Includes time-saving Command and Option Reference

Gnuplot IN ACTION

Understanding data with graphs

Philipp K. Janert

FOREWORDS BY COLIN D. KELLEY AND THOMAS WILLIAMS

MANNING

Gnuplot in Action
by Philipp K. Janert

Chapter 13

brief contents

13 Fundamental graphical methods

This chapter covers

- Investigating relationships
- Representing counting statistics
- Visualizing ranked data
- Exploring multivariate data

In this chapter and the next, I want to shift my attention: I'll now largely take gnuplot for granted, and concentrate on *applying* it to problems. Nevertheless, whenever appropriate, I'll take the opportunity to show you how a certain effect can be achieved with gnuplot. In this chapter I want to talk more generally about different graphical methods and the kinds of problems they're applicable to. In the next chapter, I'm going to take a number of different problems and walk you through the different steps that the analysis may take. If you will, this chapter introduces *la technique*, while the next chapter explains *la méthode* (with a nod to Jacques Pépin).

When faced with a new data set, there are two questions that usually dominate. The first one is, how does one quantity depend on some other quantity—how does

y vary with x? The second question (for data sets that include some form of statistical noise) asks, how is some quantity distributed—what's the character of its randomness? We'll look at graphical methods suitable for either question in the next two sections. In the last two sections of this chapter, I'll discuss two particularly challenging problems: ranked data, and methods applicable to large, unstructured, multivariate data sets.

13.1 Relationships

For many data sets, we're interested in the question of whether one quantity depends on another, and if so, how: does y grow as x grows, or does it fall, or does y not depend on x to begin with?

13.1.1 Scatter plots

A *scatter plot* is the first step in finding the answer. In a scatter plot, we just show unconnected symbols, located at the position given by x and y. It's an easy way to get a feeling for an otherwise unknown data set.

AN EXAMPLE: CAR DATA

Listing 13.1 shows the first few lines from a sample data set, containing 26 attributes for 205 different car models that were imported into the US in 1985.[1] The 14th column gives the curb-weight in pounds, and the last (26th) column the price (in 1985 dollars). We can use a scatter plot as in figure 13.1 to see how weight varies as a function of price.

In this case, the input file isn't whitespace separated, but comma separated. Instead of transforming the input file to space separated, it's more convenient to use gnuplot's `set datafile separator` option to plot this data file:

```
set datafile separator ","
plot "imports-85.data" u 26:14
```

Listing 13.1 A few lines from the Automobile data set (truncated)—see figure 13.1

```
1,158,audi,gas,turbo,four,sedan,fwd,front,105.80,192.70,71.40,55.90,...
0,?,audi,gas,turbo,two,hatchback,4wd,front,99.50,178.20,67.90,52.00,...
2,192,bmw,gas,std,two,sedan,rwd,front,101.20,176.80,64.80,54.30,2395...
0,192,bmw,gas,std,four,sedan,rwd,front,101.20,176.80,64.80,54.30,239...
0,188,bmw,gas,std,two,sedan,rwd,front,101.20,176.80,64.80,54.30,2710...
...
```

We can clearly see that weight goes up as the price increases, which is reasonable. We should also note that there are many more low-price/low-weight cars than heavy, premium vehicles. For budget cars, weight seems to increase in step (linearly) with price for a while, but for higher-priced vehicles, the gain in weight levels off. This observation may have a simple explanation: the price of mass market vehicles is largely

[1] This example comes from the "Automobile" data set, available from the UCI Machine Learning Repository: Asuncion, A. and Newman, D.J. (2007). UCI Machine Learning Repository [http://www.ics.uci.edu/~mlearn/MLRepository.html]. Irvine, CA: University of California, School of Information and Computer Science.

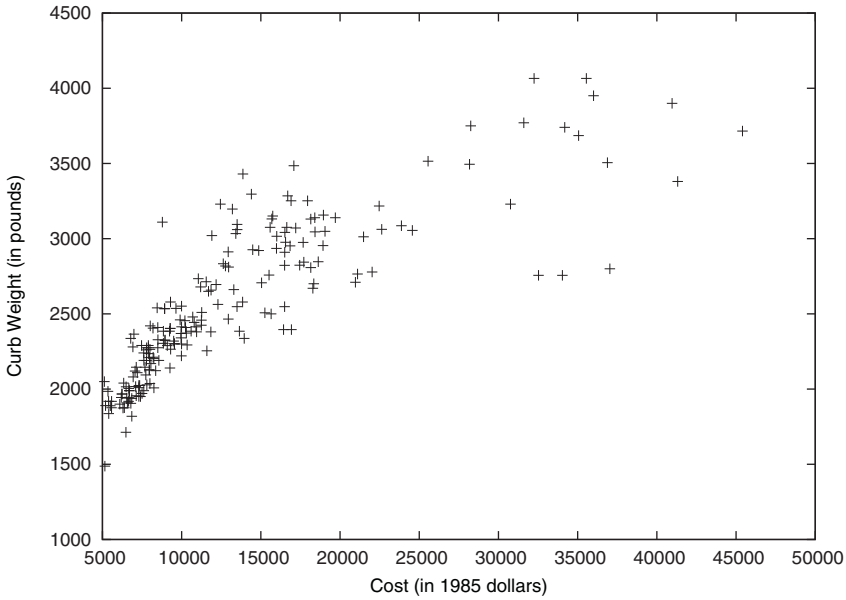

Figure 13.1 Curb weight versus price for 205 different cars. See listing 13.1.

determined by the cost of materials, so that a car that's twice as big (as measured by its overall mass) is also twice as expensive, whereas the price of luxury cars is determined by higher quality (fancier materials such as leather seats, and additional options such as more electronics), rather than by sheer bulk.

