure 16.28 shows the solution for our datasets. MDS performs similarly to SVD.

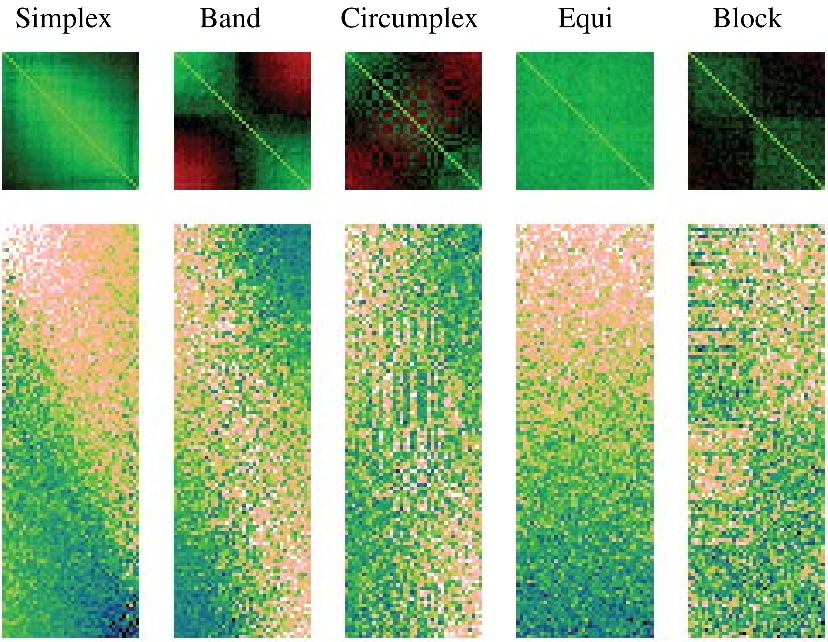


Figure 16.28 *Matrices permuted by multidimensional scaling (MDS)*

Figure 16.29 shows two-dimensional MDS solutions for the columns of our datasets. For the SVD model, correlations are mapped to cosines between vectors. For the MDS model, we work with Euclidean distances, so we do not draw vectors. The Simplex result appears surprising. Euclidean distances from a perfect simplex maps to a real number line, so why do we have a horseshoe in two dimensions? Kendall (1971) discussed this artifact. When the distances are measured with error, large distances tend to pull the ends together in MDS solutions. See Section 9.4.1 for an example of this phenomenon with real data. A similar artifact occurs for Band. This, too, appears surprising. However, Wilkinson and Ramanathan (1976) proved that logistic variates scaled in two dimensions fall within a football-shaped envelope. They also proved that the stress of the solution is a function of the slope of the logistic curves, not the number of points. Thus, tabled distributions of MDS stress values (e.g., Spence and Graef, 1974) cannot be used for testing unidimensionality in this case.

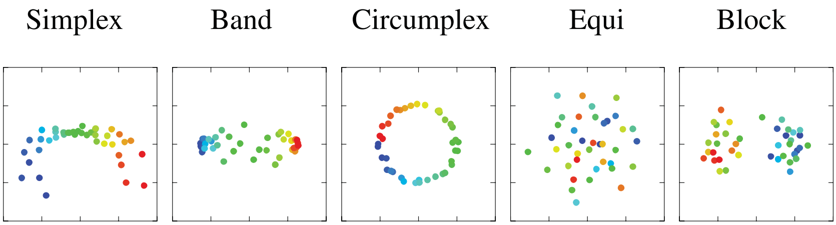


Figure 16.29 Two–dimensional MDS solutions

16.5.4.4 Permuting by Graph-Theoretic Dimension Reduction

There are several other nonlinear dimension reduction methods that can be adapted to the matrix permutation problem. Locality Preserving Projection (LPP) is a locally linear method for finding low–dimensional manifolds embedded in high–dimensional space (He and Niyogi, 2002). It is similar to metric MDS, but decomposes distances on the *k*-nearest–neighbor graph of a set of points rather than distances on the complete graph of all possible pairwise relations. Figure 16.30 shows an LPP solution for our data. The results are remarkably similar to MDS and SVD, although this similarity is due to the specific manifolds we are working with rather than the algorithms themselves.

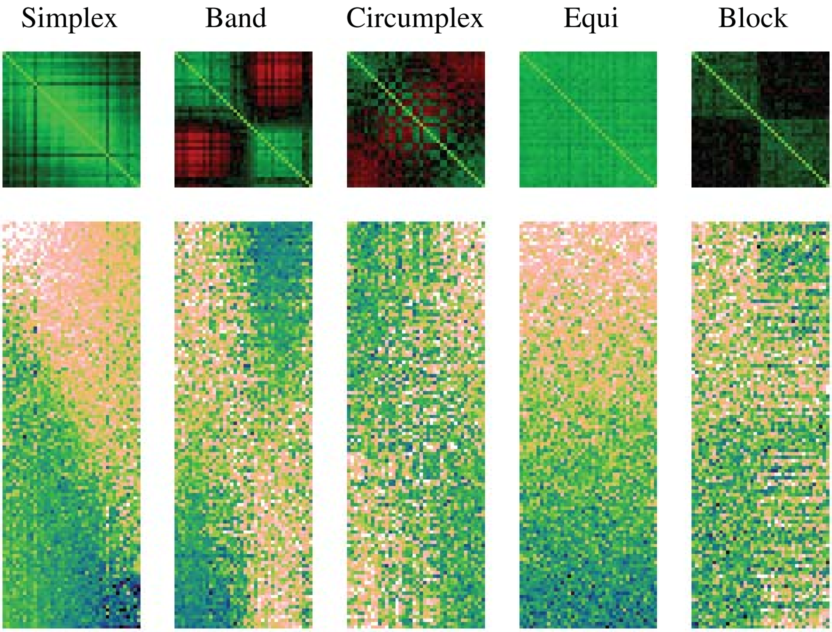


Figure 16.30 Matrices permuted by locality preserving projection (LPP)

16.5.4.5 Permuting by Minimizing Entropy

If we view a heatmap as an image whose pixel values are the row–column entries of a matrix, then it makes sense to permute by maximizing the size of local regions whose pixel values are relatively constant. Suitable measures of this state can be developed from information theory. For example, we described the resolution of an image in terms of Shannon's entropy measure in Section 10.5.5. A well–permuted image can be thought of as one requiring a minimum number of bits to encode it (compared to other permutations). In other words, a well–permuted image is more compressible.

A number of loss functions are related to entropy. Niermann (2005) proposed a variance measure for the neighborhood of pixel x_{ij}

$$S = \sum_{|k-i|<n} \sum_{|l-j|<n} (x_{ij} - x_{kl})^2$$

where n is an integer (usually 2) governing the size of the neighborhood. Wilkinson (2004) proposed a residual variance measure

$$R = \sum_{|k-i|<n} \sum_{|l-j|<n} (\hat{x}_{ij} - x_{kl})^2$$

where $\hat{x}$ is the smoothed value derived from averaging the neighboring pixel values.

Each of these measures is summed over all pixels. Minimizing these measures over all possible combinations of row–column permutations is expensive. Wilkinson proposed simulated annealing as a heuristic approach. Each annealing step consisted of an exchange of two rows or two columns. Niermann proposed a genetic evolutionary algorithm. Niermann's **crossover** (mating) stage consisted of exchanging column (or row) permutations between two parents. His **mutation** stage consisted of reversing the order of randomly selected subsequences of row and column indices. His **tournament** stage consisted of comparing stress values for randomly selected pairs of offspring.

We programmed both approaches and were disappointed to find that they did not work well on our datasets. There were several problems. First, these algorithms converge too slowly to be useful on larger datasets. Second, they are sensitive to parameter settings (annealing constants, evolutionary population sizes, mutation probabilities, *etc.*). These settings need to differ depending on the structure and size of datasets. Third, they did not overcome the difficulties we have seen elsewhere with the Circumplex and Block datasets.

Our hope was to permute initially with SVD and then use a pixel–based method to refine the solution. We saw improvements in the badness–of–fit measure, but not large enough improvements to make a visible difference. Consequently, we do not show results for these methods. For smaller datasets, as in Niermann's paper, they may be useful, although it remains to be seen how much of an improvement over SVD they make in general.

16.5.5 Summary

Figure 16.31 summarizes our results. We generated 20 samples of these five matrices and permuted them with the four methods. For each permutation, we computed a Spearman correlation between the known row indices and the permuted indices. We did the same for columns. Figure 16.31 shows the average correlations and their standard errors from the simulation.

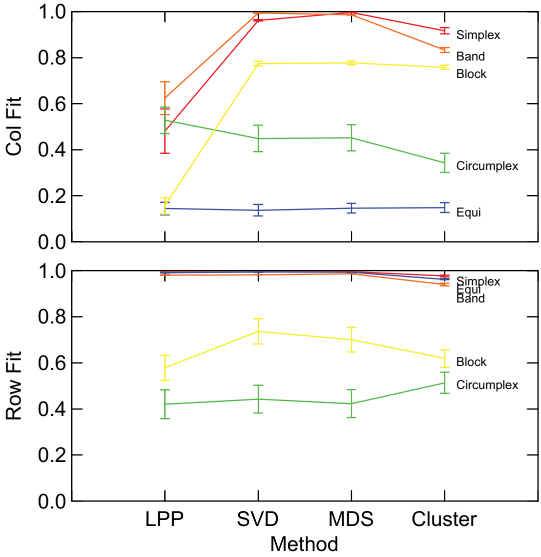


Figure 16.31 *Performance of sorting methods*

We see a number of patterns in these results. Circumplex is difficult to fit because it needs to be cut and unwound. To do this, we would have to know we have a closed circle. The Equi column order is random, so no method can be of any use. LPP does poorly with Block because points do not lie on a manifold in that configuration. In general, SVD and MDS do best on these patterns, although clustering might do better on some other structures.

We draw several conclusions from this analysis.

1) Unordered heatmaps of raw data or covariance/correlation matrices are a fairly useless form of graphic display. The information in heatmaps is limited to the spatial distribution of the color map. If rows and columns are not ordered so as to make that distribution coherent, there is little useful information available. Displays such as SPLOMs and parallel coordinate plots have a similar ordering problem, but they contain other information (*e.g.*, marginal distributions of the variates) that can be useful even when the ordering is not optimal.

2) Ordered heatmaps of data matrices are most effective when a unique ordering (based on time, space, or another variable) is known *a priori*.
3) There are no methods we know of that can recover the generated pattern in all five of these datasets. Of the methods we tested, SVD and MDS worked the best. Liu *et al*. (2003) present a robust SVD method that improves solutions for data contaminated by outliers and containing missing values. We see no advantage to using cluster analysis for seriation or matrix permutation unless the point of the display is to document a cluster analysis already used for other reasons.
4) In general, we believe these findings apply to most multivariate displays, including scatterplot matrices (SPLOMs), parallel coordinates displays, Fourier curves, and glyphs. Without a meaningful ordering, these multivariate displays are difficult to interpret. We recommend that statistical graphics software automatically seriate these displays unless the user selects a pre-determined ordering.
5) Other patterns can be constructed as the product of these patterns. Olkin (1973) discusses a product of Circumplex and Equi. The *radex* pattern of Guttman (1954) is a product of Simplex and Circumplex. We would expect to see similar results with these product sets.
6) Pictures are not always worth 1,000 words.

16.6 Sequel

The next chapter covers issues related to controlling graphics. Controllers are interface devices that allow us to change the methods of representation, change aesthetics, and explore data underlying graphics.

17

Control

The word **control** is related to the French words *contre* (from the Latin word for *against*) and *rôle* (a roll or register). To control is to keep a list of accounts so that one can regulate transactions. To control a computer is to regulate its activity through a set of controllers — pointers, sliders, buttons, brushes, scripts, and menus. There is a large literature on UI controllers, produced mainly by cognitive and computer scientists. There is also a considerable literature on chart controllers, produced mainly by statisticians. In fact, most graphics controllers used today can be traced to inventions by John Tukey and his collaborators at Bell Laboratories and elsewhere. For a history and original sources of many of these ideas, see Cleveland and McGill (1988), Cleveland (1988), Buja, Cook, and Swayne (1996), Swayne and Buja (1998), Unwin (1999), and Friedman and Stuetzle (2002).

We will discuss two aspects of user control in this chapter. First we will present methods for building charts. This area is often neglected, perhaps because it is assumed that standard operating system controllers are sufficient for chart creation. Second, we will cover interactive methods for exploring data. As with chart building, we will see that chart exploring involves many methods peculiar to the world of statistical graphics.

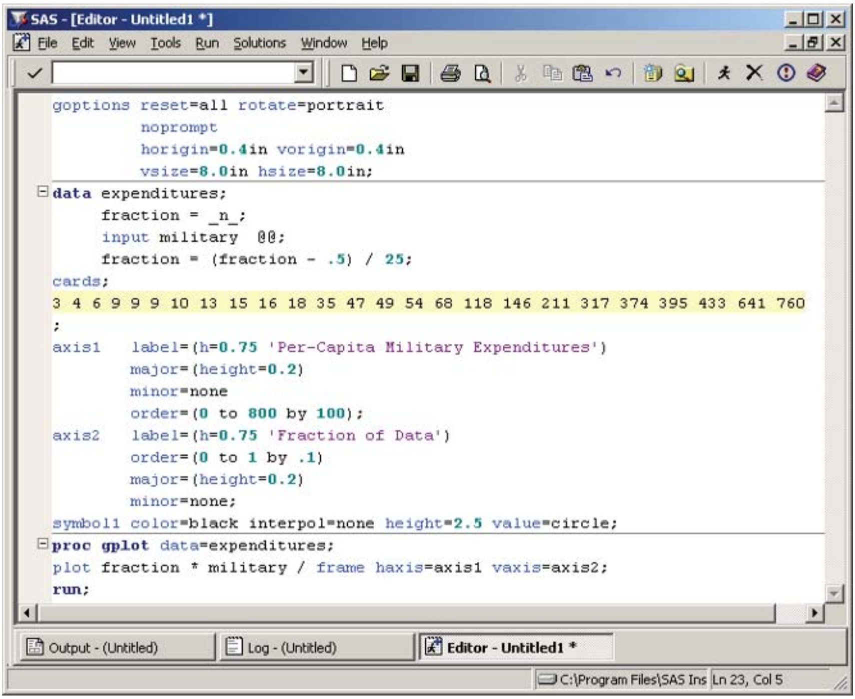
17.1 Building

Specifying to an application how to draw a graph has evolved significantly in the last 25 years. Command lines and procedural programs designed for experts have given way to dialog boxes, wizards, and drag-and-drop interfaces. Building a basic chart is now a routine task. Unfortunately, point-and-click and drag-and-drop operations have made more difficult the auditing and automating of graphics production. No interface is best for all graph-creation tasks. In this section, we will review briefly the history of graph-creation methods and indicate their strengths and weaknesses.

17.1.1 Procedural Languages

The older statistical software packages (*e.g.*, SPSS, SAS, SYSTAT) have underlying procedural languages accessible through a command-line interface. This is not only because these packages predate graphical user interfaces but also because scripting languages provide a degree of control not readily obtained in a menued system. In addition, graphics programs provide a reproducible log of an analysis that can be rerun or modified to run on a different data set. Once GUIs became popular, nearly all of the older packages added dialog boxes to generate code for their language interpreters.

Figure 17.1 shows a SAS-GRAPH® program to produce the graph in Figure 6.2. The code is verbose because it includes transformations and specifies details concerning the size of the plot and configuration of its axes.


SAS - [Editor - Untitled1 *]
File Edit View Tools Run Solutions Window Help

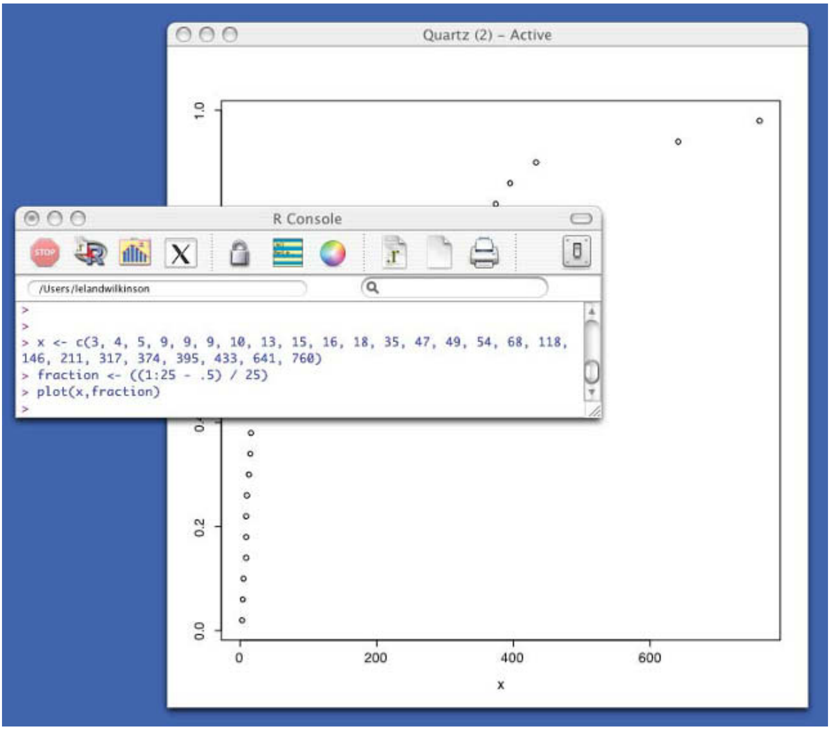
```
goptions reset=all rotate=portrait
          noprompt
          horigin=0.4in vorigin=0.4in
          vsize=8.0in hsize=8.0in;
data expenditures;
     fraction = _n_;
     input military  @@;
     fraction = (fraction - .5) / 25;
cards;
3 4 6 9 9 9 10 13 15 16 18 35 47 49 54 68 118 146 211 317 374 395 433 641 760
;
axis1    label=(h=0.75 'Per-Capita Military Expenditures')
         major=(height=0.2)
         minor=none
         order=(0 to 800 by 100);
axis2    label=(h=0.75 'Fraction of Data')
         order=(0 to 1 by .1)
         major=(height=0.2)
         minor=none;
symbol1 color=black interpol=none height=2.5 value=circle;
proc gplot data=expenditures;
plot fraction * military / frame haxis=axis1 vaxis=axis2;
run;
```

Figure 17.1 *Picture of a SAS® graph command line interface*

17.1.2 Object-Based Languages

Some statistical software tools, such as the S and R programming languages, use an object-based approach. The advantage of these tools is that they encapsulate details inside objects. The disadvantage, shared with other graphics programming languages, is learning to use all their options. In these systems, graphics and statistical functions are objects that have numerous detailed methods hidden but available. As with procedural languages, GUI interfaces with dialog boxes are often used to generate the code for the interpreter.

Figure 17.2 shows the creation of the same graph using the R console. Because R is a programming language for statistics and graphics, it works the same way on Linux, Unix, Macintosh, and Windows machines. Figure 17.2 was created on a Macintosh.



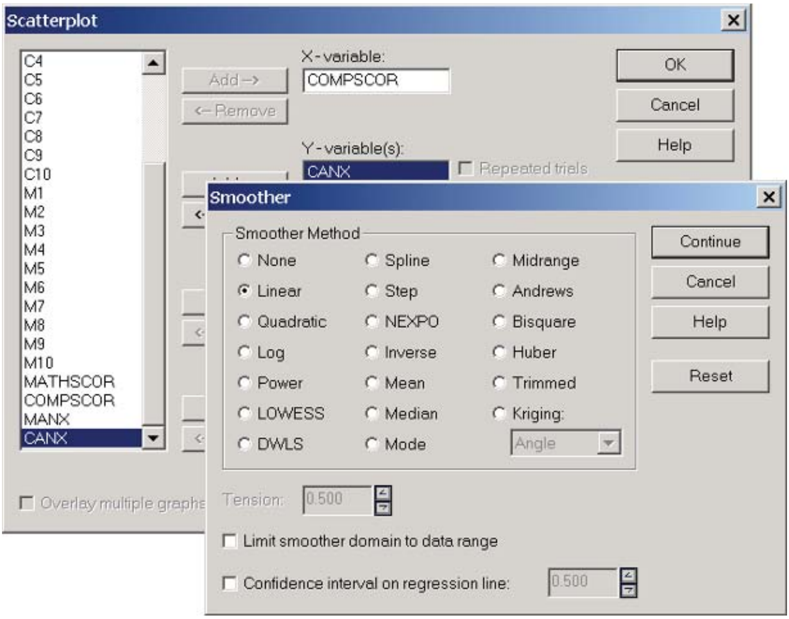
```
>
>
> x <- c(3, 4, 5, 9, 9, 9, 10, 13, 15, 16, 18, 35, 47, 49, 54, 68, 118,
146, 211, 317, 374, 395, 433, 641, 760)
> fraction <- ((1:25 - .5) / 25)
> plot(x,fraction)
>
```

Figure 17.2 *Picture of the R command line interface*

17.1.3 Dialog Boxes

The word *dialog* comes from the Greek $\delta\iota\acute{\alpha}\lambda o\gamma o\varsigma$, which means alternating speech, as in a conversation. The use of the term *dialog* to describe menus, panels of check boxes, tool bars, and other miscellaneous collections of visual controls is a misnomer. Controls involve dialogs only when they force the user to interact sequentially. To novices, dialogs can be comforting because they present available alternatives in panels of choices. To experts, dialogs can be frustrating because they are modal; they partition specifications into response classes (menus, subdialogs) that force a user to follow a fixed construction framework. Modal frameworks are especially inappropriate for constructing graphics because they compartmentalize fluid tasks.

Figure 17.3 shows an example of the scatterplot construction dialog and the smoother subdialog in SYSTAT. Because SYSTAT offers a large number of smoothers, the choices in a dialog can intimidate a novice. A large number of features drives GUIs to a large number of dialogs, which is perhaps why many graphics systems that offer only dialogs tend to skimp on features.



Scatterplot

X-variable: COMPSCOR

Y-variable(s): CANX

□ Repeated trials

Smoother

Smoother Method: ● Linear

Tension: 0.500

□ Limit smoother domain to data range

□ Confidence interval on regression line: 0.500

Figure 17.3 *SYSTAT dialog box for specifying a scatterplot*

17.1.4 Wizards

Dialogs group sets of tasks into conversational units. Despite this organization, novices can be paralyzed by choices. Wizards are dialogs taken to extremes. They force tasks into a fixed order so that a casual user has no opportunity to make a wrong choice. Needless to say, wizards are the most frustrating interface for expert users. And for richly featured graphics packages, they are impractical.

Figure 17.4 shows a panel from the Excel chart wizard. To keep things simple, the wizard employs a chart metaphor, which severely limits the capability of the package. In order to get a user to a recognizable state or sub-goal, wizard chart interfaces serialize the construction process. They then provide a chart editor to modify details of the original construction. This backtracking strategy helps novices achieve their goal but wastes steps for the expert.

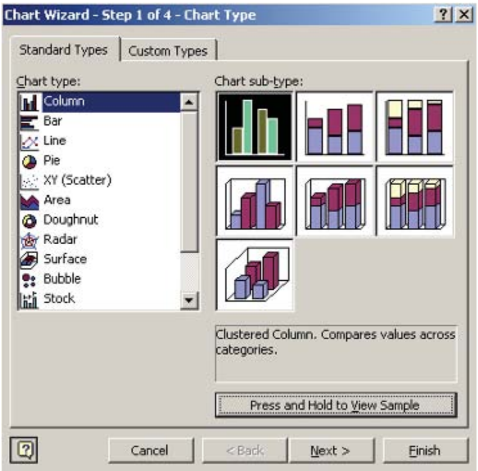


Chart type: Column

Clustered Column. Compares values across categories.

Figure 17.4 *Chart Wizard in Microsoft Excel*

17.1.5 Graphboard

The **graphboard** interface for creating and modifying graphics was introduced in the first edition of this book. Its name suggests a whiteboard, but it also incorporates rules that make objects written on it behave in a predictable manner. The graphboard is a whiteboard with a grammar.

The following sections are organized by types of interaction. The first section, *Learning*, describes available help. The second, *Playing*, describes the process of graph creation and exploration. These are not serial stages of interaction. One may play before learning or learn before playing. We have tried to eliminate dialogs, wizards, and other order-dependent devices so that both novices and experts can explore, backtrack, and modify without being forced through steps someone else imagined to be helpful.

The style of these sections is motivated by user scenarios rather than theoretical issues. As much as possible, we will present issues within the flow of actual user interactions in a session. This requires more figures, but is closer to the experience of using the real system. Macintosh users should substitute "command-click" for the term "right-button mouse click" in the following descriptions.

17.1.5.1 Learning

Figure 17.5 shows a basic graphboard. It is split into two regions: a **control panel** and a **whiteboard**. The whiteboard in Figure 17.5 contains the help screen a user sees after pressing the help (?) button in the control panel when the screen is empty (although it can be pressed at any time). The help screen contains a lot of information, but it has a transparent typeface that overlays existing graphics without obscuring them. This help overlay obviates the need for a separate hierarchical help system that requires multiple user-interactions before offering the desired assistance. And unlike some context-sensitive help, this system facilitates browsing in a single screen. This is not the only help available within the system, of course. Other help is offered after significant user delays in responding or in specific contexts where the system recognizes the need for support or as pop-up tips. This text in the basic help screen reveals the fundamental characteristics of the graphboard.

First, most user interactions with the system are based on the graphics themselves, not on dialogs or wizards. We also avoid remote controllers, which pop up as tool bars or extra windows and thus clutter the work environment. Wherever possible, we prefer to manipulate the objects themselves instead of manipulating proxies.

Second, all actions in the graphboard are reversible, so it is straightforward to return to where one was several steps back. There is no need for UNDO. This characteristic is due partly to the grammar underlying the board and partly to the design of the controls themselves.

Third, the order of operations needed to reach a goal is usually not fixed. There are often several paths to reaching the same goal. This characteristic reflects the way we draw many complex figures. We can assemble components in any order as long as the result contains all the needed components.

Controls

The control panel contains the basic tools needed to create graphics. At the top is a **scroll-list** of variable names in the data source. These are drag-and-drop objects that represent variables in the graph specification. When we wish to include a variable in a graph, we place our cursor arrow on the name and hold down our mouse button and drag a copy of the variable name into the whiteboard. To remove a variable from a graph, we do the reverse: drag the copy of the name back into the scroll-list. Some applications use a double click to signal this kind of reversal. We prefer to use the drag-out-of-frame operation to reinforce the idea that gestures are invertible.

At the bottom of the list are italicized names of synthetic variables. These names do not exist in the data source. They are needed whenever we wish to construct a dimension from functions of the data, as with histograms or bar graphs of counts. See Figure 3.6 for examples.

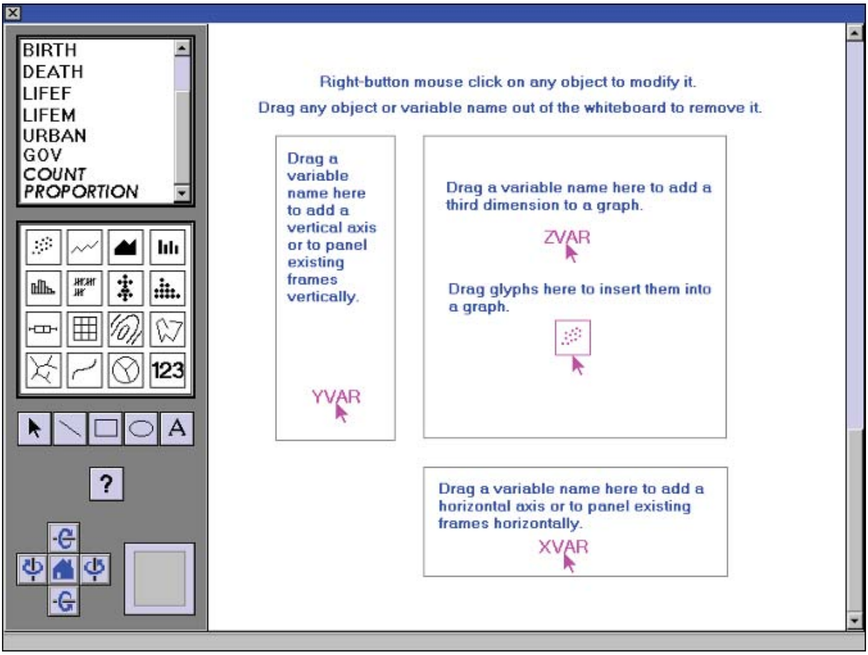
The section below the scroll-list is a **glyph tray**. Dragging an image of a graphic into the whiteboard specifies that a graph is to be represented with this object. We may drag one or more of any of these into the whiteboard in any order, before or after we have dragged variable names into this area.

The design and behavior of this tray is meant to convey to the user that it is not a list of chart types. Some graphics programs have buttons with lines, bars, and other images that enable users to choose a type of chart. The glyph tray works instead by dragging, so that a user is encouraged to try more than one graphic in a frame. We have made two conspicuous concessions to user preconceptions, however. One is the pie icon at the bottom of the tray. As we have seen in Chapter 2, a pie is a stacked *interval* in polar coordinates. This insight is too much to expect of non-technical users, however. Dragging a pie icon is therefore equivalent to dragging a *interval* icon and setting the coordinates of the frame to polar. Similarly, the smoother icon (to the left of the pie) is designed to be equivalent to dragging a *line* and setting the statistical method to a smoother. Other icons could be added to this tray for similar purposes. It is designed to grow to accommodate them. There is always a tension, however, between keeping an interface simple versus making it look like the cockpit of a fighter jet in order to impress technical users or to expose rarely used options. These decisions must be made carefully and with accompanying user testing. Finally, the numerical icon (123) is for using numerals instead of other graphics to represent quantity.

Below the glyph tray is a set of **annotation** tools. These tools are for drawing forms on a graphic, such as lines, rectangles, ellipses, or text. They are not intended to replace or duplicate paint programs. Instead, annotations are intended to link meaningfully to objects in a frame. The leftmost arrow-button is for restoring the cursor to the standard pick-arrow after drawing with a tool.

At the bottom of the control panel are two exploration tools. The one on the left is a **rotation pad** for orienting 3D images. Its central *home* button restores the orientation to the default view and the other buttons control two degrees-of-freedom of rotation. On the right is a **pan-and-zoom** controller. The central, darker square can be resized to demarcate the subset of the graphic shown in the whiteboard. For tables of graphics, this controller can be used to show anything from a single frame up to the entire table.

Finally, the **help button** (?) controls the help system. Help is always given in the whiteboard and is contingent on the material displayed. At times, this material may obscure the material in the whiteboard, but only when necessary to avoid clutter.



Right-button mouse click on any object to modify it.

Drag any object or variable name out of the whiteboard to remove it.

Drag a variable name here to add a vertical axis or to panel existing frames vertically.

Drag a variable name here to add a third dimension to a graph.

Drag glyphs here to insert them into a graph.

Drag a variable name here to add a horizontal axis or to panel existing frames horizontally.

***Figure 17.5** Graphboard help (?)*

As we shall see, the graphboard offers additional capabilities for interacting with the system, but these are not implemented with visible controls. There are several means for accomplishing this. First, as the help screen shows, dragging operations are defined by their target area in the whiteboard. Dragging variable names into the **vertical target area** at the left of the whiteboard assigns them to a vertical (*y*) dimension. If there are any variables already assigned to this vertical dimension, then dragging a variable name to this area panels the frames according to the values of the variable being dragged. We will see this operation in Section 17.1.5.2. Dragging variable names into the **horizontal target area** at the bottom of the whiteboard assigns them to a horizontal (*x*) dimension in the same manner as the vertical. Dragging variable names to the **frame target area** adds a third (*z*) dimension to the graph. Finally, dragging variable names on top of other variable names *blends* the two variables (see Section 5.1.2 for the definition of a blend).

The second method for implementing other controls is the use of delay. After 1,500 milliseconds, the system responds with prompts. These may be help messages or modal choice options presented to qualify the meaning of a gesture. This time-dependent modality, such as a delayed interruption requiring a choice between *cross* and *nest*, is employed when one option (in this case, *cross*) overwhelmingly predominates in ordinary applications. Expert users can turn off this delay feature.

17.1.5.2 Playing

We begin with the countries data used in Chapter 1. The variables in that dataset are listed in the variable scroll-list of Figure 17.6. In each figure of this section, we will represent the user action by a strobed arrow (pale blue) next to the thing being dragged (pink). The consequence of the action is what shows in the figure. In Figure 17.6, dragging a variable name (lifem) into the horizontal target area of the whiteboard creates a horizontal (*x*) axis with scale values produced from the data. If we placed our cursor arrow on the axis label ("Male Life Expectancy") in the whiteboard and dragged it back to the scroll-list, we would return to our previous state, an empty whiteboard. If, alternatively, we dragged the axis label to the vertical target area at the left edge of the whiteboard, we would have created a vertical axis. The specification produced by the action is shown above the figure.

GUIDE*: axis*(*dim*(1), *label*("Male Life Expectancy")

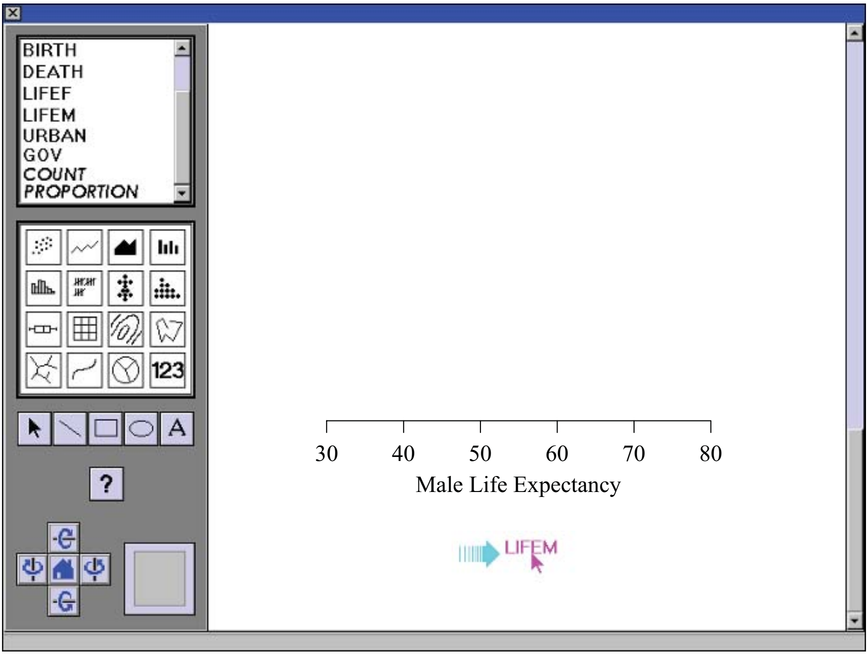


Figure 17.6 Adding a variable to create a dimension

Figure 17.7 shows the result of adding a *point* graphic to the frame. We have dragged the *point* icon from the glyph tray into the central frame target area. The result is a one-dimensional scatterplot.

GUIDE*: axis*(*dim*(1), *label*("Male Life Expectancy")
ELEMENT: *point*(*position(*lifem))

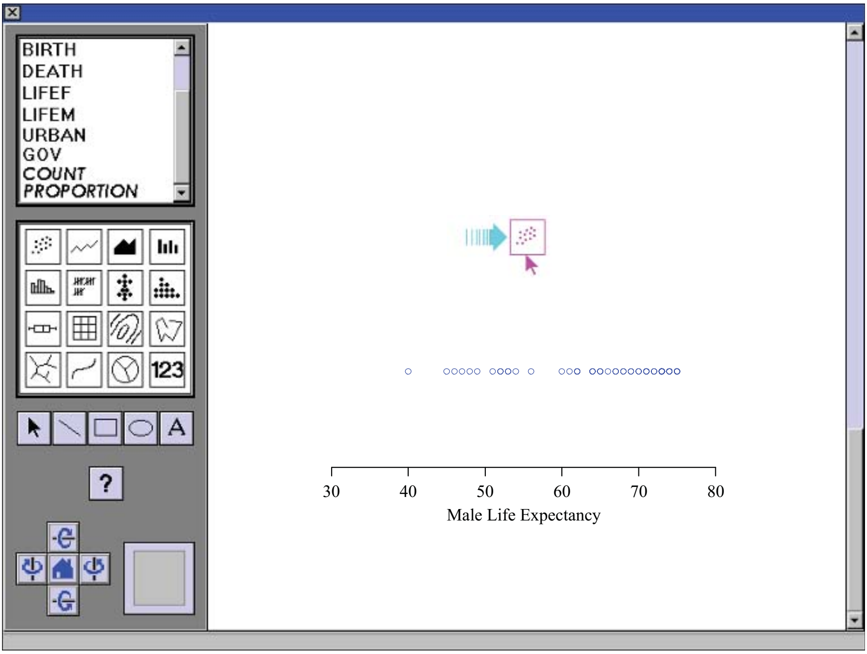


Figure 17.7 *Adding a point graphic to a frame*

Notice that the result looks like a one-dimensional scatterplot. If we had originally dragged lifem to the left-most area controlling the vertical axis, then the gesture in Figure 17.7 would have resulted in a one-dimensional vertical scatterplot. If we had wanted to put in two axes before dragging in a *point*, then we could have dragged a second variable to the other axis box, viewed two empty axes, dragged in *point*, and viewed a 2D scatterplot. This latter series of gestures is probably closer to the graph-construction script most users would expect. However, few non-technical users are aware that a one-dimensional scatterplot exists at all. Exposing users to new creatures is part of the purpose of creating a zoo-like environment that allows them to visit cages without a prior appointment. Or, even better: select the right animals and environment so that we can do away with cages.

Figure 17.8 shows the result of dragging a second variable from the scroll-list to the vertical target area. This adds a second dimension to the frame. Now we see two axes, and the point cloud expands to become a 2D scatterplot.

GUIDE: *axis*(*dim*(1), *label*("Male Life Expectancy")
GUIDE: *axis*(*dim*(2), *label*("Female Life Expectancy")
ELEMENT: *point*(*position*(lifem*lifef))

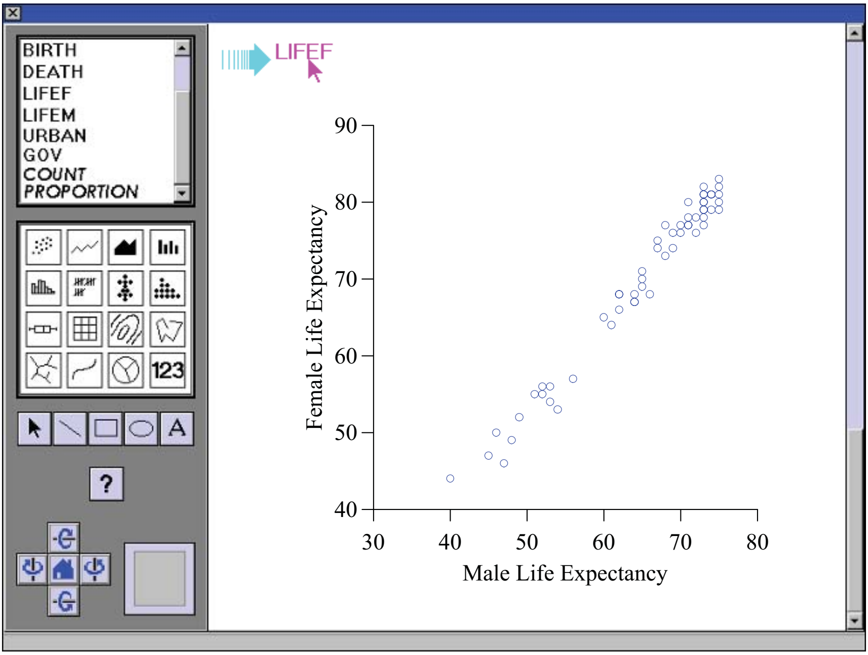


Figure 17.8 *Adding a second dimension to a frame*

At this point, we could reverse any of the actions we have taken by dragging from the whiteboard back to the control panel. For example, we could remove the *point* graphic and leave ourselves with an empty frame. Or, we could drag the horizontal axis variable lifem back to the scroll list and leave ourselves with a one-dimensional scatterplot on lifef (Female Life Expectancy). We will move forward, however.

Figure 17.9 adds a smoother to the scatterplot. We have dragged the *smoother* icon from the glyph tray to the frame target area. As we mentioned in Section 17.1.5.1, this single action is equivalent to dragging a *line* graphic icon into the frame and then right-button mouse clicking on the line to change its statistical method to a smoother. We assume the *loess* method is the default for a smoother. Figure 17.11 shows an example of this right-button gesture.