It's tempting to try to find a mathematical model to describe this behavior, but the truth of the matter is that there's not enough data here to come to an unambiguous conclusion. Various functions of the form $a\,(x\text{-}b)^{1/n} + c$ or even $a \log(x\text{-}b) + c$ fit the data about equally well, but the data alone doesn't allow us to determine which one would be the "correct" model.

USING SCATTER PLOTS

This example demonstrates what to look for when examining a scatter plot. The first question usually concerns the nature of the relationship between x and y. Does y fall as x grows or vice versa? Do the points fall approximately onto a straight line or not? Is there an oscillatory component? Whatever it is, take note of it.

The second question concerns the strength of the relationship, or, put another way, the amount of noise in the data. Do the data points jump around unpredictably as you go from one x value to the next? Are there outliers that seem to behave differently than the majority of the points? Detecting outliers is important: gone unnoticed, they'll mess up most statistical quantities (such as the mean) you may want to calculate later. And sometimes outliers indicate an interesting effect—maybe some subgroup of points follows different rules than the majority. Outliers in scatter plots should never go uninvestigated.

A third aspect to look out for in a scatter plot is the distribution of points in either dimension. Are points distributed rather uniformly, or do they cluster in a few locations? If so, do we understand the reason for the clustering, or is this something we need to investigate further? There may be a lot of information even in a humble scatter plot!

A MORE COMPLICATED EXAMPLE: THE 1970 DRAFT LOTTERY

Be warned that correlations aren't always trivial to detect. Figure 13.2 shows a famous data set, which I'll explain in a minute. But first, what do you think: is there a correlation between x and y?

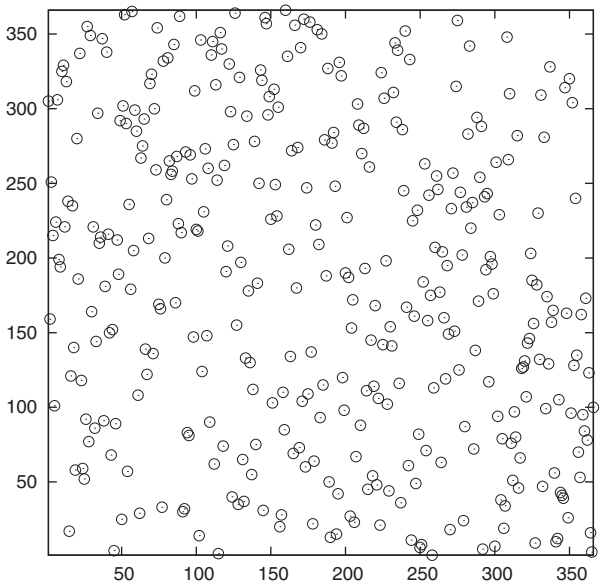

Figure 13.2 Is there any correlation between x and y in this data?

Here's the story behind the data: during the Vietnam war, men in the US were drafted into the armed forces based on their birth dates. Each possible birth date was assigned a *draft number* from 1 to 366, and men were drafted in order of their draft numbers. To ensure fairness, draft numbers were assigned to birth dates using a lottery process. Yet, allegations quickly arose that the lottery was biased, such that men born later in the year had a higher chance of being drafted early.

Figure 13.2 shows the draft numbers (as they'd been assigned by the lottery process) as a function of the birth dates. If the lottery had been fair, there should be no detectable pattern in the data.

Figure 13.3 shows the same data, but this time together with two interpolation curves, drawn using `plot ... smooth`. The curves clearly slope downward, indicating that there's a trend in the data: the later in the year the birth date falls, the lower (on average) the draft number. It was later found that the procedure used in the lottery process to mix entries was insufficient to achieve true randomness. In

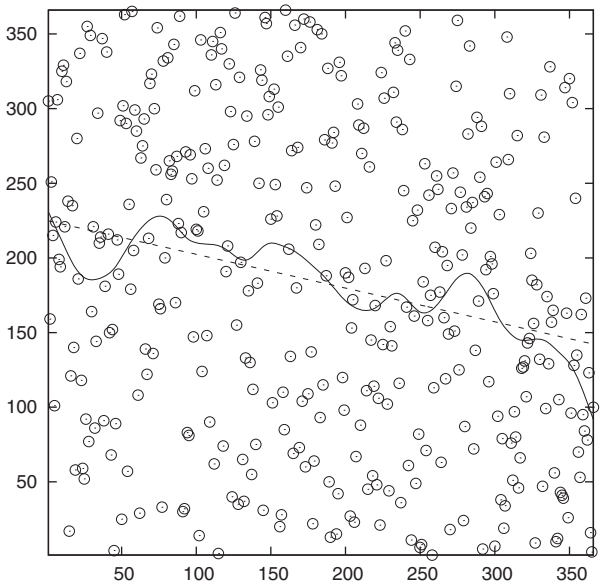

Figure 13.3 The 1970 draft lottery. Birth date (as day after Jan 01st) on the horizontal axis, draft number on the vertical axis. The lines are weighted spline approximations, with different weights. The data is the same as in figure 13.2.

later draft lotteries, this process was improved and the lottery produced truly random results.[2]

Using an interpolating line as in figure 13.3 can be a useful tool to discover otherwise invisible behavior when the input data is very noisy. It's often useful when the "stiffness" of the interpolating line can be varied continuously: If the line is very soft, it'll follow all the bumps in the data; if it's too stiff, if may flatten out relevant features in the data set. Iteration, visual inspection, and judgment are critical. In the figure, I've used gnuplot's `smooth acsplines` weighted spline feature, with two different weights: 10^{-4} for the wobbly line and 10^{-15} for the straight line (`plot "data" using 1:2:(1e-4) smooth acsplines`). The smaller the weight, the less each individual data point influences the local shape of the curve. Therefore, as the weight goes to zero, the approximation becomes increasingly global, just showing the overall trend. For more information on using locally smooth approximations to detect features and trends in data, you might want to check out the *Lowess* (or *Loess*) family of algorithms. Cleveland's books mentioned in appendix C are a good starting point.

A NOTE ON SPLINES

Splines are a way to provide a smooth approximation to a set of points. The points are called *knots*.[3]

---

[2] The 1970 draft lottery is a famous example in statistical analysis and has been analyzed in many places, for example in the introductory textbook *Introduction to the Practice of Statistics* by D. S. Moore and G. P. McCabe. The description of the lottery process can be found in *The Statistical Exorcist* by M. Hollander and F. Proschan and is well worth reading. The raw data can be found in StatLib's Data and Story Library at http://lib.stat.cmu.edu/DASL/Stories/DraftLottery.html.

[3] I'd like to thank Lucas Hart for helpful correspondence regarding this topic.

Splines are constructed from piece-wise polynomial functions, which are joined together in a smooth fashion. In the case of *interpolating* splines, the resulting curve is forced to pass *exactly* through all knots; in the case of *smoothing* or *approximating* splines, the resulting curve will in general *not* pass through the individual knots. Because in the latter case the curve doesn't have to pass through any points exactly, it can be less wiggly.

Both interpolating and approximating splines must fulfill the same smoothness conditions, but in addition, the approximating spline must strike a balance between the following two conditions:

- Passing close to the knots
- Not being too wiggly

These conditions are expressed in the following functional, which is minimized by the approximating spline $s(x)$:

$$J[s] = \int (s'')^2 \, dx + \Sigma_i^N \, w_i \, (\, s(x_i) - y_i \,)^2$$

where (x_i, y_i) are the coordinates of the knots, the w_i are the weights attached to each knot, and the prime indicates a derivative with respect to x. In this functional, the first term is large if $s(x)$ is wiggly, and the second term is large if $s(x)$ doesn't pass close to the knot points. (The form of the first term comes from a physical analogy: if the spline were made out of a real material, such as a thin strip of wood or metal, the first term would be related to the total bending energy of the strip.)

The balance between these two terms is controlled through the weight parameters w_i: if the w_i are small, the first term dominates, and the resulting spline approaches a straight line (which happens to coincide with the least-squares linear regression line for the set of knots). If the weights are large, the second term dominates and the spline approaches the interpolating spline (which passes exactly through all knots).

Another way to think about the weights is to write $w_i = 1/d_i^2$, where d_i is a measure for the uncertainty in the data of point i (such as the standard deviation in this point). We'd expect that the spline will pass through the interval $[y_i-d_i, y_i+d_i]$ at x_i. The higher our confidence in one point, the smaller we can choose this interval, and therefore the larger the weight w_i will be. By choosing $d_i = 0$ for one of the points, we can even force the curve to pass through this point exactly, although we can let the spline float more freely for the other points. Although we may choose different weights for each point, we can also use the same weight for all points, if we know all points to the same accuracy. This is what was done for all examples in this book.

One important remark: the way $J[s]$ is written in our example, the size of the second term depends on the number of knots—if you double the number of knots, the size of the second term will be (approximately) twice as large. By contrast, the first term does *not* depend on the number of knots. If the number of knots grows, the second term will therefore become larger relative to the first one, and the resulting spline will be more wiggly.

To maintain the original balance between wigglyness and closeness of approximation, the weights must be increased accordingly for data sets containing a larger

number of points. Equivalently, you might want to take the number of points into account explicitly by writing $w_i = u_i/N$, where u_i is the actual weight and N is the number of knots. With this choice for w_i, the balance between both terms will be maintained regardless of the number of knots in the data set. [4]

SCATTER PLOTS WITH LABELS: CAR DATA, AGAIN

We can add an additional dimension to a scatter plot by using gnuplot's `with labels` style. Let's come back to the car data from listing 13.1. The data set contains many more attributes than just weight and price. Column 4, for instance, gives the type of fuel used: gas or diesel. We want to include this information in our graph. Maybe diesel-powered vehicles tend to be heavier at a given price point?

We could split the data set apart (using external text-editing tools) into one set containing all diesel engines and one containing all others, and then plot these two data sets using different plot symbols. For a presentation graph, that's exactly what we should be doing, but while we're still experimenting, that's awfully inconvenient. Instead, we'll use the information in the column itself as part of the graph.

We could simply use the value of the fourth column as plotting symbol: `plot "imports-85.data" u 26:14:4 w labels`, but there are too many records in the data set so that the strings would start overlapping each other badly. So instead, we just make the diesel cars stand out more (see figure 13.4), using gnuplot's string functions:

```
plot "imports-85.data" u 26:14,
➥ "" u 26:14:(stringcolumn(4) ne 'gas' ? 'D' : '') w labels
```

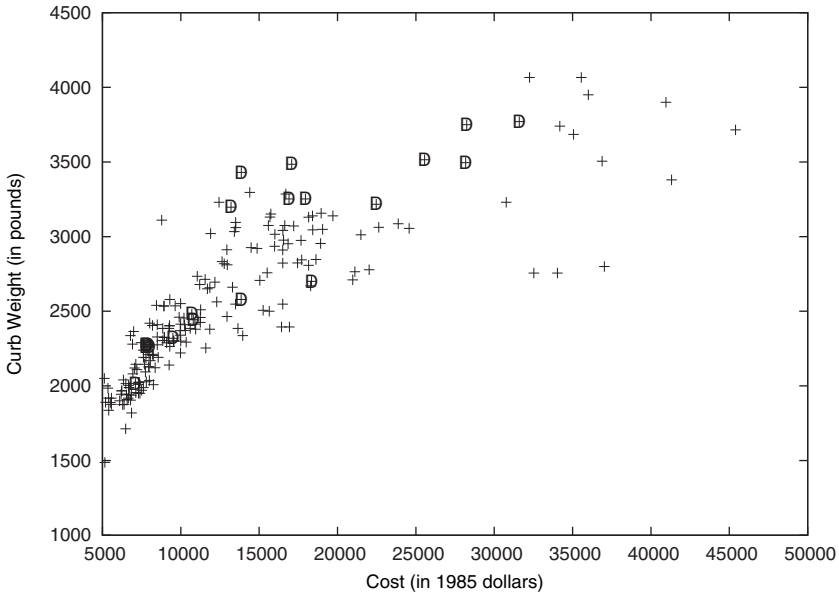

Figure 13.4 Curb weight of cars as a function of their price. Cars with diesel engines are indicated with the letter *D*. Note the distribution of diesel engines relative to gas-powered cars.