GUIDE*: axis*(*dim*(1), *label*("Male Life Expectancy")
GUIDE*: axis*(*dim*(2), *label*("Female Life Expectancy")
ELEMENT: *point*(*position*(lifem*lifef))
ELEMENT: *line*(*position*(*smooth.loess*(lifem*lifef)))

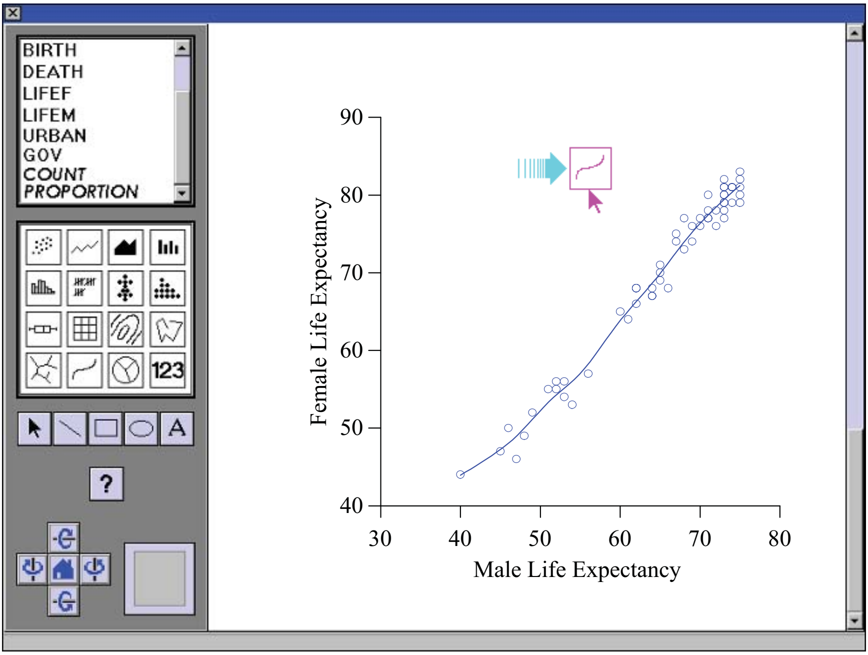


Figure 17.9 *Adding a second graphic to a frame*

So far, we have two variables in our graphical frame model. How do we add more variables? Getting a user to think about dimensions versus variables is a difficult problem. Specifically, we could represent a third variable in this graphic by using *position*() in the frame model to go to 3D, using *position*() with faceting to panel the graphic, or using some other aesthetic attribute such as *color*() to modify the *point* graph. In the graphboard, the 3D world is signalled by dragging a variable into the center of a frame. Paneling through facets is signalled by dragging a variable to one of the edges of a frame. And adding a variable to aesthetic attributes is accomplished by right-button clicking the graphic itself. By applying these rules consistently, we can add more dimensions to simple graphics, paneled graphics, and 3D graphics. We will continue this example by going to 3D first. Then we will backtrack and try attributes. Finally, we will try paneling.

Figure 17.10 shows the result of adding a third dimension to the frame by dragging birth from the scroll-list to the inside of the frame target area. This creates a 3D scatterplot with a surface smoother.

```
GUIDE: axis(dim(1), label("Male Life Expectancy"))
GUIDE: axis(dim(2), label("Female Life Expectancy"))
GUIDE: axis(dim(3), label("Birth Rate"))
COORD: rect(dim(1, 2, 3))
ELEMENT: point(position(lifem*lifef*birth))
ELEMENT: line(position(smooth.loess(lifem*lifef*birth)))
```

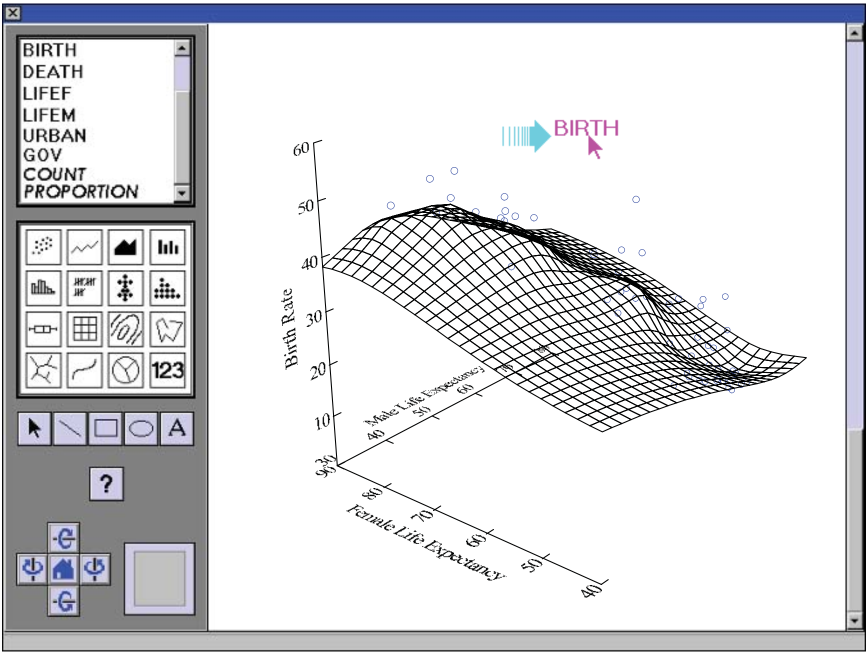


Figure 17.10 *Adding a third dimension to a frame*

Figure 17.11 shows how to remove the third dimension by dragging the birth variable name off the vertical axis and back to the scroll-list. We have also highlighted three other gestures that reveal how to change properties of graphics, frames, and variables. The **graph drop-down list** in the middle of the frame was created by pointing to the smoother in the frame area and clicking with the right mouse button. This list contains three modifiable domains: attributes (*hue*, *size*, *orientation*, etc.), statistical methods (*smooth.linear*, *smooth.loess*, etc.), and properties (associated metadata, annotations, etc.). The content of these domains is documented in Chapter 10 and Chapter 7.

An example of a **scale drop-down list** is shown at the bottom of the frame. It is produced by right-button mouse clicking on a variable label for the desired dimension. The modifiable entries in the list are transformations (listed in Table 4.1), categorization (making a scale categorical or continuous), and properties (metadata and other annotations). These actions would affect the SCALE statement of the specification.

Finally, the figure shows an example of a **frame drop-down list** accessed by right-button mouse clicking in the frame target area (but not on a graphic itself). The coordinates listing directs the user to the coordinate transformations available in the system. And the properties listing points to frame-level properties.

All of these directives arise from right-button mouse operations. In general, right-button (command-click) mousing is a *modify* operation in the whiteboard. It is always relevant to the particular object being pointed to. The right-button is never used for miscellaneous short-cuts or other operations. The specification represented in this figure is the same as for Figure 17.9.

GUIDE*: axis*(*dim*(1), *label*("Male Life Expectancy")
GUIDE*: axis*(*dim*(2), *label*("Female Life Expectancy")
ELEMENT: *point*(*position*(lifem*lifef))
ELEMENT: *line*(*position*(*smooth.loess*(lifem*lifef)))

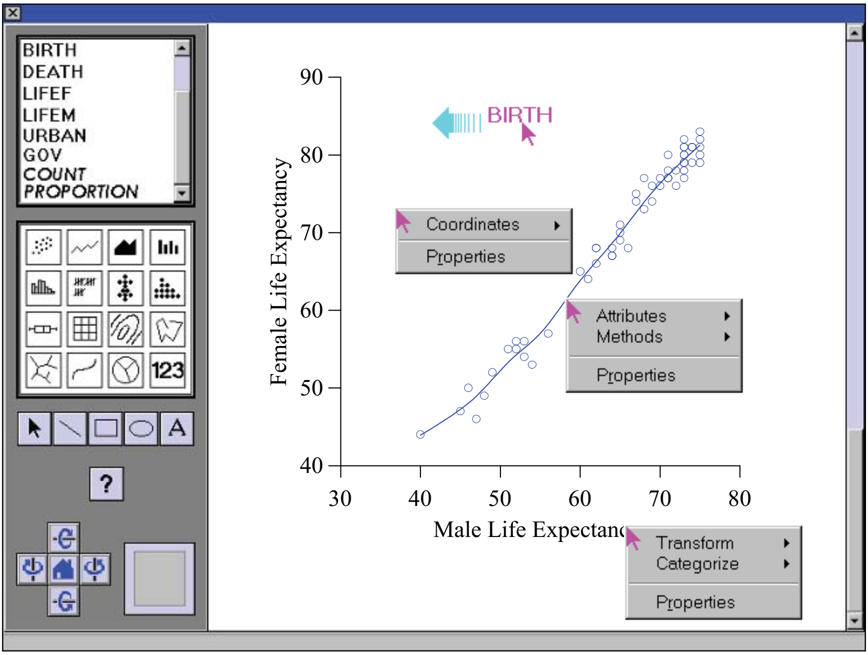


Figure 17.11 *Right button mouse clicks*

Figure 17.12 shows how to panel graphics. We have dragged a variable (gov) into the horizontal target area. This action creates three graphics, one for each value of the gov variable.

```
GUIDE: axis(dim(1), label("Male Life Expectancy"))
GUIDE: axis(dim(2), label("Female Life Expectancy"))
GUIDE: axis(dim(3), label("Type of Government"))
COORD: rect(dim(3), rect(dim(1, 2)))
ELEMENT: point(position(lifem*lifef*gov))
ELEMENT: line(position(smooth.loess(lifem*lifef*gov)))
```

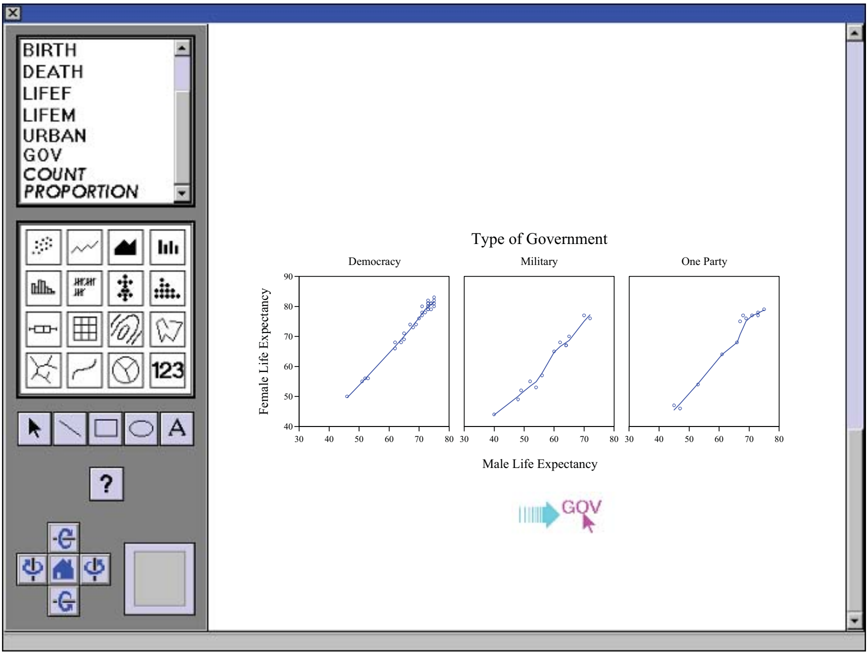


Figure 17.12 *Paneling*

This paneling operation can be carried on indefinitely. We simply drag new paneling variables to either of the margins of the current frame. This implies a structural model for tables of graphics that is documented at the beginning of Chapter 11. That is, we create frames of frames by working from the inside out, like the layers of an onion. The graphboard displays these faceted graphics as larger tables of frames. It would not be difficult to add layout templates to allow layers, or pages of tables. Layout, it must be remembered, is an aspect of the view, not the model.

Figure 17.13 shows how to panel with two variables. We have dragged a second variable (urban) into the vertical target area. This produces two rows of paneled graphics, one for each level of the urbanization variable.

```
GUIDE: axis(dim(1), label("Male Life Expectancy"))
GUIDE: axis(dim(2), label("Female Life Expectancy"))
GUIDE: axis(dim(3), label("Type of Government"))
GUIDE: axis(dim(4), label("Urbanization"))
COORD: rect(dim(3, 4), rect(dim(1, 2)))
ELEMENT: point(position(lifem*lifef*gov*urban))
ELEMENT: line(position(smooth.loess(lifem*lifef*gov*urban)))
```

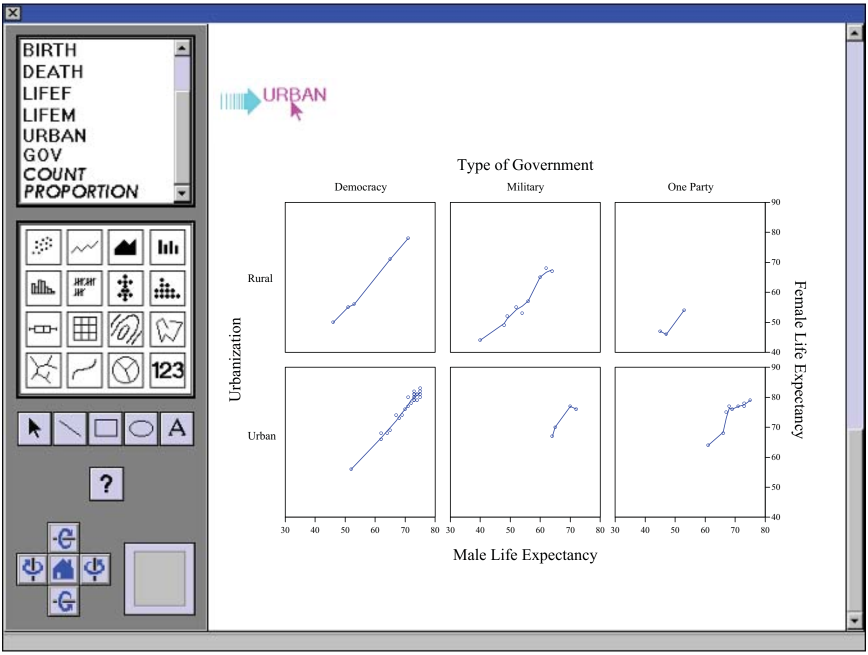


Figure 17.13 Two-way paneling

Although layout is an attribute of views, not models, there are some operations that change appearance through modifying a model. As we explained in Chapter 9, *pivoting* is an operation that exchanges the domain and range of a graph. The following example shows how this operation can be accomplished in the graphboard. This pivoting is performed by drag-and-drop of a variable name (instantiated as an axis label) onto another (on a different axis). The graphboard does not limit this operation to outer variables. Any two axes can be exchanged, thereby modifying the model.

Figure 17.14 shows how to pivot a panel. We have dragged a variable name from one paneling dimension to another paneling dimension. This flips the graphic.

GUIDE: *axis*(*dim*(1), *label*("Male Life Expectancy"))
GUIDE: *axis*(*dim*(2), *label*("Female Life Expectancy"))
GUIDE: *axis*(*dim*(3), *label*("Urbanization"))
GUIDE: *axis*(*dim*(4), *label*("Type of Government"))
COORD: *rect*(*dim*(3, 4), *rect*(*dim*(1, 2)))
ELEMENT: *point(position*(lifem*lifef*urban*gov))
ELEMENT: *line(position*(*smooth.loess*(lifem*lifef*urban*gov)))

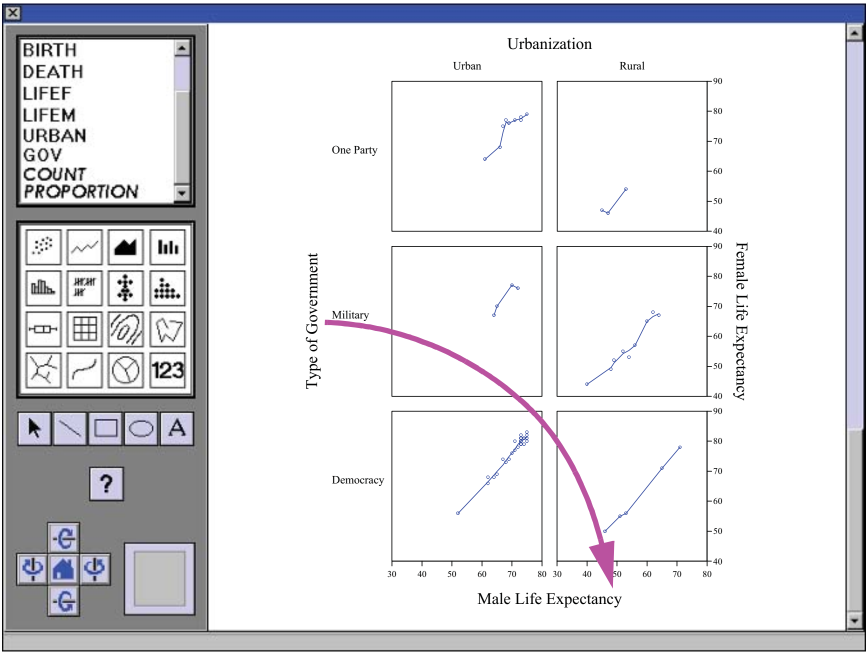


Figure 17.14 *Basic graphboard*

In order to maintain a simple drag-and-drop environment, we have constrained the graphics algebra to models that expand alternatively in vertical and horizontal directions, *e.g.*, a*b*c*d is mapped to *h**v**h**v*. Are there any expressions in Table 11.2 that cannot be created by the graphboard GUI? Can this interface be adapted to work in all the coordinate systems described in Chapter 9? Are there any drag-and-drop gestures available between the control panel and whiteboard that would not be reversible because of the result of a transformation or its non-invertibility? Finally, what special problems does a 3D coordinate system present to this GUI?

17.2 Exploring

The era of interactive exploring of online statistical data began with the PRIM9 project at the Stanford Linear Accelerator Center (Fisherkeller, Friedman, and Tukey, 1974). Among this project's innovations were real-time 3D rotation of scatterplots, built-in analytics and data flow through the PL/I computer language, 2D projections of high-dimensional data through automated projection pursuit and marginal scatterplots, and masking and filtering. Tukey (1988) later went on to describe a variety of additional controllers useful for exploring multivariate data. Many of these tools found their way into exploratory packages such as XGobi (Swayne, Cook, and Buja, 1988), DataDesk (Velleman, 1998), and Spotfire (Ahlberg and Shneiderman, 1994). They are taken for granted in modern visualization systems (Seo and Shneiderman, 2005).

We will discuss exploratory controllers in sections organized by their functionality. In most sections we will illustrate two types of controllers: an **indirect manipulation tool** (usually a widget in a separate window or palette) and a **direct manipulation tool** (usually a mouse-driven selector that works directly on the graphic). Indirect manipulation tools operate on a duplicate image or cartoon of the graphic itself; manipulations of the image are reflected in changes to the original graphic. Examples are sliders, joysticks, and buttons. Direct manipulation tools operate on the graphic itself, so user focus stays on the frame rather than elsewhere in the work environment. Examples are drag-and-drop gestures, haptic (touch) tools, and whiskers and handles for rotating and resizing 3D objects. While there are some advantages to indirect manipulation tools (programmer convenience, a game-style GUI for users under the age of 16), we generally prefer a direct manipulation tool that operates on the object itself.

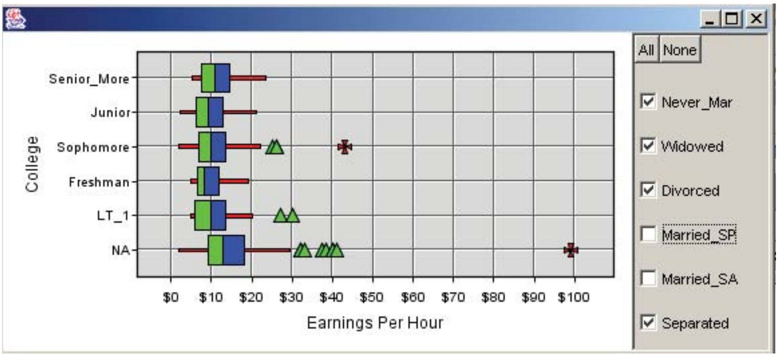
17.2.1 Filtering

Filtering is one of the most common ways of focusing on specific information within a graphic. It is usually helpful to see a certain graphic under a set of constraints that are defined either by categories or ranges of continuous values. Filtering addresses questions like: "what happens in the southeast region?" or "what about those older than 20 and younger than 35?" When filtering was first introduced in database applications with standard charts, it launched a phenomenon in the industry called **drill-down**. In fact, drill-down is simply a set of nested filters.

17.2.1.1 Categorical Filtering

Categorical filtering requires a tool for selecting categories on one or more variables. A simple indirect method is to present a series of checkboxes, one set for each variable. If the box for a category is checked, then the cases containing that category are included in the results. Figure 17.15 shows an example. If the filtering variable is also used for an aesthetic (such as color) in the graphic, it may be possible to apply the aesthetic to the checkbox labels. This allows the filter to serve as a legend, which saves precious display real estate.

If filter variables are available within the domain of the graphic, direct selection is usually preferable. We can provide a filter selection tool (such as a mouse-driven pointer) and use it to select categories directly in a legend or within the frame of a graphic. Multiple filters can be implemented with multiple selection gestures.



☑ Never_Mar
☑ Widowed
☑ Divorced
□ Married_SP
□ Married_SA
☑ Separated

Figure 17.15 *Categorical filtering with checkboxes*

17.2.1.2 Continuous Filtering

Range filtering on a continuous variable requires range selection tools. A simple indirect tool is a double-ended slider. Figure 17.16 shows an example. If an operating system GUI does not offer double-ended sliders, then input boxes for the user to supply minimum and maximum values may suffice. The filtering works by retaining cases whose values lie within the selected interval. The user may slide either the lower or upper handle to adjust the width of the interval, or may slide the whole interval to move the selection region. A useful addition to the slider is to allow for the inverse of the selection. That is, allow a simple gesture so that the user can include only the values lower than the lowest value and higher than the highest value.

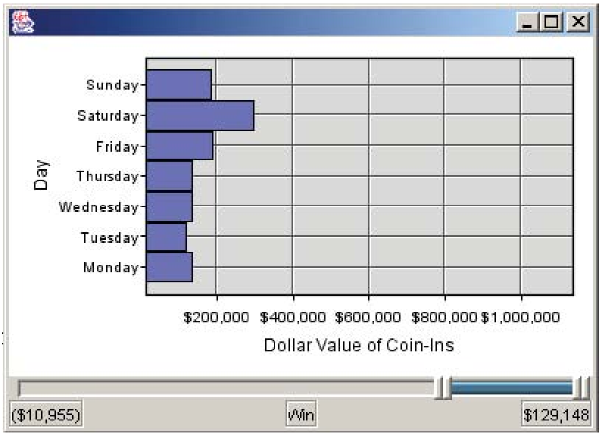


Figure 17.16 *Continuous filtering with a double-ended slider*

Figure 17.17 shows an example of direct selection. The user has drawn a loop with a **lasso tool** in Data Desk to select three outliers. Usually, a rectangular selection tool is convenient for choosing a two-dimensional selection region, but sometimes we wish to choose a joint region with a non-rectangular shape. Freehand drawing tools provide this flexibility.

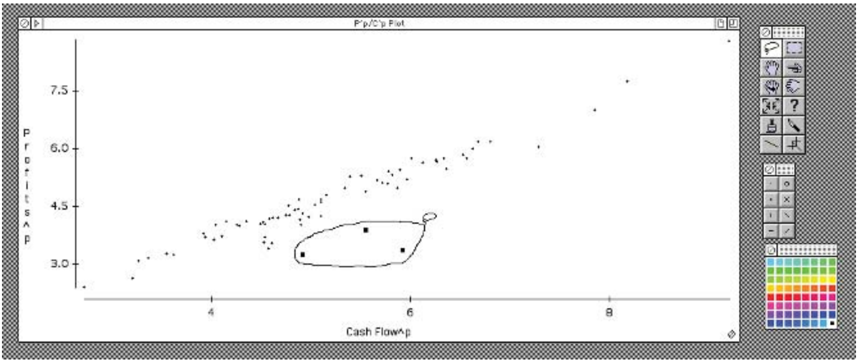
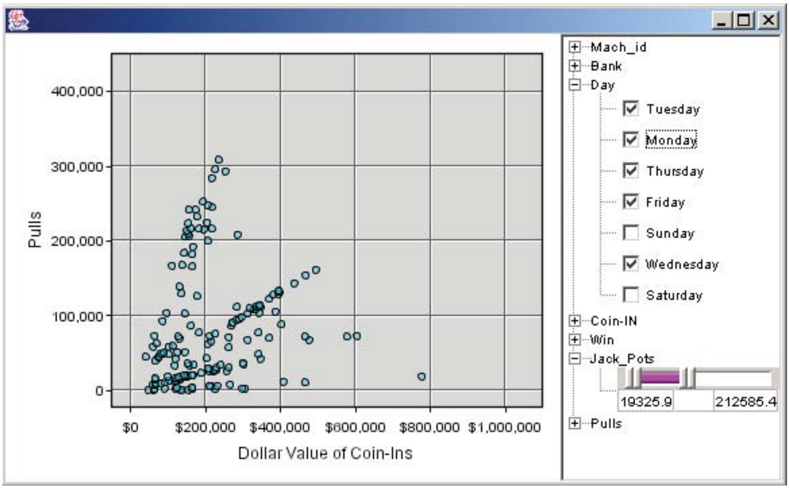


Figure 17.17 *Data Desk lasso tool for graphical filtering*

17.2.1.3 Multiple Filters

Sometimes an application may include several filters working in conjunction. Figure 17.18 contains an example. When there are multiple filters, there is a performance trade-off between redrawing immediately after each selection, or redrawing after all selections have been made. The issue arises because the application may respond faster if the user doesn't have to wait for one filter to finish being applied before moving on to the next. This may not be an issue if the application allows fast filtering (next section).



⊞ Mach_id
⊞ Bank
⊟ Day
☑ Tuesday
☑ Monday
☑ Thursday
☑ Friday
□ Sunday
☑ Wednesday
□ Saturday
⊞ Coin-IN
⊞ Win
⊟ Jack_Pots
19325.9 212585.4
⊞ Pulls

Figure 17.18 *Using multiple filters in a tree on slot machine data*

17.2.1.4 Fast Filtering

Applications that are required to filter with millisecond response time implement filtering as the graphic is rendered. This model is very fast and works very well as long as no complex statistics are required for the graphic. A classic example of a graphic well-suited for this is a scatterplot. Here, the filtering happens fast enough so that the points appear and disappear *as* the user drags the slider. This high level of interaction has the perceptual advantage of behaving like a controlled animation.

17.2.2 Navigating

Navigation is the ability to move around and to focus on interesting areas within a graphic. Navigation can be similar to filtering; the difference is that navigation moves about the graphic, while filtering moves around the data. Another difference is that navigation only involves the positional dimensions included in the graphic as opposed to filtering which can be done using any dimension that exists in the data.

17.2.2.1 Zooming

A common navigation technique is to zoom in on or pan around an area of interest. There are two ways to implement zooming: physical zooming and logical zooming. Physical zooming simply draws a selected small region of a graphic onto a larger area of the screen. This technique is commonly used in image editing, but has limited value for data analysis unless both context and focus are addressed (as in Lensing described below). Logical zooming changes the minimum and maximum values on an axis, but does not change the physical size of the elements in the graphic. Typically, zooming is done by clicking and dragging the mouse to form a rectangular region of interest that becomes the new axes limits. Figure 17.19 shows an example.

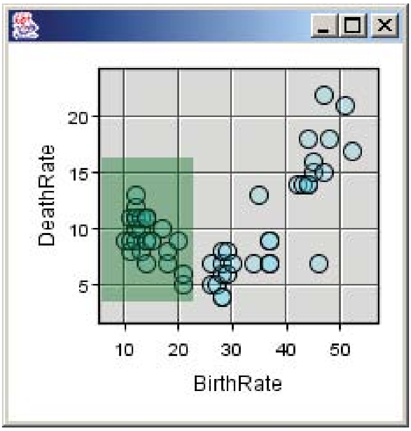


Figure 17.19 *Selecting a region of data to zoom-in on*

Zooming within non-rectangular coordinate systems requires transformed zoom-regions. This is most obvious when zooming on a map. Note how the zoom box in Figure 17.20 respects the Lambert projection.

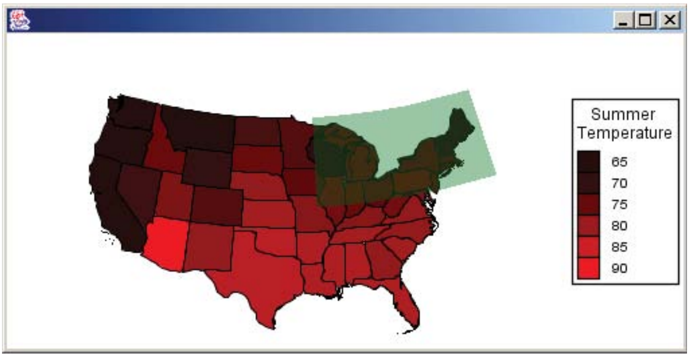


Figure 17.20 *Selecting a region of data under a Lambert projection*

There are several ways to undo a zoom. We can employ an undo button. We can implement a gesture that goes outside the zooming area to signify zooming out. Or, we can implement an indirect zooming tool

17.2.2.2 Panning

Panning is equivalent to moving a data region of fixed size along an axis. This is particularly useful for categorical axes where there simply are too many categories for the screen real estate that is available. This action can easily be tied to a scrollbar in an application; however, note the difference between panning vs. embedding an entire graphic within a scrollable region. The latter simply moves a picture of the graphic around within a fixed viewable region, thus obscuring the axes at times.

Panning can be combined with zooming. Figure 17.21 shows an example of an indirect pan-and-zoom tool that allows us to change focus and navigate at the same time. We resize the box to change the zooming region and move the box around the view-window to change relative focus.

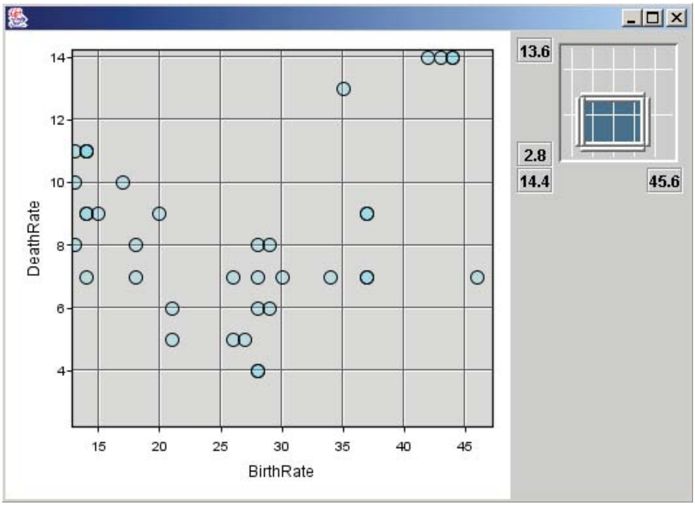


Figure 17.21 *Pan and zoom box*

17.2.2.3 Lensing

Lensing is a type of physical zoom that is able to retain focus while at the same time providing context. Lensing works like a magnifying glass that is dragged over the display. The items in focus become larger and easier to see, while the items at the fringes are pushed aside, yet still visible. This action can provide a glance into data that are highly concentrated in places. Figure 17.22 shows an example.

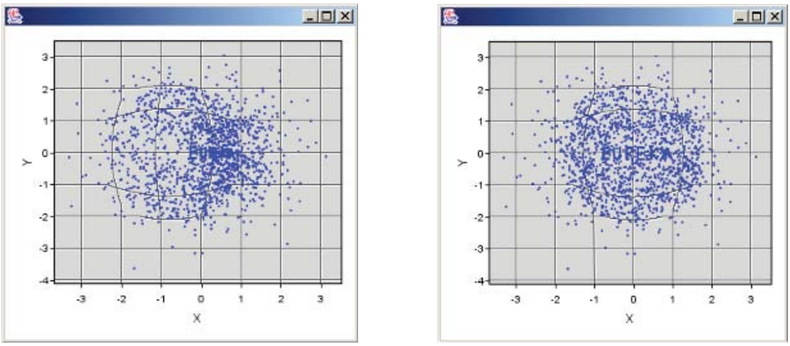


Figure 17.22 *Fisheye lens is dragged from left to right over the "Eureka" dataset*

The common mouse behavior for lensing is to tie the center of the lens to the *x* and *y* position of the mouse as the user drags the mouse over the graphic. Another useful addition when modern hardware is available is to couple the dilation factor of the lensing to the mouse wheel which allows the user to easily expand or contract the magnified region with simple forefinger movements. Figure 17.23 shows an example.

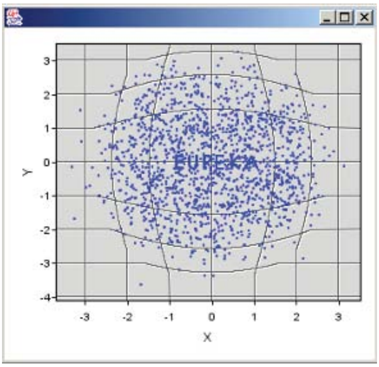


Figure 17.23 *Using the mouse wheel to increase the dilation factor*

There are some graphics where lensing can play an interesting role. Graphics containing the *link* element frequently make use of lensing. Here, lensing provides a useful tool for viewing regions that are too tangled due to a large number of edges in a local region.

Another graphic that can benefit from lensing is a time series plot. Figure 17.24 contains an example using two CPI series from the Bureau of Labor Statistics. Since the lens projection can be parameterized to work on only one dimension, lensing makes it easy to examine patterns that may have a lot of variation within relatively short time frames.

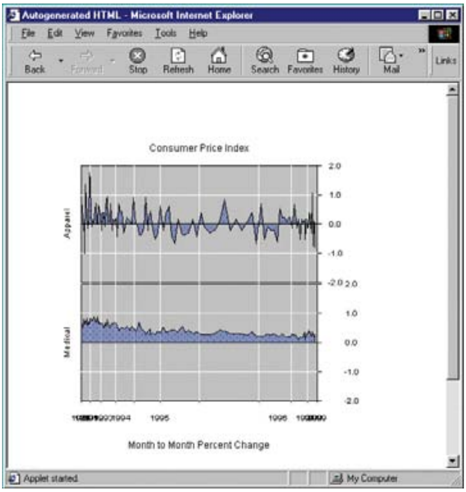


Figure 17.24 *1D Fisheye lens used in a time series*

One of the most famous lensing displays is the hyperbolic tree (Lamping, Rao, and Pirolli, 1995). Technically, however, it is not an optical lens. Instead, the hyperbolic tree transformation embeds the entire complex plane inside the Poincare disk; the edge of the disk is infinitely far from the center. The controller for this display (usually the mouse cursor) drags the tree up/down and left/right on the complex plane, but these draggings result in local translation, rotation, and dilation on the Poincare disk. Straight lines on the plane become curves on the disk. This nonlinear mapping can sometimes make it difficult for users to maintain orientation while exploring.

Figure 17.25 shows a hyperbolic tree from the UC Berkeley Herbarium website (*http://ucjeps.berkeley.edu/TreeofLife/hyperbolic.php*). This tree can be navigated online through its thousands of branches by clicking and dragging with a mouse. The root of the tree, all green plants, is at the middle-right of the display window. We have dragged the Tracheophytes toward the center so that we can see their children to the left.

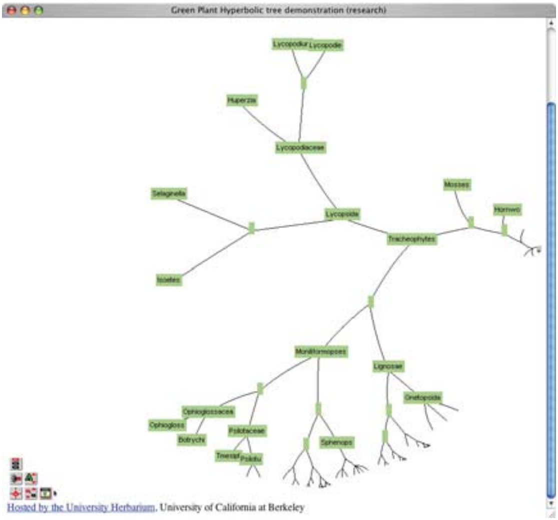


Figure 17.25 *Hyperbolic tree of green plant relationships from the Green Tree of Life website (supported by NSF Grant 0228729, used by permission of the University and Jepson Herbaria, UC Berkeley)*

In Figure 17.26, we have dragged the Moniliformopses node to the right in order to reveal more children of that node. Notice that the shape of the entire tree changes as it is dragged in any direction.

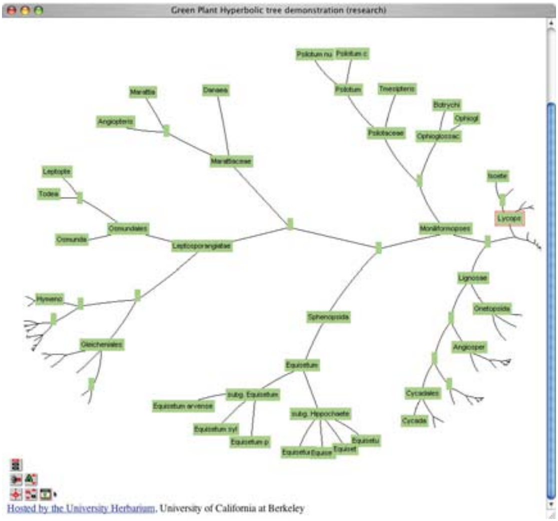


Figure 17.26 *Hyperbolic tree display of plant species dragged to right (courtesy of University and Jepson Herbaria, UC Berkeley)*

17.2.3 Manipulating

Manipulation is direct modification of a completed graphic. At first, this may sound like cheating since it implies changing the graphic to suit one's needs. But, actually, the idea here is to modify the graphic in such a way that the information is more clearly conveyed without altering the quantitative content. Manipulation also allows examination of detail by moving parts of the graphic outside the focal viewing area. This operation is in a sense the dual of lensing. Instead of manipulating our view, we manipulate the shape of the object within our view.

17.2.3.1 Node Dragging

As the name implies, node dragging is specific to graphics containing the Link element. The action here is to use the mouse to drag the nodes of a Link element into a new arrangement. Node dragging can be useful when the Link element is being used to show the results of a graph layout. This is because sometimes graph layout algorithms produce nodes that are so close to each other that they are indistinguishable and, assuming the distance between the nodes is not important, simply moving them apart does not alter the interpretation of the graphic. Figure 17.27 shows an example. Obviously, uses of the Link element where the node position is critical to the interpretation of the graphic, such as connections between cities on a map, or the results of a cluster tree where proximity to other nodes is critical should not allow the user to drag the nodes.

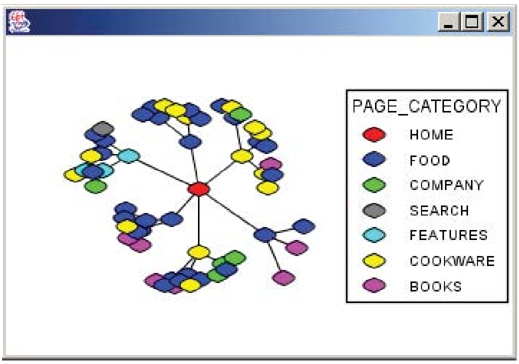


Figure 17.27 *The four nodes on the lower right have been dragged*

17.2.3.2 Categorical Reordering

Another type of manipulation is reordering the categories of a categorical variable. A category ordering controller allows categories to be reordered by directly dragging them on the graphic itself or by using a separate widget. Figure 17.28 shows an unordered categorical plot of market research results.

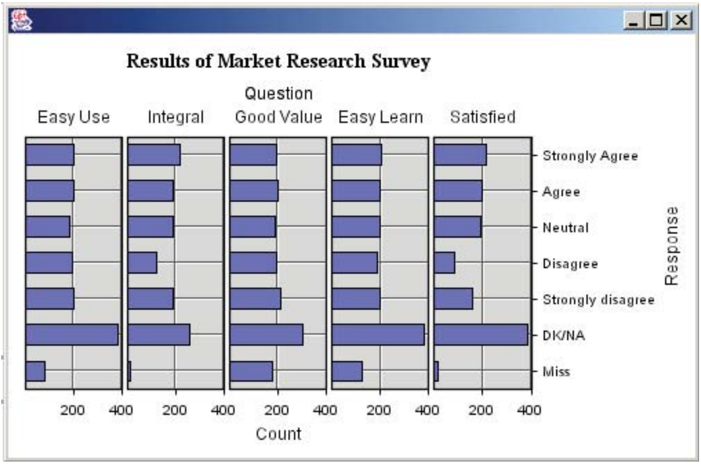


Figure 17.28 *Unordered categorical display*

Figure 17.29 shows a reordering of the display in Figure 17.28 using a category ordering tool that lets the user click on a panel's title to drag-and-drop it to the new desired location.

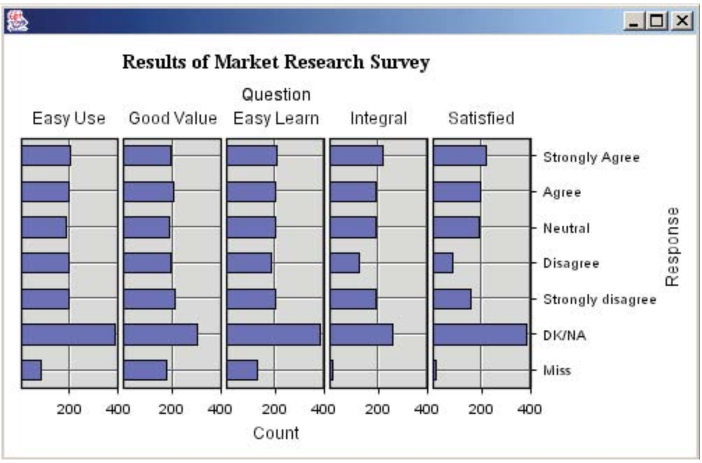


Figure 17.29 *The "Integral" panel was dragged two places to the left*

17.2.4 Brushing and Linking

Brushing highlights objects within a region of a graphic and linking connects highlighting between two or more graphics such that all of the linked graphics highlight the same cases or objects in the data. Figure 17.30 shows an example of brushing and linking between a single scatterplot and a scatterplot matrix.

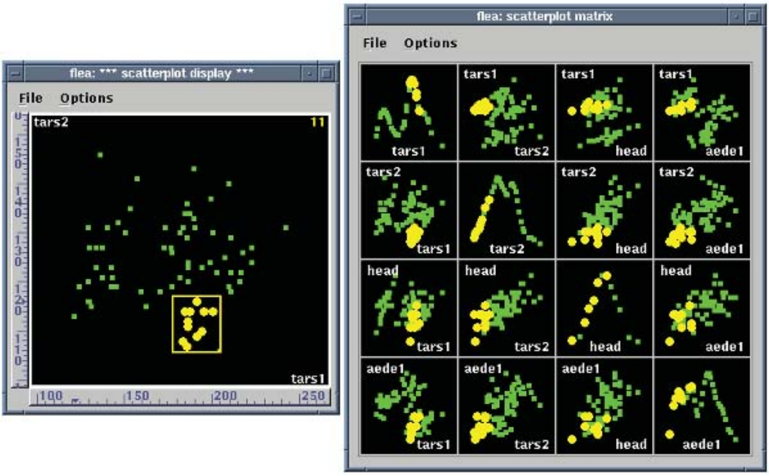


Figure 17.30 *Brushing a scatterplot matrix in GGobi*

The program used in Figure 17.30 is GGobi, a reincarnation of XGobi. The data, consisting of measurements of several body parts of flea beetles, are from Lubischew (1962). The diagonals of the SPLOM contain **averaged shifted histograms** (Scott, 1992). GGobi uses color to indicate the brushing status. Because color tables are easy to manipulate in hardware, color has been traditionally used to indicate whether something is brushed. Other aesthetics can be used as well, however. Motion (using vibrating points) is used in some graphics programs to indicate selection.

Sometimes multiple brushes are used to delineate different selections of data. Since various graphs using different statistics have geometric objects that represent or aggregate the data in different ways, we must ask which geometric object should be brushed for a given graph. Some packages avoid this question by allowing brushing and linking only on scatterplots, but this approach lacks the generality of a grammar-based system. The answer is that a highlighted region should be based on the cases of data that are relevant to the specific part of the geometry that is brushed. For example, a bar should be highlighted if any brushed points in a scatter plot correspond to points lying in the region of the bar. This mapping is invertible. When a bar is brushed, all points in a scatterplot that correspond to points lying in the region of the bar should be highlighted. More granular options are possible, but the utility of this is debatable. One approach would be to brush a portion of the bar that is equal to the proportion of data contained by the bar currently brushed.

Figure 17.31 shows brushing on a bar chart linked to a scatterplot. Brushing the bar allows us to examine the distribution of the cases in the bar on other variables.

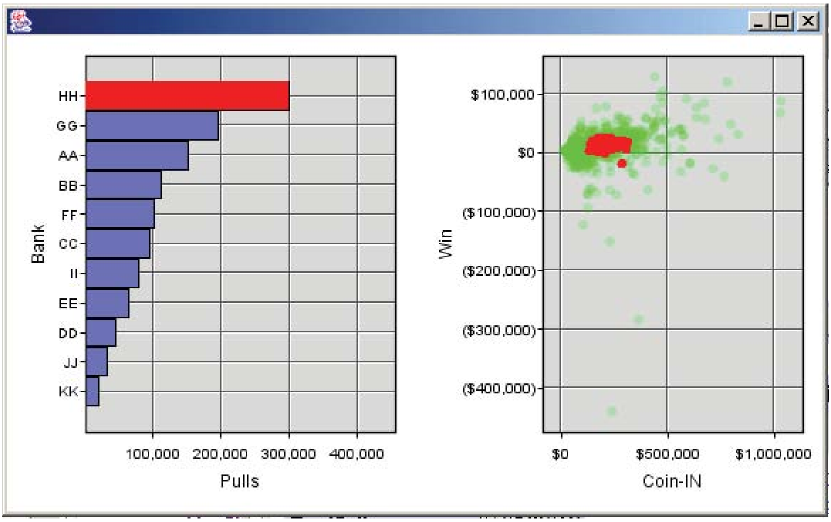


Figure 17.31 *Brushing single bar highlights points contained in bar*

Figure 17.32 shows brushing on the same scatterplot linked to the bar chart. We are able to locate the bar containing the scatterplot outlier.

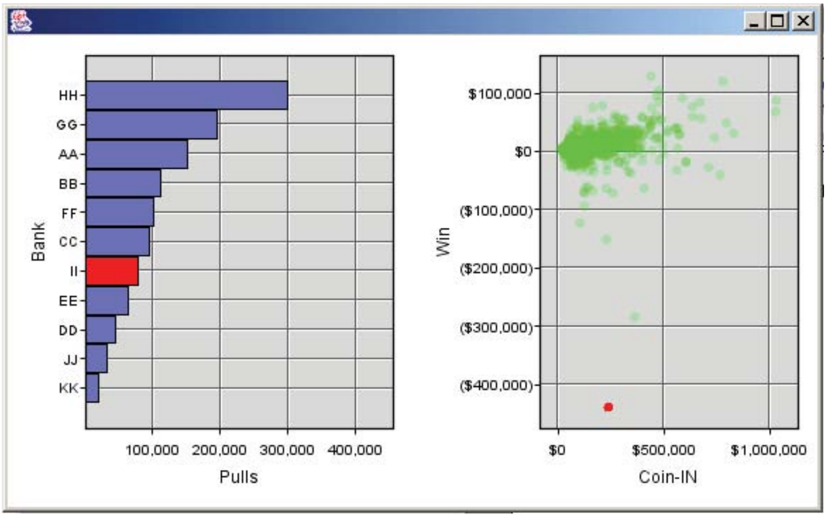


Figure 17.32 *Brushing scatterplot highlights bar(s) containing same case(s)*

Figure 17.33 shows brushing on parallel coordinates, using ExplorN (Carr, Wegman, and Luo, 1997). This brush allow the user to select an interval on any one of the axes; it then highlights all profiles in that interval.

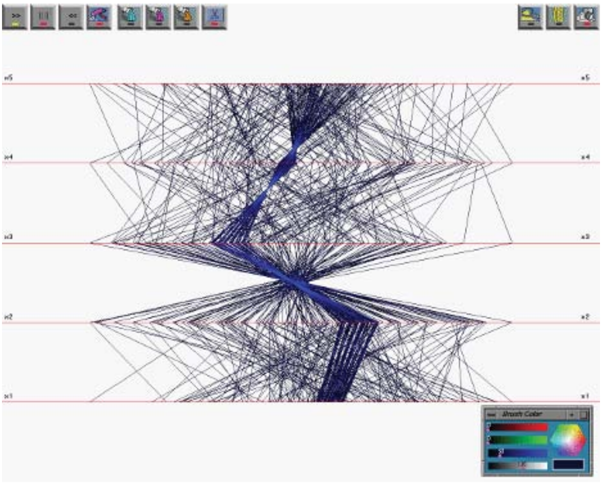


Figure 17.33 *Brushing parallel coordinates (courtesy of Ed Wegman)*

17.2.4.1 Brush Shapes

The term *brushing* was originally used in the PRIM9 system because the metaphor was to attach a brush of a selectable size onto the mouse and then objects become brushed when the mouse intersects them as it is moved around the data space. More generally, a brush is a region that is used to condition actions. A 2D brush is conventionally defined as the rectangular region $[min_x, max_x] \times [min_y, max_y]$. This definition simplifies hit testing, but it may not always serve our needs.

We must remember that a region is transformable under the coordinate system in which it is embedded (see Figure 17.20 for an example of a transformed rectangular zoom region). In fact, any of the isocontours shown in Section 13.1.3 could be used to define a 2D brush. A brush for capturing nearest neighbors in the Euclidean metric would be circular in 2D or would be a ball in higher dimensions. This is the brush we use for a pointer. When the mouse cursor is within a delta neighborhood (Euclidean distance less than delta) we signal a hit. In other words, we use a tiny circular brush for picking objects within a frame. What would a brush for capturing nearest neighbors in a city block metric look like? How would you design a brush for parallel coordinates? (Winkler, 2000; Hauser *et al.*, 2002).

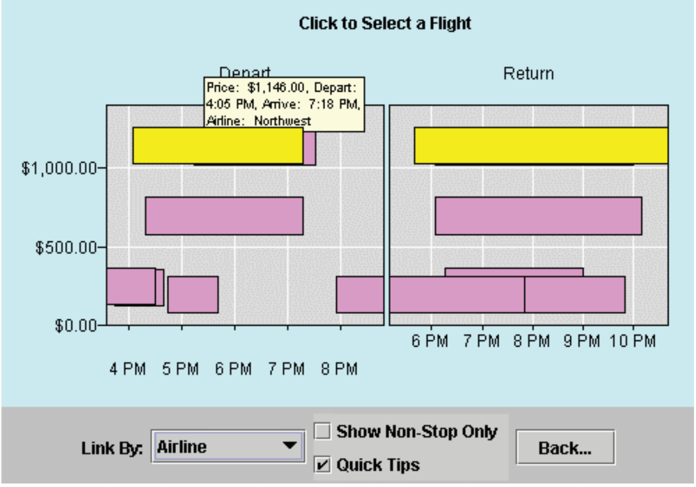
17.2.4.2 Brush Logic

We can add logic to brushes by offering setwise operations. Union, intersection, complement, and other operations give us the ability to combine separate brushes or brush inclusion criteria. These selection modes allow the user to fine tune the intended brush performance. Wilhelm (2003) discusses these issues further. Chen (2003) discusses applications of fuzzy logic to brushing.

17.2.4.3 Fast Brushing

Brushing always needs to happen with sub-second response time; however, some brushing can happen faster than other brushing. For example, when using color for brushing, the highlight color can be drawn on top of the existing plot instead of re-rendering the entire scene or the color table itself can be manipulated in hardware. Other aesthetics used for brushing might not work well this way since the original graph may still show through. For example, if a triangle shape is used to brush something that is a circle, pieces of the circle might still show through.

Traditionally, fast brushing has been used by statisticians in desktop data analysis packages. However, since both the brushing action as well as the results that brushing produces are so intuitive, brushing can be embedded within business applications. Figure 17.34 shows an example for choosing an airline flight. The horizontal axis is time and the vertical axis is price of the ticket. The link colors yellow those flights with a common airline. This facilitates choosing a round trip with the same airline.



Price: $1,146.00, Depart: 4:05 PM, Arrive: 7:18 PM, Airline: Northwest

Link By: Airline

□ Show Non-Stop Only

☑ Quick Tips

Figure 17.34 Using brushing to choose an airline flight

Figure 17.35 shows an example of brushing and linking for a text analysis system. The nodes of the graph in the left panel are defined by a set of news topics. Their coloring is based on a measure of concordance among news sources reporting on those topics. The darkness of the edges (links) between the topic nodes is proportional to the number of sources covering both topics. The user of this software clicks on a particular edge and the concordance information for the news sources is shown in the right panel. Because there can be only one bar graph in the right panel for one edge, this display is intrinsically interactive. It is highly responsive so that the user can check any edge at will.

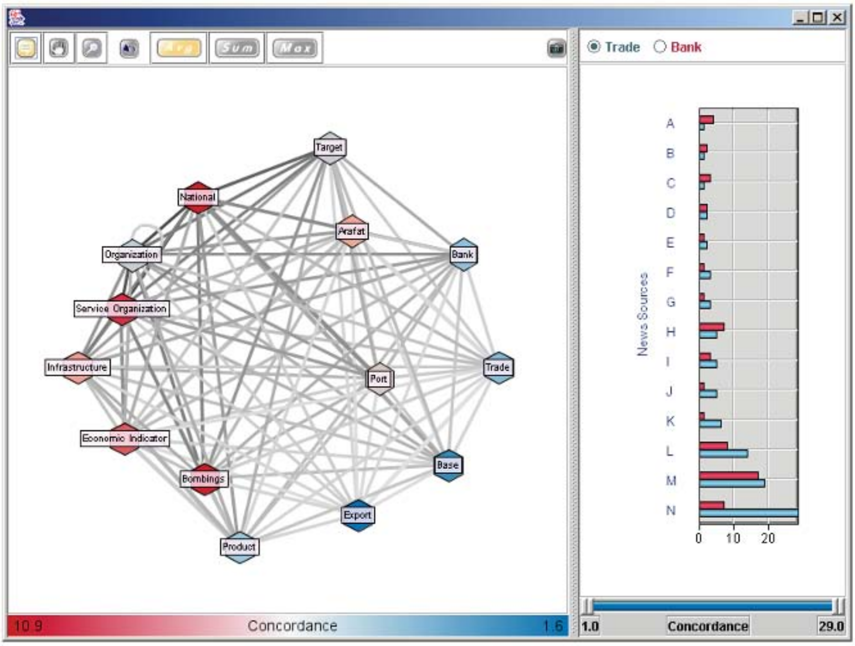


Figure 17.35 *Text analysis brushing application*

Finally, Figure 17.36 shows a treemap linked to a bar graph. The data underlying the graphics are trading statistics for a set of stocks. The user has select trading volume as the variable of interest and the treemap displays the companies on this measure. The color of the tiles is based on percentage change in volume. The bar graph on the left displays the volumes directly. The user can brush either the bar graph or the treemap to link to the other panel. This linkage facilitates exploration of the variables in two different graphical layouts. Other options of this application allow the user to choose different variables and cluster analytic methods to explore the underlying data.

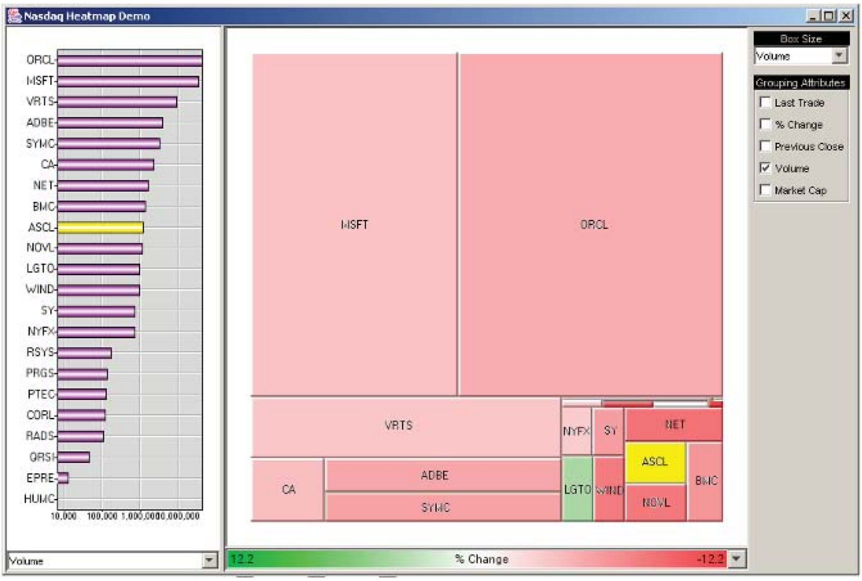


Figure 17.36 Linked treemap

17.2.5 Animating

Animation can add insight to statistical graphics. Typically, systems that provide control over animation are more useful than ones that just let the animation play over and over in a loop. This section provides a brief overview of animation under user control.

17.2.5.1 Frame Animation

Graphics can be animated over variables intrinsic or extrinsic to the graph. In this case, the animation does not necessarily have to be fast enough for smooth-motion video. More importantly, the user should have control over the frames and be able to pause, move forward, or move backward in the animation at will. This can be accomplished with a pause button and a slider that the user can move where each tick on the slider corresponds to a single frame. If the animation is allowed to continue in a loop, there should be some indication of restarts.

Usually animation is implemented by generating a set of static pictures and then playing them in sequence. This is because performance is critical and some applications and datasets may require extensive time to render frames. When an animation is paused, however, it is often useful to be able to interact with the graphic in the usual ways. This means that tooltips, brushing and linking, or other interactive features should be enabled when the animation is stopped. Figure 17.37 shows an example. This animation is paused near the middle of a time series. Popup metadata (for each state) and linking to other graphs of the data are available during this pause. The unusual map projection was designed by Dan Carr to reveal patterns in the smaller states.

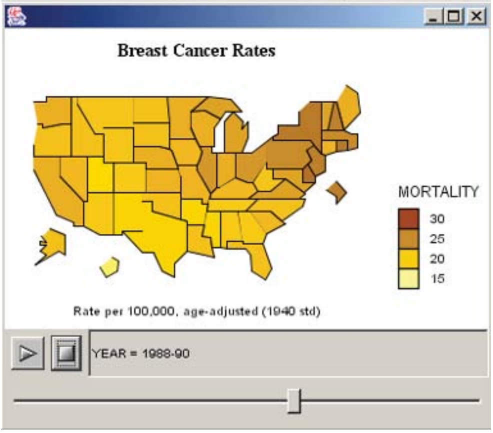


Figure 17.37 *Controlled animation of a map showing breast cancer mortality rates over time*

17.2.6 Rotating

3D graphics often require rotation to get a sense of their structure or to see, as it were, the dark side of the moon. Controlling rotation is not simple, however. Physical joysticks and trackballs can be used to get true three degrees of freedom, but screen devices (virtual sliders, arcballs, and thumbwheels) can offer only two degrees of freedom. Designers of virtual rotation devices derive a third degree of freedom from the velocity of mouse movements. Interestingly, Hinckley *et al.* (1997) found no difference in rotation-to-target accuracy between physical and virtual devices. Physical devices did improve rotation speeds, however. The first, and probably still definitive, study of virtual rotation devices was done in the Apple Computer labs (Chen *et al.*, 1988). For unconstrained rotation, the Apple authors found a virtual trackball to be most effective. For constrained rotation, sliders and other single degree-of-freedom devices worked comparatively well.

We must remember, however, that we seldom wish to use unconstrained rotation on 3D graphics. There has been a mini-controversy over whether exploratory 3D scatterplot rotation is effective compared to other 2D and 3D graphical methods, but most of the evidence supporting arguments on either side is anecdotal or derived from poorly designed experiments. For 3D scatterplot rotation, probably the best controller (on the basis of the Apple results) is a virtual trackball implemented by a mouse cursor operating directly on the cloud of points (as if the points were inside a transparent trackball sphere).

For other charts, such as 3D bars and surfaces, we need only a two degree-of-freedom rotation: up-down or left-right (from the user point of view). Moreover, we probably never want to allow a user to rotate one of these functional graphics below the horizon. That is, in a well-designed system we should not allow the user to get underneath a 3D bar graph. Consequently, our rotation world for most 3D graphics is the upper hemisphere. In this case, two virtual thumb wheels or sliders are sufficient and help prevent users from becoming disoriented by Necker cube and other perspective illusions.

17.2.7 Transforming

In Chapter 7, we gave several reasons for placing statistics under the control of graphs by making them graph methods. This is different enough from the way most statistical graphics packages work that we need to examine it further. The following example illustrates the behavior this design produces.

Most statistical graphics systems perform transformations (when requested) and pass through the data in one step to accumulate basic statistics that are used in creating a chart. In these systems, the scene assembly and rendering components are fed aggregated data to make *bars* and other graphics. The motivation for doing this is to keep the architecture simple by computing statistics before scene assembly so that different graphic elements can share common aggregated statistics. This approach severely limits the capabilities of a graphics system because it makes statistics determine functionality for graphs rather than the other way around. Ironically, limitation is not required to maintain simplicity. In a properly designed graphics system, we can employ data views and proxies to make sure that multiple graphic clients of the same statistic do not force the system to duplicate the computations.

The following scenario illustrates why graphs must control their own statistics. It is an aspect of a dynamic interface that is essential for exploration. Unlike other dynamic scripts, this scenario requires the recalculation of a different statistic for each separate graphical element in a common frame every time a button is pressed or a slider moved. And each recalculation requires a pass through the raw data, because a nonlinear transformation is involved. Even though such interactions are fundamental to real data mining, where MOLAPs have been heavily promoted by some, this scenario cannot be implemented by a MOLAP or data cube model (see Section 3.5.1).

This scenario is due to Sanford Weisberg (personal communication), who choreographed it using Lisp-Stat (Tierney, 1991). The application involves one of the most widely used modeling procedures: linear regression. Weisberg connected a residual plot to a transformation controller in order to examine dynamically the distribution of residuals under different values of a power transformation applied to the dependent variable in a linear regression. The reason for doing this is to be able to identify a proper power transformation by examining the residuals directly. Pressing a button or slider to determine the value of the power parameter (p) helps the investigator understand the behavior of the transformation over a plausible range of values. The alternative would be to run hundreds of analyses and then to examine the residuals for each analysis by plotting them separately. Although this example involves power transformations, it is only one of many similar operations that are required in a graphical system for data modeling and discovery.

17.2.7.1 Specification

We will use the sleep dataset in Allison and Cicchetti (1976) to predict brain weight from the average number of hours of sleep an animal takes per day (no causal relation is implied). The specification is similar to Weisberg's, but we will add a regression plot so that we can see the residuals and the regression at the same time.

```
DATA: zero = constant(0)
TRANS: bp = pow(brainweight, p)
TRANS: resid = residual.linear.student(sleep, bp)
ELEMENT: link(position.edge.join(sleep*resid + sleep*zero))
ELEMENT: line(position.smooth.loess(sleep*resid), color(color.red))
GUIDE: position(dim(2), label("Linear Regression Residual"))

SCALE: position(pow(dim(2), exponent(p)))
ELEMENT: point()
ELEMENT: line(position.smooth.linear(sleep*brainweight), color(color.red))
GUIDE: position(dim(1), label("Daily Hours of Sleep"))
GUIDE: position(dim(2), label("Brainweight"))
```

The first specification produces the upper facet in Figure 17.38 and the second specification produces the lower. The lower plot includes a *point* graphic for the scatterplot cloud and a *line*(*position.smooth.linear*()) graphic for the regression line. The upper plot contains the residuals. The residuals are calculated with the *residual.linear.student*() function, which computes studentized residuals so that we can compare them to a *t* distribution. Instead of *point*(), we have used spikes to zero with a *link*(*position.edge.join*()) in the residual plot so that we can see the distribution of the residuals more clearly. In addition, we have included a *line*(*position.smooth.loess*()) smoother in order to highlight any trend in the residuals across values of the predictor (there is not supposed to be a trend). The transformations will be accomplished through the *pow*() function applied as a scale specification in the lower frame and a data transformation in the upper. Since both functions share the same parameter (*p*), the transformation will be shared by both frames.

The position function for the *line* smoother in the upper plot requires some explanation. It includes a blend of sleep*resid and miss*miss. We used the same device in Figure 7.34 to prevent points from plotting. Here, it prevents the smoother from trying to include the zero values in its computation. Since the blend operation is like stacking columns on top of each other, we use the missing values to force the graphing function to ignore the extra cases.

17.2.7.2 Assembly

The assembly of the scene parallels the structure of the specification. First, each Frame object is constructed. This involves (1) parsing the frame model and executing the algebra, (2) linking variables to appropriate data, (3) associating the variables with the dimensions specified in the model, (4) registering the transforms, and (5) registering the scale for each dimension. Next, the graphs are set into action. Once the Frames are known, the graph computations may be done in any order, or may even be done in parallel if the environment is **multi-threaded** (allowing separate processes to execute independently or simultaneously). The *point* graph, for example, asks its Frame for values on brainweight and sleep. Frame notices that a scale transformation is registered for brainweight and requests DataView to return transformed values. Since the value of the exponent is 1, no transformation is performed. The *point* graph now has data to assemble itself, so it puts together a package of tuples and attribute values (default symbol shape and color) and sends the package off to Display to render a cloud whenever it is ready.

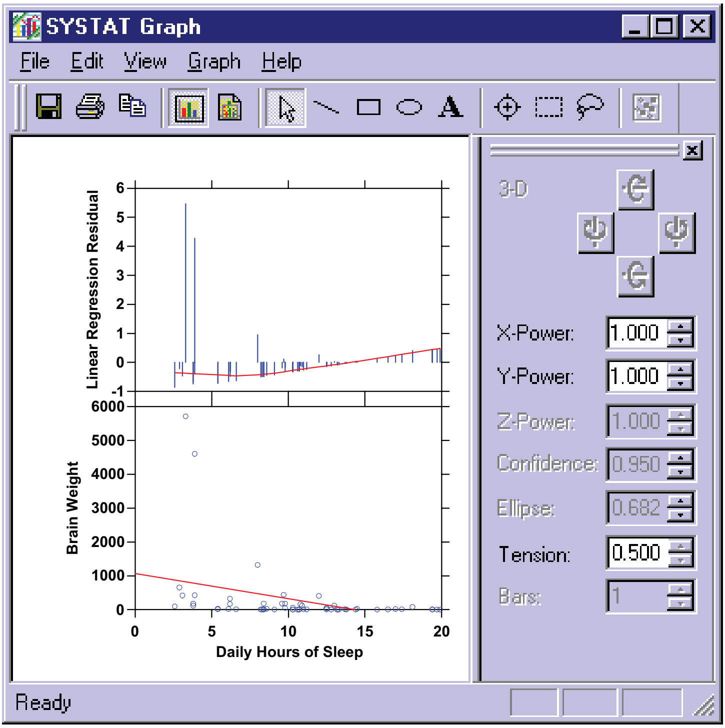
The *line*(*position.smooth.loess*()) graphing function asks its Frame to return values on resid and zero and sleep. Frame has a *residual*() transform registered for resid, so DataView must perform linear regression calculations and return studentized residuals to *line*(*position.smooth.loess*()). Because a *pow*() transformation is registered for bp, which is used in *residual*(), the data values must be transformed before doing the regression (although the value of the exponent is still 1 at this time). As far as the graphs are concerned, they are receiving values for their own calculations. They don't care how those values were assembled. The *line* graphing function will proceed to do *another* regression on the residuals using the LOESS locally parametric regression smoother, and *link* will connect the residuals to a zero level on resid.

A similar process is followed by the other graphs. If more than one graph requests values on the same variable, DataView can **cache** the requests (collect the requests for execution at one time) or **persist** the view (hold the values until clients stop registering requests for them) in order to make things more efficient in a client-server or Web environment. The essential thing to notice about this assembly process is that each graph must do something different with its data. Geometry drives data. However, there is no implication here that every graph must waste time reading data and doing the same work over and over again. Frame is a central clearing-house for graph activity and maintains the knowledge needed to keep graphs from thrashing.

17.2.7.3 Display

Figure 17.38 shows the graphic from this specification in the SYSTAT graphics controller window (Wilkinson, 1998). This program has a display system that implements **widgets**, or graphical toolbar controls, for dynamic graphics. The tool we will use is the *Y-Power* spinner widget, which sets *p* for the scale specification for brainweight. (The other tools are irrelevant to the scenario).

The lower graphic in the window shows a regression line that does not pass through the center of the *y* values for most *x* values. This is because the brainweight data are severely skewed. The residuals plot directly above corroborates the finding; the LOESS smoother reveals a trend in the residuals due to this skewness. If our regression were appropriate, the residuals would be fairly symmetrically distributed about zero across the whole horizontal range.



X-Power: 1.000

Y-Power: 1.000

Z-Power: 1.000

Confidence: 0.950

Ellipse: 0.682

Tension: 0.500

Bars: 1

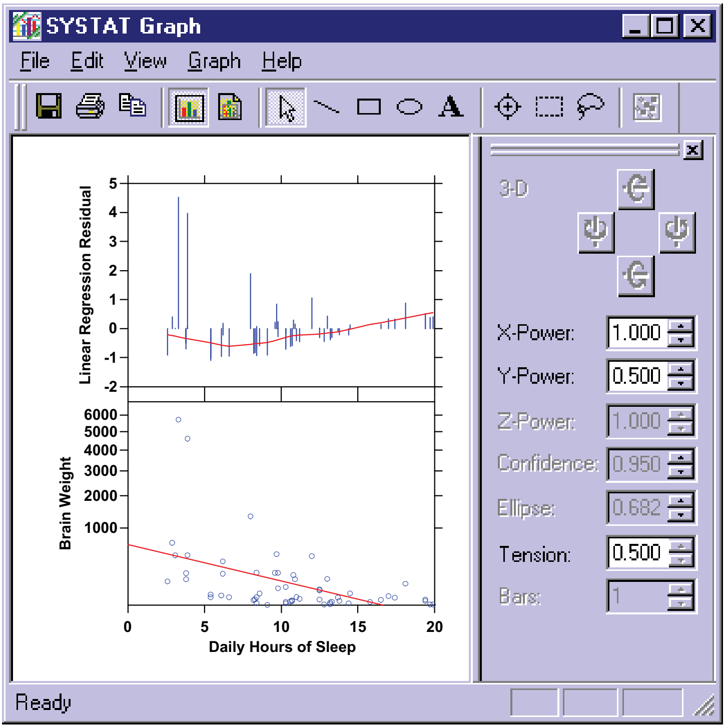
Figure 17.38 Regression and residuals

Tap

Now we are ready to press the *Y-Power* spinner on the controller toolbar. We want to lower the *p* exponent toward zero to see if transforming brainweight with $f : y^p$ will improve our regression fit. Figure 17.39 shows the result after we have tapped the *Y-Power* spinner to change its value to .5 (square-root). The regression now appears better behaved. The residual plot shows less trend in the LOESS smooth and the values are more evenly spread around zero.

How did this happen? First of all, the controller for the *Y-Power* spinner sent out a ScaleChangeEvent message. Each Frame has a listener for this and other messages and orders a redraw because the graphic no longer reflects the state of the specification. All the graphs receive this redraw message and put together a new set of geometry based on the changed situation.

The only difference between the ensuing events and the assembly we described in Section 17.2.7.2 is that the transformation is now applied to **brainweight** because $p = .5$. For example, the *line*(*position.smooth.loess*()) graphing function asks its Frame to return values on **resid** and **sleep**. That Frame has a *residual*() transform registered for **resid**, so DataView will compute a linear regression and return studentized residuals to *line*(*position.smooth.loess*()). Before it can do that, however, Frame has a power transform registered on **brainweight**. So, the **brainweight** values are square-rooted before the regression is computed. At this point, *line* is looking at the residuals from a power-transformed regression. It proceeds to do its own LOESS regression smoothing on these residuals and presents the resulting geometric curve to Displayer.



X-Power: 1.000

Y-Power: 0.500

Z-Power: 1.000

Confidence: 0.950

Ellipse: 0.682

Tension: 0.500

Bars: 1

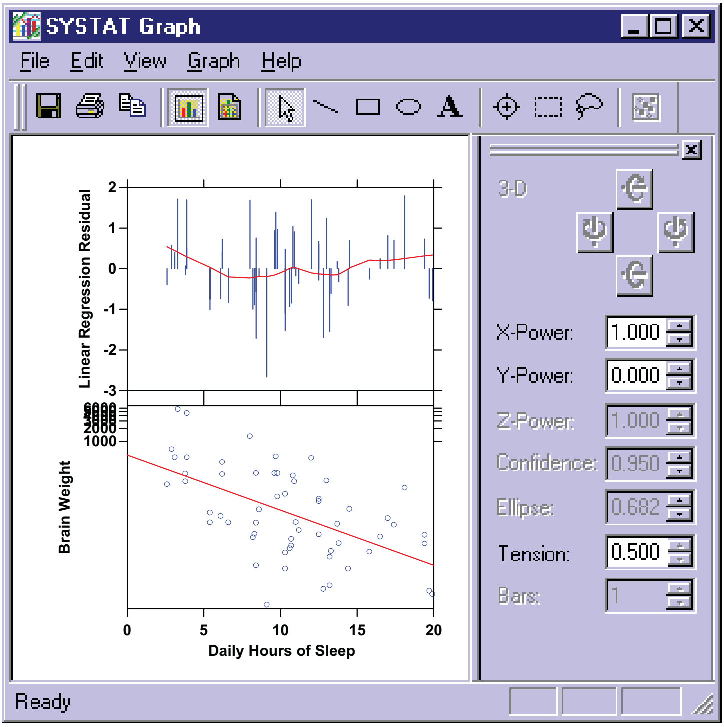
Ready

Figure 17.39 Regression and residuals under square-root transformation

Notice that some elements, such as *axis1*(), get values that are unchanged by the event that originally forced the redraw. They remain unchanged because there is no power transformation hooked up to their data. We can save some of this effort by having Frame keep track of regions on the Canvas where graphs are being drawn, so that unchanged areas can be left untouched. In our experience, this is not worth the extra accounting effort.

Tap Tap

We tap the *Y-power* spinner again and its value drops to zero. The display refreshes with the graphic shown in Figure 17.40. At this point, we have a fairly good model. The LOESS smoother and spikes show some irregularity in the residuals, but not enough to worry us. Incidentally, the scale numbers on the lower graphic are intended to collide at the top of the vertical axis. This signals the extent of the transformation. On the computer display, they move in real time as the button is pressed, so their behavior is smooth and predictable.



X-Power: 1.000

Y-Power: 0.000

Z-Power: 1.000

Confidence: 0.950

Ellipse: 0.682

Tension: 0.500

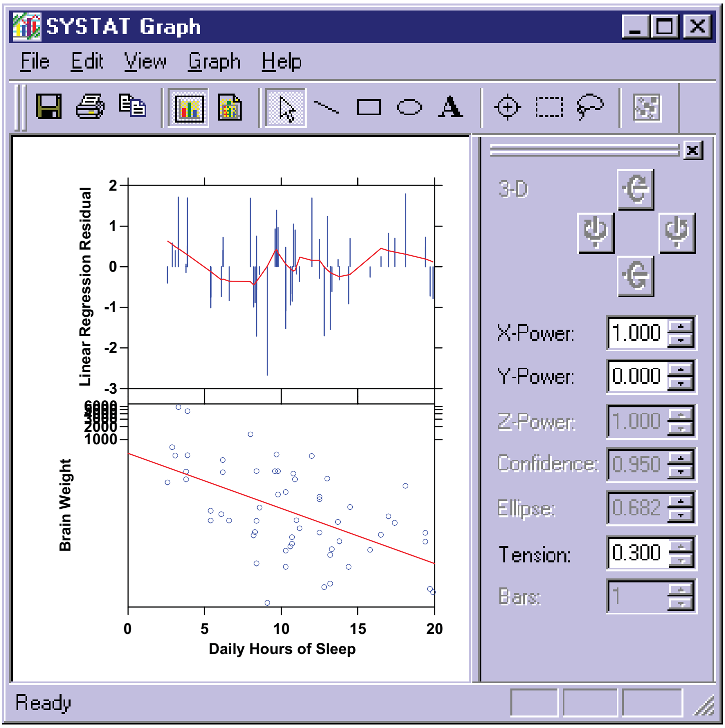
Bars: 1

Ready

Figure 17.40 Regression and residuals under log (pow=0) transformation

Tap Tap Tap

Figure 17.41 shows the result of a tap on the *Tension* spinner near the bottom of the control panel. The controller for this spinner sends out a message to any smoothers or statistical procedures that have a tension parameter. This parameter governs the amount of smoothing desired: large tension values correspond to more smoothing, smaller values to less. Its default value is $t = .5$, halfway between zero and one.



X-Power: 1.000

Y-Power: 0.000

Z-Power: 1.000

Confidence: 0.950

Ellipse: 0.682

Tension: 0.300

Bars: 1

Figure 17.41 *Regression and residuals under log transformation, tension=.3*

When notified of a TensionChangeEvent, each Frame broadcasts a redraw to the graphs. This time, the *line*(*position.smooth.loess*()) graph relaxes a bit because the value of one of its parameters has been ordered changed through its Frame. Notice, again, that none of the other graphs are modified, because they do not have tension parameters to listen for. This type of local behavior under the control of widgets through Frame messaging can be used to produce animations over statistical parameters. We can, in effect, see a movie of the behavior of a graph like *line* when tension changes in small increments over a larger range in real time. To save space, we have illustrated only one tension event over a large step. In SYSTAT, these taps change all the spinners in increments of .1 so that we can see small changes in tension or power instead of sudden jumps.

The point of this scenario is to demonstrate the value of putting statistical calculations under the control of graphs so that we can fine-tune a system and represent behavior that is far more nuanced than what we see in typical data mining displays. What our general design attempts to do is to get away from static graphic entities and instead to treat graphs and their statistical methods like little creatures that respond to different messages and do their calculations in their own peculiar ways. After all, these statistical methods evolved over centuries in the interplay between mathematics and engineering. This evolutionary perspective is not simply poetic license. We believe, following Gould (1996) and Pinker (1997), that evolutionary theory can provide clues to good design: build an ecosystem of many small organisms instead of a single dinosaur. Dinosaurs were adaptive, but nobody beats insects.

Finally, this scenario was intended to illustrate the importance of autonomy and cooperation among graphs in assembling and using statistical methods. Some contemporary graphics and statistics programs can perform part of this scenario already because they are hard-wired to animate, link, drill–down, and brush within certain widely used graphics such as bar charts and scatterplots. The real trick, however, is to replicate this behavior for any graph, on any scale, in any coordinate system, in any ensemble of graphics. A graphics grammar gives us the means.

17.3 Sequel

The next chapter concerns automating graphics. How do we construct jobs that produce a large number of graphics automatically? How do we allow components of a larger production system to access graphical methods and produce their own displays without user intervention?

18

Automation

The word *automation* comes from the Greek $\alpha\dot{\upsilon}\tau\acute{o}\mu\alpha\tau\alpha$ (self-moving machines). When we view graphics, we tend to think of them as created individually, like a painting. The computer tools for creating statistical graphics — spreadsheets, statistics packages, and charting programs — are usually used this way. A few, such as SAS, SPSS, Stata, and SYSTAT, are designed to be able to produce large numbers of graphics in a single run. These packages are often used for automated production graphics. Market research companies, for example, routinely construct reports containing thousands of pages of tables and graphics. Alternatively, scientific and media sites sometimes need to update a single graphic every minute, hour, or day.

Such intensive applications are most effectively implemented with a programming language designed specifically for graphics (as opposed to a general automation facility or macro language). Statistics packages such as SAS and SYSTAT implement a proprietary graphics programming language. This can be effective if one stays within a single programming environment. Linking graphic and analytical applications from different vendors and locations requires a more open standard, however.

This chapter covers two complementary methods for automating production graphics. The first is called Graphics Production Language (GPL). This language originated in a project at the US Bureau of Labor Statistics. The BLS wanted a system for graphics that would mirror its Table Producing Language (Mendelssohn, 1974) so that the Bureau could produce reports containing tables of graphics. Dan Rope at the BLS worked with Dan Carr (1994) to develop an implementation in Java. Leland Wilkinson joined them in 1997 and continued the work jointly under the support of SPSS to reshape GPL into a grammar-based system. Andy Norton developed the GPL parser.

The second method is called Visualization Markup Language (ViZml). ViZml is designed to automate aesthetics. ViZml is also designed as an implementation layer so that complete graphic specifications can be transported from machine to machine or network to network. ViZml was designed by Graham Wills and Roger Dubbs at SPSS to enable thin-client XML-based implementations of the grammar.

18.1 Graphics Production Language

GPL consists of statements. Statements may span several lines and several statements may appear on one line. There is no termination character for a statement. A new statement is denoted by a label followed by a colon. GPL is not case sensitive, except for characters inside quotation marks (").

The form of a statement is

<LABEL>: <name> = <*function*>

- The <LABEL> element and its associated colon are required. It is used to identify a statement. A <LABEL> element consists of one of the following labels: COMMENT, SOURCE, DATA, TRANS, SCALE, COORD, GUIDE, ELEMENT, DO, PAGE, and GRAPH.
- There are four types of statements.

  1) Comment statements (COMMENT).
  2) Data definition statements (SOURCE, DATA, TRANS).
  3) Specification statements (SCALE, COORD, GUIDE, and ELEMENT).
  4) Control statements (DO, PAGE, GRAPH).

- COMMENT statements may appear anywhere.
- SOURCE, DATA, and TRANS statements apply throughout an entire program (they have global scope) and may appear anywhere.
- Specification statements apply within a block defined by a GRAPH statement (they have local scope). If no GRAPH statement exists, there is a single implied GRAPH statement. Specification statements may appear in any order within a block defined by a GRAPH statement.
- The <name> element and its associated equal sign may only be used for DO, SOURCE, DATA and TRANS statements. Names consist of one or more letter characters followed by zero or more letter characters, digits, and/or the underscore character (_). A <variable name> is a name defined by a DO, SOURCE, DATA, or TRANS statement.
- The <*function*> element is required. It consists of a function name followed by an argument enclosed in parentheses.
- An argument consists of

  1) a primitive list, or
  2) a function set.

- A primitive list is a list of zero or more primitives separated by commas.
- A function set is a set of zero or more functions separated by commas.
- A list is an ordered collection of elements (duplicates allowed).
- A set is an unordered collection of unique elements (no duplicates).

- The collection [1, 2, 1] contains a duplicate. The collection [1, 2, 3] does not. The collection [*f*(), *f*()] contains a duplicate. The collection [*f*(), *g*()] does not. For GPL, the collection [*f*(*a*), *f*(*b*)] is considered to contain a duplicate, even though the arguments to the functions are different. In other words, (*position*(x), *color*(y), *position*(z)) is not a proper GPL argument because it contains a duplicate (by GPL definition) and sets may not contain duplicates.
- A primitive consists of

  1) a string enclosed in quotation marks, or
  2) a number, or
  3) a name.

- A string is an ordered list of characters.
- A number is a string representing an integer, real, or complex number.
- A name is a <name>, namely, a <source name>, a <field name>, a <variable name>, or the *unity* value (**1**).
- A graphics algebra expression consists of one or more <variable name>, or <system variable name> elements separated by graphics algebra operators.
- Graphics algebra operators are the characters "*", "/", and "+".
- In GPL, a graphics algebra expression is considered a function. For convenience, the function *op*(x*y) and the expression x*y are equivalent.

The syntax for each of the statement types follows.

Comment Statements

COMMENT: <*comment*>

- The optional COMMENT statement consists of a comment, which may contain any string of characters except one that includes a <LABEL> followed by a colon.

Data Definition Statements

SOURCE: <source name> = <*fn*>(<*args*>)
DATA: <variable name> = <*fn*>(<*args*>)
TRANS: <variable name> = <*fn*>(<*args*>)

- The SOURCE statement specifies a data source. It is optional if the source is specified in a DATA statement.

- The DATA statement assigns a variable name to a column or field in a data source. It is optional if variable names are already defined through context (such as a database access utility). Permissible data functions are listed in Chapter 3. The <variable name> element specifies a single variable, but some of the data functions (*e.g., shape.low*) have multiple columns for the <field name> argument list.
- The optional TRANS statement specifies a variable transformation. Permissible transformations are listed in Chapter 4.

Specification Statements

SCALE: <scale type>(*aesthetic*("<aesthetic type>"), <*args*>)
COORD: <coordinate type>(*aesthetic*("<aesthetic type>"), <*args*>)
GUIDE: <guide type>(*aesthetic*("<aesthetic type>"), <*args*>)
ELEMENT: <element type>(<*args*>)

- The optional SCALE statement specifies a scale for a dimension. The default <aesthetic type> is *aesthetic*("position"), so

  SCALE: <scale type>() and
  SCALE: <scale type>(*aesthetic*("<aesthetic type>")) are equivalent.