[4] More information on splines can be found in chapter 1 of *Handbook on Splines for the User* by E. V. Shikin and A. I. Plis (CRC Press, 1995).

The `stringcolumn(4)` function returns the value of column 4 *as a string*, which is then compared to the standard fuel (namely "gas"). Only if the fuel isn't regular gasoline, a text label ("D" for diesel) is placed onto the graph in addition to the usual plot symbol.

And, yes, overall diesel-powered vehicles seem to be slightly on the heavy side. We should also take note that diesel is most prevalent in the mid-price sector: there are a few cheap diesels, but none of the true luxury cars use it.

13.1.2 Logarithmic scales

Logarithmic scales are one of the most versatile tools in the graphical analyst's toolbox. I introduced them already in section 3.6.1 and discussed how they work. Now let's put them into action.

Logarithmic scales serve three purposes when plotting:

- They rein in large variations in the data.
- They turn multiplicative deviations into additive ones.
- They reveal exponential and power-law behavior.

To understand the meaning of the first two items, let's study the daily traffic pattern at a web site. Figure 13.5 shows the number of hits per day over approximately three months. There's tremendous variation in the data, with alternating periods of high and low traffic. During periods of high traffic, daily hit counts may reach close to half a million hits, but then fall to very little traffic shortly thereafter. On the scale of the graph, the periods of low traffic seem barely different from zero, with little fluctuation.

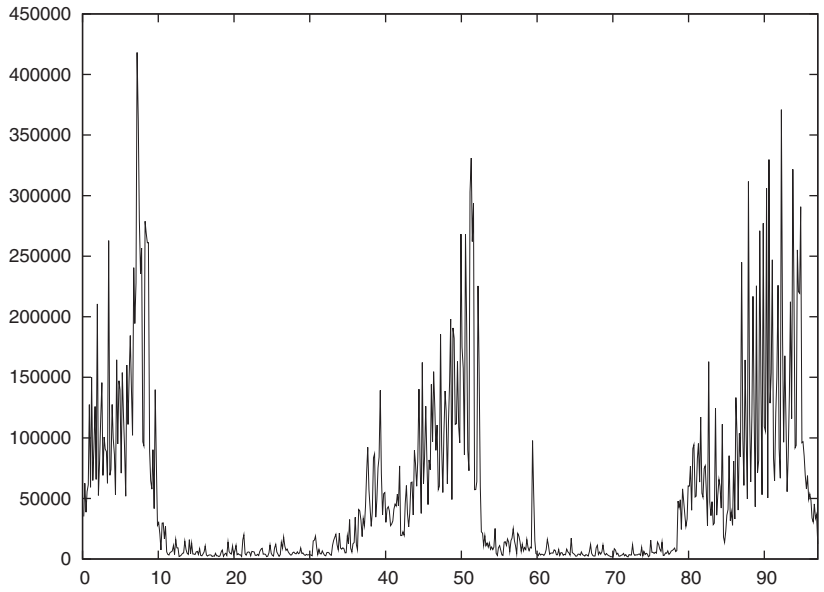

Figure 13.5 Traffic patterns at some web site. Daily hit count versus day of the year. Note the extreme variation in traffic over time.

In figure 13.6 we see the same data, but now on a semi-logarithmic scale. The logarithmic scale helps to dampen the extreme variation of the original data set (two orders of magnitude), so that we can now see the structure both during the high- and the low-traffic season. That's the first effect of logarithmic plots: they help to make data spanning extreme ranges visible, by suppressing high-value outliers and enhancing low-value background.

Furthermore, we can see that the *relative* size of the day-to-day fluctuations is about equal during both phases. The absolute size of the fluctuations is quite different, but their size as a percentage of the average value is roughly the same (very approximately, during low season, traffic varies between 2,000 and 20,000 hits a day, a factor of 10; whereas during high season it varies between 30,000 and 300,000 hits a day, again a factor of 10). That's the second effect of logarithmic plots: they turn multiplicative variations into additive ones.

Figure 13.7 tries to demonstrate the last point in a different way. The bottom panel shows the web traffic on consecutive days (like figure 13.5), displaying great seasonal variance, but the top panel shows the ratio of the difference in traffic on consecutive days divided by the actual value—*(current day - previous day)/current day*—which does *not* exhibit a seasonal pattern: further proof that the daily fluctuation, viewed as a percentage of the overall traffic, is constant throughout.

Finally, let's look at a curious example that brings together two benefits of logarithmic plots: the ability to display and compare data of very different magnitude, and the ability to turn power-law behavior into straight lines.

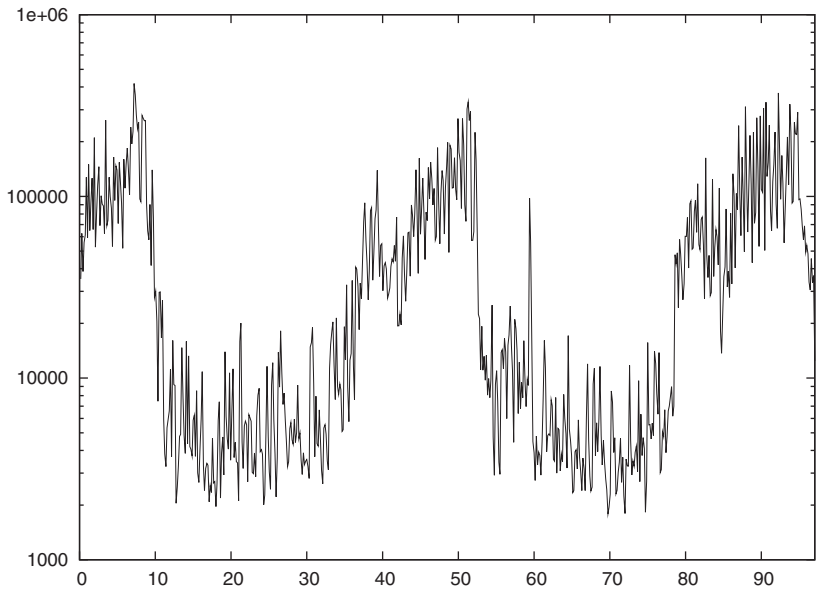

Figure 13.6 The same data as in figure 13.5, but on a semi-logarithmic scale. Note how the high-traffic outliers have been suppressed and the low-traffic background has been enhanced. In this presentation, data spanning two orders of magnitude can be compared easily.

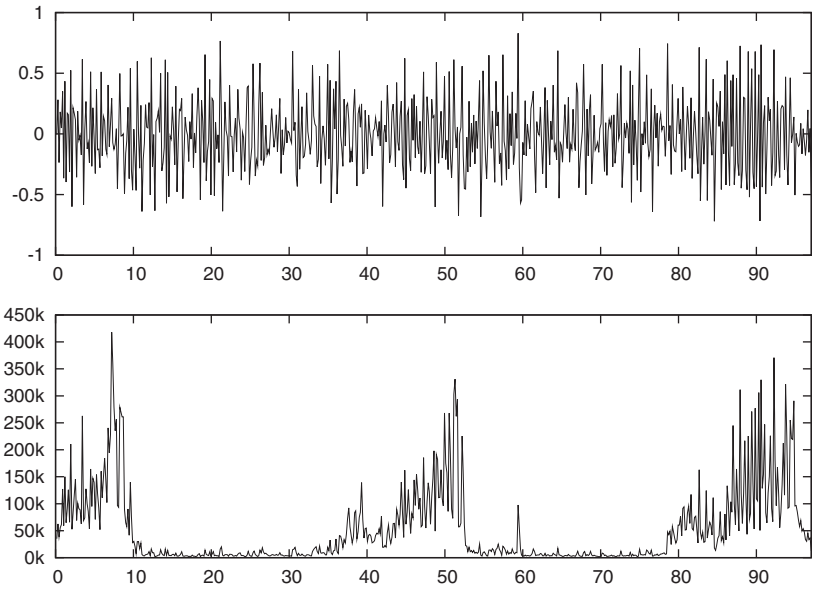

Figure 13.7 Bottom panel: hits per day over time (as in figure 13.5); top panel: change in traffic between consecutive days, divided by the total traffic. Note how the relative change (top panel) doesn't exhibit any seasonal pattern, indicating that the relative size of the variation is constant.

Mammals come in all shapes and sizes, from tiny rodents (the smallest known land mammal is the Pygmy Shrew, which weighs only a few grams, but some bats found in Thailand are apparently smaller still) to the largest of whales (weighing several hundreds of tons). It's a curious empirical fact that there seem to be fixed relationships between different metabolic quantities—basically, the larger an animal is, the slower its bodily functions progress. Figure 13.8 shows an example: the duration (in seconds) of a single resting heartbeat, as a function of the typical body mass. The regularity of the data is remarkable—spanning *eight orders of magnitude* for the mass of the animal. What's even more amazing is how well the data is represented by the simple function $T \sim m^{1/4}$. This law isn't limited to the examples shown in the graph: if you added further animals to the list, they'd also fall close to the straight line (I didn't just pick the best ones).

The existence of such scaling relations in biological systems has been known for a long time and seems to hold generally. For example, it turns out that the typical lifetime of a mammal also obeys a quarter-power scaling law relation against the body mass, leading to the surprising conclusion that the total number of heartbeats in the life of a single organism is fixed—no matter what the typical resting heart rate is. (In case you care, the number comes out to about 1.5 billion heartbeats during a typical lifetime.)

Recently these observations have been explained in terms of the geometrical constraints that must exist in the vascular networks (the veins and arteries), which supply

nutrients to all parts of the organism.[5] As it turns out, you can derive the quarter-power scaling laws starting from only three simple assumptions, namely that the support network must be a space-filling fractal, reaching all parts of the organism; that the terminal capillaries where nutrients are actually exchanged are the same size in all animals; and finally that organisms have evolved in such a way that the energy required for the transport of nutrients through their bodies is minimized. I think it's amazing how such a powerful result can be derived from such simple assumptions, but on the other hand, we shouldn't be surprised: generally applicable laws (such as the quarter-power scaling in this example) must stem from very fundamental assumptions disregarding any specifics.

Let's come back to figure 13.8. The double-logarithmic scales make it possible to follow the data over eight orders of magnitude. (Had we used linear scales, all animals except for the whale would be squished against the left side of the graph—literally crushed by the whale.) So again, logarithmic scales can help to deal with data spanning a wide range of values. In addition, the double-logarithmic plot turns the power law relationship $T \sim m^{1/4}$ into a straight line and makes it possible to read off the exponent from the slope of the line. I explained how this works in detail in section 3.6.1 and won't repeat it here.