  Only one SCALE statement per dimension and one dimension per SCALE statement are allowed. Permissible scale types are discussed in Chapter 6. If a SCALE statement is omitted, the default scale is *interval*().
- The optional COORD statement specifies a coordinate system for graphics. The default <aesthetic type> is *aesthetic*("position"), so

  COORD: <coordinate type>() and
  COORD: <coordinate type>(*aesthetic*("<aesthetic type>")) are equivalent.

  Only one COORD statement per aesthetic is allowed. Coordinate functions are covered in Chapter 9. If a COORD statement is omitted, the default is rectangular coordinates. Coordinate functions may be nested. The expresion *rect*(*dim*(3, 4), *polar*(*dim*(1, 2))) is a rectangular layout of polar plots.
- The optional GUIDE statement specifies axes, legends, and other guides. Only one GUIDE statement per dimension and one dimension per GUIDE statement is allowed. Guide functions are discussed in Chapter 12. If a GUIDE statement is omitted, the default guide is one axis for each position dimension and one legend for each non-position aesthetic dimension. Guides may be suppressed with a *null*() argument.
- The optional ELEMENT statement specifies a geometric element. Permissible geometric elements are discussed in Chapter 7. Graphics algebra expressions are discussed in Chapter 5. If an ELEMENT statement is omitted, only an empty frame is produced.

Control Statements

```
DO: <variable name> = step(from(<number>), to(<number>), by(<number>))
DO: if(<expression>)
DO: end
PAGE: begin
PAGE: end
GRAPH: begin(origin(<measure, measure>), scale(<measure, measure>))
GRAPH: end
```

- The optional DO statement acts as an iterator or conditional executor. Either form requires a closing DO:*end* statement. The <*number*> argument for the iterator form of DO may be an integer or real number.
- The optional PAGE statement groups all graphics within its range to a single display area with a fixed origin. This statement does not set or presume a page size. Graphs are sized and placed with the GRAPH statement. The PAGE statement simply groups graphs together on a single page with a common origin. A measure is a number expressed in length units, *e.g.*, 1cm, 2in, 72pt. Measures increase to the right and up relative to the page origin.
- The optional GRAPH statement groups all elements within its range to a single frame for each aesthetic. It also sizes and locates graphics on a page using origin and scale values expressed in inch or centimeter units. GRAPH statements may not be nested, but multiple graph statements may be nested within a PAGE statement.

18.1.1 Examples

We begin with a few examples that illustrate the syntax. We omit the graph in these examples to save space. After these introductory examples, we show graphs that depend on the use of GPL itself, particularly to do overlays.

First, an example of a data source specification:

```
SOURCE: u = source("url")
DATA: d = col(source(u), name("date"), unit.time())
DATA: r = col(source(u), name("revenue"), unit.currency())
DATA: c = col(source(u), name("company"), unit.category())
DATA: s = col(source(u), name("state"), unit.category())
DATA: p = col(source(u), name("price"), unit.currency())
DATA: e = col(source(u), name("earnings"), unit.currency())
DATA: lon = col(source(u), name("longitude_coordinate"))
DATA: lat = col(source(u), name("latitude_coordinate"))
DATA: longitude, latitude = map(source(s), id(s))
TRANS: pe = ratio(p, e)
```

To create a paneled scatterplot, we need only one statement:

```
 ELEMENT: point(position(d*r*c*s))
```

The following verbose program creates a similar graph with coloring to mark the price–earnings ratios. We have added optional statements to indicate how they can be used.

```
PAGE: begin
 SCALE: time(dim(1))
 SCALE: interval(dim(2), min(0), max(1.0d8), unit(currency.dollar))
 SCALE: cat(dim(3))
 SCALE: cat(dim(4))
 COORD: rect(dim(3,4), rect(dim(1,2)))
 GUIDE: axis(dim(1), label("Date"), format("mm/dd/yy"))
 GUIDE: axis(dim(2), label("Revenue"))
 GUIDE: axis(dim(3), label("Company"))
 GUIDE: axis(dim(4), label("State"))
 GUIDE: legend(aesthetic(color.blue), dim(1))
 ELEMENT: point(position(d*r*c*s), color(pe))
PAGE: end
```

The following program is similar to the preceding, but it uses a different COORD statement to create a one-dimensional paneling of 3D graphs.

```
PAGE: begin
 SCALE: time(dim(1))
 SCALE: interval(dim(2), min(0), max(1.0d8), unit(currency.dollar))
 SCALE: cat(dim(3))
 SCALE: cat(dim(4))
 COORD: rect(dim(4), rect(dim(1,2,3)))
 GUIDE: axis(dim(1), label("Date"), format("mm/dd/yy"))
 GUIDE: axis(dim(2), label("Revenue"))
 GUIDE: axis(dim(3), label("Company"))
 GUIDE: axis(dim(4), label("State"))
 GUIDE: legend(aesthetic(color.blue), dim(1))
 ELEMENT: point(position(d*r*c*s), color(pe))
PAGE: end
```

The following program places two graphs on a page.

```
PAGE: begin
 GRAPH: begin(origin(0, 0), scale(10cm, 5cm))
  ELEMENT: point(position(d*r))
 GRAPH: end
 GRAPH: begin(origin(0, 8cm), scale(5cm, 5cm))
  ELEMENT: interval(position(pe*c))
 GRAPH: end
PAGE: end
```

18.1.1.1 Syntax for Typical Graphs

For review, we now show examples of GPL programs for typical graphs.

scatterplot
ELEMENT: *point*(*position*(d*r))

line chart
ELEMENT: *line*(*position*(d*r))

bar chart
ELEMENT: *interval*(*position*(d*r))

horizontal bar chart
COORD: *rotate*(270)
ELEMENT: *interval*(*position*(d*r))

clustered bar chart
ELEMENT: *interval.dodge*(*position*(d*r), *color*(c))

stacked bar chart
ELEMENT: *interval.stack*(*position*(*summary.proportion*(r)), *color*(c))

stacked bars chart
ELEMENT: *interval.stack*(*position*(*summary.proportion*(d*r)), *color*(c))

pie chart
COORD: *polar.theta*(*dim*(1))
ELEMENT: *interval.stack*(*position*(*summary.proportion*(r)), *color*(c))

paneled pie charts
COORD: *rect*(*dim*(2), *polar.theta*(*dim*(1)))
ELEMENT: *interval.stack*(*position*(*summary.proportion*(d*r)), *color*(c))

map
ELEMENT: *polygon*(*position*(longitude*latitude))

choropleth map
ELEMENT: *polygon*(*position*(longitude*latitude, *split*(s)),
color.hue(*summary.mean*(y)))

scatterplot on top of map
ELEMENT: _polygon_(_position_(longitude*latitude))
ELEMENT: _point_(_position_(longitude*latitude))

tree
ELEMENT: _edge_(_position_(_link.join_(xparent*yparent + xchild*ychild)))

minimum spanning tree
ELEMENT: _edge_(_position_(_link.mst_(xnode*ynode))
contour
ELEMENT: _contour_(_position_(_smooth.quadratic_(_x_*y*z)),
color(_smooth.quadratic_(x*y*z)))

histogram
ELEMENT: _interval_(_position_(_summary.count_(_bin.rect_(y))))

frequency polygon
ELEMENT: _area_(_position_(_summary.count_(_bin.rect_(y))))

kernel density
ELEMENT: _area_(_position_(_smooth.density.kernel_(y)))

3D histogram
COORD: _rect_(_dim_(1,2, 3))
ELEMENT: _interval_(_position_(_summary.count_(_bin.rect_(x*y))))

2D hex-binned scatterplot
ELEMENT: _polygon_(_position_(_bin.hex_(x*y)), _color_(_summary.count_(_bin.hex_(x*y))))

dot histogram
ELEMENT: _point.dodge_(_position_(_bin.dot_(y)))

symmetric dot plot
ELEMENT: _point.dodge.symmetric_(_position_(_bin.dot_(y))

loess smoother
ELEMENT: _point_(_position_(_smooth.loess_(x*y)))

The following sections contain overlaid graphs that depend on the use of GPL

18.1.1.2 Bordering Scatterplots

Overlays can be used to position graphics relative to frames. They are especially useful for representing marginal variation in rectangular plots. Figure 18.1 shows an example for a quantile plot on the same data used in Figure 5.1. We have added dot plots to the top and right borders of the plot so that we can see the skewness of the distribution.

```
TRANS: rmil = prank(military)
GRAPH: begin(origin(0, 0), scale(10cm, 10cm))
  ELEMENT: point(position(military*rmil))
GRAPH: end
GRAPH: begin(origin(0, 10cm), scale(10cm, 5cm))
  ELEMENT: point.dodge(position(military))
GRAPH: end
GRAPH: begin(origin(10cm, 0cm), scale(5cm, 10cm))
  COORD: position(transpose(dim(1, 2)))
  ELEMENT: point.dodge(position(rmil))
GRAPH: end
```

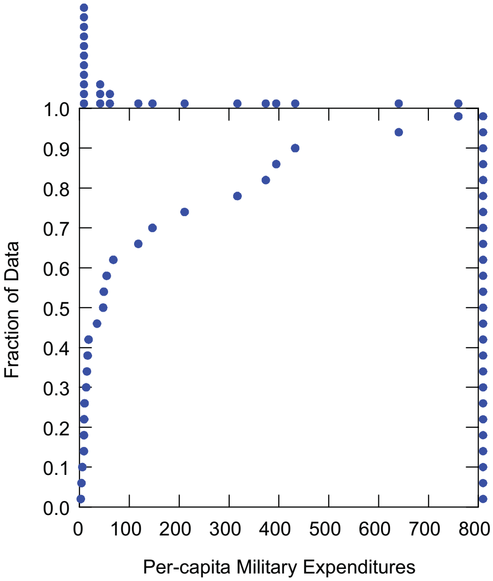


Figure 18.1 *Quantile plot of military expenditures*

Densities are especially suited for bordering. They help reveal skewness in scatterplots and can be helpful for assessing the need for power or log transformations. Bordering in three dimensions requires us to consider facets and viewing angle for displaying graphics appropriately.

18.1.1.3 Bordering Heatmaps with Cluster Trees

Figure 18.2 shows an example of permuting a small matrix whose values are represented by colored tiles. The display is bordered by cluster trees.

```
DATA: unode = col(source("col tree"))
DATA: vnode = col(source("col tree"))
DATA: xnode = col(source("row tree"))
DATA: ynode = col(source("row tree"))
DATA: x = reshape.rect(climate, economic, recreation, transport,
          arts, health, education, crime, housing, "colname")
DATA: y = reshape.rect(climate, economic, recreation, transport, arts,
          health, education, crime, housing, "rowname")
DATA: d = reshape.rect(climate, economic, recreation, transport,
          arts, health, education, crime, housing, "value")
GRAPH: begin(origin(0, 0), scale(4cm, 8cm))
  ELEMENT: polygon(position(bin.rect(x*y)), color.hue(d))
GRAPH: end
GRAPH: begin(origin(0, -2cm), scale(4cm, 2cm))
  COORD: transpose(dim(1, 2), reflect(dim(2)))
  ELEMENT: edge(position(link.join(unode*vnode)))
GRAPH: end
GRAPH: begin(origin(4cm, 0cm), scale(2cm, 8cm))
  ELEMENT: edge(position(link.join(xnode*ynode)))
GRAPH: end
```

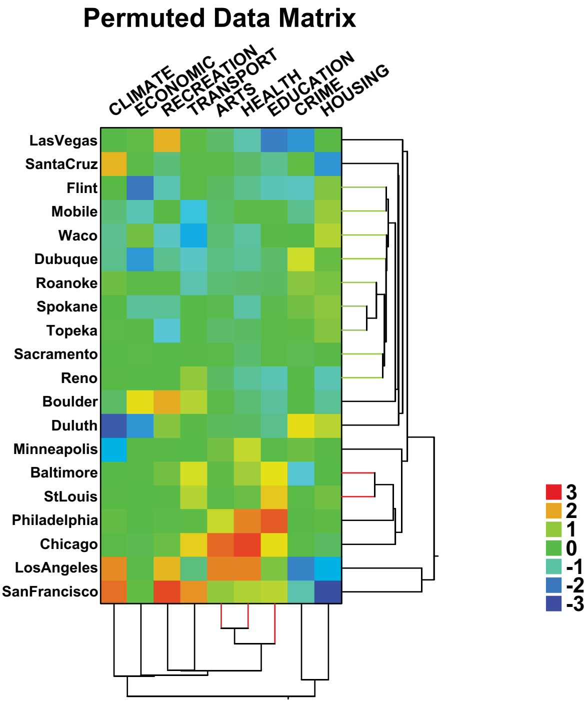
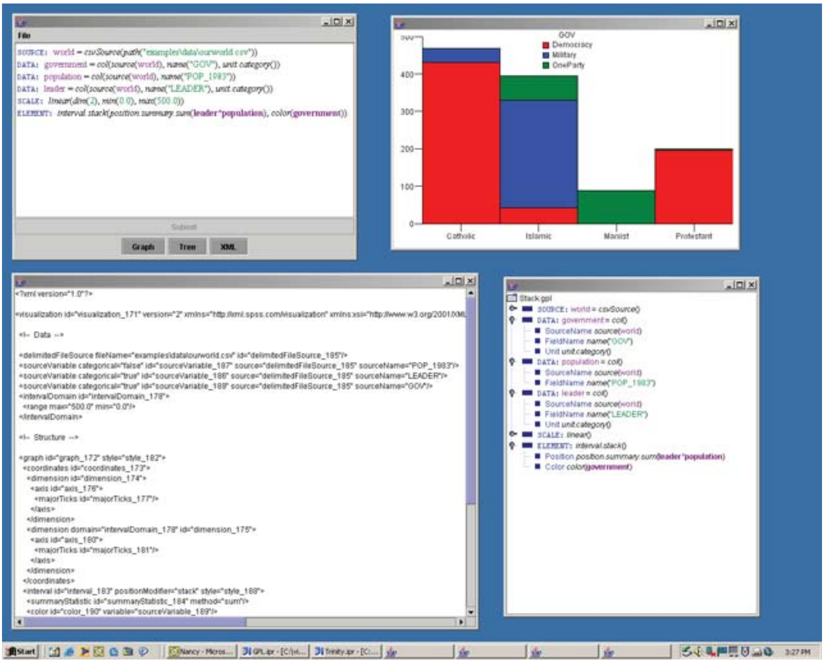


Figure 18.2 *Permuted tiling*

18.1.2 GPL Development Environment

Figure 18.2 shows an example of a development environment for GPL. The upper left window shows a program for generating a stacked bar chart on the countries data. The window below it shows the generated ViZml. The output graphic is in the upper right window. The lower right window shows a tree control for editing and updating the program. A key–value pair structure allows users to modify the graph without knowing GPL.



```
SOURCE: world = csvSource(path("examples\data\ourworld.csv"))
DATA: government = col(source(world), name("GOV"), unit.category())
DATA: population = col(source(world), name("POP_1983"))
DATA: leader = col(source(world), name("LEADER"), unit.category())
SCALE: linear(dim(2), min(0.0), max(500.0))
ELEMENT: interval.stack(position.summary.sum(leader*population), color(government))
```

```
<?xml version="1.0"?>

<visualization id="visualization_171" version="2" xmlns="http://xml.spss.com/visualization" xmlns:xsi="http://www.w3.org/2001/XML

 <!-- Data -->

 <delimitedFileSource fileName="examples\data\ourworld.csv" id="delimitedFileSource_185"/>
 <sourceVariable categorical="false" id="sourceVariable_187" source="delimitedFileSource_185" sourceName="POP_1983"/>
 <sourceVariable categorical="true" id="sourceVariable_186" source="delimitedFileSource_185" sourceName="LEADER"/>
 <sourceVariable categorical="true" id="sourceVariable_189" source="delimitedFileSource_185" sourceName="GOV"/>
 <intervalDomain id="intervalDomain_178">
  <range max="500.0" min="0.0"/>
 </intervalDomain>

 <!-- Structure -->

 <graph id="graph_172" style="style_182">
  <coordinates id="coordinates_173">
   <dimension id="dimension_174">
    <axis id="axis_176">
     <majorTicks id="majorTicks_177"/>
    </axis>
   </dimension>
   <dimension domain="intervalDomain_178" id="dimension_175">
    <axis id="axis_180">
     <majorTicks id="majorTicks_181"/>
    </axis>
   </dimension>
  </coordinates>
  <interval id="interval_183" positionModifier="stack" style="style_188">
   <summaryStatistic id="summaryStatistic_184" method="sum"/>
   <color id="color_190" variable="sourceVariable_189"/>
```

```
Stack.gpl
  SOURCE: world = csvSource()
  DATA: government = col()
     SourceName source(world)
     FieldName name("GOV")
     Unit unit.category()
  DATA: population = col()
     SourceName source(world)
     FieldName name("POP_1983")
  DATA: leader = col()
     SourceName source(world)
     FieldName name("LEADER")
     Unit unit.category()
  SCALE: linear()
  ELEMENT: interval.stack()
     Position position.summary.sum(leader*population)
     Color color(government)
```

Figure 18.3 *Editing a GPL program*

18.2 Visualization Markup Language

ViZml is an **extensible markup language** (XML) for specifying graphics using the grammar-based concepts. XML is not a language; it is a specification mechanism based on a particular data structure — a tree. Because it is based on a tree, XML is suited for graphics grammar specifications (see Figure 1.2). XML allows us to automate the production of graphics with a simple structure that provides an arbitrarily level of detail.

There are many reasons for adopting XML as a graphics specification medium. First, XML parsers offer syntax checking and editing tools. Second, there are numerous GUI tools available for working with XML; one need not type XML itself. Third, many open-source and commercial systems include XML utilities or are based on XML for their Web services. Fourth, XML allows us to remove stylistic details from a language like GPL. XML offers a hierarchical key-value pair mechanism that lets us specify a huge number of stylistic details in a parsimonious manner. A designer can create style templates in ViZml, for example, for a GPL programmer to employ in a production system. Fifth, by writing a GPL parser that generates ViZml, we enhance portability, improve debugging, and open the system to future modification and extension.

ViZml is an XML **schema** that takes the basic grammar concepts and places them within an XML framework. Core grammar concepts are mapped to high-level nodes within the schema, and a hierarchy is defined so that only charts that are grammatically correct can be specified. With this schema, XML-aware editors can be used directly to construct specifications, and programs can read specifications that use the schema and automatically have default values provided. Using an XML schema, we can be confident, for example, that no one can set the color of gridlines on a nonexistent axis.

Figure 18.4 shows a graphic based on the same model underlying Figure 6.5. We have added the kind of stylistic features demanded in many business applications — drop shadows, color flows, text formatting, and 3D effects. The bars have been colored by intraday volatility (difference between high and low price). This coloring, of course, is redundant with the length of the bars. Some may object to such redundancy, but it generally does no harm to, and often enhances, perceptual decoding.

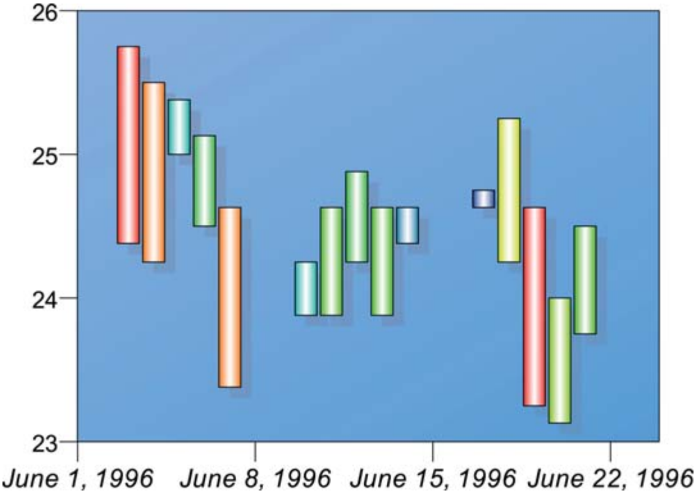


***Figure 18.4** Bar chart of SPSS stock price*

The XML that generated Figure 18.4 follows. We have colorized the three main sections of the specification. The Data components are in green. The Structure components are in blue. And the Style components are in red.

```
<?xml version="1.0"?>
<graph>
   <source id="data">
      <fileAccess fileName="stockValues.csv" separator=","
header="true"/>
   </source>
   <variable id="date" sourceName="date" source="data" cate-
gorical="false">
      <meta>
         <dateTimeFormat>
            <dateFormat>
               <dayOfMonth/>
               <monthName long="true"/>
               <year showCentury="true"/>
            </dateFormat>
         </dateTimeFormat>
      </meta>
   </variable>
   <variable id="high" sourceName="high" source="data" cate-
gorical="false"/>
   <variable id="low" sourceName="low" source="data" cate-
gorical="false"/>
   <variable id="close" sourceName="close" source="data"
            categorical="false"/>
   <variable id="range" categorical="false" expres-
sion="high+low"
            source="data"/>

   <domain categorical="false" id="dateDomain">
      <interval min="1996-06-02" max="1996-06-23"/>
   </domain>

   <frame id="outer" style="cell">
      <dimension role="x" domain="dateDomain" lowerMar-
gin="4.4%"
         upperMargin="4%" upperMapping="exact"
         lowerMapping="exact">
         <axis id="xaxis" baselineStyle="hidden">
            <majorTicks markStyle="ms" delta="7"
               tickLabelStyle="ital"/>
         </axis>
      </dimension>
      <dimension role="y">
         <axis id="yaxis" baselineStyle="hidden">
            <majorTicks markStyle="ms" delta="1"
               tickLabelStyle="fonts"/>
         </axis>
      </dimension>

      <element type="interval" id="bars" style="element-
Style">
         <x variable="date"/>
         <y variable="range">
            <statistic method="range"/>
         </y>
         <color variable="range" low="navy" high="red">
            <statistic method="range"/>
         </color>
      </element>
   </frame>

   <style id="fonts" value="font-family:Arial;font-
size:8pt">
      <style id="ital" value="font-style:italic"/>
   </style>
   <style id="hidden" value="visibility:hidden"/>
   <style id="f" value="fill:transparent; stroke:transpar-
ent;margin:20px"/>
   <style id="cell" value="stroke:black;fill:#4af;stroke-
width:0.5px;
         gradient:#8bf; gradient-angle:60"/>
   <style id="ms" value="stroke:black;stroke-width:0.5px"/>
   <style id="elementStyle" value="stroke-
width:0.5px;width:0.27cm;fill:
         navy; gradient:white; gradient-focus:-0.5;shad-
ow:gray;
         shadow-dx:5;shadow-dy:5;shadow-opacity:0.4">
      <style id="element2Style" value="stroke:navy; stroke-
width:2px"/>
   </style>
</graph>
```

18.2.1 Data Definition

Nodes in this section define grammar entities from Chapter 3 and Chapter 4 (data and variables). They are colored green in the above listing.

The **`source`** node defines where the actual data values can be found. This may be embedded in the XML as row and value nodes, making up a simple table, or can refer to an external source such as a database, a native object or, as in this case, a file. The columns of data in the source, which will correspond to variables, are not typed by the source, so the same column may be used as a category and as a date or scalar value. There may be multiple sources of data for a chart.

The **`variable`** node defines how the source data will be used in the chart. Its necessary attributes are `id` of the variable, which is effectively its name, the `sourceName,` and `source.` The latter two attributes state which column of which source to use. The only access to the sources is via the variable, so modifying ViZml to use different sources or change the target data does not require any changes outside this section. This strong separation between data and the use of data in the chart definition is an important concept; it not only makes it easy to change the underlying data, but it also allows data to be used in different ways without needing to go to the source to redefine the data.

The final required attribute is the `categorical` flag. This defines whether the variable defines categories or numbers. From the viewpoint of the grammar, this is the single most important piece of metadata that the system needs to know. It indicates how a variable is used as the second operand in a blend, determines the type of axis and controls which aesthetics may use the variable. Therefore ViZml requires this flag be defined for each variable.

The variable node may contain other nodes that define metadata, sort order, display characteristics and formatting, whether it is to be used as a weight variable, categories to combine or drop, and formulas to apply to create derived variables.

Finally, the **`domain`** node defines the domain of a dimension. Domains are used for several purposes, including the following.

Comparability: A common use of charts is to look at the same data across time or across another variable. In such a case we want the charts to look as similar as possible. By defining a same domain for the axes, we ensure that the charts using the same ViZml definitions but different data do not change the axes' ranges. This is not just a case of ensuring that a scalar axis runs from, say, 0 to 100 for each chart, but also ensures that even if a category is missing from a given set of data, a bar chart will allocate space on the axis for that category.

Filtering: If we are uninterested in certain categories for a chart, or want to concentrate on certain ranges, we may define a domain to include only those categories or ranges we are interested in. The "out-of-domain" values are omitted.

Including important values in a plot: The most common use here is to ensure that zero is included in bar charts or other charts which measure additive quantities.

Exact aesthetic mapping: If we are using a variable to map to size, for example, we may want to ensure that zero maps to zero, even if no value in the data actually is zero. A domain including zero solves this problem.

18.2.2 Structure

This section, colored blue in the listing, defines the essence of the chart. It is where the algebra, geometry, aesthetics, statistics, scales, coordinates, facets and guides are all defined. If this section is changed, the resulting chart will be fundamentally different. If this section is not changed, all that can change are style details and underlying data.

The outer node of this section is a **`frame`** node. Each frame node may contain some text (to form a title, annotation, footnote or other label), a legend (for an aesthetic), sub-frames, or, as in this case, nodes defining a graph. The frames may be given a location in absolute or percentage coordinates, allowing some simple layouts to be created. In this example, there is a single frame containing the chart.

To define the chart, we first define a coordinate system consisting of one **dimension** or a set of such nodes. These define the inner coordinate system (that is, the coordinate system within each panel of a faceted chart, if faceting is defined). For this example, we simply define two dimensions, with a *role* of "x" and a *role* of "y". We could have chosen a **coordinateTransform** node with a *transpose* method if we wished to transpose the plot. Other coordinate transforms include *polar* and *fisheye* transforms and some 3D projection transforms.

One other option we did not define in this plot is a **scale** for each dimension. If missing, the scale defaults to a linear scale, and since no domain is defined, the domain is chosen so that all the points defined by the element node below will be visible.

Next is the **element** node. We have a very simple element; an *interval* element. It is an error to define an element dimension that does not have a corresponding frame dimension. It is not an error to leave one out. The rule is that an element must be able to be embedded within the frame. If a dimension is missing the element can be drawn assuming a constant value for that dimension, typically either in the middle of the dimension's domain (for points, lines, and similar) or filling the entire domain (for intervals).

We have added a *color* aesthetic to this element with the node

```
<color variable="range" low="navy" high="red">
   <statistic method="range"/>
</color>
```

Notice we used the same statistical method for *color* as we did for *position*. We can add statistics in two ways — either individually on a dimension (or an aesthetic) or we can replace all the element dimension information by a position statistic node, which allows complex, multi-dimensional statistics to be used. Thus on a scatterplot we could have a line element with position defined by:

```
<x variable="xv"/>
<y variable="yv">
   <statistic method="median"/>
</y>
```

This will draw a line connecting the median values of `yv` for each group defined by having the same value of `xv`. Under the second formulation, we could have

```
<position>
   <statistic method="smooth"/>
   <x variable="xv"/>
   <y variable="yv"/>
</position>
```

This will create a set of `x` and `y` values that define a smooth fit to the data. For the default case shown, the default smooth is a simple linear regression, so the actual data generated by the smooth will simply be the endpoints of a line.

In this example, we have not defined any faceting. To do so we would add a **`tableFacet`** or **`coordinateFacet`** node as a peer of the dimension definitions. Each graph must have the same faceting for all the elements of the chart, and this mechanism enforces that. It is legal to have different variables being used to display the points, lines or other elements *within* each cell, but they must all share the same layout of facets.

18.2.3 Style

The final section defines styles that are used to make cosmetic appearance modifications to the chart. In the structure section, any node that defines a visible entity may have a `style` attribute. This reference is to a **`style`** node in the final section of the XML. The decision was made to allow style modifications only by reference to style in this section. Other XML systems using styles allow inline embedding of styles, but ViZml requires styles to be outside the structure section. This is slightly more cumbersome if only minor changes are required, but it makes it much easier to replace styles and create, in effect, style sheets that can be used with multiple charts.

The implementation of styles in ViZml is based on the Cascading Style Sheet (CSS) standard that is extensively used to define styles for web pages. Each style node contains an `id` naming it and a string `value` that defines the style. As in CSS, the style string is a semicolon separated list of *key*:*value* pairs, each of which changes the default appearance. We have supplemented the CSS definitions with additional ones that are useful in our specific domain. Where possible we have based our extensions on similar CSS attributes. We also used Scalable Vector Graphics (SVG) and Vector Markup Language (VML) as a reference and guide to defining attributes, with the intent that people familiar with these markup languages will be able to read and write ViZml styles more easily.

In Figure 18.4, the bars have been given a style that modifies their appearance. Child nodes inherit all the properties of their parent and so hierarchies of styles may be defined.

The set of styles attributes that can be set is given in Table 18.1. In several cases the definitions are quite complex, and the reader is referred to the CSS definition for the attribute. Attributes of type `color` conform to either the CSS '#RRGGBB' syntax or may be a name from the SVG list of color names (which expands the smaller CSS list of names). Length attributes consist of a number and a unit. Units may be inches, centimeters, millimeters, pixels, points or percentages. Fractional amounts are legal.

Table 18.1 **Style Attributes**

Attribute	Region	Target
fill	Color	The shape is filled with this color
fill-opacity	[0,1]	Sets the transparency of the fill color
stroke	Color	Color to outline shape (a.k.a. "border")
stroke-opacity	[0,1]	Sets the transparency of the stroke color
stroke-width	Length	The width of lines
stroke-linecap	see CSS	Sets the line end and join caps
stroke-dasharray	see CSS	Sets line dash style
gradient	Color	Sets the second color to use with fill as a gradient
gradient-angle	[0,360)	The gradient angle
gradient-focus	[-1,1]	Where the fill color occurs along the gradient line. Negative inverts the gradient
shadow	Color	the shape is drawn with this color shadow
shadow-opacity	[0,1]	Sets the transparency of the shadow
shadow-dx	Length	Shadow horizontal offset
shadow-dy	Length	Shadow vertical offset
border-bevel	Length	Size of bevel effect for a shape
fill-pattern	Integer	Index of pattern to use to fill object
visibility	hidden | visible	Whether to show the object
width	Length	Width of an object
depth	Length	Depth of an object
background-image	URL	An image to fill in the background
background-repeat	see CSS	How to repeat the image (tile, etc)
glyph-type	many options	The name of a glyph to use
glyph-sides	Integer	Defines number of sides or points for polygon, star and flower glyphs
glyph-size	Length	Size of the glyph
glyph-angle	[0,360)	Angle to rotate glyph
glyph-aspect	Number	Aspect ratio of the glyph
font-family	See CSS	S list of fonts in preference order
font-size	Length	Size of font to use
font-weight	see CSS	bold, normal, etc.

Table 18.1 **Style Attributes (Continued)**

Attribute	Region	Target
`font-style`	`see CSS`	`italic, normal, etc.`
`label-location-x`	`positive | negative | center`	`How to position a label relative to a shape`
`label-location-y`	`positive | negative | center`	`How to position a label relative to a shape`
`label-location-inside`	`Boolean`	`If true, the label is inside. If not, it is outside.`
`label-location-theta`	`[0,360)`	`Angle of label text.`

18.2.4 Levels of Specification

If someone were to describe the composition of Figure 18.4, the description might read something like the following.

> The figure shows a plot with vertical bars placed along the horizontal axis at data locations. The data shown are dates. The horizontal axis has tick marks at weekly intervals, starting on June 1, 1996 and continuing to June 22, 1996. There is no title for the plot or for the axes (haven't you read Tufte?).

If we pressed for more detail, we might also get the following.

> The dates are formatted as the US English comma-delimited form of the month, day, and year. The tick labels for the dates are drawn horizontally. The plot is drawn against a blue background. It looks like the font used is Helvetica or some other san-serif font. All the lines look about normal thickness.

In both natural language and in the general grammar, there is an importance ranking for chart features. Although a typesetter might think of fonts as preëminent, the axes are a parent class of the labels on the scales. One of the core principles of ViZml is that the XML description of a chart should reflect this type of ranking. There are three levels of importance within ViZml:

Named Nodes: Core elements of the specification are represented in the XML as nodes with a unique name (or "tag" in XML nomenclature). They are named to parallel the grammar. Many of them have a required `id` attribute that allows easy programmatic manipulation of the XML. Some of these nodes may also have attributes that must be specified and have no default value. These required attributes are at the same level of importance and will radically affect chart appearance.

Optional Child Elements and Attributes: These modify a parent node's appearance. The rule for ViZml is that an optional attribute or child should not radically modify a chart. Typically an optional child element will add something to the graphic representation of the parent node, whereas an optional attribute will typically modify a default value.

Styles: All XML nodes that have a visible representation have a default style. That default style can be modified by referring to a style that overrides some of the attributes of the style (attributes are given in Table 18.1). While some of them might dramatically affect the visual appearance of a chart, they do not affect the basic representation or the mapping from the data.

18.2.5 Default Settings

One of the more difficult decisions underlying ViZml was the conclusion that a missing node implies that the corresponding graphic representation is to be absent. Because it is very common for axes to have tick marks, it would seem to make more sense to have them visible by default and to add an option to hide them. This was in fact the initial standard, but the decision was changed for two reasons. The first reason is consistency. It puts a burden on the user (and on software processing XML) to differentiate between items that default to hidden and ones that default to visible. For example, what should be the default title for a blended dimension? Supporting this consistency is the behavior of other markup languages, such as HTML, where the default behavior is only to show items if explicitly defined. The second reason is more pragmatic. In an interactive system, it is usually important for a graphic system to be able to report information that the user has clicked on (to perform drill-down, editing or other interactions). It is much easier and makes more sense to be able to report the node that the user clicked on (*e.g.* the user clicked on the majorTicks node) rather than report that the axis node was clicked on and pass extra information to indicate it was the ticks that were clicked on. As a general principle, then, ViZml assumes that if an element could be defined, but is not, it will not be shown. This is extended to attributes as far as possible, so that the default value for all Boolean attributes is *false*.

18.2.6 Locales

We handle locales in as parsimonious manner as possible. That is to say, we specify locale for the entire graph and let the local system determine appropriate formatting. In most formatting systems, a pattern is given for dates, such as “mm/dd/yyyy” which is taken to be a two-digit month, followed by a forward slash, followed by a two-digit day of month, followed by a forward slash followed by a four-digit year. Although adequate within some locales, this approach fails when used in other locales. In ViZml, we require that a chart created in one locale be as intelligible as possible in another locale. Therefore we do not use the pattern model in ViZml, but instead specify those date elements that are wanted in the tick labels. It is possible to define a format that is nonsensical using this scheme, as it is with a pattern-based scheme, but that is the price of flexibility. Following our general principle we only display those date segments that occur in the XML, so if a user specifies only week and year, “Friday, 1999” will be the result.

Figure 18.5 shows an example. We have made a minor modification to set the locale of the entire graph to Chinese:

```
<graph lang="zh">
   ...
</graph>
    .
```

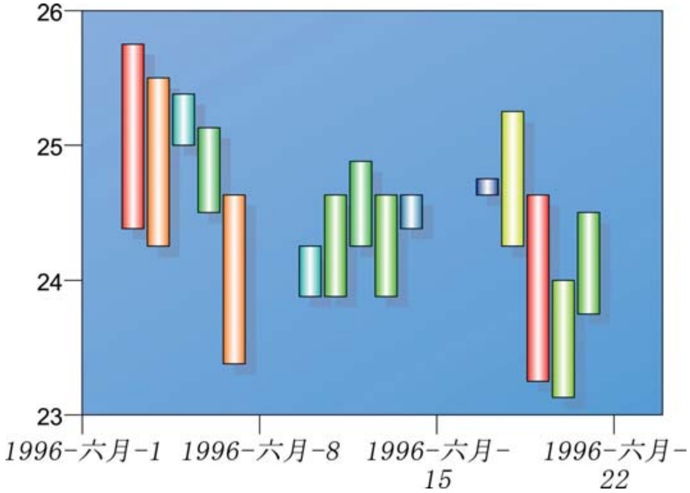


Figure 18.5 Bar chart of SPSS stock price in Chinese

Figure 18.6 shows the same graph in Latvian.

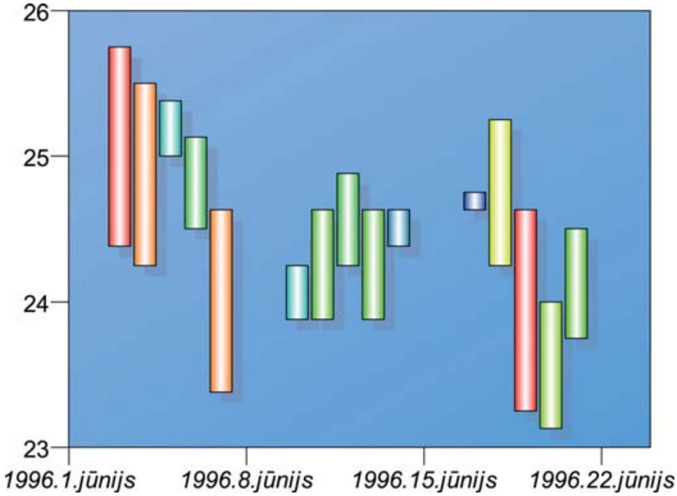


Figure 18.6 *Bar chart of SPSS stock price in Latvian*

18.2.7 Extensibility

In graphics grammar, we may construct many combinations of graphic elements, coordinate systems, aesthetics, statistics, styles, and so on. ViZml allows for these combinations, but the intent is not simply to be able to specify all the currently imaginable charts; it is also to be extensible to other charts that use pieces yet to be invented. The ‘X’ in XML stands for extensible, and ViZml employs the following techniques for extensibility.

18.2.7.1 Schema Support

ViZml makes use of a feature of XML schemas that allows nodes to be defined as extensible. A user of ViZml who wants to be able to add a new attribute to an element simply has to write a small schema that includes the ViZml schema and defines a new node based on the element node. That new node can then be used wherever the original one was used.

In a similar way, if a new aesthetic is desired, that aesthetic can be defined in the new schema and added to the new node derived from element. In some cases, there are a number of definitions all of which share similar specifications (aesthetics require a *variable* and an *id* and may have a *domain*). ViZml has been defined with a base node that encompasses this required behavior, thus making it simple to create a new aesthetic.

In summary, schemas allow a fairly simple object-oriented design approach, which ViZml takes advantage of. A user who wants to extend ViZml need only design a new schema that includes the ViZml schema, and adapt it as desired. In fact, the schema is also open to restriction, so that an application that cannot cope with certain aspects of the grammar may explicitly forbid them. However, in most cases, it would be preferable simply to ignore XML that cannot be processed, perhaps logging a warning message. Thus a very simple engine that only knew how to create bar charts, scatterplots and histograms has two options:

1) Define an XML that includes ViZml and restricts the elements so that only those charts may be defined.
2) When processing input XML, ignore all irrelevant XML.

The latter option is easier to implement, but has the disadvantage that validation cannot be used prior to processing an XML specification. Fortunately, the approaches are consistent with each other, so the second approach can be taken initially and the first added later.

18.2.7.2 New Decorations: Modifying Style

Modifying style specifications is trivial; all that is needed is to document a new key-value pair. For example, when gradients were added to the list of available style attributes, there was no change made to the XML schema, and the following style was immediately legal.

```
<style id="s" value="fill: blue;
 gradient: white; gradient-angle: 45;
 gradient-focus: 0.5"/>
```

Following the second rule in Section 18.2.7.1, all the `gradient` terms were ignored and the graph displayed without gradients. When the code was written to display gradient fills, the gradient style definition was applied and used. Users of previous application versions can read the new XML and will see essentially the same charts, with some style features ignored. This behavior is similar to the behavior of web browsers when confronted with more recent HTML or CSS definitions than the ones they were written to accommodate.

18.2.7.3 Nodes for Generalization: Parameter and Method

The grammar defines some concepts, such as axes, dimensions, variables, and faceting, that are unlikely to need much extension. Axes are well understood and ViZml provides means of specifying margins, rounding, formatting of numbers/dates/strings, tick frequency and base, tick length, label angle, and staggering. It is unlikely that many applications will need to override these. Dimensions, variables, and faceting have specific meaning in the grammar, and so are not amenable to much change. For these types, modifying the schema is the preferred means of extension.

In contrast, there are other structural entities defined that are practically certain to be extended. A prime candidate for extension is the **`statistic`** node, which is used to define any statistic that could be calculated on data. This is a huge class and is constantly growing. Even were we to define all the statistics currently available, next year would see the list incomplete. Rather than attempt such a definition, ViZml explicitly defines statistic in terms of one or more **`parameter`** nodes, which contain a textual *key* attribute and a completely general *value* attribute. This freedom allows any parameter to be defined, at the expense of losing the ability to validate that the parameter makes sense. The `statistic` node also has a required `method` attribute, which allows the main type of statistic to be defined. Using this system, a robust linear fit is defined as follows.

```
<statistic method="smooth">
  <parameter name="model" value="linear"/>
  <parameter name="robust"
   value="biweight"/>
</statistic>
```

and a *loess* smooth with uniform kernel and window of width 25% of the data is defined by the following.

```
<statistic method="smooth">
  <parameter name="model" value="loess"/>
  <parameter name="kernel"
   value="uniform"/>
  <parameter name="width" value="25%"/>
</statistic>
```

The disadvantage of such flexibility is that the following XML is valid, but unlikely to be useful.

```
<statistic method="smooth">
  <parameter name="model"
   value="polynomial"/>
  <parameter name="degree"
   value="should be a positive integer"/>
</statistic>
```

Another example of the use of the method / parameter approach is **`CoordinateTransform`**. This node takes a set of defined dimensions and transforms them. The initial version of ViZml defined the following transforms.