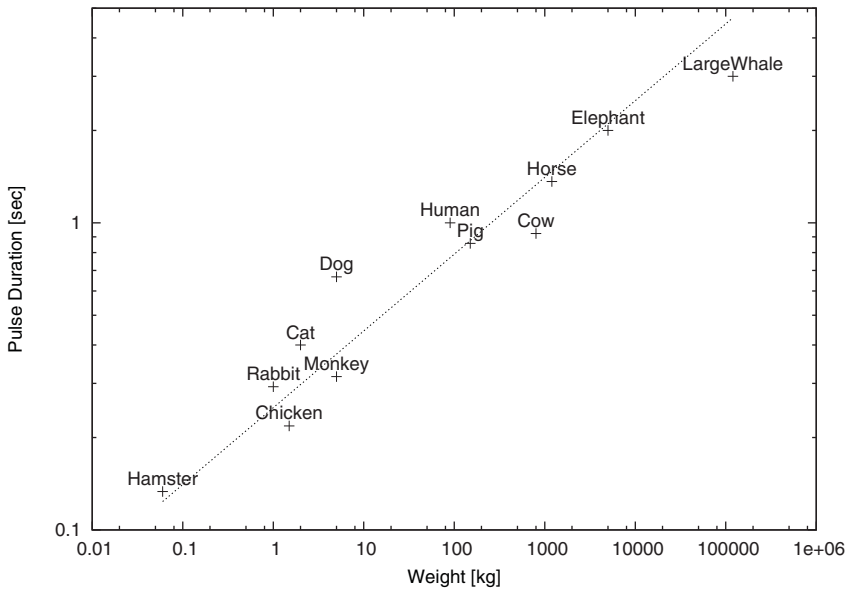

Figure 13.8 Allometric scaling: the duration of a average resting heartbeat as a function of the typical body mass for several mammals. Note how the data points seem to fall on a straight line with slope 1/4.

[5] The original reference is the paper "A General Model for the Origin of Allometric Scaling Laws in Biology" by G. B. West, J. H. Brown, B. J. Enquist in the journal *Science* (Volume 276, page 122 (1997)). Additional references can be found on the web.

Finally, figure 13.8 is a nice example for the power of gnuplot's `with labels` plot style. The graph was generated using

```
plot "mammals" u 2:3 w points, "" u 2:(1.1*$3):1 w labels
```

The first part of the command draws the symbols (`with points`); the second adds the labels. All the labels are shifted a bit upward so as not to obscure the symbols themselves. In this example, the vertical offset is *multiplicative*, because of the logarithmic scale of the graph (remember: logarithms turn multiplicative offsets into linear ones).

13.2 Counting statistics

Besides detecting relationships between quantities, we may want to understand how data points that are somehow random are distributed. Are data points spread out evenly or are they clustered in a few spots? Are distributions symmetric or are they skewed? How much weight is contained in the tails of a distribution, compared to its center?

Let's say we have a file containing a set of measurements—these can be anything: interarrival times for requests at a web server, completion times of database queries, weights of potatoes, heights of people—whatever. What can we say about them?

13.2.1 Jitter plots and histograms

One easy way to get a visualization of a collection of random points is to generate a *jitter plot*, which is really a one-dimensional scatter plot, but with a twist (as in the bottom part of figure 13.9).

This graph was created by shifting each data point vertically by a random amount. (The `rand(0)` function returns a random number in the range [0:1].) If we'd just plotted the data in a true, one-dimensional fashion, too many of the points would've overlapped, making it difficult to detect clustering. Such jittering by a random amount is a good trick to remember whenever creating scatter plots of larger data sets!

```
plot "random-points" u 1:(0.25*rand(0)-.35)
```

We can see that the distribution of points is skewed. It's strictly bounded by zero on the left, with points clustering around one, and as we move to the right, points become increasingly sparse. But it's hard to say something more definite by just looking at the jitter plot. For instance, is there a second cluster of points between three and four? This does seem possible, but it's hard to tell for sure using this representation.

The next step when investigating the properties of a distribution usually involves drawing a histogram. To create a histogram, we assign data points to buckets or *bins* and count how many events fall into each bin. It's easiest to make all bins have equal width, but with proper normalization per bin, we can make a histogram containing bins of differing widths. This is sometimes useful out in the tails of a distribution where the number of events per bin is small.

Gnuplot doesn't have an explicit histogramming function, but we can use the `smooth frequency` functionality (see section 3.2) to good effect. Recall: `smooth frequency` sorts the x values by size, and then plots the sum of y values per x value. That's what we need to build a histogram.

In the following code, I introduce a function `bin(x,s)` of two parameters. The first parameter is the x value we'd like to bin, and the second parameter is the bin width. Note that the bins obtained in this way are flush left—you can use the `binc(x,s)` function for bins centered at the bin value.

The `smooth frequency` feature forms the sum of all y values falling into each bin. If all we care about is the overall shape of the histogram, we may supply any constant, such as `(1)`, but if we want to obtain a normalized histogram (one including a total surface area equal to unity), we need to take into account the number of points in the sample and the bin width. You can convince yourself easily that the proper y value for a normalized histogram is

*1/(bin-width * number-of-points-in-sample)*

We can use the `with boxes` style to draw a histogram (see figure 13.9), but we want to fix the width of the boxes in the graph to coincide with the bin width. (By default, the boxes expand to touch their neighbors, which leads to a faulty graphical representation if some of the internal bins are empty.) The bin width is 0.1 and there are 300 points in the sample. (We have to count them ourselves—unfortunately, gnuplot currently doesn't have the ability to report the number of records in a data file.)

```
bin(x,s)  = s*int(x/s)
binc(x,s) = s*(int(x/s)+0.5)

set boxwidth 0.1
plot "random-points"
➥ u (bin(1,0.1)):(1./(0.1*300)) smooth frequency with boxes
```