`transpose` – reflects the coordinate system around the line $y = x$
`polar` – interprets the first two coordinates as theta and radius
`rectangular` – performs a metric transform into 3D
`oblique` – performs an oblique transform in 3D
`triangular` – converts 3D coordinates to 2D barycentric coordinates
`inset` – insets dimensions to change the aspect ratio or trim edges

These coordinate transformations can be chained. For example, if we initially have a single stacked bar and apply the following transforms, we create what we might call a "3D doughnut chart." The first transform takes the stacked bar and converts it to polar coordinates (a pie chart). The second transform insets the lower radius so as to remove a radius from the center of the pie. The final transform locates the result in 3D.

```
<coordinateTransform method="polar"/>
<coordinateTransform method="inset">
  <parameter name="r-min" value="10%"/>
</coordinateTransform >
<coordinateTransform method="rectangular">
  <parameter name="phi" value="45"/>
  <parameter name="theta" value="15"/>
</coordinateTransform >
```

Figure 18.7 shows the result of a rather unusual `CalendarTransform` applied to a scatterplot that includes a *path* element. This figure is not easily recognizable as a scatterplot, yet each day in the calendar is a simple frame.

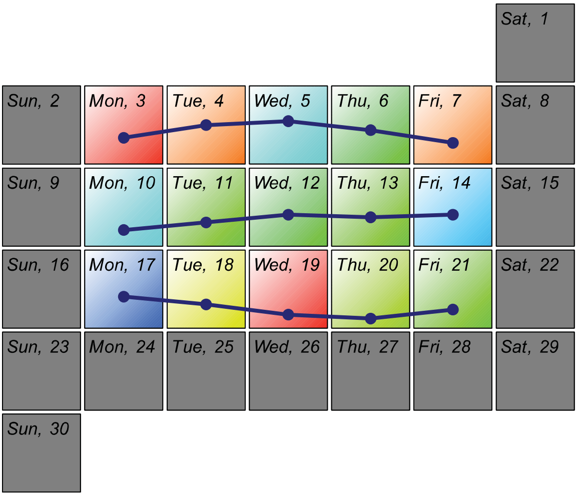


Figure 18.7 *A calendar rendition of the SPSS stock data (color represents intraday volatility and height of dot represents closing price)*

The complete XML for the graphic in Figure 18.7 is as follows:

```
<?xml version="1.0"?>
<graph>
   <source id="data">
   </source>
   <variable id="date" sourceName="date" source="data" cate-
gorical="false">
      <meta>
         <dateTimeFormat>
            <dateFormat>
               <dayOfMonth/>
               <monthName long="true"/>
               <year showCentury="true"/>
            </dateFormat>
         </dateTimeFormat>
      </meta>
   </variable>
   <variable id="high" sourceName="high" source="data" cate-
gorical="false"/>
   <variable id="low" sourceName="low" source="data" cate-
gorical="false"/>
   <variable id="close" sourceName="close" source="data"
categorical="false"
      domain="vd"/>
   <variable id="exist" sourceName="close" source="data"
      categorical="true"/>
   <variable id="range" categorical="false" expres-
sion="high+low"
         source="data"/>

   <domain categorical="false" id="dateDomain">
      <interval min="1996-06-01" max="1996-06-30"/>
   </domain>

   <domain categorical="false" id="vd">
      <interval min="23" max="27"/>
   </domain>

   <frame id="outer">
      <frame id="main" style="f">
         <dimension role="x" upperMapping="exact" lowerMap-
ping="exact">
         </dimension>
         <dimension role="y">
         </dimension>
         <coordinateTransform method="extension">
            <parameter name="class"
               value="com.spss.vis.sample.CalendarTrans-
form"/>
            <parameter name="start" value="1996-06-01"/>
            <parameter name="end" value="1996-06-30"/>
         </coordinateTransform>

         <element type="point" id="p" style="elementStyle">
            <x variable="date"/>
            <styleBy variable="exist" styleCycle="a"/>
            <color variable="range">
               <statistic method="range"/>
            </color>
            <labeling variable="date" style="ital">
               <dateTimeFormat>
                  <dateFormat>
                     <dayOfWeek long="false"/>
                     <dayOfMonth fill="false"/>
                  </dateFormat>
               </dateTimeFormat>
            </labeling>
         </element>
         <element type="path" id="pp" style="spp">
            <x variable="date"/>
            <y variable="close"/>
         </element>
         <element type="point" id="ppp" style="spp">
            <x variable="date"/>
            <y variable="close"/>
         </element>
      </frame>
   </frame>

   <styleCycle id="a">
      <cycle>
         <style id="a" value="fill:navy; gradient:white;
               gradient-angle:45"/>
      </cycle>
   </styleCycle>
   <style id="fonts" value="label-location-x:negative;label-
location-y:
         positive;label-location-inside:true;font-fami-
ly:Arial;
         font-size:8pt">
      <style id="ital" value="font-style:italic;font-
size:7pt;margin:1px"/>
   </style>
   <style id="hidden" value="visibility:hidden"/>
   <style id="f" value="fill:transparent; stroke:transpar-
ent;margin:20px"/>
   <style id="cell" value="stroke:transparent;fill:transpar-
ent"/>
   <style id="ms" value="stroke:black;stroke-width:0.5px"/>
   <style id="elementStyle" value="fill:grey;glyph-
type:square;
      glyph-size:1.50cm;stroke-width:0.5px"/>
   <style id="spp" value="stroke-width:2px;stroke:navy;
fill:navy;
      glyph-size:4px"/>
</graph>
```

This XML shows the method/parameter extensibility approach for both statistics and for coordinate transforms.

18.2.8 Comparison with Other XMLs

As we have discussed, the approach taken in defining ViZml was starting with core grammar entities in a central structure section, then adding optional child elements and attributes to expand possibilities and override defaults. The specification forms a clear separation between data, structure and styles. Style specifications are based on open standards, primarily cascading style sheets (CSS) definitions. Data specifications are based on tables, either embedded or external, with extensive metadata definable. We considered a number of other markup languages in the design of ViZml and used them for motivation. In this section we compare them to ViZml.

18.2.8.1 HTML

The most extensively used markup language, even if it is not actually an XML, is Hypertext Markup Language (HTML). In combination with cascading style sheets, HTML provided motivation for the following.

1) The separation of styles and structure (HTML style sheets and the common injunction not to embed style information in HTML).
2) Simple methods for creating frames which can be relatively or absolutely positioned.
3) Minimal formatting and styling attributes that solve most users' problems.
4) Syntax for specifying lengths in multiple units and colors.

18.2.8.2 VML and SVG

The Vector Markup Language is an XML designed by Microsoft for the display of vector images within their Office suite and for display of vector images on the web in Internet Explorer. SVG is the acronym for Scalable Vector Graphics, an open-source initiative supported by Adobe that has similar goals. Both systems allow objects such as rectangles, ovals, polygons and text to be defined. Fills, gradients, patterns, transparency, dashing, *etc*., are all extensively defined. VML has a few more high-level operations, such as extrusion and shadows, whereas SVG has more low-end operations, such as better path support and clipping.

ViZml defines items at a higher level than these languages, and has more semantic information. An axis in VML or SVG is simply a set of styled lines and text. However, the structure of items in SVG and VML was used to motivate the design of similar items in ViZml. This means that the ViZml engine can produce VML and SVG output that aligns with the source ViZml. In particular, when VML output is requested from the ViZml engine, the VML produced by an axis is grouped logically, with groups given `id` tags and class attributes so that, for example, a JavaScript click handler can identify the part clicked on. This is helpful in building interactive thin-client web applications.

Also borrowed from these two XMLs were the names of general attributes and style attributes which were not specified by CSS. The names of the gradient attributes presented in Table 18.1 were based on VML names, for example.

18.2.8.3 MathML

MathML is a low-level specification for formatting mathematical expressions and encoding them for machine-to-machine communication. It also provides a foundation for the inclusion of mathematical expressions in Web pages. As a language for expressions, ViZml adopts the syntax of MathML for the creation of faceting expressions. The notion of operators and operands with simple rules was adopted after some trial and error with other systems. To represent a complex facet expression such as: (x*y + w*z) / (a*b), we use the following XML, which aligns closely with the style of MathML.

```
<tableFacet>
  <nest>
    <blend>
      <cross>
        <facetVariable variable="x"/>
        <facetVariable variable="y"/>
      </cross>
      <cross>
        <facetVariable variable="w"/>
        <facetVariable variable="z"/>
      </cross>
    </blend>
    <cross>
      <facetVariable variable="a"/>
      <facetVariable variable="b"/>
    </cross>
  </nest>
</tableFacet>
```

18.2.8.4 PMML

Predictive Model Markup Language (PMML), was a natural choice for examination when constructing an XML for statistical graphics. The standard is documented at *www.dmg.org*. The intent of PMML is rather different from ViZml, however. PMML describes the output from a model, rather than specifying the type of a model, and so it is less of a prescriptive language than a descriptive one. Some of the features of the language are similar (notably the similar importance of distinguishing variables (PMML DataFields) based on whether or not they are categorical, but in general PMML forms a separate XML that can be used in conjunction with ViZml, but not as a similar system.

18.2.9 GraphML

GraphML is a language for describing the structural properties of graphs in the sense of node-link or node-edge diagrams. It incorporates a flexible extension mechanism for handling application-specific data. As node-edge graphs are a subset of the graphs that ViZml can produce, GraphML was examined closely. GraphML is a later version of GraphXML. Because of its specific focus, many of the GraphML concepts do not generalize well and were not considered sufficiently useful to include in ViZml. Perhaps the biggest contribution of GraphML was in the negative sense. GraphML ties the data extremely tightly to the structure and presentation of the graph. As such it is very flexible — nodes can be given explicit representations, links can be styled in an arbitrary fashion, and so on. However, this makes it very hard to use GraphML to build a template — a specification that can be applied to many datasets. In a strong sense, GraphML is an output XML like PMML; it describes a single instantiation of a node-edge graph very well, but provides little support for specifying general presentations of graphs.

18.3 Summary

Although the reasons to create statistical graphics are varied, and although there are many different tasks that might be attempted using such graphics, the fundamental function of a statistical graphic is to present data visually. Whether the user will view the graphic as printed on a high–quality printer, or whether they will interact with it on a 96 dpi screen with 32-bit color, in either case the user should be viewing something that presents the data in a truthful way, does not obscure features that may be of interest, and is aesthetically pleasing.

In any system apart from demoware or research systems, it will be necessary to store specifications of charts. Paper or online reports will need to be run at regular intervals with the same specification, users will want to save their explorations and come back to them later, etc. The grammar of graphics provides a simple specification aimed at describing the basic structure of a chart; this chapter has expanded the basic specification to a wider set of possibilities. Users from different countries, users with different abilities to perceive color, and users with different aesthetic sensibilities will all need to specify charts with the same fundamental structure, *vis-à-vis* the grammar, but with different presentation characteristics. And, although it may sound trivial, if there is any aspect of a chart that you cannot control, and you dislike the original designer's choice, it will be a constant source of irritation.

The use of XML for specifying graphics is natural since charts are tree-like, charts are thought of in terms of major elements such as axes and bars, and because the extensibility of XML allows us to plan for unthought of future graphics. The many freely available tools and resources for working with

XML are a useful bonus. We divide the ViZml schema specification into sections so that we can isolate data, structure and style. By doing so we can apply style sheets to existing specifications in a similar way to the way CSS style sheets are applied to web page, we can change the structure of a chart without needing to worry about the effect on data and styles, and we can easily adapt a specification without needing to understand the structure of the chart.

Finally, by coupling GPL with ViZml, we provide two alternative methods for constructing and editing graphics in a production environment. Since GPL generates ViZml as its intermediate code, we allow inspection and debugging with a wider array of tools. And for those more interested in models than styles, we allow GPL users to employ ViZml templates to specify consistent default styles. This keeps the GPL language simple *and* customizable.

18.4 Sequel

The next chapter turns everything inside-out. We will learn how to produce data from graphics.

19

Reader

The word *read*, according to the *Oxford English Dictionary*, has obscure origins in Old English, French, and other Indo-European languages. Its earlier meanings have mostly to do with giving counsel, considering, expounding, or explaining something obscure. Its modern association with text derives from the older sense of understanding, rather than just looking. This interpretation suits our purposes, because a reader in this sense is someone who can parse and understand a graphic, including *both* text and image. This chapter introduces the design of a graphics reader.

This task is formidable. As Pinker (1997) says,

> A seeing machine must solve a problem called inverse optics. Ordinary optics is the branch of physics that allows one to predict how an object with a certain shape, material, and illumination projects the mosaic of colors we call the retinal image. Optics is a well-understood subject, put to use in drawing, photography, television engineering, and more recently, computer graphics and virtual reality. But the brain must solve the *opposite* problem. The input is the retinal image, and the output is a specification of the objects in the world and what they are made of — that is, what we know we are seeing. And there's the rub. Inverse optics is what engineers call an "ill-posed problem." It literally has no solution. (p. 28)

In designing a graphics reader, our goals are less ambitious. First, we will limit the problem to a particular decoding task. Then we will leverage what we have learned so far about the organization of objects in **graphics world**. Specifically, we will *invert* the functions we have already used to create graphics. Because we designed these functions to be invertible, this becomes a "well-posed problem." We have been making graphics from data. Now we are going to examine how to make data from graphics.

The function

$$f\colon S \to T$$

is invertible if there exists a function

$$g\colon T \to S$$

such that for every x in S, $g(f(x)) = x$ and for every y in T, $f(g(y)) = y$. We call g the *inverse* of f and denote it as f^{-1}. In a sense, inversion is how to get from there to here assuming we know how to get from here to there. Not every function is invertible. And not every system or composition of functions is invertible. If all the functions in a chain of functions are invertible, however, then the chain itself is invertible by executing the inverse functions in reverse order.

In the development of this system, we have been concerned about invertibility because it keeps objects clean and domesticates their behavior in an interactive environment where we often need to know where we came from. Sometimes it is not easy to invert a function without doing a lot of extra work or carrying along extra information (as with 3D to 2D perspective projections), but this should not deter us from pursuing the goal wherever we can.

In the next section, we will present the problem. Then, we will summarize Steven Pinker's propositional model of graphics reading to provide psychological background. Finally, we will outline the approach to a solution through a graphics grammar.

19.1 The Problem

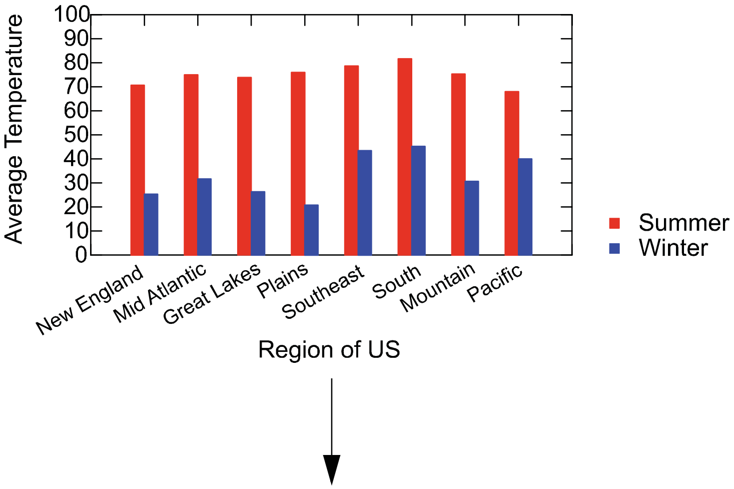
Suppose we have the graphic shown in Figure 19.1. Our problem is how to derive the table at the bottom of this figure by scanning the graphic. We may bring to the task prior knowledge about what graphs and graphics are, but not knowledge about this specific graphic.

There are several constraints we must place on the problem. First of all, we accept as a solution any table whose organization allows us to decode the data correctly. For example, if summer and winter were stacked in a single column called temperature and we had an extra index column called season, this would be a feasible solution. Without metadata, we have no way to determine whether summer/winter denotes independent groups or repeated measures. In other words, this graphic could have been produced by either

```
DATA: s = string("SUMMER")
DATA: w = string("WINTER")
ELEMENT: interval(position(region*(summer+winter)), color(s+w))
```

or

ELEMENT: *interval*(*position*(region*temperature), *color*(season))



Region	Summer	Winter
New England	71	25
Mid Atlantic	75	32
Great Lakes	74	26
Plains	76	21
Southeast	79	44
South	82	45
Mountain	75	31
Pacific	68	40

Figure 19.1 *From graphic to data*

Second, we expect rounding error. But we assume this error is uniform over the field of the graphic if we are working in rectangular coordinates. If a scale or coordinate transformation is involved, then we expect error to be transformed in the process.

Third, we are not concerned with general understanding. The problems Pinker addresses — detecting relationships, reasoning about trends, proportions, and so on — are not part of our problem domain. Of course, if we produce a graphics reader capable of decoding data, then we can feed the data mining and statistical engines already designed to address these more general problems in an automated system. People are better than computers when finding patterns in images, but computers are better than people when finding patterns in numbers.

19.2 A Psychological Reader Model

Bertin (1967), Simkin and Hastie (1987), Kosslyn (1989), Pinker (1990), Cleveland (1993), and MacEachren (1995) have all developed models to explain how people decode graphics. Pinker's model goes farthest toward solving the specific decoding problem we have posed. Pinker's model also resembles in many respects the object-oriented approach we have taken later in this chapter. This is not surprising, because Pinker used a propositional system developed by mathematical logicians (see Copi, 1967; Epp, 1990) and implemented by cognitive psychologists for modeling memory (see Rumelhart, 1977; Anderson, 1983) that influenced the development of object-oriented design itself.

A **propositional calculus** is a system that involves the following.

1) a set of variables whose truth or falsity is assumed,
2) symbolic operators, and
3) syntactical rules for producing expressions.

Expressions consisting of variables and operators following proper syntactical rules are called *well-formed formulas* (*wff*). A *wff* produces a truth table that can be used for a variety of purposes ranging from testing a logical argument to producing an electronic circuit. A **predicate calculus**, by contrast, is a propositional system that involves variables whose truth or falsity is *not* assumed. A predicate calculus consists of a set of *objects* (variables) and *predicates* (functions). The predicate functions operate on variables. These functions are unary, binary, or *n*-ary. Some examples are *circle*(x), *part*(x, y), and *parallel*(x, y). If x = '*body*' and y = '*head*', then *part*(x, y) is true, but *part*(y, x) is false.

It is often convenient to represent object-predicate relationships in a graph consisting of a network of labeled nodes and edges. Nodes are labeled with object names and edges are labeled according to predicates. Figure 19.2 shows Pinker's general graph schema based on a predicate calculus system. Pinker has omitted variable names inside the nodes (except for iterators, which we will discuss later) and has labeled the graph with predicate function names. We will use the name of the unary predicate function operating on a node to describe the node itself. The node at the top of the graph, for example, represents *scene_graph*(x). This entity stands in a part/whole relationship with everything below it. The *framework*() node stands in relation to *scene_graph*() via the function *part*(*scene_graph*(), *framework*()). There is a formal duality between the graph and the predicate calculus. This formalism distinguishes Figure 19.2 from the loosely-drawn network diagrams that people use to make *ad hoc* collections of objects look like information processing systems.

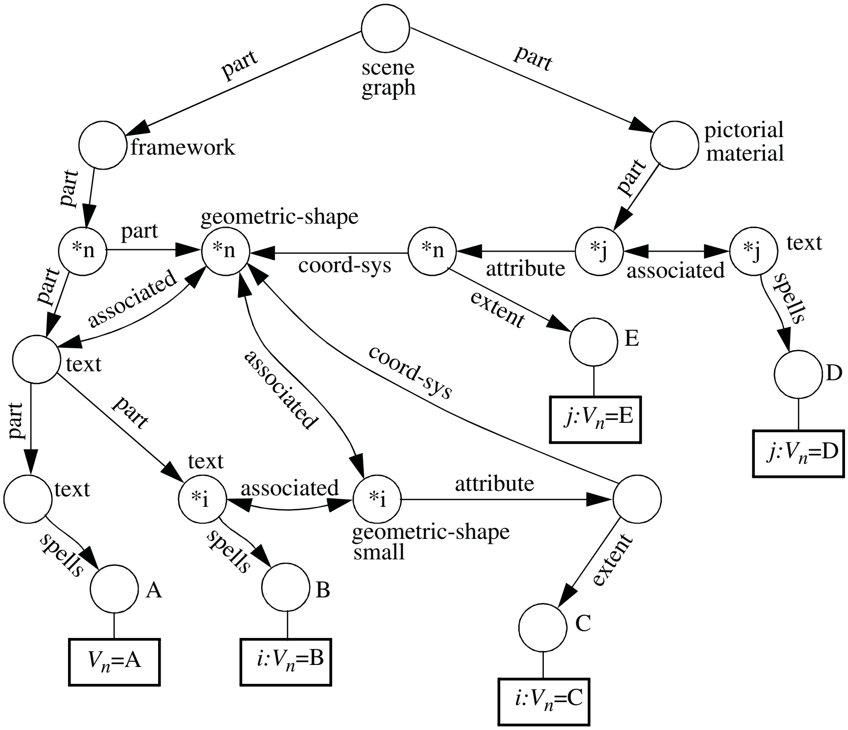


***Figure 19.2** Propositional reader model (adapted from Pinker, 1990)*

Some of the nodes in Figure 19.2 contain iterator variable names, which are denoted by an asterisk. The **n* iterator, for example, specifies an iterated replication of the node plus all its predicate relationships. Because the iterators are variables, their values are known to other functions linked to them. For example, each iteration of the *geometric_shape*(**n*) node is associated with a result value V_n at the bottom of the graph.

Another notation peculiar to Figure 19.2 is the capital letter designation for predicate functions at the bottom of the figure. These are *unknown predicates*, which are functions that are instantiated at "run-time" when specific values of the variables are available.

The output of this system is defined in the bold outlined boxes below the unknown predicates at the bottom of the figure. The two extent measures *C*() and *E*() are for comparisons of size or intensity that are used by graph readers to answer questions like "Is the third bar taller than the second?". The *A*(), *B*() and *D*() predicates yield the numbers or strings that we need to enter into our data table of Figure 19.1.

Let's focus on the problem of decoding the leftmost red bar in Figure 19.1. In the following explanation, we will supply the predicate function names specific to a bar graph schema that are missing in the general graph schema of Figure 19.2. We will try to do this in a series of sentences. First of all, the **j* node directly below and to the left of *pictorial_material*() would be *bar*(**j*) in a bar graph schema. The first bar would thus be recognized by the reader as *part*(*pictorial_material*(), *bar*(1)). To the left of *bar*(**j*) would be several attributes (*color*, *size*, *position*, etc.). The horizontal *position*() attribute would be linked through a *coord_sys*() rectangular coordinate system to a horizontal axis element *geometric_shape*(1) that is *part*() of *framework*(). This axis *geometric_shape*(1) is *associated*() with *text*() that *spells*() "New England" and is assigned to the first variable V_1. The *bar*(1) also has an *attribute*() of vertical *position*() which is linked through a *coord_sys*() to a vertical axis *geometric_shape*(2) that is *part*() of the same *framework*(). The location on the vertical axis line *geometric_shape*(2) that is indexed by the *position*() of the top of the *bar*() has *near*(*tick*(), *text*()) that *spells*() "70." We enumerate and assign a value of 70 to the second variable V_2. Finally, the variable names "Region of US" and "Average Temperature" would have been located through the *A*() function tied to the *part*() of the vertical and horizontal *geometric_shape*() axes that is *near*() the *text*() containing them.

To make our description more rigorous, we would have to collect all the predicate functions in a linked list. We chose a sentence format to make this complex sequence more readable. Nevertheless, we should be able to get an idea of how this system functions from our description. There are several critical aspects to note.

First, look-up and scale decoding is done for every object and part-object in the system. The reader must locate and examine text every time a direct evaluation on a categorical or continuous scale is required. This apparently inefficient process fits empirical data of Pinker (1990), Simken and Hastie (1987), and other researchers quite well. As we shall see, it is not a model we need for efficient decoding in an automated system. Once coordinates are known and scales calculated, every measurement of position is a simple affine transformation for a system that is capable of doing linear algebra in its head (obviously not human).

Second, the propositional system says nothing about the number of schemas required for its implementation. There are presumably bar graph schemas, line graph schemas, and so on. Each of these may be a close relative of another and all are presumably children of the general schema in Figure 19.2. This plethora of schemas probably fits the human data better than the system we will develop in the next section. As Pinker indicates, efficient decoding of graphics is a learned skill that improves with the acquisition of new schemas rather than the refinement of a single global algorithm.

Third, Pinker's coordinate system is driven by a computational vision model outlined in Marr and Nishihara (1978). This means that graphic evaluation is confined to rectangular and polar coordinates. Again, this may follow the human decoding model more closely. It may be next to impossible for a human to decode raw values in a graph represented by a projectively transformed logarithmic scale, for example. This is no more difficult for a machine than handling rectangular coordinates.

Fourth, Pinker's model has no provision for composing functions outside of the network. The bar dodging in Figure 19.1, for example, is seen by humans in a variety of clustered bar charts, but inducing the rule for other graphs and coordinate systems is nontrivial. As Pinker (1997) discusses, people do not compose even simple functions in the way we are proposing.

None of these characteristics of Pinker's model makes it less rigorous or adequate for describing how we humans read graphics. Indeed, as we have said, it almost certainly fits experimental data better than the system we will propose in the next section. Nor does this imply that it would be futile to build a graphics reading system along the lines Pinker suggests. Pinker's model can be implemented in a closed system. One might learn a lot from unexpected results and from comparing machine to human data. This is a venerable tradition in cognitive psychology.

Two lessons, we believe, can be taken from examining this model, however. First, it is not always desirable to use the strategies people actually bring to well-defined problems as methods for solving these problems automatically. In engineering, we should use psychological research for guidance, but not as a template. Secondly, well-articulated psychological models based on empirical research give us a head start in defining the basic objects we need for an automated system. We are more impressed by the similarities of Pinker's model to the one we will propose than by the differences.

19.3 A Graphics Grammar Reader Model

Figure 19.3 presents our design of a graphics reader. The notation differs from Figure 19.2. We have adopted a subset of the Booch–Jacobson–Rumbaugh **Unified Modeling Language** (UML) notation (see Fowler and Scott, 1997). The triangular–shaped arrows represent "is a" relationships (inheritance). The diamond–shaped arrows represent "has a" relationships (aggregation). The hollow arrows represent interfaces. An interface is a specification for how a task is to be performed. This specification can be fulfilled by a variety of different implementations or devices. We use these interfaces to avoid making the objects they point to part of the system. Finally, italic type denotes abstract classes. These are classes that are containers for concrete classes. They specify how a class is to behave and what are its interfaces, but exist only as patterns for creating concrete classes. Using abstract classes helps to keep the design free of implementation details and provides an economical method for grouping classes that inherit from a common ancestor and whose differences in functionality are not substantively important.

The bold boxes for Image and Data show the input and output of the system. Image contains a bitmap or other representation of the graphic image. Data contains some representation of the table in Figure 19.1. The first thing we need is a Scanner. This object converts Image into an object discernible by the children of Reader. Since Reader has a Scanner, the converted graphic Image is now owned by the Reader.

Reader has three main components: an Identifier, a Measurer, and a Framer. The general strategy for converting geometric entities to numerical and textual tables is to measure extents or other aesthetic attributes and convert these to numbers or text. To do this, Measurer needs to have two packets of information. First Measurer must know how to identify an object in order to assemble a measuring strategy and methods. Second, Measurer must know about the coordinate system and frame that defines the dimensions on which graphs are measured. we will review these two in turn.

Identifier examines the shapes and attributes of objects and compares them to its list of known graphic elements. This is not a simple pattern matching or template procedure, however. Because Identifier knows about the coordinate system (through Reader), it is capable of knowing that a section of a divided bar in rectangular coordinates is the same object as a pie slice in polar coordinates. It uses an inverse polar-to-rectangular transformation to make this comparison. Identifier requires shape recognition methods for accomplishing its work, but because it has top-down information (like the rest of the system) about the array of shapes it is seeking, Identifier's methods can be customized for this purpose. Identifier supervises several children. AxisIdentifier, for example, looks for ordered text near rules and tick marks with an associated label. PointIdentifier looks for clouds of points inside a frame. It knows that a cloud may consist of points that do not look like geometric points, but it seeks to determine if they are a collection of similar atoms within a common pattern and grouped inside the frame. Finally, Identifier evaluates the votes of each of its children and picks the strongest vote. This follows the Pandemonium model described in Selfridge (1959) and later employed as a foundation for neural networks.

Framer is responsible for establishing the model underlying the graphic. It must associate variables with dimensions, set the coordinate system, and determine the scales on which dimensions are mapped. The first of these three tasks is accomplished by Modeler. Through Framer, Modeler can parse text and determine whether more than one variable name is associated with an axis or guide. Modeler, like the other agents in the system, knows where to look for an axis or guide because of the information provided by Identifier through Reader, the common parent. In addition to recognizing blends by parsing text, Modeler recognizes crossings by looking for perpendicular axes (after inverse transformation to rectangular coordinates). With legended variables, Modeler looks for all possible combinations of attributes to recognize a crossing. Modeler recognizes nestings by looking for subscales with different ranges on common variables. The CoordinateSetter depends on the presence of a Guide to determine coordinates. For example, a circular axis with appropriate ticks would signal CoordinateSetter that it is looking at polar coordinates. Additional methods for error checking (seeing that angles radiate from a central origin, for example) are available to CoordinateSetter to verify its hypothesis. Finally, Scaler computes scales by examining the spacing of tick marks on axes and their associated numerical labels. It inverse transforms the tick locations to see if the physical spacing of ticks in the transformed metric can be made proportional (after adjusting for arbitrary constant) to the numerical values at the ticks. Scaler, like Modeler, requires prior work by CoordinateSetter. Thus, Framer is usually given first priority by Reader and CoordinateSetter is given first priority by Framer.

The result of this process is that Measurer can do its work through a simple chain of coordinate and scale transformations without further reference to text or other cues in the scene. Measurer is an abstract class because it is a collection of different measuring objects that all return a common data structure. The actual measurements require location methods that are specific to the graphics being measured. For example, PointMeasurer requires a centroid computation to find the center of a *point* graphic. LinkMeasurer requires a skeletonizer to assure that the links are measured at their central mass (assuming they have thickness). ColorMeasurer requires a colorimeter. Once these readings are taken and collected, Measurer returns to Reader the information that Reader must pass to Data. This is the collection of numbers and text and field identifiers that is the dataset we seek.

How does this process work for the graphic in Figure 19.1? Let's consider AxisIdentifier first. This animal hunts around for an axis and barks when it finds one. The components it is looking for are a *rule*, *ticks*, tick *labels* (letters or numbers), and an axis *title* (*e.g.*, "Average Temperature"). The rule may be curved, so this little beast has to sniff along line segments that do not cross themselves and look for the associated elements attached to or near the line, much the way a contour-following algorithm works. When AxisIdentifier barks, it must then furnish to other clients the coordinates (in Image space) of the components of an axis. At this point, we are conversing in Image World coordinates, which we will call *w* coordinates.

How are these *w* coordinates and their object descriptors used by clients? Let's examine CoordinateSetter first. This calculating engine must examine one or more axis objects and try a number of inverse transformations of the plane that transform the axis rules to a set of orthogonal or parallel lines. In the case of Figure 19.1, an identity transformation yields the result immediately. At this point, we are conversing in rectangular coordinates, which we will call *r* coordinates.

AxisScaler examines the spacing of the tick marks on the plane in *r* coordinates and applies a set of inverse transformations (*log*(), *exp*(), etc.) to get them to be equally spaced. Then it uses TextConverter to convert the numerals near the ticks (located by AxisIdentifier) to numbers, assuming the scale is numerical. Through these operations, it can determine the scale transformation.

Again, for Figure 19.1 this would be an identity. At this point, we are conversing in scaled coordinates, which we will call *s* coordinates.

We still do not have data values, which we will call *d* coordinates. For this last step, which is an affine transformation, we need to bring in the scale values. A simple rescaling involves the transformation

$$d_i = s_i\left(\frac{d_{max} - d_{min}}{s_{max} - s_{min}}\right) + d_{min}$$

For the values on the horizontal axis, we need to remember that the character string labels correspond to the integers 1 through *n*. In the case of Figure 19.1, n=8, so d_{min}=0 and d_{max}=9.

The importance of inversion is nowhere more apparent than with coordinate systems. We *draw* the graphics in Chapter 9 through the composition chain $d \rightarrow s \rightarrow r \rightarrow w$. We *read* the graphics in Chapter 10 by inverting our transformations in the chain $w \rightarrow r \rightarrow s \rightarrow d$.

We will not discuss the remaining objects in Figure 19.3 with respect to the reading of the graphic in Figure 19.1. How does Reader recognize that the bars are dodging, and how does it link this information to the legend? Can Reader use the axis titles to infer whether blending is in the model? What text would it look for to identify this situation? Is AxisIdentifier fooled by the top and right axes or can it recognize that these are associated with the other two? How does color in the legend get linked to color in the bars? How are LegendIdentifier, ColorMeasurer, LegendScaler, and LegendModeler involved in this inference?

Additional questions are raised by scenarios involving other figures in this book. How does AxisIdentifier avoid thinking the *path* graphic in Figure 8.10 is an axis? Assume this path did not cross itself; could AxisIdentifier then recognize it as a graphic rather than a guide? What does Reader produce when there are no axes, such as in Figure 10.60? How does Reader approach facets, from simple Trellises to the complex polar graphic in Figure 11.15? Could Reader be modified to handle facets? If so, could Reader be modified to handle facet graphs, such as the tree in Figure 11.11? What problems are introduced by reading 3D graphics? Can this be done at all with the model we have developed? What does Reader do with Table 20.1? Would it have to take a course with Ed Tufte or Bill Cleveland first?

Going through the scenarios necessary to understand parsing for these instances is the best set of exercises one can pursue in learning the aspects of the system in this book. The Reader is not only a useful application, it is also an exercise constructed to test the limits of a graphics grammar.

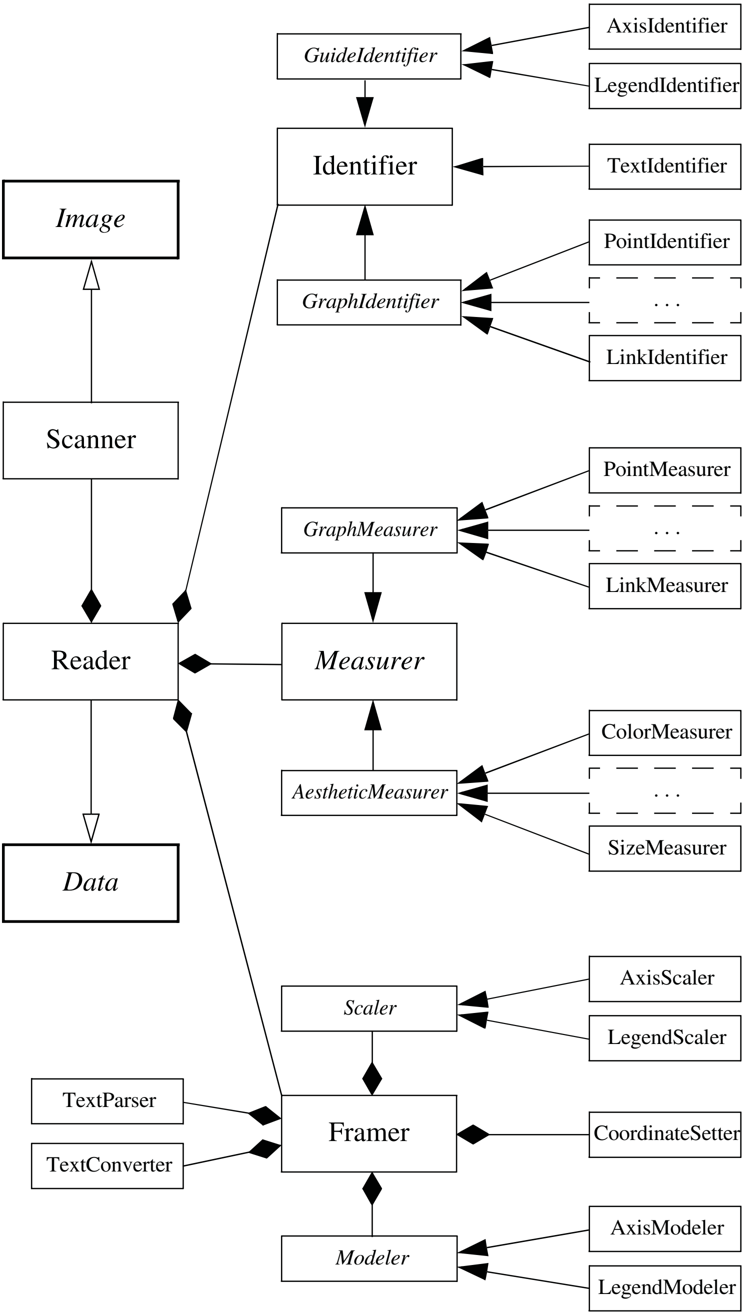


Figure 19.3 *A graphics grammar reader model*

19.4 Research

Automated graphics reading has recently become a focal area in the more general field of document recognition. The International Workshop on Graphics Recognition (GREC) has been sponsored by the The International Association for Pattern Recognition (IAPR). In the five biannual meetings held so far, several papers specifically devoted to chart recognition (pie, bar, line, scatter) have been presented. Springer Verlag has been publishing the proceedings of this conference in its *Lecture Notes in Computer Science* series. Springer also publishes the *International Journal on Document Analysis and Recognition*, which carries occasional articles on graphics recognition. The IAPR also helps to organize the International Conference on Document Analysis and Recognition, which meets every two years.

19.5 Sequel

The next, final chapter will analyze two unusual graphics in detail in order to show how a specification and data contain the meaning of a graphic.

20

Coda

The word *coda* comes from the Latin *cauda*, or tail. In music, a coda is the tail end of a composition. In grammar, it is the last of the three phonetic pieces of a syllable — the onset, nucleus, and coda. In this concluding chapter, we intend to show how the *grammar* of graphics can inform our understanding of the *meaning* of graphics. With Pinker (1990) and MacEachren (1995), we believe that understanding the meaning of a graphic is a lexical task. The syntactical information is expressed in its grammar and its semantic information is encapsulated in its associated data. We will examine this proposition through a detailed analysis of two statistical graphics.

20.1 Napoleon's March

Minard's "Figurative map of the successive losses of men in the French army during the Russian campaign, 1812–1813" is now one of the most famous statistical graphics, thanks to Tufte (1983). Tufte said of Minard's creation, "It may well be the best statistical graphic ever drawn." Tufte's devotion to this and other historical graphics is extraordinary; his books are worth owning for the quality of the reproductions alone. It would help during this discussion if you kept a copy of Tufte's book nearby. Also helpful is the fine analysis of this same graphic in Roth *et al.* (1997).

20.1.1 The Data

Table 20.1 shows the data behind Minard's graphic. We produced this table through a combination of digitizing the map itself, analyzing the annotations, and locating the geographic landmarks. Table 20.1 consists of three subtables — the city data, the temperature data, and the army data. We have omitted the data that define the rivers. There are other ways to organize these data, but we believe the structure of Table 20.1 is closest to the way Minard himself would have arranged the information. The data in Table 20.1 are designed to reproduce Minard's graphic, not to represent accurately the historical record.

Table 20.1 **Napoleon's March Data**

lonc	latc	city	lont	temp	date	lonp	latp	survivors	direction	group
24.0	55.0	Kowno	37.6	0	Oct 18	24.0	54.9	340,000	A	I
25.3	54.7	Wilna	36.0	0	Oct 24	24.5	55.0	340,000	A	I
26.4	54.4	Smorgoni	33.2	–9	Nov 9	25.5	54.5	340,000	A	I
26.8	54.3	Molodexno	32.0	–21	Nov 14	26.0	54.7	320,000	A	I
27.7	55.2	Gloubokoe	29.2	–11		27.0	54.8	300,000	A	I
27.6	53.9	Minsk	28.5	–20	Nov 28	28.0	54.9	280,000	A	I
28.5	54.3	Studienska	27.2	–24	Dec 1	28.5	55.0	240,000	A	I
28.7	55.5	Polotzk	26.7	–30	Dec 6	29.0	55.1	210,000	A	I
29.2	54.4	Bobr	25.3	–26	Dec 7	30.0	55.2	180,000	A	I
30.2	55.3	Witebsk				30.3	55.3	175,000	A	I
30.4	54.5	Orscha				32.0	54.8	145,000	A	I
30.4	53.9	Mohilow				33.2	54.9	140,000	A	I
32.0	54.8	Smolensk				34.4	55.5	127,100	A	I
33.2	54.9	Dorogobouge				35.5	55.4	100,000	A	I
34.3	55.2	Wixma				36.0	55.5	100,000	A	I
34.4	55.5	Chjat				37.6	55.8	100,000	A	I
36.0	55.5	Mojaisk				37.7	55.7	100,000	R	I
37.6	55.8	Moscou				37.5	55.7	98,000	R	I
36.6	55.3	Tarantino				37.0	55.0	97,000	R	I
36.5	55.0	Malo-jarosewli				36.8	55.0	96,000	R	I
						35.4	55.3	87,000	R	I
						34.3	55.2	55,000	R	I
						33.3	54.8	37,000	R	I
						32.0	54.6	24,000	R	I
						30.4	54.4	20,000	R	I
						29.2	54.3	20,000	R	I
						28.5	54.2	20,000	R	I
						28.3	54.3	20,000	R	I
						27.5	54.5	20,000	R	I
						26.8	54.3	12,000	R	I
						26.4	54.4	14,000	R	I
						25.0	54.4	8,000	R	I
						24.4	54.4	4,000	R	I
						24.2	54.4	4,000	R	I
						24.1	54.4	4,000	R	I
						24.0	55.1	60,000	A	II
						24.5	55.2	60,000	A	II
						25.5	54.7	60,000	A	II
						26.6	55.7	40,000	A	II
						27.4	55.6	33,000	A	II
						28.7	55.5	33,000	A	II
						28.7	55.5	33,000	R	II
						29.2	54.2	30,000	R	II
						28.5	54.1	30,000	R	II
						28.3	54.2	28,000	R	II
						24.0	55.2	22,000	A	III
						24.5	55.3	22,000	A	III
						24.6	55.8	6,000	A	III
						24.6	55.8	6,000	R	III
						24.2	54.4	6,000	R	III
						24.1	54.4	6,000	R	III

Digitizing Minard's graphic reveals several anomalies. First, Minard gave no date for the –11 degree temperature measurement at Bobr (the dates are labeled on the temperature line at the bottom of the graphic). Second, Minard's temperature line indicates that the last retreating troops covered more than 50 miles in only one day (between December 6 and December 7). This is implausible. It also does not fit the contemporary descriptions of the events. One of Minard's sources (de Fezensac, 1849) wrote of those early December days,

> Imagine vast snow-covered plains stretching as far as the eye can see, deep pine forests, half-burned and deserted villages, and marching through this mournful countryside an immense column of miserable wretches, almost all without weapons, moving along rag-tag and bobtail, slipping on the ice at each step and falling down beside the carcasses of horses and the corpses of their comrades.