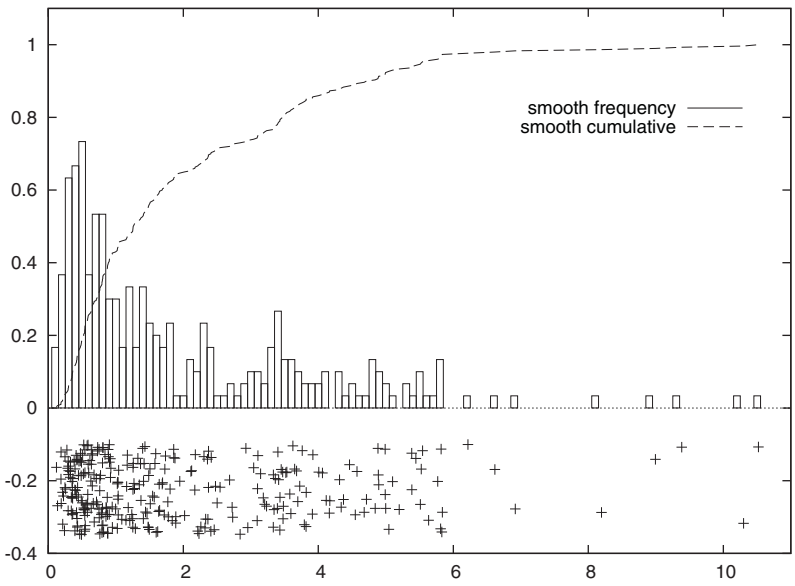

Figure 13.9 Three ways to represent a distribution of random points: jitter plot (bottom), histogram (with boxes), and cumulative distribution function (dashed line)

Before leaving this section, I should mention another graphical device you may encounter in the literature, the so-called *box-and-whiskers* plot or *box plot* for short. Basically, a box-plot is similar to the `with candlesticks` style (see section 5.2.3), with the box representing the upper and lower quartiles (see section 13.2.4 if you aren't familiar with percentiles). The "whiskers" extending from the central box are a measure of the outliers and may represent, for example, the 10 and 90 percent percentiles.

I must say that I never use box plots for the purpose of representing a single distribution of points. They give you all the hassles of a graph, but don't add much information that couldn't be expressed by the sheer percentile numbers alone. Furthermore, if I'm looking for a graphical representation, a histogram or even a jitter plot tells me so much more than a box plot. This comment doesn't apply to situations where I want to compare a large number of distributions, such as the time series plots, where box plots can be very useful.

13.2.2 Kernel density estimates

The apparent simplicity of the histogramming method hides some pitfalls. The first concerns the width of the bins: make them too narrow and the resulting histogram will be bumpy; make them too wide and you lose relevant features. There's also ambiguity in regard to the placement of the bins: is the first bin centered at zero (or any other value) or flush left there? The overall appearance of the histogram can depend quite sensitively on these details!

A better method to generate distribution curves from individual data points goes under the name *kernel density estimation*. Rather than counting how many data points fall into each bin, we place a strongly peaked, but smooth function (a *kernel*) centered at the location of each data point. We then sum the contributions from all these curves and plot the result. Mathematically, the kernel estimate $f(x)$ for a data set consisting of N points x_i is

$$f(x) = (1/N)\Sigma_i^N \ (1/h)\, K((x-x_i)/h)$$

Here, $K(x)$ is any smooth, peaked, normalized function, and h is the *bandwidth*: a measure of the width of the kernel function. A popular example is the Gaussian kernel:

$$K(x) = (2\,\pi)^{-1/2} \exp(-x^2/2)$$

The current development version of gnuplot (version 4.3) contains code to generate such curves, using the `smooth kdensity` functionality. It works in much the same way as the `smooth frequency` feature we saw earlier:

```
plot "random-points" u 1:(1./300.):(0.05) smooth kdensity
```

The first column specifies the location; the second gives the weight each point should have. For a normalized histogram, this should be the inverse of the number of data points—since the kernel functions are normalized themselves, you don't have to worry about the bandwidth at this point as you did for histograms using `smooth frequency`. The third parameter is optional and fixes the bandwidth of the kernels. If it's omitted (or negative), gnuplot calculates a default bandwidth, which would be

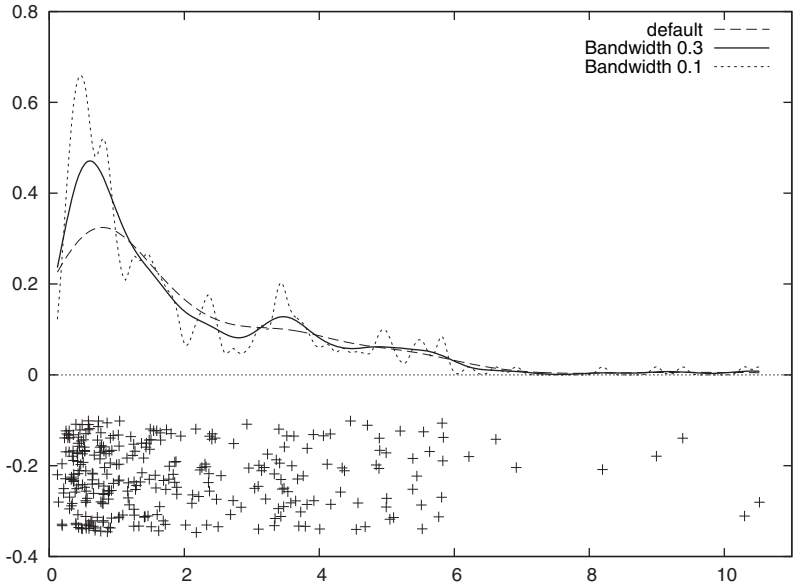

Figure 13.10 An alternative to histograms: kernel density estimates using `smooth kdensity`. Curves for three different bandwidths are shown. A bandwidth of 0.3 seems to give the best trade-off between smoothing action and retention of details. Note how it brings out the secondary cluster near x=3.5.

optimal if the data were normally distributed. This default bandwidth tends to be quite conservative (that means, rather broad).

Figure 13.10 shows several curves drawn using `kdensity` for the same data set we've already seen in figure 13.9, for a variety of bandwidth parameters. Studying this graph carefully, we may conclude that there's indeed a second cluster of points, located near 3.5. Note how the choice of bandwidth can hide or reveal features in the distribution of points.