M. de Fezensac's detailed timeline (with locations and events) does not match Minard's. It would appear Minard inconsistently amalgamated dates from the four sources he mentioned in his annotations to the graphic. Minard took other minor liberties with the numbers and the historical record. He combined some events in the campaign, as he says, "in order to make it easier to judge by eye the diminution of the army." For example, Minard drew one thin path for the final retreat west of Smorgoni and labeled it with numbers from 12,000 to 14,000 to 8,000 without changing its thickness.

Checking Minard's graphic against authoritative historical sources (*e.g.*, the statistical appendix in Chandler, 1966) makes several things clear. First, it must have taken him considerable effort to assemble these data from the published diaries, memoirs, and histories of the campaign that he cites in his comments. The authors of these works did not tabulate the troop losses consistently or in a readily usable form. Minard had to make quantitative inferences based on descriptions, estimates, and anecdotes. At best, the numbers printed alongside the paths in his map (especially the intermediate ones) are speculative. Second, he took liberties with the events in order to simplify his graphic. More happened in the campaign — routs, excursions, regroupings, bivouacs — than the graphic suggests. As Minard himself notes, he omitted major segments of the southern front. Altogether, his troop figures are about 20 percent short of the numbers reported in his sources.

One cannot argue that all these liberties are not significant. There are contemporary 19th-century maps of the campaign that lack these imprecisions, including some in the works Minard cites in his annotations. Minard's graphic is even less accurate in its portrayal of the central army's movements. The army retraced its path during retreat in several places (*e.g.*, Mojaisk to Smolensk), but Minard kept the advance–retreat paths geographically separate. This was unnecessary because he used separate colors for advance and retreat and the paths would have superimposed nicely (as is shown, for example, in the revision of Roth *et al.*, 1997)

In short, Minard's graphic cannot be taken as an accurate geographic or statistical summary of the campaign. It is, as he says, a "figurative map," a popular chart (like ones in newspapers and magazines today) intended to make a political point. One that ignores details. These observations do not diminish the credit Minard deserves for a beautiful piece of work. Although he did not invent the idea of representing a variable by thickness of a path (Playfair, among others, had done that a century earlier), he used it to great effect. Minard was an engineer, not a statistician or cartographer. Despite the liberties he took, however, Minard's chart is not meaningless. It is well-formed and based on a clearly structured dataset. The test of this assertion is that a properly designed computer program can draw his graphic from the data and a properly structured specification. Now we will examine that specification.

20.1.2 The Graphic

Figure 20.1 shows the specification and a graphic that reproduces the deep structure and content of Minard's original. Tufte says, "*Six* variables are plotted: the size of the army, its location on a two-dimensional surface, direction of the army's movement, and temperature on various dates during the retreat from Moscow." Tufte's statement refers to variable sets or dimensions in the specification, because there is a blending. The six dimensions are survivors, longitude (a blend of lonp, lonc, lont), latitude (a blend of latp, latc), direction, temp, and date.

In fact, there are *seven* dimensions in this plot. The seventh is group. This is not obvious from looking at the graphic. It only becomes apparent when we attempt to choose a graph for representing the troop movements from the data. In addition to *path*, possible candidates are *edge* and *polygon*. The last two are clearly inappropriate; they would require structures not easily computable from the original data and certainly not meaningful in terms of the variables. The *polygon* graph would require polygon vertices describing the outline of the path. The *edge.link.tree* graph would require a scheme for producing the branching. Obviously, the physical movement of an army *is* a path. Our problem is that there is no single path that would describe the actual movements. It is clear that there are three paths required — each corresponding to collected divisions of the army (as Minard indicates in his annotations). These are (1) the path of the central army group, (2) the path of the left flank taking the excursion northward between Kaunas and Vilnius, and (3) another central army group consisting of the second and sixth corps that marched from Vilnius to Polotzk and rejoined the others at Bobr (we combined the remnant of this group with the central army's retreat west of the Berezina River because Minard provides only aggregate numbers after this point). It is likely that Minard chose not to assign a visible attribute to the **group** variable in order to keep the aesthetic simple and conserve color. His focus was on *overall* attrition. If we did not have to be explicit about specifying a graph correctly, we would have taken the structure of this part of the graphic for granted.

The *position* frame specification for the upper *point* graphic is based on the longitudes and latitudes of the cities (lonc*latc) through and around which the troops passed. The *position* frame specification for the *path* itself is based on the coordinates of the itinerary (lonp*latp). Each of the three *path* graphics is determined by setting *split*() to the value of group. Since group is categorical, this splits one *path* into three. Since Minard chose not to represent group through *size*, *shape*, *color*, *texture*, or any other aesthetic attribute, we must use the *split*() function explicitly to split the paths. Otherwise, the *path*() graphing function would display a jumble of paths.

The *size* of each segment of the *path* graphics is determined by survivors. And the *color* is determined by direction (advance or retreat). The advance is red and the retreat is black. Tufte's faithful reproduction shows the advance path in pale brown. The original was almost certainly bright red for two reasons. First, Minard said it was red in his annotations on the graphic. Second, red in a lithograph oxidizes and fades to brown in months under direct sunlight and in years under indirect light. Saturated reds fade faster than dull. Tufte's comment, "Minard's refined use of color contrasts with the brutal tones often seen in current-day graphics" is probably not historically correct. It is similar to the mistakes archaeologists once made in failing to notice that frescoes fade.

Unlike Minard's graphic, the two north–south black retreat paths are superimposed on the red advances in Figure 20.1. If Minard intended to show that these southerly retreats occurred earlier in time by putting them underneath the red paths, then this is puzzling. The black path from Polotsk to Bobr, for example, consisted primarily of the second corps under Marshal Oudinot. They rejoined the central army at the Berezina River near Bobr months after the central army had passed through the region on the way to Moscow. More likely, Minard did not want the black to cut the red path into sections because he wished to emphasize the drive toward Moscow in a single swath of red. We cannot do this without changing the definition of a path, however. A path wants to be drawn from beginning to end in sequence. In a statistical graphic, the choice of position cannot be made on the artistic merits. This is one of the few instances in the chart where Minard is graphically ungrammatical.

There remains the other major element in Minard's display: the temperature graphic below. This is defined by a GRAPH specification containing a *path* based on lont*temp. Ordinarily, we would use a *line* to represent temperature. We used a *path* instead because we wanted to modify each segment through another variable. As we discussed in Chapter 8, a *path* must be used in place of a *line* if we wish to modify the appearance of each separate segment. To do this, we added an eighth dimension to Minard's seven and realized it with the aesthetic function *texture.granularity*(days). We did this because it is difficult to discern the pace of the armies' retreat from the date labels on the temperature graphic.

The days variable is created by differencing the lagged dates with themselves; the arithmetic is done by *diff*() after converting to numerical day-of-century values (numeric functions are overloaded to be able to handle string time variables). Notice that we had to fill in the missing value by linear interpolation using the *miss*() function before we could do this. Otherwise, the missing date between Nov 14 and Nov 28 would have caused missing values for days after using the *lag*() function. Once we have the days values, we use them to make each dash in the temperature line correspond to one day of marching. The implausible distance covered in the last day is immediately apparent.

The horizontal scale for the temperature graphic is linear in longitude and nonlinear in time — an ingenious linking of space and time on a single physical dimension. We have made this link more perceivable by supplementing the date annotations with the dash lengths. The two graphics (the march and the temperature) share a single horizontal dimension yet they vie for the vertical space in the display area. This is the same situation we encounter with bordered graphics, where we link on a common variable.

```
TRANS: date = miss(date, lont, "linear")
TRANS: ldate = lag(date, –1)
TRANS: days = diff(date, ldate)
GRAPH: begin(origin(0, 0), scale(12cm, 4cm))
  ELEMENT: point(position(lonc*latc), label(city), size(0))
  ELEMENT: path(position(lonp*latp), size(survivors), color(direction),
                split(group))
  GUIDE: legend.color(dim(1))
GRAPH: end
GRAPH: begin(origin(0, -2cm), scale(12cm, 2cm))
  ELEMENT: path(position(lont*temp), label(date), texture.granularity(days)
                color.brightness(.5))
  GUIDE: axis(dim(1), label("Longitude"))
  GUIDE: axis(dim(2), label("Temperature"))
GRAPH: end
```

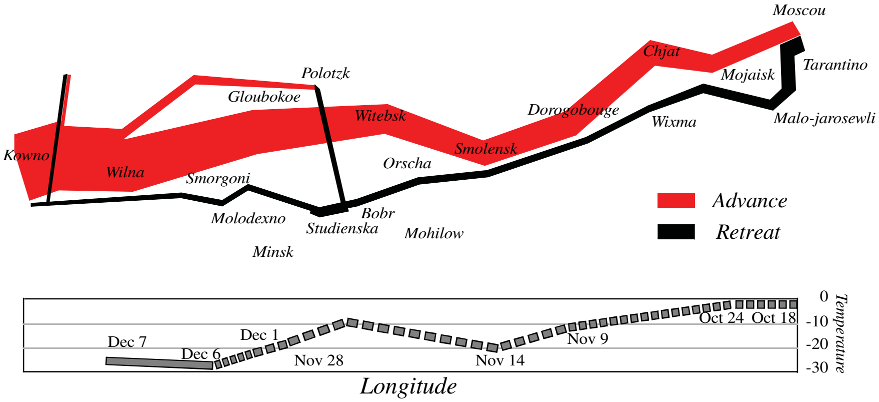


Figure 20.1 *Napoleon's Russian campaign (after Minard)*

20.1.3 The Meaning

Understanding the meaning of Minard's graphic is a matter of attending to the data and metadata underlying the graphic and then parsing the specification. There are other ways to specify this graph with a graphics algebra, but we have found none that is simpler and none that alters the substantive interpretation. The subtleties of this graphic (including Minard's errors and omissions) are revealed through a detailed analysis of the data in the light of the specification. Constructing a valid specification forces us to reconstruct the data properly.

Aside from the details that emerged from the process of producing Table 20.1, what is the main structure of the graph underlying this graphic? First of all, we know the data are geographic and temporal. We *could* hook up this specification to non-spatio/temporal data, but the *path* and *line* graphics would have a different meaning if this were the case. The *path* graphic is related to time only by indirection and only for half its extent (during retreat). The indirection is accomplished through the `date` variable.

Thus, Minard's graph is a *path* through geographical space linked to a meta-variable (temperature) through the time needed to cover the path. And his graphic is an aesthetically superb (better than mine) realization of that graph. What is unusual about the graphic is the positional sharing of the horizontal dimension (longitude) between the path frame and the temperature frame. A similar positional device is used to link panels of a SPLOM, but Minard employs it in a geographic context.

What other data could we link to a geographical *path* specification? We could use this specification (as Minard did) to represent the campaigns of other armies. We could use it to represent trade routes in the Hellenistic era (Koester, 1982). We could use it to represent the resettling of refugees in Germany, Eastern Europe, and Russia after World War II (Barraclough, 1984). Or, we could use it to represent migrations, as in the following example. As we will show, however, migration data do not always allow us to use a *path*.

20.2 Monarch Butterfly Migration

In the early 1970's, Fred Urquhart and his associates at the University of Toronto began tagging Monarch butterflies (*Danaus plexippus*) in an effort to track their southward migratory movements in eastern North America. The tags returned to his laboratory pointed to a site somewhere in Mexico. This led Urquhart to advertise in Mexico for assistance in locating the site more precisely. Kenneth Brugger, a textile engineer working in Mexico, contacted Urquhart after seeing one of the advertisements in a Mexican newspaper. Brugger joined the team, and in January 1975 he and his wife found the site in the Sierra Madre mountains west of Mexico City. This spectacular discovery revealed trees veiled in orange by countless Monarchs waiting to return north in the spring.

Many newspapers, magazines, and Web sites covering this story have presented a graphic consisting of a tree whose root is anchored at the Mexican site and whose trunk splits just north of Mexico into four branches. This branching tree portrays the butterflies heading north and splitting into four major migratory routes. The data do not support such a graphic. With Minard's graphic, we might dispute the accuracy of his data but not his grammar. In this case, we may accept the data but we dispute the grammar. We will summarize the data and then present an alternative graphic based on grammatical rules.

20.2.1 *The Data*

The data underlying Figure 20.2 were collected in the first six months of 1997 by children, teachers, and other observers reporting to the Journey North Project under the auspices of the Annenberg/CPB Learner Online program (*www.learner.org*). Each data point is an observer's first sighting of an adult Monarch butterfly during the period of northward migration in 1977. The date of the sighting was coded in two-week intervals and the location of the sighting was given in latitude and longitude. We omitted a few observations from the West Coast because these Monarch populations generally do not migrate to the same site in Mexico. Data and theory indicate that the butterflies are fenced in by the Rocky Mountains — both by climate and the lack of their critical nesting and food source, the milkweed.

There are other datasets on which we could build a migration graphic. Capture–recapture data arise when we capture a butterfly, tag it, release it, and capture it again. This allows us to infer survival, spatial distribution, and other aspects of a population distributed in space and time. It is particularly difficult with a fragile and dispersed migrating population such as Monarch butterflies, however. Further information on Monarch capture-recapture is available at *www.monarchwatch.org*, established by Orley R. Taylor at the University of Kansas Department of Entomology and co-sponsored by the University of Minnesota Department of Biology. Recently, Wassenaar and Hobson (1998) have linked additional variables to Monarch data by analyzing chemical signatures in butterfly wings. This method has helped to identify the US origins of butterflies wintering in Mexico.

20.2.2 *The Graphic*

Figure 20.2 presents our alternative to the prevalent tree-form butterfly graphic. We will discuss this alternative first and then show why a *tree* graphic (or a *path*) is ungrammatical. The specification of Figure 20.2 is simpler than the one for Minard's. This graphic consists of a map of the continental US overlaid with traces of the advancing front of Monarch northerly migration. The Journey North Web site has a map of the raw data plotted as colored points, one point for each sighting. Figure 20.2 shows more clearly the contours of the advance over time. These contours were computed by nonparametric smoothing.

Although we used the word *contours* to describe the butterflies' advance, the graphic we used in the specification is *line* instead of *contour*. This is because we chose to recognize the observations as containing random errors and thus ran the lines through the concentration of points at each different time interval rather than through their northernmost edges. Since the **date** scale is categorical, each smoother is computed separately on the subgroup of sightings occurring in each time interval.

Alternatively, we could have chosen *point* to represent every separate sighting. This would have reproduced the original Journey North map. Or, we could have chosen *polygon* to aggregate the points into a regional representation of the concentration of sightings. Finally, we could have chosen *contour*, as we mentioned earlier. As we shall see, however, there are other glyphs we cannot choose for these data, including the *path* graphic Minard used and the *tree* graphic used in the popular maps of the Monarch migration. The grammar of graphics prevents us from doing so.

```
DATA: longitude, latitude = map(source("US states"))
COORD: project.stereo(dim(1, 2))
ELEMENT: line(position(smooth.quadratic.cauchy(lonp*latp)), color(date))
ELEMENT: polygon(position(longitude*latitude),
         pattern(texture.pattern.dash))
```

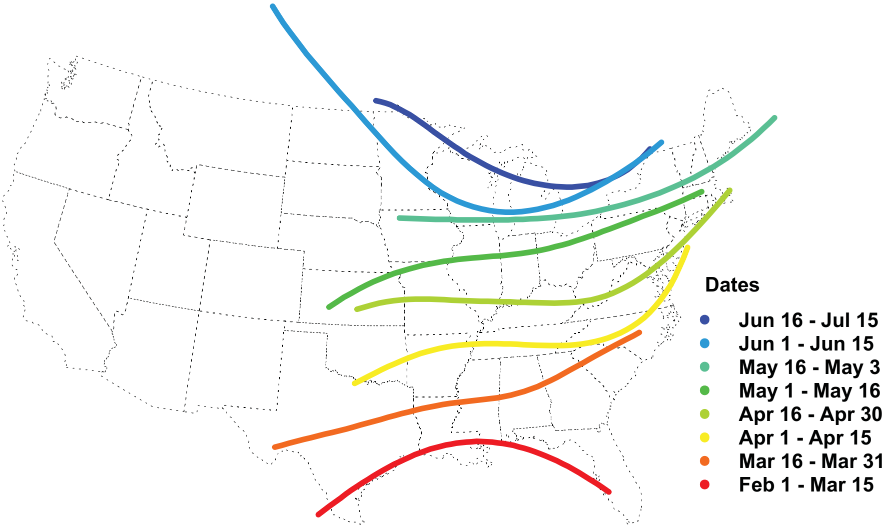


Figure 20.2 *Northerly migration of the Monarch butterfly*

20.2.3 The Meaning

The butterfly graphic shares substantial elements of meaning with Minard's. Both are migration graphics. There is an important difference, however. Minard used a *path* graphic to represent troop migration and we have used a *line* graphic to represent butterfly migration. In each case, the organization of the data constrains this usage.

We have already discussed why Minard's data require a set of *path* or *edge* graphics based on a group variable. However, we cannot use a *path* or *edge* graphic on the Monarch data. The reason the Monarch data cannot support a *path* or *edge* graphic is because there is no information to delineate the branching. To gather such information, we would have to tag butterflies and recapture them repeatedly along the northward migration route (not only at their final destination) to detect whether branching occurs. And if there were only four branches, we would expect to see large regions in central and northeastern US devoid of Monarchs. The Journey North dataset suggests that the pattern is otherwise. The sightings are continuously distributed across the eastern US in an advancing wave rather than in four widely separated branching migratory routes. Given the genesis of the data, we would have to call a tree representation not simply misleading, but meaningless.

20.3 Conclusion

Do we care whether we use a tree or contours to represent a migration? Don't both convey the same sense of movement in some direction? The answers to these questions depend on whether we intend to construct a graphic to represent *ideas* or to represent *variables*. Pictograms, ideograms, and thematic maps have their uses. Many abstract ideas are communicated especially well through figurative graphics (Herdeg, 1981; Lieberman, 1997). If we want to use graphics to represent magnitudes and relations among variables, however, we are not dealing with the grammar of ornament. We are dealing with the grammar of graphics.

Do we need a grammar of graphics for this task? Obviously, we think we do. Programs to draw graphics abound. They are built into spreadsheets, databases, and statistical packages. While many designers of these applications take pains to insure data integrity, fewer seem concerned about graphical mistakes, such as confusing pivoting with transposing. Graphical errors are subtle but serious, and remind one of Lincoln's saying that one can fool some of the people all of the time and all of the people some of the time. As with psychology, too many designers appear to consider themselves experts in graphics because they have good visual instincts. Just as psychological theory and experiments expose mistaken notions about graphical perception, however, the grammar of graphics exposes mistaken ideas about graphical structure.

20.3.1 The Grammar of Graphics

The grammar of graphics determines how algebra, geometry, aesthetics, statistics, scales, and coordinates interact. In the world of statistical graphics, we cannot confuse aesthetics with geometry by picking a tree graphic to represent a continuous flow of migrating insects across a geographic field simply because we like the impression it conveys. We can color a whole tree red to suit our design preferences, but we cannot use a tree in the first place unless we have the variables to generate one. By contrast, Minard took liberties with the data to influence the appearance of his graphic. His specification is tightly coupled to the data, however. We might disagree with his data manipulation, but we cannot call his graphic meaningless. For the Monarch data, by contrast, the variables drive us toward a representation like that in Figure 20.2.

The rules we have been invoking to make these claims constitute what we call the grammar of graphics. It is important not to confuse this grammar with the particular language we have used to describe it. The grammar of graphics is not about whether the function *directedGraph*() is a better way to implement *edge*(*position*(*link.mst*()))*.* It is not about whether *plot*(*smoother, method*) is preferable to *line*(*position*(*smooth.method*()))*.* It is not about whether a graphical system should use functions like *near*(), *enclose*(), or *intersect*() to define geometric relations. It is not about Java. It is not even about whether a different syntactical system can produce many or all of the graphics shown in this book.

We can make these statements because we did not invent this grammar or even discover it by analyzing a collection of charts and deciding how they resemble each other. We began instead by constructing definitions of data, variables, and other primitives that underlie what we call statistical charts. These definitions are embedded in the mathematical history that determined the evolution of statistical charts and maps. One cannot separate that history from the functionality that we see in printed and computer charts today, as Collins (1993) and others have made clear.

Figure 2.2 summarizes the grammar of graphics. It tells us that we cannot make a graphic without defining and using the objects shown in that figure — in the implicational ordering denoted there. Its consequences are many. Figure 2.2 epitomizes not only the rules of graphics usage, but also the domain of statistical graphics. A graphic without a concept of a variable is not a statistical graphic. A graphic without an associated coordinate system is not a statistical graphic.

Everything after Figure 2.2 involves the details that follow from this general structure. These are most evident in the examples. We cannot flip a bar chart from vertical to horizontal without making assumptions about its domain and range. We cannot log the scale of a bar chart without re-aggregating the data. A *line* cannot be used to represent Minard's path. A *tree* cannot be used to represent the butterfly data. These assertions and others we have made in the previous chapters are grounded in the grammar of graphics that follows from Figure 2.2.

20.3.2 The Language of Graphics

We now have the outline for designing a language of graphics. A scatterplot is a *point* graphic embedded in a frame. A bar chart is an *interval* graphic bound to an aggregation function embedded in a frame. A pie chart is a polar, stacked, *interval* graphic mapped on proportions. A radar chart is a *line* graphic in polar parallel coordinates. A SPLOM is a crossing of nested scatterplots in rectangular coordinates. A trellis display is a graphic faceted on crossed categorical variables in a rectangular coordinate system.

If these descriptions make sense to you, then we have come a long way since Chapter 1. Deciding whether they are simpler than the ordinary language found in handbooks on statistical graphics is a matter of evaluating the parsimony and generality of the specifications at the head of the figures in this book. This effort should help us to see why statements like "a trellis employs small multiples" are too general to be useful and statements like "a pie slice is a wedge" are too particular.

Some of these specifications require study; they were not meant for light reading. Rather, they were designed to encapsulate the definitions and algorithms all of us use — consciously or unconsciously — when we create statistical graphics. The ultimate goal of this effort is to understand how graphics work by designing a system that can understand the content of graphics, commute effortlessly between the world of tabular data and the world of charts, and respond to creative and inquisitive users interacting through a graphical interface. By thinking about how to design such a system, we take steps toward understanding what graphics mean. And consequently, we may teach a computer to understand what they mean.

We return to the beginning. Grammar gives language rules. Graphics are generated by a language. The syntax of graphics lies in their specification. The semantics of graphics lies in their data.

20.4 Sequel

There is an important detail that does not appear in the Monarch specification. It is the connection between thousands of teachers and school children, outdoors in the spring searching for butterflies, and a Web site that provides them with the opportunity to share what they found. God is in the details.

References

Abelson, R.P. (1995). *Statistics as Principled Argument.* Hillsdale, NJ: Lawrence Erlbaum.

Abiteboul, S., Fischer, P.C., and Schek, H.J. (1989). *Nested relations and complex objects in databases*. Springer–Verlag, New York.

Agrawal, R., and Srikant, J. (1995). Mining sequential patterns. *Proceedings of the 11th International Conference on Data Engineering,* 3–14.

Agrawal, R., Gupta, A., and Sarawagi, S. (1997). Modeling multidimensional databases. *Proceedings of the 13th International Conference on Data Engineering,* 232–243.

Ahlberg, C., and Shneiderman, B. (1994) Visual information seeking: Tight coupling of dynamic query filters with starfield displays. *ACM CHI 94 Conference Proceedings*, 313–317.

Aitkin, M., and Aitkin, I. (2004). Bayesian inference for factor scores. In *Contemporary Advances in Psychometrics*. Hillsdale, NJ: Lawrence Erlbaum.

Allison, T. and Cicchetti, D. (1976). Sleep in mammals: Ecological and constitutional correlates. *Science, 194*, 732–734.

Alon, U., Barkai, N., Notterman, D.A., Gish, K., Ybarra, S., Mack, D., and Levine, A. J. (1999). Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide arrays. *Proceedings of the National Academy of Sciences, 96,* 6745–6750.

Altschul, S.F., Bundschuh, R., Olsen, R., and Hwa, T. (2001). The estimation of statistical parameteers for local alignment score distributions. *Nucleic Acids Research, 29*, 251-261.

Alvey, N.G., Banfield, C.F., Baxter, R.I., Gower, J.C., Krzanowski, W.J., Lane, P.W., Leech, P.K., Nelder, J.A., Payne, R.W., Phelps, K.M., Rogers, C.E., Ross, G.J.S., Simpson, H.R., Todd, A.D., Wedderburn, R.W.M., and Wilkinson, G.N. (1977). *GENSTAT: A General Statistical Program.* The Statistics Department, Rothamsted Experimental Station. Harpenden, UK.

Anderson, E. (1935). The irises of the Gaspe Peninsula. *Bulletin of the American Iris Society, 59*, 2–5.

Anderson, J.R. (1983). *The Architecture of Cognition*. Cambridge, MA: Harvard University Press.

Anderson, J.R. (1995). *Cognitive Psychology and its Implications* (4th ed.). New York: W.H. Freeman and Company.

Andreeva, I.G., and Vartanyan, I.A. (2004). White noise masks some temporal parameters of the auditory localization of radially moving targets. *Human Physiology, 30*, 159–165

Andrews, D.F. (1972). Plots of high dimensional data. *Biometrics, 28*, 125–136.

Andrews, D.F. and Herzberg, A.M. (1985). *Data: A Collection of Problems from Many Fields for the Student and Research Worker*. New York: Springer–Verlag.

Asimov, D. (1985). The grand tour: A tool for viewing multidimensional data. *SIAM Journal on Scientific and Statistical Computing, 6*, 128–143.

Attneave, F. (1959). *Applications of Information Theory to Psychology: A Summary of Basic Concepts, Methods and Results*. New York: Holt, Rinehart and Winston.

Baker, R.J., and Nelder, J.A. (1978). *GLIM*. Oxford, UK: Numerical Algorithms Group and Royal Statistical Society.

Bahrick, H.P., Bahrick, P.O., and Wittlinger, R.P. (1975). Fifty years of memory for names and faces: A cross-sectional approach. *Journal of Experimental Psychology: General, 104*, 54–75.

Banchoff, T.F. (1996). *Beyond the Third Dimension: Geometry, Computer Graphics, and Higher Dimensions*. New York: W.H. Freeman.

Barnett, V. and Lewis, T. 1994. *Outliers in Statistical Data* (3rd ed.). John Wiley & Sons, New York.

Barraclough, G., (Ed.) (1984). *The Times Atlas of World History* (rev. ed.). London: Times Books Limited.

Beck, K. (2000). *Extreme Programming Explained: Embrace Change.* Reading, MA: Addison–Wesley.

Becker, R.A. and Cleveland, W.S. (1987). Brushing scatterplots. *Technometrics, 29*, 127–142.

Becker, R.A. and Cleveland, W.S. (1991). Take a broader view of scientific visualization. *Pixel, 2*, 42–44.

Becker, R.A., Cleveland, W.S., and Shyu, M-J (1996). The design and control of Trellis display. *Journal of Computational and Graphical Statistics, 5,* 123–155.

Bellman, R.E. (1961). *Adaptive Control Processes*. Princeton, NJ: Princeton University Press.

Beniger, J.R., Robyn, D.L. (1978). Quantitative graphics in statistics: A brief history. *The American Statistician, 32*, 1–11.

Benjamin, L.S. (1979). Use of structural analysis of social behavior (SASB) and Markov chains to study dyadic interactions. *Journal of Abnormal Psychology, 88,* 303–319.

Benjamini, Y., and Braun, H. (2002). John W. Tukey's contributions to multiple comparisons. *Annals of Statistics, 30*, 1576–1594.

Berger, J.O. (1985). *Statistical Decision Theory and Bayesian Analysis* (2nd ed.). New York: Springer–Verlag.

Berger, J.O., Boukai, B., and Wang, Y. (1997). Unified frequentist and Bayesian testing of a precise hypothesis. *Statistical Science, 12*, 133–160.

Berlin, B., and Kay, P. (1969). *Basic Color Terms: Their Universality and Evolution.* Berkeley: University of California Press.

Bertin, J. (1967). *Sémiologie Graphique*. Paris: Editions Gauthier–Villars. English translation by W.J. Berg as *Semiology of Graphics*, Madison, WI: University of Wisconsin Press, 1983.

Bertin, J. (1977). *La Graphique et le Traitement Graphique de l'Information.* Paris: Flammarion. English translation by W.J. Berg and P. Scott as *Graphics and Graphic Information Processing*. Berlin: Walter de Gruyter & Co., 1981.

Besag, J. (1986). On the statistical analysis of dirty pictures. *Journal of the Royal Statistical Society, Series B, 3*, 259–302.

Beshers, C., and Feiner, S. (1993). AutoVisual: Rule-based design of interactive multivariate visualizations. *IEEE Computer Graphics and Applications, 13*, 41–49.

Biederman, I. (1972). Perceiving real-world scenes. *Science, 177*, 77–80.

Biederman, I. (1981). On the semantics of a glance at a scene. In M. Kubovy and J.R. Pomerantz (Eds.), *Perceptual Organization* (pp. 213–253). Hillsdale, NJ: Lawrence Erlbaum.

Bird, R., and De Moor, O. (1996). *Algebra of Programming*. Upper Saddle River, NJ: Prentice–Hall.

Bishop, G., Fuchs, H., McMillan, L., Scher-Zagier, E.J. (1994). *Frameless rendering: Double buffering considered harmful*. Proceedings of SIGGRAPH '94, 175–176.

Blattner, M., Sumikawa, D. and Greenberg, R. (1989) Earcons and icons: Their structure and common design principles. *Human-Computer Interaction, 4*, 11–44.

Bly, S.A. (1983). Interactive tools for data exploration. *Computer Science and Statistics: Proceedings of the 15th Symposium on the Interface*, 255–259.

Bly, S.A., Frysinger, S.P., Lunney, D., Mansur, D.L., Mezrich, J.J., and Morrison, R.C. (1985). Communicating with sound. *Human Factors in Computing Systems: CHI 85 Conference Proceedings*, 115–119.

Bock, R.D. (1975). *Multivariate Statistical Methods in Behavioral Research.* New York: McGraw–Hill.

Booch, G. (1994). *Object-oriented Analysis and Design* (2nd ed.). Redwood City, CA: Benjamin/Cummings Publishing.

Bookstein, F. L. (1991). *Morphometric Tools for Landmark Data: Geometry and Biology*. New York: Cambridge University Press.

Borg, I. and Groenen, P. (1997). *Modern Multidimensional Scaling: Theory and Applications*. New York: Springer–Verlag.

Borg, I. and Staufenbiel, T. (1992). The performance of snow flakes, suns, and factorial suns in the graphical representation of multivariate data. *Multivariate Behavioral Research, 27*, 43–55.

Boring, E.G. (1950). *A History of Experimental Psychology*. New York: Appleton–Century–Crofts.

Bornstein, M.H., Kessen, W., and Weiskopf, S. (1976). Color vision and hue categorization in young human infants. *Journal of Experimental Psychology: Human Perception and Performance, 2*, 115–129.

Bornstein, M.H. (1987). Perceptual categories in vision and audition. In S. Harnad (Ed.), *Categorical Perception: The Groundwork of Cognition (*pp. 287–300). Cambridge: Cambridge University Press.

Bowman, C.D. (2005). Difference and Democracy Survey. Pew Center on Religion and Democracy, University of Virginia.

Boyer, R., and Savageau, D. (1996). *Places Rated Almanac*. Chicago, IL: Rand McNally.

Box, G.E.P. and Cox, D.R. (1964). An analysis of transformations. *Journal of the Royal Statistical Society, B, 26*, 211–252.

Box, G.E.P., and Jenkins, G.M. (1976). *Time series analysis: Forecasting and control* (rev. ed.). Oakland, CA: Holden–Day.

Box, G.E.P., and Tiao, G.C. (1973). *Bayesian Inference in Statistical Analysis*. New York: John Wiley & Sons.

Boynton, R.M. (1988). Color vision. *Annual Review of Psychology, 39*, 69–100.

Bregman, A. S. (1990). *Auditory Scene Analysis: The Perceptual Organization of Sound*. Cambridge, MA: MIT Press.

Breiger, R. L., Boorman, S.A. and Arabie, P. (1975). An algorithm for clustering relational data with applications to social network analysis and comparison with multidimensional scaling. *Journal of Mathematical Psychology,12*, 328–383.

Breiman, L., Friedman, J.H., Olshen, R.A., and Stone, C.J. (1984). *Classification and Regression Trees*. Belmont, CA: Wadsworth.

Brewer, C.A. (1994). Color use guidelines for mapping and visualization. In A. M. MacEachren and D.R.F. Taylor (Eds.), *Visualization in Modern Cartography* (pp. 123–147). Oxford: Pergamon Press.

Brewer, C.A. (1996). Guidelines for selecting colors for diverging schemes on maps. *The Cartographic Journal, 33*, 79–86.

Brewer, C.A., MacEachren, A.M., Pickle, L.W., and Herrmann, D. (1997). Mapping mortality: Evaluating color schemes for choropleth maps. *Annals of the Association of American Geographers, 87*, 411–438.

Brillinger, D.R. and Irizarry, R.A. (1998). An investigation of the second- and higher- order spectra of music. *Signal Processing, 65*.161–178.

Brodlie, K.W. (Ed.) (1980). *Mathematical Methods in Computer Graphics and Design*. London: Academic Press.

Brodlie, K.W. (1993). A classification scheme for scientific visualization. In R.A. Earnshaw and D. Watson (Eds.), *Animation and Scientific Visualization: Tools and Applicaitons* (pp. 125–140). New York: Academic Press.

Brooks, R. (1987). Planning is just a way of avoiding figuring out what to do next. Technical Report 303, MIT AI Labs.

Brown, G.W. (1986). Inverse regression. In S. Kotz and N.L. Johnson (Eds.), *Encyclopedia of Statistical Sciences, Vol. 7* (pp. 694–696). New York: John Wiley & Sons.

Browne, M.W. (1977). The analysis of patterned correlation matrices by generalized least squares. *British Journal of Mathematical and Statistical Psychology 30*, 113–124.

Browne, M.W. (1992). Circumplex models for correlation matrices. *Psychometrika, 57,* 469–497.

Bruckner, L.A. (1978). On Chernoff faces. In P.C.C. Wang (Ed.), *Graphical Representation of Multivariate Data* (pp. 93–121). New York: Academic Press.

Buja, A., and Asimov, D. (1986). Grand tour methods: an outline. *Computer Science and Statistics: Proceedings of the 18th Symposium on the Interface*, 171–174.

Buja, A., Cook, D., and Swayne, D.F. (1996). Interactive high-dimensional data visualization. *Journal of Computational and Graphical Statistics, 5*, 78–99.

Burns, E.M., and Ward, W.D. (1978). Categorical perception — phenomenon or epiphenomenon: Evidence from experiments in the perception of melodic musical intervals. *Journal of the Acoustical Society of America, 63*, 456–468.

Butler, D.M. and Pendley, M.H. (1989). A visualization model based on the mathematics of fibre bundles. *Computers in Physics, 3*, 45–51.

Buttenfield, B.P. (2000). Mapping ecological uncertainty. In C.T. Hunsaker, M.F. Goodchild, M.A. Friedl, and T.J. Case (Eds.) *Spatial Uncertainty in Ecology* (pp. 116–132). New York: Springer–Verlag.

Campbell, D.T., and Fiske, D.W. (1959). Convergent and discriminant validation by the multitrait-multimethod matrix. *Psychological Bulletin, 56,* 81-105.

Carlis, J.V., and Konstan, J.A. (1998). Interactive visualization of serial periodic data. *Proceedings of the 11th Annual ACM Symposium on User Interface Software and Technology* (*UIST*)*,* 29–38.

Carpendale, M.S.T., Cowperthwaite, D.J., and Fracchia, F.D. (1997). Extending distortion viewing from 2D to 3D. *IEEE Computer Graphics and Applications, 17*, 42–51.

Carr, D.B. (1994). Converting Tables to Plots. Technical Report 101, Center for Computational Statistics, George Mason University.

Carr, D.B. (2002). 2-D and 3-D coordinates for m-mers and dynamic graphics. *Computing Science and Statistics: Proceedings of the 34th Symposium on the Interface.*

Carr, D.B. and Nicholson, W.L. (1988). EXPLOR4: A program for exploring four-dimensional data using stereo-ray glyphs, dimensional constraints, rotation, and masking. In W.S. Cleveland and M.E. McGill, (Eds.), *Dynamic Graphics for Statistics* (pp. 309–329). Belmont, CA: Wadsworth.

Carr, D.B., Littlefield, R.J., Nicholson, W.L., and Littlefield, J.S. (1987). Scatterplot matrix techniques for large N. *Journal of the American Statistical Association, 82*, 424–436.

Carr, D.B., Olsen, A.R., and White, D. (1992). Hexagon mosaic maps for display of univariate and bivariate geographical data. *Cartography and Geographic Information Systems, 19*, 228–236.

Carr, D.B., Olsen, A.R., Courbois, J-Y.P., Pierson, S.M., and Carr, D.A. (1998). Linked micromap plots: Named and described. *Statistical Computing & Statistical Graphics Newsletter, 9,* 24–32.

Carr, D.B., Olsen, A.R., Pierson, S.M, and Courbois, J.P. (1999). Boxplot variations in a spatial context: An Omernik ecoregion and weather example. *Statistical Computing & Statistical Graphics Newsletter, 9,* 4–13.

Carr, D.B., Wegman, E.J., and Luo, Q. (1997). ExplorN: Design Considerations Past and Present. Technical Report 137, Center for Computational Statistics, George Mason University.

Carroll, J.D. and Chang, J.J. (1976). Spatial, non-spatial and hybrid models for scaling. *Psychometrika, 41*, 439–463.

Carswell, C.M. (1992). Choosing specifiers: An evaluation of the basic tasks model of graphical perception. *Human Factors, 4*, 535–554.

Casdagli, M.C. (1997). Recurrence plots revisited. *Physica D, 108*, 12–44.

Chambers, J.M. (1977). *Computational Methods for Data Analysis*. New York: John Wiley & Sons.

Chambers, J.M., Cleveland, W.S., Kleiner, B., and Tukey, P.A. (1983). *Graphical Methods for Data Analysis.* Monterey, CA: Wadsworth.

Chandler, D.G. (1966). *The Campaigns of Napoleon*. New York: The Macmillan Company.

Chartrand, J. M. (1997). National sample survey. Unpublished raw data.

Chelton, D.B., Mestas-Nunez, A.M., and Freilich, M.H. (1990). Global wind stress and Sverdrup circulation from the Seasat Scatterometer. *Journal of Physical Oceanography, 20*, 1175–1205.

Chen, H. (2003). Compound brushing. *Proceedings of the IEEE Symposium on Information Visualization.*

Chen, M., Mountford, S. J., and Sellen, A. (1988). A study in interactive 3-D rotation using 2-D control devices. *Computer Graphics, 22*, 121–129.

Chernoff, H. (1973). The use of faces to represent points in k-dimensional space graphically. *Journal of the American Statistical Association, 68*, 361–368.

Chernoff, H. (1975). Effect on classification error of random permutations of features in representing multivariate data by faces. *Journal of the American Statistical Association, 70*, 548–554.

Chi, E.H., Konstan, J., Barry, P., and Riedl, J. (1997). A spreadsheet approach to information visualization. *Proceedings of the 10th Annual ACM Symposium on User Interface Software and Technology* (*UIST*)*,* 79–80.

Chomsky, N. (1956). Three models for the description of language. *IRE Transactions on Information Theory, 2*, 113–124.

Christensen, J., Marks, J., and Shieber, S. (1995). An empirical study of algorithms for point-feature label placement. *ACM Transactions on Graphics, 14*, 203–232.

Clarkson, D.B. and Jennrich, R.I. (1988). Quartic rotation criteria and algorithms. *Psychometrika, 53*, 251–259.

Cleveland, W.S. (1984). Graphs in scientific publications. *The American Statistician, 38*, 19–26.

Cleveland, W.S. (1985). *The Elements of Graphing Data*. Summit, NJ: Hobart Press.

Cleveland, W.S. (Ed.) (1988). *The Collected Works of John W. Tukey, Vol. 5: Graphics*. New York: Chapman & Hall.

Cleveland, W.S. (1993). A model for studying display methods of statistical graphics (with discussion). *Journal of Computational and Graphical Statistics, 2*, 323–343.

Cleveland, W.S. (1995). *Visualizing Data*. Summit, NJ: Hobart Press.

Cleveland, W.S., and Devlin, S. (1988). Locally weighted regression analysis by local fitting. *Journal of the American Statistical Association, 83*, 596–640.

Cleveland, W.S., Diaconis, P., and McGill, R. (1982). Variables on scatterplots look more highly correlated when the scales are increased. *Science, 216*, 1138–1141.

Cleveland, W.S., Harris, C.S., and McGill, R. (1981). Judgments of circle sizes on statistical maps. Unpublished paper. Murray Hill, NJ: Bell Laboratories.

Cleveland, W.S. and McGill, M.E. (1988). *Dynamic Graphics for Statistics*. Belmont, CA: Wadsworth.

Cleveland, W.S., McGill, M.E., and McGill, R. (1988). The shape parameter of a two-variable graph. *Journal of the American Statistical Association, 83,* 289–300.

Cleveland, W.S. and McGill, R. (1984a). Graphical perception: Theory, experimentation, and application to the development of graphical methods. *Journal of the American Statistical Association, 79,* 531–554.

Cleveland, W.S. and McGill, R. (1984b). The many faces of a scatterplot. *Journal of the American Statistical Association, 79,* 807–822.

Codd, E.G. (1970). A relational model of data for large shared data banks. *Communications of the ACM, 13*, 377–387.

Collins, B.M. (1993). Data visualization — has it all been seen before? In R.A. Earnshaw and D. Watson (Eds.), *Animation and Scientific Visualization: Tools and Applications* (pp. 3–28). New York: Academic Press.

Conway, J.H., and Guy, R.K. (1996). *The Book of Numbers*. New York: Springer–Verlag.

Cook, R.D. and Weisberg, S. (1994). *An Introduction to Regression Graphics*. New York: John Wiley & Sons.

Coombs, C.H. (1964). *A Theory of Data*. New York: John Wiley & Sons.

Coombs, C.H. and Avrunin, G.S. (1977). Single-peaked functions and the theory of preference. *Psychological Review, 84*, 216–230.

Coombs, C.H., Dawes, R.M., and Tversky, A. (1970). *Mathematical Psychology: An Elementary Introduction*. Englewood Cliffs, NJ: Prentice–Hall.

Copi, I.M. (1967). *Symbolic Logic* (3rd ed.). New York: Macmillan Co.

Coren, S., and Girgus, J.S. (1978). *Seeing is Deceiving: The Psychology of Visual Illusions*. Hillsdale, NJ: Lawrence Erlbaum.

Cornell, J.A. (1990). *Experiments with Mixtures: Designs, Models, and the Analysis of Mixture Data*. New York: John Wiley & Sons.

Corter, J.E. (1982). ADDTREE/P: A PASCAL program for fitting additive trees based on Sattath and Tversky's ADDTREE algorithm. *Behavior Research Methods and Instrumentation, 14*, 353–354.

Cox, P.R. (1986). Population pyramid. In S. Kotz and N.L. Johnson (Eds.), *Encyclopedia of Statistical Sciences, Vol 7.* (pp. 113–116). New York: John Wiley & Sons.

Cox, T.F., and Cox, M.A.A. (1994). *Multidimensional Scaling*. London: Chapman & Hall.

Cramer, C.Y. (1956). Extension of multiple range tests to group means with unequal numbers of replications. *Biometrics, 12*, 309–310.

Cressie, N.A.C. (1991). *Statistics for Spatial Data*. New York: John Wiley & Sons.

Cruz-Neira, C., Sandin, D.J., and DeFanti, T.A. (1993). Surround-Screen projection-based virtual reality: The design and implementation of the CAVE. *ACM Siggraph '93 Conference Proceedings*, 135–142.

Dallal, G., and Finseth, K. (1977). Double dual histograms. *The American Statistician, 31*, 39–41.

Daly, C., Neilson, R.P. and Phillips, D.L. (1994). A statistical-topographic model for mapping climatological precipitation over mountainous terrain. *Journal of Applied Meteorology, 33*, 140–158.

Datar, M., Gionis, A., Indyk, P., and Motwani, R. (2002) Maintaining stream statistics over sliding windows. Annual ACM–SIAM Symposium on Discrete Algorithms, San Francisco.

Date, C.J. (1990). What is a domain? In C.J. Date (Ed.), *Relational Database Writings 1985–1989*. Reading, MA: Addison–Wesley, 213–313.

Date, C.J. (1995). *An Introduction to Database Systems* (6th ed.). Reading: MA: Addison–Wesley.

Date, C.J., and Darwen, H. (1992). Relation-valued attributes. In C.J. Date and H. Darwen (Eds), *Relational Database Writings 1989–1991* (pp. 75–98). Reading, MA: Addison–Wesley.

Davis. J.A., Smith, T.W., and Marsden, P.V. (1993). *The General Social Survey*. Chicago: National Opinion Research Center.

Dawson, R.J.M. (1995). The "unusual episode" data revisited. *Journal of Statistics Education, 3*(*3*).

Defays, D. (1978). A short note on a method of seriation. *British Journal of Mathematical and Statistical Psychology, 31*, 49–53.

DeGroot, M.H. (1970). *Optimal Statistical Decisions*. New York: McGraw–Hill.

Dempster, A.P., Laird, N.M., and Rubin, D.B. (1977). Maximum likelihood from incomplete data via the EM algorithm (with discussion). *Journal of the Royal Statistical Society, B, 39*, 1–38.

Dennis, J.E. Jr., and Schnabel, R.B. (1983). *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Englewood Cliffs, NJ: Prentice–Hall.

Deregowski, J.B., and McGeorge, P. (1998). Perceived similarity of shapes is an asymmetrical relationship: A study of typical contours. *Perception, 27*, 35–46.

Derrick, W.R. (1984). *Complex Analysis and Applications* (2nd ed.). Belmont, CA: Wadsworth.

Derthick, M., Kolojejchick, J., and Roth, S.F. (1997). An interactive visual query environment for exploring data. *Proceedings of the 10th Annual ACM Symposium on User Interface Software and Technology* (*UIST*), 189–198.

De Soete, G. (1986). A perceptual study of the Flury-Riedwyl faces for graphically displaying multivariate data. *International Journal of Man-Machine Studies, 25*, 549–555.

Deutsch, D. (1996). The perception of auditory patterns. In W. Prinz and B. Bridgeman (Eds.), *Handbook of Perception and Action, Vol. 1* (pp. 253–296). London: Academic Press.

deValois, R.L., and Jacobs, G.H. (1968). Primate color vision. *Science, 162*, 533–540.

Di Battista, G., Eades, P., Tamassia, R., and Tollis, I.G. (1999). *Graph Drawing: Algorithms for the Visualization of Graphs*. Upper Saddle River, NJ: Prentice Hall.

Dierckx, P. (1993). *Curve and Surface Fitting with Splines*. Oxford: Clarendon Press.

Dirschedl, P. (1991). Klassifikationsbaume — Grundlagen und Neuerungen. In W. Flischer, M. Nagel, and R. Ostermann (Eds.), *Interaktive Datenalyse mit ISP* (pp. 15–30). Essen: Westart Verlag, Djurcilov, S., Kim, K., Lermusiaux, P., and Pang, A. (2001). Volume rendering data with uncertainty information. In D. Ebert, J. M. Favre, and R. Peikert, (Eds.), *Data Visualization 2001* (pp. 243–252). New York: Springer Verlag

Doane, D.P. (1976). Aesthetic frequency classifications. *The American Statistician, 30,* 181–183.

Doerr, A., and Levasseur, K. (1985). *Applied Discrete Structures for Computer Science*. Chicago: Science Research Associates.

Donoho, A.W., Donoho, D.L., and Gasko, M. (1988). MacSpin: Dynamic graphics on a desktop computer. In W.S. Cleveland and M.E. McGill, (Eds.), *Dynamic Graphics for Statistics* (pp. 331–351). Belmont, CA: Wadsworth.

Draper, N.R., and Smith, H. (1981). *Applied Regression Analysis* (2nd ed.). New York: John Wiley & Sons.

Duda, R.O., Hart, P.E., and Stork, D.G. (2001). *Pattern Classification* (2nd ed.). New York: John Wiley & Sons.

Dunn, R. (1987). Variable-width framed rectangle charts for statistical mapping. *The American Statistician, 41*, 153–156.

DuMouchel, W. (1990). Bayesian Metaänalysis. In D.A. Berry (Ed.). *Statistical Methodology in the Pharmaceutical Sciences* (pp. 509–529). New York: Marcel Dekker, Inc.

Eames, C., and Eames, R. (1996) *Powers of Time* (motion picture). Santa Monica CA: Eames Office.

Earnshaw, R.A., and Watson, D. (1993). *Animation and Scientific Visualization: Tools and Applications.* New York: Academic Press.

Eckmann, J.P., Kamphorst, S.O., and Ruelle, D. (1987). Recurrence plots of dynamical systems. *Europhysics Letters, 4*, 973–977

Eddy, D.M. (1982). Probabilistic reasoning in clinical medicine: Problems and opportunities. In D. Kahneman, P. Slovic, and A. Tversky (Eds.). *Judgment under uncertainty: Heuristics and biases* (pp. 249–267). Cambridge: Cambridge University Press.

Eddy, W.F., and Oue, S. (1995). Dynamic three-dimensional display of U.S. air traffic, *Journal of Computational and Graphical Statistics, 4*, 261–280.

Edelman S. (1995). Representation, similarity, and the chorus of prototypes. *Minds and Machines, 5*, 45–68.

Edgar, G.A. (1990). *Measure, Topology, and Fractal Geometry*. New York: Springer Verlag.

Edwards, W. (1968). Conservatism in human information processing. In B. Kleinmuntz (Ed.), *Formal Representation of Human Judgment* (pp. 17–52). New York: John Wiley & Sons.

Edwards, W., Lindman, H., and Savage, L.J. (1963). Bayesian statistical inference for psychological research. *Psychological Review, 70*, 193–242.

Efron, B. (1979). Bootstrap methods: Another look at the jackknife. *Annals of Statistics, 7*, 1–26.

Efron, B., and Tibshirani, R.J. (1993). *An Introduction to the Bootstrap*. New York: Chapman & Hall.

Egenhofer, M.J., Herring, J.R., Smith, T., and Park, K.K. (1991). A framework for the definition of topological relationships and an algebraic approach to spatial reasoning within this framework. Technical Paper 91–7, National Center for Geographic Information and Analysis/NCGIA.

Eick, S.G., Steffen, J.L., and Sumner, E.E. (1992). SeeSoft — A Tool for Visualizing Software. *IEEE Transactions on Software Engineering, 18*, 957–968.

Eimas, P. (1974). Auditory and phonetic coding of the cues for speech: Discrimination of the [r–l] distinction by young infants. *Perception and Psychophysics, 18*, 341–347.

Ekman, G. (1954). Dimensions of color vision. *Journal of Psychology, 38*, 467–474.

Ekman, G. (1964). Is the power law a special case of Fechner's law? *Perceptual and Motor Skills, 19*, 730.

Ekman, P. (1984). Expression and the nature of emotion. In K. Scherer and P. Ekman (Eds.), *Approaches to Emotion* (pp. 319–343). Hillsdale, NJ: Lawrence Erlbaum.

Ekman, P. and Friesen, W. (1978). *Facial Action Coding System: A Technique for the Measurement of Facial Movement*. Palo Alto, CA: Consulting Psychologists Press.

Ekman, P., Friesen, W.V., and O'Sullivan, M. (1988). Smiles when lying. *Journal of Personality and Social Psychology, 54*, 414–420.

Emmer, M. (Ed.) (1995). *The Visual Mind: Art and Mathematics*. Cambridge, MA: MIT Press.

Epp, S.S. (1990). *Discrete Mathematics with Applications*. Belmont, CA: Wadsworth.

Estes, W.K. (1994). *Classification and Cognition.* Oxford: Oxford University Press.

Falmagne, J-C. (1985). *Elements of Psychophysical Theory*. Oxford: Oxford University Press.

Fan, J., and Gijbels, I. (1996). *Local Polynomial Modelling and Its Applications*. London: Chapman & Hall.

Feamster, N.G. (2001). Adaptive delivery of real-time streaming video. Master's thesis, Department of Electrical Engineering and Computer Science, MIT.

Fechner, G.T. (1860). *Elemente der Psychophysik*. Leipzig: Breitkopf & Härtel. English translation of Vol. 1 by H.E. Adler, New York: Holt, Rinehart and Winston.

Feigenbaum, J., Kannan, S., and Zhang, J. (2004). Computing diameter in the streaming and sliding-window models. *Algorithmica*, in press.

Ferster, C.B., and Skinner, B.F. (1957). *Schedules of Reinforcement.* New York: Appleton–Century–Crofts.

de Fezensac, R.A.P.J. (1849). *Journal de la campagne de Russie en 1812*. Paris. English translation by L.B. Kennett as *The Russian Campaign, 1812*, Athens, GA: The University of Georgia Press.

Fienberg, S. (1979). Graphical methods in statistics. *The American Statistician, 33*, 165–178.

Feiner, S. and Beshers, C. (1990). Worlds within worlds: Metaphors for exploring n-dimensional virtual worlds. *Proceedings of the ACM UIST '90*. 76–83.

Fisher, L. and Van Ness, J. (1971). Admissible cluster procedures. *Biometrika*, *58*, 91–104.

Fisher, P.F. (1994). Animation and sound for the visualization of uncertain spatial information. In H.H. Hearnshaw and D.J. Unwin (Eds.), *Visualization in geographical information systems* (pp. 181–185). New York: John Wiley & Sons.

Fisher, R.A. (1915). Frequency distribution of the values of the correlation coefficient in samples from an indefinitely large population. *Biometrika, 10*, 507–521.

Fisher, R.A. (1925). *Statistical Methods for Research Workers*. London: Oliver and Boyd.

Fisher, R.A. (1935). *The Design of Experiments*. Edinburgh: Oliver and Boyd.

Fisherkeller, M.A., Friedman, J.H., and Tukey, J.W. (1974). PRIM9: An interactive multidimensional data display and analysis system. SLAC–PUB–1408. Reprinted in W.S. Cleveland (Ed.), *The Collected Works of John W. Tukey, Vol 5* (pp. 307-327). New York: Chapman & Hall.

Flaherty, M. (1999). *A watched pot: How we experience time*. New York: New York University Press.

Flury, B., and Riedwyl, H. (1981). Graphical representation of multivariate data by means of asymmetrical faces. *Journal of the American Statistical Association, 76*, 757–765.

Foley, J.D., Van Dam, A., Feiner, S.K., and Hughes, J.F. (1993). *Introduction to Computer Graphics*. Reading, MA: Addison–Wesley.

Fortner, B. (1995). *The Data Handbook: A Guide to Understanding the Organization and Visualization of Technical Data* (2nd ed.). New York: Springer–Verlag.

Fowler, M., and Scott, K. (1997). *UML Distilled: Applying the Standard Object Modling Language*. Reading, MS: Addison–Wesley.

Freedman, D. and Diaconis, P. (1981). On the histogram as a density estimator: L_2 theory. *Zeitschrift für Wahrscheinlichkeitstheorie und verwandt Gebiete, 57*, 453–476.

Friedman, J., and Stuetzle, W. (1981). Projection pursuit regression. *Journal of the American Statistical Association, 76*, 817–823.

Friedman, J. (1987). Exploratory projection pursuit. *Journal of the American Statistical Association, 82*, 249–266.

Friedman, J. (1997). Data mining and statistics: What's the connection? *Computing Science and Statistics: Proceedings of the 29th Symposium on the Interface*, 3–9.

Friedman, J., and Stuetzle, W. (2002). John W. Tukey's work on interactive graphics. *Annals of Statistics, 30*, 1629–1639.

Friendly, M. (1994). Mosaic displays for n-way contingency tables. *Journal of the American Statistical Association, 89*, 190–200.

Friendly, M. (2002). A brief history of the mosaic display. *Journal of Computational and Graphical Statistics, 11*, 89–107.

Fruchterman, T.M.J., and Reingold, E.M. (1991). Graph drawing by force-directed placement. *Software – Practice and Experience, 21*. 1129–1164.

Funkhouser, H.G. (1937). Historical development of the graphical representation of statistical data. *Osiris, 3*, 269–404.

Furnas, G.W. (1986). Generalized fisheye views. *Human Factors in Computing Systems: CHI 86 Conference Proceedings*, 16–23.

Gabriel, K.R. (1971). The biplot graphical display of matrices with application to principal component analysis. *Biometrika, 58*, 45–467.

Gabriel, K.R. (1995). Biplot display of multivariate categorical data, with comments on multiple correspondence analysis. In W.J. Krzanowski (Ed.), *Recent Advances in Descriptive Multivariate Analysis* (pp. 190–225). Oxford: Clarendon Press.

Gamma, E., Helm, R., Johnson, R., Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison–Wesley.

Gantt, H.L. (1903). A graphical daily balance in manufacture. *ASME Transactions, 24*, 1322–1336.

Gao, J. and Cai, H. (2000). On the structures and quantification of recurrence plots. *Physics Letters, Series A, 270*, 75–87.

Garner, W.R. (1970). The simulus in information processing. *American Psychologist, 25*, 350–358.

Garner, W.R. (1974). *The Processing of Information and Structure*. Hillsdale, NJ: Lawrence Erlbaum.

Garner, W.R. (1981). The analysis of unanalyzed perceptions. In M. Kubovy and J.R. Pomerantz (Eds.), *Perceptual Organization* (pp. 119–139). Hillsdale, NJ: Lawrence Erlbaum.

Garner, W.R. and Felfoldy, G.L. (1970). Integrality of stimulus dimensions in various types of information processing. *Cognitive Psychology, 1,* 225–241.

Gastner, M.T., and Newman, M.E.J. (2004). Diffusion-based method for producing density-equalizing maps. *Proceedings of the National Academy of Sciences, 101*, 7499–7504.

Gelman, A., Carlin, J.B., Stern, H.S., and Rubin, D.B. (2003). *Bayesian Data Analysis* (2nd ed.). New York: Chapman & Hall/CRC.

Gibson, E.J. (1991). *An Odyssey in Learning and Perception.* Cambridge, MA: MIT Press.

Gibson, J.J. (1966). *The Senses Considered as Perceptual Systems*. Boston: Houghton Mifflin.

Gibson, J. J. (1975). Events are perceivable but time is not. In J. T. Fraser and N. Lawrence (Eds.), *The Study of Time II* (pp. 295–301). New York: Springer–Verlag.

Gibson, J.J. (1979). *The Ecological Approach to Visual Perception*. Boston: Houghton Mifflin.

Gillies, G.T. (1997). The Newtonian gravitational constant: recent measurements and related studies. *Reports on Progress in Physics, 60*, 151–225.

Glenberg, A.M. (1997). What memory is for. *Behavioral and Brain Sciences, 20,* 1–19.

Glymour, C., Madigan, D., Pregibon, D., and Smyth, P. (1996). Statistical inference and data mining. *Communications of the ACM, 39*, 35–41.

Goldstone, R. (1994). Influences of categorization on perceptual discrimination. *Journal of Experimental Psychology: General, 123*, 178–200.

Gomes, J., and Costa, B. (1998). *Warping and Morphing of Graphical Objects*. New York: Morgan Kaufmann.

Gonnelli, S., Cepollaro, C., Montagnani, A., Monaci, G., Campagna, M.S., Franci, M.B., and Gennari, C. (1996). Bone alkaline phosphatase measured with a new immunoradiometric assay in patients with metabolic bone diseases. *European Journal of Clinical Investigation, 26*, 391–396.

Gonzalez, R.C., and Wintz, P. (1977). *Digital Image Processing*. Reading, MA: Addison–Wesley.

Goodman, L.A. (1978). *Analyzing Qualitative/Categorical Data*. Cambridge, MA: Abt Books.

Gould, S.J. (1980). *The Panda's Thumb: More Reflections in Natural History*. New York: W.W. Norton & Company.

Gould, S.J. (1996). *Full House: The Spread of Excellence from Plato to Darwin*. New York: Harmony Books.

Gower, J.C. (1995). A general theory of biplots. In W.J. Krzanowski, Ed., *Recent Advances in Descriptive Multivariate Analysis* (pp. 283–303). Oxford: Clarendon Press.

Gower, J.C., and Digby, P.G.N. (1995). Expressing complex relationships in two dimensions. In Vic Barnett (Ed.), *Interpreting Multivariate Data* (pp. 83–118). Chichester: John Wiley & Sons.

Green, P.J. and Silverman, B.W. (1994). *Nonparametric Rression and Generalized Linear Models: A Roughness Penalty Approach*. London: Chapman and Hall.

Gregory, R.L. (1978). *Eye and Brain* (3rd ed.). New York: McGraw–Hill.

Gregson, R.A.M. (1988). *Nonlinear Psychophysical Dynamics*. Hillsdale, NJ: Lawrence Erlbaum.

Grigoryan, G., and Rheingans, P. (2004). Point-based probabilistic surfaces to show surface uncertainty. *IEEE Transactions on Visualization and Computer Graphics , 10*, 564–573.

Grimm. S., Widmann, A., and Schroger, E. (2004). Differential processing of duration changes within short and long sounds in humans. *Neuroscience Letters, 12;356*, 83–86.

Grondin, S. (2001). From physical time to the first and second moments of psychological time. *Psychological Bulletin, 127*, 22–44.

Gruvaeus, G., and Wainer, H. (1972). Two additions to hierarchical cluster analysis. *British Journal of Mathematical and Statistical Psychology, 25*, 200–206.

Guha, S., Mishra, N., Motwani, R., and O'Callaghan, L. (2000). Clustering data streams. In *Proceedings of the Annual Symposium on Foundations on Computer Science (FOCS 2000).*

Gusfield, D. (1997). *Algorithms on Strings, Trees, and Sequences*. Cambridge: Cambridge University Press.

Guttman, L. (1954). A new approach to factor analysis: The radex. In P.F. Lazarsfeld (Ed.), *Mathematical Thinking in the Social Sciences*. New York: Free Press.

Guttman, L. (1955) The determinacy of factor score matrices with implications for five other basic problems of common-factor theory. *British Journal of Statistical Psychology, 8*, 65–81.

Guttman, L. (1971). Measurement as structural theory. *Psychometrika, 36*, 329–347.

Guttman, L. (1977). What is not what in statistics. *The Statistician, 26*, 81–107.

Gyssens, M., Lakshmanan, L. V. S., and Subramanian, I. N. (1996). Tables as a paradigm for querying and restructuring (extended abstract). *Proceedings of the fifteenth ACM SIGACT–SIGMOD–SIGART symposium on principles of database systems*, 93–103.

Haber, R.B. and McNabb, D.A. (1990). Visualization idioms: A conceptual model for scientific visualization systems. In G.M. Nielson, B.D. Shriver, and L.J. Rosenblum (Eds.), *Visualization in Scientific Computing* (pp. 74–93). Los Alamitos, CA: IEEE Computer Society Press.

Haber, R.N. and Wilkinson, L. (1982). Perceptual components of computer displays. *IEEE Computer Graphics and Applications, 2*, 23–35.

Haith, M.M. (1980). *Rules that Babies Look by: The Organization of Newborn Visual Activity*. Hillsdale, NJ: Lawrence Erlbaum.

Hand, D. (1996). Statistics and the theory of measurement. *Journal of the Royal Statistical Society, A, 159*, 486–487.

Hand, D.J., Mannila, H., and Smyth, P. (2001). *Principles of Data Mining (Adaptive Computation and Machine Learning).* Cambridge, MA: MIT Press.

Harary, F. (1969). *Graph Theory*. Reading, MA: Addison–Wesley.

Härdle, W. (1990). *Applied Nonparametric Regression*. Cambridge: Cambridge University Press.

Harman, H.H. (1976). *Modern Factor Analysis,* (3rd ed.). Chicago: University of Chicago Press.

Harnad, S. (1987). *Categorical Perception*. Cambridge: Cambridge University Press.

Harris, J.E. (1987). Who should profit from cigarettes? *The New York Times*. Sunday, March 15, Section C, 3.

Hartigan, J.A. (1972). Direct clustering of a data matrix. *Journal of the American Statistical Association, 67*, 123–129.

Hartigan, J.A.(1975a). *Clustering Algorithms*. New York: John Wiley & Sons.

Hartigan, J.A. (1975b). Printer graphics for clustering. *Journal of Statistical Computation and Simulation, 4,* 187–213.

Hartigan, J.A. (1983). *Bayes Theory*. New York: Springer–Verlag.

Hartigan, J.A. and Kleiner, B. (1981). Mosaics for contingency tables. In *Computer Science and Statistics: Proceedings of the 13th Symposium on the Interface*, 268–273.

Hartigan, J.A. and Kleiner, B. (1984). A mosaic of television ratings. *The American Statistician, 38*, 32–35.

Hastie, T. and Stuetzle, W. (1989). Principal curves. *Journal of the American Statistical Association, 84*, 502–516.

Hastie, T. and Tibshirani, R. (1990). *Generalized Additive Models*. London: Chapman and Hall.

Hastie, T., Tibshirani, R., and Friedman, J.H. (2001). *The Elements of Statistical Learning*. New York: Springer Verlag.

Hauser, H., Ledermann, F., and Doleisch, H. (2002). Angular Brushing of Extended Parallel Coordinates. *Proceedings of the IEEE Symposium on Information Visualization, 127.*

He, X., and Niyogi, P. (2002). Locality preserving projections. Technical Report TR–2002–09, Department of Computer Science, University of Chicago.

Heckbert, P.S. (1990). Nice numbers for graph labels. In A.S. Glassner (Ed.), *Graphics Gems* (pp. 61–63). Boston: Academic Press.

Heckscher, W. (1958). *Rembrandt's Anatomy of Dr. Nicolaas Tulp.* New York: NYU Press.

Hedges, L.V. and Olkin, I. (1985). *Statistical Methods for Meta-Analysis.* Orlando, FL: Academic Press.

Heiberger, R.M. (1989). *Computation for the Analysis of Designed Experiments*. New York: John Wiley & Sons.

Heiberger, R.M., and Holland, B. (2004). *Statistical Analysis and Data Display*. New York: Springer Verlag.

Hellström, Å. (1985). The time-order error and its relatives: Mirrors of cognitive processes in comparing. *Psychological Bulletin, 97*, 35–61.

von Helmholtz, H. (1866). *Handbuch der Physiologischen Optik*. English translation by J. Southall as *Treatise on Physiological Optics*, *Vol 3*. New York: Dover Books, 1962.

Henley, N.M. (1969). A Psychological Study of the Semantics of Animal Terms. *Journal of Verbal Learning and Verbal Behavior, 8*, 176–184.

Henzinger, M.R., Raghavan, P., and Rajagopalan, S. (1998). Computing on data streams. Technical Report 1998–011, Digital Equipment Corporation Systems Research Center.

Herdeg, W. (Ed.) (1981). *Graphis Diagrams: The Graphic Visualization of Abstract Data*. Zürich, Switzerland: Graphis Press Corp.

Hill, M.A. and Wilkinson, L. (1990). Dissecting the Alaskan king crab with SYSTAT and SYGRAPH. *Proceedings of the Section on Statistical Graphics of the American Statistical Association*, 108–113.

Hinckley, K., Tullio, J., Pausch, R., Proffitt, D., and Kassell, N. (1997). Usability analysis of 3D rotation techniques. *Proceedings of the 10th annual ACM symposium on User interface software and technology*,1–10.

Hochberg, J., and Krantz, D.H. (1986). Perceptual properties of statistical graphs. *Proceedings of the Section on Statistical Graphics of the American Statistical Association*, 29–35.

Hogarth, R.M. (1975). Cognitive processes and the assessment of subjective probability distributions. *Journal of the American Statistical Association, 70*, 271–289.

Holland, P.W. (1986). Statistics and causal inference. *Journal of the American Statistical Association, 81*, 945–960.

Hollander, A.J. (1994). An exploration of virtual auditory shape perception. Unpublished Masters' thesis, Engineering, University of Washington.

Holman, E.W. (1972). The relation between hierarchical and Euclidean models for psychological distances. *Psychometrika, 37,* 417–423.

Holmes, N. (1991). *Designer's Guide to Creating Charts and Diagrams*. New York: Watson–Guptill Publications.

Hopgood, F.R.A., Duce, D.A., Gallop, J.R., and Sutcliffe, D.C. (1983). *Introduction to the Graphical Kernel System (GKS)*. New York: Academic Press.

Hsu, J.C. (1996). *Multiple Comparisons: Theory and Methods.* New York: Chapman & Hall.

Hsu, J.C., and Peruggia, M. (1993). Graphical representations of Tukey's multiplc comparison method. *Journal of Computational and Graphical Statistics, 3*, 143–161.

Hubel, D.H. and Wiesel, T.N. (1962). Receptive fields, binocular interaction, and functional architecture in the cat's visual cortex. *Journal of Physiology, 166*, 106–154.

Huber, P.J. (1972). Robust statistics: A review. *Annals of Mathematical Statistics, 43*, 1041–1067.

Hubert, L.J. (1974a). Some applications of graph theory and related non-metric techniques to problems of approximate seriation: The case of symmetric proximity measures. *British Journal of Mathematical and Statistical Psychology, 27*, 133–153.

Hubert, L.J. (1974b). Problems of seriation using a subject by item response matrix. *Psychological Bulletin, 81*, 976–983.

Hubert, L.J. (1976). Seriation using asymmetric proximity measures. *British Journal of Mathematical and Statistical Psychology, 29*, 32–52.

Hubert, L.J., and Golledge, R.G. (1981). Matrix reorganization and dynamic programming: Applications to paired comparisons and unidimensional seriation. *Psychometrika, 46*, 429–441.

Huff, D. (1954). *How to Lie with Statistics.* New York: Penguin Books.

Hull, C.L. (1943). *Principles of Behavior*. D. Appleton–Century.

Hurley, C.B., and Oldford, R.W. (1991). A software model for statistical graphics. In A. Buja and P. Tukey (Eds.), *Computing and Graphics in Statistics* (pp. 77–94). New York: Springer Verlag.

Indow, T. (1988). Multidimensional studies of Munsell color solid. *Psychological Review 88*, 456–470

Inselberg, A. (1984). The plane with parallel coordinates. *The Visual Computer, 1*, 69–91.

International Organization for Standardization (1993). *Guide to the Expression of Uncertainty in Measurement.* Geneva, Switzerland: ISO.

Jasso, G. (1985). Marital coital frequency and the passage of time: Estimating the separate effects of spouses' ages and marital duration, birth and marriage cohorts, and period influences. *American Sociological Review, 50*, 224–241.

Jasso, G. (1986). Is it outlier detection or is it sample truncation? Notes on science and sexuality. *American Sociological Review, 51*, 738–742.

Jimenez, W.H., Corrêa, W.T., Silva, C.T., Baptista, A.M. (2003). Visualizing spatial and temporal variability in coastal observatories. *Proceedings of the IEEE Symposium on Information Visualization.*

Jobson, J.D. (1992). *Applied Multivariate Data Analysis, Vol. 2: Categorical and Multivariate Methods*. New York: Springer Verlag.

Johnson, B.S. (1993). Treemaps: Visualizing Hierarchical and Categorical Data. Dissertation, The University of Maryland Department of Computer Science.

Johnson, B.S. and Shneiderman, B. (1991). Treemaps: A space-filling approach to the visualization of hierarchical information structures. *Proceedings of the IEEE Symposium on Information Visualization*, 275–282.

Johnson, C,R., and Sanderson, A.R. (2003). A next step: Visualizing errors and uncertainty. *IEEE Computer Graphics and Applications 23*, 2–6.

Johnson, M.H., and Morton, J. (1991). *Biology and Cognitive Development: The Case of Face Recognition*. Oxford, UK: Blackwell.

Johnson, S.C. (1967). Hierarchical clustering schemes. *Psychometrika, 32*, 241–254.

Jones, O. (1856). *The Grammar of Ornament*. Reprinted 1989, Dover Publications.

Jourdain, P.E.B. (1919). *The Nature of Mathematics*. London: T. Nelson. Reprinted in J.R. Newman, *The World of Mathematics*. New York: Simon and Schuster, 1956.

Judd, D.B. (1951). Basic correlates of the visual stimulus. In S.S. Stevens (Ed.), *Handbook of Experimental Psychology* (pp. 811–867). New York: John Wiley & Sons.

Julesz, B. (1965). Texture and visual perception. *Scientific American, 212*, 38–48.

Julesz, B. (1971). *Foundations of Cyclopean Perception.* Chicago: University of Chicago Press.

Julesz, B. (1975). Experiments in the visual perception of texture. *Scientific American, 232*, 34–43.

Julesz, B. (1981). Figure and ground perception in briefly presented isodipole textures. In M. Kubovy and J.R. Pomerantz (Eds.), *Perceptual Organization* (pp. 119–139). Hillsdale, NJ: Lawrence Erlbaum.

Julesz, B., and Levitt, H. (1966). Spatial chords. *Journal of the Acoustical Society of America, 40*, 1253.

Kahn, J.R., and Udry, J.R. (1986). Marital coital frequency: Unnoticed outliers and unspecified interactions lead to erroneous conclusions. *American Sociological Review, 51*, 734–737.

Kahneman, D., Slovic, , P., and Tversky, A. (Eds.) (1982). *Judgment under uncertainty: Heuristics and biases.* Cambridge: Cambridge University Press.

Kahneman, D., and Tversky, A. (1973). On the psychology of prediction. *Psychological Review, 80*, 237–251.

Kahneman, D., and Tversky, A. (1979). Prospect theory: An analysis of decision under risk. *Econometrika, 47*, 263–291.

Kamada, T., and Kawai, S. (1989). An algorithm for drawing general undirected graphs. *Information Processing Letters, 31*, 7–15.

Kany, R. (1985). Lo sgardo filologico: Aby Warburg e i dettagli. *Annali della Scuola Normale Superiore di Pisa, 15*, 1265–83.

Kaufman, L. (1974). *Sight and Mind: An Introduction to Visual Perception*. New York: Oxford University Press.

Kaufman, L., and Kaufman, J.H. (2000). Explaining the moon illusion. *PNAS, 97*, 500–505.

Kehrer, L., and Meinecke, C. (1996). Perceptual organization of visual patterns. In W. Prinz and B. Bridgeman (Eds), *Handbook of Perception and Action, Vol. 1* (pp. 25–70). London: Academic Press.

Keim, D.A., North, S.C., and Panse, C. (2004). CartoDraw: A fast algorithm for generating contiguous cartograms. *IEEE Transactions on Visualization and Computer Graphics , 10*, 95–110.

Kendall, D.G. (1971). Seriation from abundance matrices. In F.R. Hodson, D.G. Kendall, and T. Tautu (Eds.), *Mathematics in the Archaeological and Historical Sciences*. Edinburgh: Edinburgh University Press, 1971.

Kendall, D.G., Barden, D., Carne, T.K., and Le, H. (1999). *Shape and Shape Theory*. New York: John Wiley & Sons.

Kennedy, W.J., and Gentle, J.E. (1980). *Statistical Computing*. New York: Marcel Dekker.

Killeen, P. R., and Fetterman, J. G. (1988). A behavioral theory of timing. *Psychological Review, 95*, 274–295.

Kleiner, B., and Hartigan, J.A. (1981). Representing points in many dimensions by trees and castles. *Journal of the American Statistical Association, 76*, 260–269

Knuth, D.E. (1969). *The Art of Computer Programming, Vol. 1*: *Fundamental Algorithms*. Reading, MA: Addison–Wesley.

Koester, H. (1982). *Introduction to the New Testament, Vol. 1*: *History, Culture, and Religion of the Hellenistic Age.* Philadelphia: Fortress Press, and Berlin: Walter de Gruyter & Co.

Kooijman, S.A.L.M. (1979). The description of point patterns. In R.M. Cormack and J.K. Ord (Eds.), *Spatial and Temporal Analysis in Ecology* (pp. 305–332). Fairland, MD: International Co-operative Publishing House.

Koshy, T. (2001). *Fibonacci and Lucas Numbers with Applications*. New York: John Wiley & Sons.

Kosslyn, S.M. (1980). *Image and Mind.* Cambridge, MA: Harvard University Press.

Kosslyn, S.M. (1985). Graphics and human information processing: A review of five books. *Journal of the American Statistical Association, 80*, 499–512.

Kosslyn, S.M. (1989). Understanding charts and graphs. *Applied Cognitive Psychology, 3*, 185–225.

Kosslyn, S.M. (1994). *Elements of Graph Design*. New York: W.H. Freeman.

Kosslyn, S.M., Ball, T.M., and Reiser, B.J. (1978). Visual images preserve metric spatial information: Evidence from studies of image scanning. *Journal of Experimental Psychology: Human Perception and Performance, 4*, 47–60.

Kramer, G., (Ed.) (1994). *Auditory Display: Sonification, Audification and Auditory Interfaces*. Reading: MA: Addison–Wesley.

Krueger, L.E. (1982). Single judgments of numerosity. *Perception and Psychophysics, 31*, 175–182.

Kruja, E., Marks, J., Blair, A., Waters, R. (2001). A short note on the history of graph drawing. In P. Mutzel, M. Jünger, and S. Leipert (Eds.), *Graph Drawing : 9th International Symposium, GD 2001, Lecture Notes in Computer Science Vol. 2265* (pp. 272–278). Heidelberg: Springer–Verlag.

Krumhansl, C.L. (1979). The psychological representation of musical pitch in a tonal context. *Cognitive Psychology, 11*, 346–374.

Krumhansl, C. L. (1978). Concerning the applicability of geometric models to similarity data: The interrelationship between similarity and spatial density. *Psychological Review, 85*, 445–463.

Kruskal, J.B. (1964). Multidimensal scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika, 29*, 1–27.

Kruskal, J.B., and Seery, J.B. (1980). Designing network diagrams. In *Proceedings of the First General Conference on Social Graphics* (pp. 22–50). U.S. Department of the Census.

Krygier, J.B. (1994). Sound and geographic visualization. In Alan MacEachren and D.R.F. Taylor (Eds.), *Visualization in Modern Cartography* (pp. 149–166). New York: Pergamon Press.

Kutner, M.H., Neter, J., Nachtsheim, C.J., and Wasserman, W. (2003). *Applied Linear Regression Models* (4th ed.). Homewood, IL: McGraw–Hill / Irwin.

LaForge, R. and Suczek, R.F. (1955). The interpersonal dimension of personality: III. An interpersonal checklist. *Journal of Personality, 24,* 94–112.

Lamping, J., Rao, R., and Pirolli, P. (1995). A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. *Human Factors in Computing Systems: CHI 95 Conference Proceedings,* 401–408.

Lancaster, P., and Salkauskas, K. (1986). *Curve and Surface Fitting: An Introduction*. London: Academic Press.

Lauritzen, S.L. (1996). *Graphical Models*. Oxford: Oxford University Press.

Lausen, B., Sauerbrei, W., and Schumacher, M. (1994). Classification and regression trees (CART) used for the exploration of prognostic factors measured on different scales. In P. Dirschedl and R. Ostermann (Eds). *Computational Statistics* (pp. 483–496). Heidelberg: Physica–Verlag.

Lazarsfeld, P.F. and Henry, N.W. (1968). *Latent Structure Analysis*. Boston: Houghton Mifflin.

Leary, T. (1957). *Interpersonal Diagnosis of Personality.* New York: Ronald Press.

L'Ecuyer, P. (1988). Efficient and portable combined random number generators. *Communications of the ACM, 31*, 742–774.

Lee, C.H., and Varshney, A. (2002). Representing thermal vibrations and uncertainty in molecular surfaces. In R.F. Erbacher, P.C. Chen, M. Groehn, J.C. Roberts, and C.M. Wittenbrink (Eds.), *Visualization and Data Analysis 2002: Proceedings of SPIE, Vol. 4665* (pp. 80–90) , San Jose, CA.

Lee, P.M. (1989). *Bayesian Statistics: An Introduction*. New York: John Wiley & Sons.

Legrand, Stephen (2002). *The Snack Sound Toolkit*.

Lenstra, J.K. (1974). Clustering a data array and the traveling salesman problem. *Operations Research, 22*, 413–414.

Leung, Y.K., and Apperly, M.D. (1994). A review and taxonomy of distortion-oriented presentation techniques. *ACM Transactions on CHI, 1*, 126–160.

Levine, M.W., Frishman, L.J., and Enroth-Cugell, C. (1987). Interactions between the rod and the cone pathways in the cat retina. *Vision Research, 27*, 1093–1104.

Levine, M. W. (2000). *Fundamentals of Sensation and Perception* (3rd ed.). Oxford: Oxford University Press.

Levkowitz, H. (1997). Color Theory and Modeling for Computer Graphics, Visualization, and Multimedia Applications. Boston, MA: Kluwer Academic Publishers.

Lewandowsky, S., and Myers, W.E. (1993). Magnitude judgments in 3D bar charts. In R. Steyer, K.F. Wender, and K.F. Widaman (Eds.), *Psychometric Methodology. Proceedings of the 7th European Meeting of the Psychometric Society in Trier* (pp. 266–271). Stuttgart: Gustav Fischer Verlag.

Lewandowsky, S., and Spence, I. (1989). Discriminating strata in scatterplots. *Journal of the American Statistical Association, 84,* 682–688.

Lewis, J.L., Askew, M.J., and Jaycox, D.P. (1982). A comparative evaluation of tibial component designs of total knee prostheses. *The Journal of Bone and Joint Surgery, 64-A,* 129–135.

Liber Usualis : Missae et Officii pro Dominicis et Festis cum Cantu Gregoriano (1953). Tournai, Belgium: Desclée & Co.

Lieberman, H. (1997). Intelligent graphics. *Communications of the ACM, 39*, 38–48.

Ling, R.F. (1973). A computer generated aid for cluster analysis. *Communications of the ACM, 16*, 355–361.

Linhart, H. and Zucchini, W. (1986). *Model Selection*. New York: John Wiley & Sons.

Lisker, L., and Abramson, A. (1970). The voicing dimension: Some experiments in comparative phonetics. *Proceedings of Sixth International Congress of Phonetic Sciences, Prague*. Cited in J.R. Anderson (1995), *Cognitive Psychology and its Implications*. (p. 59). New York: W.H. Freeman.

Little, R.J.A., and Rubin, D.B. (2002). *Statistical Analysis with Missing Data* (2nd ed.). New York: John Wiley & Sons.

Liu, L., Hawkins, D.M., Ghosh, S., and Young, S.S. (2003). Robust singular value decomposition analysis of microarray data. *Proceedings of the National Academy of Sciences, 100,* | 13167–13172.

Lockhead, G.R. (1992). Psychophysical scaling: Judgments of attributes or objects? *Behavioral and Brain Sciences, 15*, 543–601.

Lohse, G.L. (1993). A cognitive model for understanding graphical perception. *Human-Computer Interaction, 8*, 353–388.

Lohse, G.L., Biolsi, K., Walker, N, and Rueter, H.H. (1994). A classification of visual representations. *Communications of the ACM, 37*, 36–49.

Long, L.H. (Ed.) (1971). *The World Almanac*. New York: Doubleday.

Lord, F. and Novick, M.R. (1968). *Statistical Theories of Mental Test Scores*. Reading, MA: Addison–Wesley.

Lorenz, E. (1963). Deterministic nonperiodic flows, *Journal of the Atmospheric Sciences, 20*, 130–41.

Lubischew, A.A. (1962). On the use of discriminant functions in taxonomy, *Biometrics*, *18*, 455–477.

Luce, R.D. and Suppes, P. (1991). Measurement theory. *The New Encyclopaedia Britannica, 23, 792–798*,

Luce, R.D., and Tukey, J.W. (1964). Simultaneous conjoint measurement: A new type of fundamental measurement. *Journal of Mathematical Psychology, 1*, 1–27.

MacEachren, A.M. (1992). Visualizing uncertain information. *Cartographic Perspective, 13*, 10–19.

MacEachren, A.M. (1995). *How Maps Work*. New York: The Guilford Press.

Mackinlay, J. (1986). Automating the design of graphical presentations of relational information. *ACM Transactions on Graphics (TOG) 5*, 110–141.

Maindonald, J.H. (1984). *Statistical Computation.* New York: John Wiley & Sons.

Makridakis, S., and Wheelwright, S.C. (1989). *Forecasting Methods for Management* (5th ed.). New York: John Wiley & Sons.

Maling, D.H. (1992). *Coordinate Systems and Map Projections.* Oxford: Pergamon Press.

Marr, D. (1982). *Vision.* San Francisco: W.H. Freeman.

Marr, D., and Nishihara, H.K. (1978). Representation and recognition of the spatial organization of three dimensional shapes. *Proceedings of the Royal Society, 200*, 269–294.

Massaro, D.W. (1992). Broadening the domain of the fuzzy logical model of perception. In H.L. Pick, Jr., P. Van den Broek, and D.C. Knill (Eds.), *Cognition: Conceptual and Methodological Issues* (pp. 51–84). Washington, DC: American Psychological Association.

McCloskey, B. (2003). *Illustrated Glossary of Sea Anemone Anatomy*, http://web.nhm.ku.edu/tol/glossary/intro.html.

McCormick, W.T., Schweitzer, P.J., and White, T.J. (1972). Problem decomposition and data reorganization by a clustering technique. *Operations Research, 20*, 993–1009.

McLain, D.H. (1974). Drawing contours from arbitrary data points. *The Computer Journal, 17,* 318–324.

McNish, A.G. (1948). Terrestrial magnetism. *The Encyclopaedia Britannica, 21*, 959–970.

McQuitty, L.L. (1968). Multiple clusters, types, and dimensions from iterative intercolumnar correlational analysis. *Multivariate Behavioral Research 3*, 465–477.

Meehl, P.E. (1950). Configural scoring. *Journal of Consulting Psychology, 14*, 165–171.

Mendelssohn, R.C. (1974). The Bureau of Labor Statistic's table producing language (TPL). *Proceedings of the 1974 annual conference*, *Bureau of Labor Statistics, Washington, DC*, 116–122.

Messick, S.M., and Abelson, R.P. (1956). The additive constant problem in multidimensional scaling. *Psychometrika, 21,* 1–15.

Meyer, B. (1988). *Object-Oriented Software Construction*. Englewood Cliffs, NJ: Prentice–Hall.

Mezrich, J.J., Frysinger, S., and Slivjanovski, R. (1984). Dynamic representation of multivariate time series data. *Journal of the American Statistical Association, 79*, 34–40.

Mihalas, D., and Binney, J. (1981). *Galactic Astronomy: Structure and Kinematics*. San Francisco: W.H. Freeman & Co.

Mihalisin, T., Timlin, J., and Schwegler, J. (1991). Visualizing multivariate functions, data, and distributions. *IEEE Computer Graphics and Applications 11,* 13, 28–35.

Mirkin, B. (1996). *Mathematical Classification and Clustering*. Dordrecht, NL: Kluwer Academic Publishers.

Morrison, D.F. (1976). *Multivariate Statistical Methods*. New York: McGraw–Hill.

Morton, H.C. and Price, A.J. (1989). *The ACLS Survey of Scholars: Final Report of Views on Publications, Computers, and Libraries*. Lanham, MD: University Press of America.

Mosteller, F. and Parunak, A. (1985). Identifying extreme cells in a sizable contingency table: Probabilistic and exploratory approaches. In D.C. Hoaglin, F. Mosteller, and J. W. Tukey (Eds.), *Exploring Data Tables, Trends, and Shapes* (pp. 189–224). New York: John Wiley & Sons.

Mosteller, F. and Tukey, J.W. (1968). Data analysis, including statistics. In G. Lindzey and E. Aronson (Eds.), *The Handbook of Social Psychology* (2nd ed.) (pp. 80–203). Reading, MA: Addison–Wesley.

Munkres, J.R. (1975). *Topology: A First Course.* Englewood Cliffs, NJ: Prentice–Hall.

Munzner, T. (1997). Laying out large directed graphs in 3D hyperbolic space. *Proceedings of the IEEE Symposium on Information Visualization.*

Nadaraya, E.A. (1964). On estimating regression. *Theory of Probability and its Applications, 10*, 186–190.

Needham, T. (1997). *Visual Complex Analysis*. Oxford: Clarendon Press.

Nelder, J.A. (1965). The analysis of randomised experiments with orthogonal block structure (Parts I and II). *Proceedings of the Royal Society of London, Series A, 283*, 147–178.

Nelder, J.A. (1976). Scale selection and formatting. *Algorithm AS 96. Applied Statistics, 25*, 94–96.

Nelder, J.A., and Wedderburn, R.W.M. (1972). Generalised linear models. *Journal of the Royal Statistical Society, Series A, 135*, 370–384.

Neter, J, Wasserman, W., and Kutner, M.H. (1990). *Applied Linear Statistical Models* (3rd ed.). Homewood, IL: Richard D. Irwin.

Neumann, O. (1996). Theories of attention. In O. Neumann and A.F. Sanders (Eds), *Handbook of Perception and Action, Vol. 3* (pp. 389–446). London: Academic Press.

Nielsen, D.R., Biggar, J.W., and Erh, K.T. (1973). Spatial variability of field-measured soil-water properties. *Hilgardia, 42*, 215–259.

Niermann, S. (2005). Optimizing the ordering of tables with evolutionary computation. *The American Statistician, 59*, 41–46.

Norton, A. Rubin, M., and Wilkinson L. (2001). Streaming graphics. *Statistical Computing and Graphics Newsletter*, *12*, 11–15.

Okabe, A., Boots, B., and Sugihara, K. (1992). *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. New York: John Wiley & Sons.

Olkin, I. (1973), Testing and estimation for structures which are circularly symmetric in blocks. In D.G. Kabe and R.P. Gupta (Eds.), *Multivariate Statistical Inference (*pp. 183–195). Amsterdam: North–Holland.

Olkin, I. and Press, S. (1969), Testing and estimation for a circular stationary model. *The Annals of Mathematical Statistics 40*, 1358–1373.

Olson, J.M., and Brewer, C.A. (1997). An evaluation of color selections to accommodate map users with color-vision impairments. *Annals of the Association of American Geographers, 87*, 103–134.

Omernik, J.M. (1987). Ecoregions of the coterminous United States. *Annals of the Association of American Geographers}, 77,* 118–25.

Omernik, J.M. (1995). Ecoregions: a spatial framework for environmental management. In W.S. Davis, and T.P. Simon (Eds.), *Biological Assessment and Criteria: Tools for WaterResource Planning and Decision Making* (pp. 49–62). Boca Raton, FL: Lewis Publishers.

O'Neill, R. (1971).Function minimization using a simplex procedure. *Algorithm AS 47. Applied Statistics, 20*, 338.

O'Rourke, J. (1998). *Computational Geometry in C* (2nd ed.). Cambridge: Cambridge University Press.

Packer, C.V. (1989). Applying row-column permutation to matrix representations of large citation networks. *Information Processing Management, 25*, 307–314.

Pang, A. (2001). Visualizing uncertainty in geo-spatial data. Paper prepared for a committee of the Computer Science and Telecommunications Board, National Research Council. Computer Science Department, University of California, Santa Cruz.

Pang, A., Wittenbrink, C.M., and Lodha, S. K. (1997). Approaches to uncertainty visualization. *The Visual Computer*, *13*, 370–390.

Papadopoulos, A. (2002). Mathematics and music theory: From Pythagoras to Rameau. *The Mathematical Intelligencer, 24*, 65–73.

Papantonakis, A., and King, P.J.H. (1995). Syntax and semantics of GQL, a graphical query language. *Journal of Visual Languages and Computing, 6*, 3–25.

Papathomas, T.V. and Julesz, B. (1988). The application of depth separation to the display of large data sets. In W.S. Cleveland and M.E. McGill (Eds.), *Dynamic Graphics for Statistics* (pp. 353–377). Belmont, CA: Wadsworth.

Pearl, J. (2001). *Causality : Models, Reasoning, and Inference*. Cambridge: Cambridge University Press.

Pearson, K. (1892). *The Grammar of Science*. London: Walter Scott.

Pedersen, D., Riis, K., and Pedersen, T.B. (2002). A powerful and SQL-compatible data model and query language for OLAP. *Proceedings of the thirteenth Australasian conference on Database technologies*, 121–130.

Pinker, S. (1990). A theory of graph comprehension. In R. Freedle (Ed.), *Artificial Intelligence and the Future of Testing (*pp. 73–126). Hillsdale, NJ: Lawrence Erlbaum Associates.

Pinker, S. (1997). *How the Mind Works*. New York: W.W. Norton & Company.

Plateau, J. (1872). Sur la mesure des sensations physiques, et sur la loi qui lie l'intensité de ces sensations à l'intensité de la cause excitante. *Bulletins de l'Académie Royale de Belgique, 33*, 376–388.

Plutchik, R. and Conte, H.R. (1997). *Circumplex Models of Personality and Emotions*. Washington, DC: American Psychological Association.

Preparata, F.P., and Shamos, M.I. (1985). *Computational Geometry: An Introduction.* New York: Springer Verlag.

Press, W., Flannery, B., Teukolsky, S., and Vetterling, W. (1986). *Numerical Recipes: The Art of Scientific Computing*. New York: Cambridge University Press.

Rabenhorst, D.A. (1993). Interactive exploration of multidimensional data. Yorktown Heights, NY: *IBM T.J. Watson Research Center.*

Ramsay, J.O, and Silverman, B.W. (1997). *Functional Data Analysis*. New York: Springer Verlag.

Rao, C.R. (1973). *Linear Statistical Inference and Its Applications* (2nd ed.). New York: John Wiley & Sons.

Rasch, G. (1960). *Probabilistic Models for Some Intelligence and Attainment Tests*. Copenhagen: The Danish Institute for Educational Research.

Ratanamahatana, C.A., and Keogh, E. (2004). Everything you know about dynamic time warping is wrong. Third Workshop on Mining Temporal and Sequential Data, Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD–2004).

Reed, S.K. (1972). Pattern recognition and categorization. *Cognitive Psychology, 3*, 382–407.

Reeves, A (1996). Temporal resolution in visual perception. In W. Prinz and B. Bridgeman (Eds), *Handbook of Perception and Action, Vol. 1* (pp. 11–24). London: Academic Press.

Rips, L.J., and Collins, A. (1993). Categories and resemblance. *Journal of Experimental Psychology: General, 122*, 468–486.

Robert, C.P., and Casella, G. (2004). Monte Carlo Statistical Methods (2nd. ed.). New York: Springer Verlag.

Roberts, W.A. (2002). Are animals stuck in time? *Psychological Bulletin, 128*, 473–489.

Robinson, A.H. (1974). A new map projection: its development and characteristics. *International Yearbook of Cartography, 14*, 145–155.

Robinson, A.H. (1982). *Early Thematic Mapping in the History of Cartography*. Chicago: University of Chicago Press.

Robinson, A.H., Morrison, J.L., Muehrcke, P.C. (1995). *Elements of Cartography* (6th ed.). New York: John Wiley & Sons.

Robinson, W.S. (1951). A method for chronologically ordering archaeological deposits. *American Antiquity, 16*, 293–301.

Rogers, D.F. and Adams, J.A. (1990). *Mathematical Elements for Computer Graphics* (2nd ed.). New York: McGraw–Hill.

Ronan, C.A. (1991). Measurement of time and types of calendars. *The New Encyclopaedia Britannica, 15, 432–435.*

Rosch, E. (1975). Cognitive representations of semantic categories. *Journal of Experimental Psychology: General*, 104, 192–223.

Roth, M.A., Korth, H.F., and Silberschatz, A. (1988). Extended algebra and calculus for nested relational databases. *ACM Transactions on Database Systems, 13*, 389–417.

Roth, S. F., Kolojejchick, J., Mattis, J., Chuah, M.C., Goldstein, J., and Juarez, O. (1994). SAGE tools: a knowledge-based environment for designing and perusing data visualizations. *Human Factors in Computing Systems: CHI 94 Conference Proceedings*, 27–28.

Roth, S.F., Kolojejchick, J., Mattis, J., and Chuah, M. (1995). SageTools: An intelligent environment for sketching, browsing, and customizing data graphics. *Human Factors in Computing Systems: CHI 95 Conference Proceedings*, 409–410.

Roth, S.F., Chuah, M.C., Kerpedjiev, S., Kolojejchick, J.A., and Lucas, P. (1997). Towards an information visualization workspace: Combining multiple means of expression. *Human-Computer Interaction Journal, 12,* 131–185.

Rothkopf, E.Z. (1957). A measure of stimulus similarity and errors in some paired associate learning tasks. *Journal of Experimental Psychology, 53*, 94–101.

Rousseeuw, P.J., Ruts, I., Tukey, J.W. (1999). The Bagplot: A Bivariate Boxplot. *The American Statistician, 53*, 382–387.

Roweis, S.T. and Saul, L.K. (2000). Nonlinear dimensionality reduction by locally linear embedding. *Science*, *290*, 2323– 2326.

Rubin, D.B. (1974). Estimating causal effects of treatments in randomized and nonrandomized studies. *Journal of Educational Psychology, 66*, 688–701.

Rubin, D.B. (1976). Inference and Missing Data. *Biometrika, 63,* 581-592.

Rubin, D.B. (1978). Multiple imputations in sample surveys — a phenomenological Bayesian approach to nonresponse. *Proceedings of the Survey Research Methods Section of the American Statistical Association*, 20–34.

Rubin, D.B. (1987). *Multiple Imputation for Nonresponse in Surveys*. New York: John Wiley & Sons.

Rugg, R.D., Egenhofer, M.J., and Kuhn, W. (1995). Formalizing behavior of geographic feature types. Technical Report 95-7, National Center for Geographic Information and Analysis/NCGIA.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorensen, W. (1991). *Object-Oriented Modeling and Design.* Englewood Cliffs, NJ: Prentice–Hall.

Rumelhart, D.E. (1977). *An Introduction to Human Information Processing.* New York: John Wiley & Sons.

Russell, J.A. (1980). A circumplex model of affect. *Journal of Personality and Social Psychology, 39,* 1161–1178

Russell, J.A., and Fernandez-Dols, J.-M. (Eds.) (1997). *The Psychology of Facial Expression.* New York: Cambridge University Press.

SAS Institute Inc. (1976). *A User's Guide to SAS*. Raleigh, NC: Sparks Press.

Sacks, O. (1995). *An Anthropologist on Mars.* New York: Alfred A. Knopf.

Safire, W. (1997). *Watching My Language: Adventures in the Word Trade*. New York: Random House.

Sakoe, H., and Chiba, S. (1978). Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing, 26*, 43–49.

Sall, J. (1992). Graphical comparison of means. *Statistical Computing and Statistical Graphics Newsletter, 3*, 27–32.

Sammon, J.W. (1969). A nonlinear mapping for data analysis. *IEEE Transactions on Computers, 18.* 401–409.

Sankoff, D., and Kruskal, J.B. (Eds.) (1983). *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*. Reading, MA: Addison–Wesley.

Santini, S., and Jain, R. (1997). Similarity is a geometer. *Multimedia Tools and Applications, 5*, 277–306.

Sarkar, M., and Brown, M.H. (1994). Graphical fisheye views. *Communications of the ACM, 37*, 73–83.

Sattath, S., and Tversky, A. (1977). Additive similarity trees. *Psychometrika, 42*, 319–345.

Schaefer, E.S. (1959). A circumplex model for maternal behavior. *Journal of Abnormal and Social Psychology, 59,* 226–235

Schmid, C.F., and Schmid, S.E. (1979). *Handbook of Graphic Presentation* (2nd ed.). New York: Ronald Press.

Schmid, C.F. (1983). *Statistical Graphics: Design Principles and Practices.* New York: John Wiley & Sons.

Schneider, B., Parker, S., Ostrosky, D., Stein, D., and Kanow, G. (1974). A scale for the psychological magnitude of number. *Perception & Psychophysics, 16*, 43–46.

Schroeder, M. (1991). *Fractals, Chaos, Power Laws*. New York: W.H. Freeman.

Scott, D.W. (1979). On optimal and data-based histograms. *Biometrika, 66*, 605–610.

Scott, D.W. (1992). *Multivariate Density Estimation: Theory, Practice, and Visualization.* New York: John Wiley & Sons.

Selfridge, O.G. (1959). Pandemonium: A paradigm for learning. In D.V. Blake and A.M. Uttley (Eds.), *Symposium on the mechanization of thought processes* (pp. 511–529). London: H.M. Stationery Office.

Seo, J., and Shneiderman, B. (2005). A Rank-by-feature framework for interactive exploration of multidimensional data. *Information Visualization*, in press.

Setterfield, B., and Norman, T. (1987). *The Atomic Constants, Light, and Time*. Blackwood, Australia: Privately published.

Shannon C.E. (1948). A mathematical theory of communication. *Bell System Technical Journal, 27*, 379–423.

Shepard, D. (1965). A two-dimensional interpolation function for irregularly spaced data. *Proceedings of the 23rd National Conference of the ACM,* 517.

Shepard, R.N. (1962). The analysis of proximities: Multidimensional scaling with an unknown distance function. I. *Psychometrika, 27*, 125–139.

Shepard, R.N. (1964). Circularity in judgments of relative pitch. *The Journal of the Acoustical Society of America, 36*, 2346–2353.

Shepard, R.N. (1978). The circumplex and related topological manifolds in the study of perception. In S. Shye (Ed.), *Theory Construction and Data Analysis in the Behavioral Sciences* (pp. 29–80). San Francisco: Jossey–Bass.

Shepard, R.N. and Carroll, J.D. (1966). Parametric representation of nonlinear data structures. In P.R. Krishnaiah (Ed.), *International Symposium on Multivariate Analysis*. New York: Academic Press.

Shepard R.N. and Cermak G.W. (1973). Perceptual-cognitive explorations of a toroidal set of free-form stimuli. *Cognitive Psychology, 4*, 351–377.

Shepard R.N. and Chipman S. (1970). Second order isomorphism of internal representations: Shapes of states. *Cognitive Psychology, 1*, 1–17.

Shepard, R.N. and Cooper, L.A. (1982). *Mental Images and their Transformations*. Cambridge, MA: MIT Press.

Shepard, R.N. and Metzler, J. (1971). Mental rotation of three-dimensional objects. *Science, 171*, 701–703.

Shiffman, H.R. (1990). *Sensation and Perception: An Integrated Approach.* New York: John Wiley & Sons.

Shneiderman, B. (1992). Tree visualization with tree-maps: A 2-d space-filling approach. *ACM Transactions on Graphics, 11*, 92–99.

Shoshani, A. (1997). OLAP and statistical databases: similarities and differences. *Proceedings of the Sixteenth ACM SIGACT–SIGMOD–SIGART Symposium on Principles of Database Systems,* 185–196.

Silverman, B.W. (1986). *Density Estimation for Statistics and Data Analysis.* New York: Chapman & Hall.

Simkin, D., and Hastie, R. (1987). An information processing analysis of graph perception. *Journal of the American Statistical Association, 82*, 454–465.

Simonoff, J.S. (1996). *Smoothing Methods in Statistics*. New York: Springer–Verlag.

Simonoff, J.S. (1997). The “unusual episode” and a second statistics course. *Journal of Statistics Education, 5*(*1*).

Skinner, B.F. (1961). *Cumulative Record*. New York: Appleton–Century–Crofts.

Skinner, B.F. (1969). *Contingencies of Reinforcement: A Theoretical Analysis*. New York: Appleton–Century–Crofts.

Skiena, S.S. (1998). *The Algorithm Design Manual*. New York: Springer Verlag.

Slagel, J.R., Chang, C.L., and Heller, S.R. (1975). A clustering and data reorganizing algorithm. *IEEE Transactions on Systems, Man, and Cybernetics, 5*, 125–128.

Sloane, N.J.A., and Plouffe, S. (1995). *The Encyclopedia of Integer Sequences*. New York: Academic Press.

Slovic, P., Fischhoff, B., and Lichtenstein, S. (1982). Facts versus fears: Understanding perceived risk. In D. Kahneman, P. Slovic, and A. Tversky (Eds.). *Judgment Under Uncertainty: Heuristics and Biases* (pp. 463–489). Cambridge: Cambridge University Press.

Smith, S., Bergeron, R., and Grinstein, G. (1990). Stereophonic and surface sound generation for exploratory data analysis. *Human Factors in Computing Systems: CHI 90 Conference Proceedings*, 125–32.

Smookler, J. (1966). *Invention and Economic Growth*. Cambridge, MA: Harvard University Press.

Sneath, P.H.A. (1957). The application of computers to taxonomy. *Journal of General Microbiology, 17*, 201–226.

Snyder, J.P. (1989). *An Album of Map Projections.* US Geological Survey: Professional Paper 1453.

Spearman, C. (1904). “General intelligence” objectively determined and measured. *American Journal of Psychology, 15,* 201–293.

Spence, I. (1990). Visual psychophysics of simple graphical elements. *Journal of Experimenal Psychology: Human Perception and Performance, 16*, 683–692.

Spence, I., and Graef, J. (1974). The determination of the underlying dimensionality of an empirically obtained matrix of proximities. *Multivariate Behavioral Research, 9*, 331–342.

Spence, I., and Lewandowsky, S. (1991). Displaying proportions and percentages. *Applied Cognitive Psychology, 5*, 61–77.

Spoehr, K.T. and Lehmkuhle, S.W. (1982). *Visual Information Processing*. San Francisco: W.H. Freeman and Company.

SPSS Inc. (1996). *SPSS Reference Guide*. Upper Saddle River, NJ: Prentice–Hall.

Standing, L. (1973). Learning 10,000 pictures. *Quarterly Journal of Experimental Psychology, 25*, 207–222.

Steiger, J. H., and Schönemann, P.H. (1978). A history of factor indeterminacy. In Shye (Ed.), *Theory construction and data analysis* (pp. 136–178). San Francisco: Jossey–Bass.

Stevens, A., and Coupe, P. (1978). Distortions in judged spatial relations. *Cognitive Psychology, 10*, 422–437.

Stevens, S.S. (1946). On the theory of scales of measurement. *Science, 103*, 677–680.

Stevens, S.S. (1961). To honor Fechner and repeal his law. *Science, 133*, 80–86.

Stevens, S.S. (1985). *Psychophysics: Introduction to its Perceptual, Neural, and Social Prospects*. New Brunswick, NJ: Transaction Books.

Stevenson, D. (1981). A Proposed Standard for Binary Floating-Point Arithmetic: Draft 8.0 of IEEE Task P754. *IEEE Computer, 14*, 51–62.

Stigler, S. (1983). *The History of Statistics: The Measurement of Uncertainty before 1900*. Cambridge, MA: Harvard University Press.

Stillwell, J. (1992). *Geometry of Surfaces*. New York: Springer–Verlag.

Stirling, W.D. (1981). Scale selection and formatting. *Algorithm AS 168. Applied Statistics, 30*, 339–344.

Stolte, C., Tang, D., and Hanrahan, P. (2000). Polaris: A System for Query, Analysis and Visualization of Multi-dimensional Relational Databases. *Proceedings of the Sixth IEEE Symposium on Information Visualization.*

Stoyan, D., Kendall, W.S., and Mecke, J. (1987). *Stochastic Geometry and its Applications*. New York: John Wiley & Sons.

Stouffer, S.A., Guttman, L., Suchman, E.A., Lazarsfeld, P.F., Staf, S.A., and Clausen, J.A. (1950). *Measurement and Prediction.* Princeton, NJ: Princeton University Press.

Strothotte, T., and Schlechtweg, S. (2002). *Non-Photorealistic Computer Graphics: Modeling, Rendering, and Animation*. New York: Morgan Kaufmann.

Sturges, H.A. (1926). The choice of a class interval. *Journal of the American Statistical Association, 21*, 65–66.

Swayne, D.F., and Buja, A. (1998). Missing data in interactive high-dimensional data visualization. *Computational Statistics, 13,* 15–26.

Swayne, D.F., Cook, D., and Buja, A. (1998). XGobi: Interactive dynamic data visualization in the X Window system. *Journal of Computational and Graphical Statistics, 7*, 113–130.

Swets, J.A., Tanner Jr., W.P., and Birdsall, T.G. (1961). Decision processes in perception. *Psychological Review, 68*, 301–340.

Symanzik, J., Cook, D., Kohlmeyer, B.D., Lechner, U., Cruz-Neira, C. (1997). Dynamic Statistical Graphics in the C2 Virtual Environment. *Computing Science and Statistics, 29*(2), 35–40.

Taylor, B.N. (1997). The international system of units (SI). In *Special Publication 330*, NIST. Washington, DC, 171–182.

Tenenbaum, J.B., de Silva, V., and Langford, J.C. (2000). A global geometric framework for nonlinear dimensionality reduction. *Science, 290*, 2319–2322.

Theus, M. (1998). MONDRIAN: Data visualization in Java. AT&T Conference on Statistical Science and the Internet, Madison, NJ, Drew University, July 12–14.

Thisted, R.A. (1988). *Elements of Statistical Computing: Numerical Computation*. New York and London: Chapman and Hall.

Thompson, D'Arcy (1942). *On Growth and Form: A New Edition*. Cambridge: Cambridge University Press. First published in 1917.

Thurstone, L.L. (1927). A law of comparative judgment. *Psychological Review, 34*, 273–286.

Thurstone, L.L. (1945). *Multiple Factor Analysis.* Chicago: University of Chicago Press.

Tidmore, F.E. and Turner, D.W. (1977). Clustering with Chernoff–type faces. *Communications in Statistics — Theory and Methods, 12*, 397–408.

Tierney, L. (1991). *Lisp-Stat*. New York: John Wiley & Sons.

Tilling, L. (1975). Early experimental graphis. *The British Journal for the History of Science, 8*, 193–213.

Tobler, W. (1993). Speculations on the geometry of geography. Technical Report 93-1, National Center for Geographic Information and Analysis. Santa Barbara, CA.

Tobler, W. (1997). Visualizing the impact of transportation on spatial relations. Paper presented at the Western Regional Science Association meeting. Honolulu, Hawaii.

Tobler, W. (1999). Unusual map projections. Paper presented at the 1999 AAG Convention, Honolulu, Hawaii.

Toffoli, T., and Margolus, N. (1987). *Cellular Automata Machines*. Cambridge, MA: MIT Press.

Torgerson, W.S. 1952. Multidimensional scaling: I. theory and method. *Psychometrika, 17*, 401–419.

Travis, D. (1991). *Effective Color Displays: Theory and Practice.* New York: Academic Press.

Treinish, L.A. (1993). Unifying principles of data management for scientific visualization. In R.A. Earnshaw and D. Watson (Eds.), *Animation and Scientific Visualization: Tools and Applicaitons* (pp. 141–170). New York: Academic Press.

Treisman, M. (1993). On the structure of the temporal sensory system. *Psychologica Belgica, 33*, 271–293.

Troje, N., and Bülthoff, H. (1996). How is bilateral symmetry of human faces used for recognition of novel views? Technical Report 38, Max-Planck-Institut für biologische Kybernetik.

Tufte, E.R. (1983). *The Visual Display of Quantitative Information*. Cheshire, CT: Graphics Press.

Tufte, E.R. (1990). *Envisioning Data*. Cheshire, CT: Graphics Press.

Tufte, E.R. (1997). *Visual Explanations*. Cheshire, CT: Graphics Press.

Tukey, J.W. (1953). The problem of multiple comparisons. Unpublished manuscript. Princeton University Department of Statistics.

Tukey, J.W. (1957). On the comparative anatomy of transformations. *Annals of Mathematical Statistics, 28*, 602–632.

Tukey, J.W. (1958). Bias and confidence in not quite large samples. *Annals of Mathematical Statistics, 29*, 614.

Tukey, J.W. (1974). Mathematics and the picturing of data. *Proceedings of the International Congress of Mathematicians*, 523–531.

Tukey, J.W. (1977). *Exploratory Data Analysis*. Reading, MA: Addison–Wesley.

Tukey, J.W., and Tukey, P.A. (1981). Graphical display of data sets in three or more dimensions. In Vic Barnett (Ed.), *Interpreting Multivariate Data* (pp. 189–275). Chichester: John Wiley & Sons.

Tukey, J.W., and Tukey, P.A. (1985). Computer graphics and exploratory data analysis: An introduction. In *Proceedings of the Sixth Annual Conference and Exposition: Computer Graphics'85* (pp. 773–785). Fairfax, VA: National Computer Graphics Association. Reprinted in W.S. Cleveland (Ed.), *The Collected Works of John W. Tukey, Vol 5* (pp. 419-436). New York: Chapman & Hall.

Tukey, P.A. (1993). Scagnostics. Unpublished paper presented at IMA Conference on Graphics. Minneapolis, MN: Institute for Mathematics and its Applications.

Turk, M., and Pentland, A., (1991) Eigenfaces for Recognition. *Journal of Cognitive Neuroscience, 3*, 71–86.

Tversky, A. (1977). Features of similarity. *Psychological Review, 84*, 327–352.

Tversky, A. (2003). *Preference, Belief, and Similarity*. Ed. E. Shafir. Cambridge, MA: MIT Press.

Tversky, A., and Kahneman, D. (1971). The belief in the "law of small numbers." *Psychological Bulletin, 76*, 105–110.

Tversky, A., and Kahneman, D. (1973). Availability: A heuristic for judging frequency and probability. *Cognitive Psychology, 5*, 207–232.

Tversky, A., and Kahneman, D. (1974). Judgments under uncertainty: Heuristics and biases. *Science, 185*, 1124–1131.

Tversky, A., and Kahneman, D. (1983). Extensional versus intuitive reasoning: The conjunction fallacy in probability judgment. *Psychological Bulletin, 90*, 293–315.

Tversky, B., and Schiano, D.J. (1989). Perceptual and cognitive factors in distortions in memory for graphs and maps. *Journal of Experimental Psychology: General, 118*, 387–398.

Unwin, A.R. (1999). Requirements for interactive graphics software for exploratory data analysis. *Computational Statistics, 14*, 7–22.

Unwin, A.R., Hawkins, G., Hofmann, H., and Siegl, B (1996). Interactive graphics for data sets with missing values — MANET. *Journal of Computational and Graphical Statistics, 5*, 113–122.

Upson, C., Faulhaber, T., Kamins, D., Schlege, D., Laidlaw, D., Vroom, J., Gurwitz, R., and vanDam, A. (1989). The application visualization system: A computational environment for scientific visualization. *IEEE Computer Graphics and Applications, 9*, 30–42.

Upton, G.J.G. and Fingleton, B. (1989). *Spatial Data Analysis by Example, Vol 2*: *Categorical and Directional Data.* New York: John Wiley & Sons.

Urbanek, S. (2003). Many faces of a tree. *Computing Science and Statistics: Proceedings of the 35th Symposium on the Interface.*

Urbanek, S., and Unwin, A.R. (2001). Making trees interactive — KLIMT. *Computing Science and Statistics: Proceedings of the 33rd Symposium on the Interface.*

US Department of Labor, Bureau of Labor Statistics, Office of Systems and Standards (1979). *Table Producing Language, Version 5: Language Guide and Print Control Language*. Washington, DC: Government Printing Office.

Vach, W. (1995). Classification trees. *Computational Statistics, 10*, 9–14.

Velleman, P.F., and Hoaglin, D.C. (1981). *Applications, Basics, and Computing of Exploratory Data Analysis.* Belmont, CA: Duxbury Press.

Velleman, P.F., and Wilkinson, L. (1996). Nominal, ordinal, interval, and ratio typologies are misleading for classifying statistical methodology. *The American Statistician, 47*. 65–72.

Velleman, P.F. (1998). *Data Desk*. Ithaca, NY: Data Description Inc.

Voss, R.F., and Clarke, J. (1978). 1/*f* noise in music: Music from 1/*f* noise. *Journal of the Acoustical Society of America, 63*, 258–263.

Wadsworth, H.M. Jr., Stephens, K.S., and Godfrey, A.B. (1986). *Modern Methods for Quality Control and Improvement*. New York: John Wiley & Sons.

Wahba, G. (1990). *Spline Models for Observational Data*. Philadelphia, PA: SIAM.

Wainer, H. (1995). A rose by another name. *Chance*, 8, 46–51.

Wainer, H. (1997). *Visual Revelations: Graphical Tales of Fate and Deception from Napoleon Bonaparte to Ross Perot*. New York: Springer–Verlag.

Wainer, H., and Francolini, C.M. (1980). An empirical inquiry concerning human understanding of two-variable color maps. *The American Statistician, 34*, 81–93.

Walker, B.N. (2002). Magnitude estimation of conceptual data dimensions for use in sonification. *Journal of Experimental Psychology: Applied, 8*, 211–221.

Wallgren, A., Wallgren, B., Persson, R., Jorner, U., and Halland, J-A. (1996). *Graphing Statistics & Data*. Thousand Oaks, CA: Sage Publications.

Wassenaar, L.I., and Hobson, K.A. (1998). Natal origins of migratory monarch butterflies at wintering colonies in Mexico: New isotopic evidence. *Proceedings of the National Academy of Sciences, 95,*15436.

Watson, G.S. (1964). Smooth regression analysis. *Sankhya, Series A, 26*, 359–372.

Watson, J.B. (1924). *Behaviorism*. New York: W.W. Norton & Company.

Watt, R. (1991). *Understanding Vision*. London: Academic Press.

Webb, E.J., Campbell, D.T., Schwartz, R.D., Sechrest, L., and Grove, J.B. (1981). *Nonreactive Measures in the Social Sciences* (2nd ed.). Boston, MA: Houghton Mifflin.

Wegman, E.J. (1990). Hyperdimensional data analysis using parallel coordinates. *Journal of the American Statistical Association, 85*, 664–675.

Wegman, E.J., and Luo, Q. (2002). On methods of computer graphics for visualizing densities. *Journal of Computational and Graphical Statistics, 11*, 37–162.

Wegman, E.J., and Symanzik, J. (2002). Immersive projection technology for visual data mining. *Journal of Computational and Graphical Statistics, 11*, 163–188.

Wertheimer, M. (1958). *Principles of Perceptual Organization*. In D.C. Beardslee and M. Wertheimer (Eds.), *Readings in Perception*. Princeton, NJ: Van Nostrand–Reinhold. (Originally published in German, 1923).

Whorf, B.L. (1941). Languages and logic. In J.B. Carroll (Ed.), *Language, Thought, and Reality: Selected Papers of Benjamin Lee Whorf* (pp. 233–245). Cambridge, MA: MIT Press.

Wickens, C.D., Hollands, J.G. (1999). *Engineering Psychology and Human Performance* (3rd ed.). Upper Saddle River, NJ: Prentice Hall.

Wichman, B.A., and Hill, I.D. (1982). An efficient and portable pseudo-random number generator. Algorithm AS 183. *Applied Statistics, 311*, 188–190.

Wiggins, J.S. (1979). A psychological taxonomy of trait-descriptive terms: The interpersonal domain. *Journal of Personality and Social Psychology, 37,* 395–412.

Wiggins, J.S. (1982). Circumplex models of interpersonal behavior in clinical psychology. In P.C. Kendall and J.N. Butcher (Eds.), *Handbook of Research Methods in Clinical Psychology* (pp. 183–221). New York: John Wiley & Sons.

Wiggins, J.S. (1996). An informal history of the interpersonal circumplex tradition. *Journal of Personality Assessment, 66,* 217–233.

Wilf, H.S. (1994). *generatingfunctionology* (2nd ed.). Boston: Academic Press, Inc.

Wilhelm, A. (2003). User interaction at various levels of data displays. *Computational Statistics and Data Analysis, 43*, 471–494.

Wilkinson, G.N., and Rogers, C.E. (1973). Symbolic description of factorial models for analysis of variance. *Journal of the Royal Statistical Society, Series C, 22*, 392–399.

Wilkinson, L. (1975). Similarity and Preference Structures. Unpublished dissertation, Yale University.

Wilkinson, L. (1979), Permuting a matrix to a simple pattern, *Proceedings of the Statistical Computing Section of the American Statistical Association*, 409–412.

Wilkinson, L. (1982). An experimental evaluation of multivariate graphical point representations. *Human Factors in Computer Systems: Proceedings*, 202–209.

Wilkinson, L. (1983a). SYSTAT: System Statistics. *Proceedings of the Statistical Computing Section of the American Statistical Association*, 312–314.

Wilkinson, L. (1983b). Fuzzygrams. Paper presented at Harvard Computer Graphics Week. Cambridge, MA.

Wilkinson, L. (1988). Cognitive science and graphic design. In *SYGRAPH: The System for Statistics*. Evanston, IL: SYSTAT Inc.

Wilkinson, L. (1992). Graphical displays. *Statistical Methods in Medical Research 1*, 3–25.

Wilkinson, L. (1993a). Algorithms for choosing the range and domain of plotted functions. In A. Buja and P. Tukey (Eds.), *Computing and Graphics in Statistics* (pp. 231–237). New York: Springer Verlag.

Wilkinson, L. (1993b). Comment on “A model for studying display methods of statistical graphics.” *Journal of Computational and Graphical Statistics, 2*, 355–360.

Wilkinson, L. (1996). A graph algebra. *Computing Science and Statistics: Proceedings of the 28th Symposium on the Interface*, 341–351.

Wilkinson, L. (1997). *SYSTAT 7*. Chicago, IL: SPSS Inc.

Wilkinson, L. (1998). *SYSTAT 8*. Chicago, IL: SPSS Inc.

Wilkinson, L. (1999). Dot plots. *The American Statistician, 53*, 276–281.

Wilknson, L. (2004). Heatmaps. Paper presented at MGA Workshop III: Multiscale Structures in the Analysis of High-Dimensional Data. UCLA Institute for Pure and Applied Mathematics. October 25 - 29.

Wilkinson, L., Blank, G., and Gruber, C. (1996). *Desktop Data Analysis with SYSTAT*. Upper Saddle River, NJ: Prentice–Hall.

Wilkinson, L., and Dallal, G. E. (1981), Tests of significance in forward selection regression with an F-to-enter stopping rule, *Technometrics, 23*, 25–28

Wilkinson, L., Gimbel, B.R., and Koepke, D. (1982). Configural self-diagnosis. In N. Hirschberg and L.G. Humphreys (Eds.), *Multivariate Applications in the Social Sciences* (pp. 103–115). Hillsdale, NJ: Lawrence Erlbaum.

Wilkinson, L., Grossman, R., and Anand, A. (2005). Graph-theoretic scagnostics. Technical report, Department of Computer Science, University of Illinois at Chicago.

Wilkinson, L., and McConathy, D. (1990). Memory for graphs. *Proceedings of the Section on Statistical Graphics of the American Statistical Association*, 25–32.

Wills, G.J. (1999). NicheWorks — Interactive visualization of very large graphs. *Journal of Computational and Graphical Statistics, 8,* 190–212.

Wilson, E.B. (1928). On hierarchical correlation systems. *Proceedings of the National Academy of Sciences*, *14*, 283–291.

Winkler, S. (2000). Parallel Coordinates and Data Analysis with the Software CASSATT. Dissertation, Institute of Mathematics, University of Augsburg.

Winston, P.H. (1984). *Artificial Intelligence*. Reading, MA: Addison–Wesley.

Wittenbrink, C.M., Pang, A.T., and Lodha, S.K. (1996). Glyphs for visualizing uncertainty in vector fields. *IEEE Transactions on Visualization and Computer Graphics, 2*, 266–279.

Wolfram, S. (2002). *A New Kind of Science*, Champaign, IL: Wolfram Media, Inc.

Wong, P.C., Cowley, W., Foote, H., Jurrus, E., and Thomas, J. (2000). Visualizing sequential patterns for text mining, *Proceedings of the IEEE Symposium on Information Visualization,* 105.

Woodruff, A., Landay, J., and Stonebraker, M. (1998). Constant density visualizations of non-uniform distributions of data. *Proceedings of the 11th Annual ACM Symposium on User Interface Software and Technology* (*UIST*)*,* 19–28.

Young, F.W., and Hamer, R.M. (1987). *Multidimensional Scaling: History, Theory and Applications*. New York: Erlbaum.

Young, G., and Householder, A.S. (1938). Discussion of a set of points in terms of their mutual distances. *Psychometrika, 3*, 19–22.

Young, M.P., and Yamane, S. (1992). Sparse population coding of faces in the inferotemporal cortex. *Science, 56*, 1327–1331.

Young, T. (1802). The Bakerian lecture: On the theory of lights and colours. *Philosophical Transactions of the Royal Society, 92*, 12–48.

Zhang, D., and Lu, G. (2004). Review of shape representation and description techniques. *Pattern Recognition*, *37*, 1–19.

Author Index

Subject Index