13.2.3 Cumulative distribution functions

Histograms and density estimates have the advantage of being intuitive: they show us directly the probability for a certain value to occur. But they have some disadvantages when it comes to making quantitative statements. For example, based on the histogram in figure 13.9, it's hard to determine how much "weight" is in the tail of the distribution: how likely are values larger than 4 to occur? How about values larger than 6? We can guess that the probability will be small, but it's hard to be more precise. To answer such questions, we need to know the *area under the histogram* within certain bounds. In other words, we want to look at the *cumulative distribution function* (or simply *distribution function* for short).

The value of the cumulative distribution function at position x gives us the fraction of events that have occurred with x_i less than x. In figure 13.9, I already showed the distribution function together with the histogram. To repeat: the value of the

cumulative distribution function at position x is equal to the area under the (normalized) histogram from its left border to the position x.

Cumulative distribution functions are part of the current development version of gnuplot (version 4.3), and are accessible using `smooth cumulative`. The `smooth cumulative` feature is similar to `smooth frequency`: first, all points are sorted in order of ascending x value, then the sum of all y values to the left of the current position is plotted as smoothed value. To obtain a normalized distribution function, we must supply $1/number\text{-}of\text{-}points$ as y value. In contrast to histograms or density estimates, distribution functions don't depend on a width parameter.

```
plot "random-points" u 1:(1./300.) smooth cumulative
```

Cumulative distribution functions can be a little unintuitive at first, but they're well worth becoming familiar with. They make it easy to answer questions such as those raised at the beginning of this section. From figure 13.9 we can immediately see that there's a 3 percent chance of finding a point at $x > 6$ and about a 15 percent chance for $x > 4$. We can also find more proof for the second cluster of points between 3 and 4: at this point, the distribution function seems to make a jump, indicating an accumulation of points in this interval.

The cumulative distribution function is sufficiently useful that you may want to dump it into a file, in order to study it in its own right. Listing 13.2 shows how to do that and also demonstrates a good graphical method to check whether the points in a data set are normally (Gaussian) distributed. If they are, the resulting plot will be a straight line. Moreover, we can read off the width of the Gaussian from the slope of the line and the location of the Gaussian from the intersection of the line with the y axis. (Here's how it works: if the data points were normally distributed, then the cumulative distribution function would be the Gaussian one, so that $y = \Phi((x\text{-}\mu)/\sigma)$, where $\Phi()$ stands for the Gaussian distribution function, μ is the mean and σ is the standard deviation. Now operate on both sides with the inverse of the distribution function: $\Phi^{-1}(y) = (x\text{-}\mu)/\sigma$. Rearranging terms, we end up with $\sigma\, \Phi^{-1}(y) + \mu = x$. A similar argument also holds for non-Gaussian distributions functions, although few of them have a closed form for the inverse of their distribution functions, and μ and σ need to be taken as the appropriate location and scale parameters for the distribution under consideration.)

Listing 13.2 Testing whether a cumulative distribution function is normal

```
plot "data" u 1:(1./100.) s cumul  # Distribution function of
                                   #     original data

set table "cdf" # Re-direct output to file 'cdf'
replot          # ... write to file
unset table     # ... and switch file output off again.

plot "cdf" u (invnorm($2)):1 w l   # Should be straight line
                                   #     if data is Gaussian
```

13.2.4 Consider using median and percentiles

When faced with a random distribution of points or events, it's natural to look for some form of summary statistics that'll give a good estimate for the *location* of the distribution and its *spread*. The best-known measures of location and spread are the *mean* and the *standard deviation*:

$$\mu = (1/N) \Sigma_i^N x_i$$

$$\sigma = [\ (1/N) \Sigma_i^N (x_i - \mu)^2\ ]^{1/2}$$

One major reason for their popularity is the relative ease with which they can be calculated. Both can be found by iterating over the data set, updating the sum $\Sigma_i^N x_i$ and the sum of squares $\Sigma_i^N x_i^2$ (it's not hard to show that the standard deviation can be found from the sum and the sum of squares: $\sigma^2 = (1/N) (\Sigma_i^N x_i^2) - \mu^2$). In particular, it's never necessary to manipulate the entire data set at once; elements need only be accessed one by one and in any order.

The problem is that mean and standard deviation may not be good estimators of location and spread. If the distribution of points is asymmetrically skewed, the mean won't be a good measure of the location, and if the distribution has so-called *fat tails* (so that relatively many events occur far away from the center), the standard deviation won't be of much value. If the distribution is bi- or multimodal (has more than one peak), both of these measures are basically meaningless.

I therefore strongly recommend that you become familiar with the median and percentile statistics. To find the median, sort all values in ascending order: the median is the element exactly in the middle, so that half of all points are below the median and the other half above it. (This rule holds if the overall number of points is odd. If it's even, take the average of the two points closest to the middle.) Percentiles work in a similar fashion: the *10 percent percentile* is the value below which 10 percent of points fall, and so on. The lower and upper quartiles are the values below which 25 and 75 percent of points fall, respectively.

The median is a much more reliable estimator of the true center of the distribution in the presence of asymmetry than the mean. For the same reason, the quartiles are a better measure of the spread than the standard deviation. The latter are also much less sensitive to the occasional "crazy" outlier (in contrast to the standard deviation).

The problem with median and percentiles is that they're computationally expensive: the entire data set must be read and sorted. In particular, this requirement makes it impossible to process a data set point by point—instead, the entire file must be slurped and processed at once.

Many actual distributions that one encounters in the wild aren't well represented by mean and standard deviation. Skewed and multimodal distributions are the rule rather than the exception. Heavy-tail phenomena occur frequently and their effects tend to be important; outliers, both real and accidental, are widespread and need to be dealt with. Any one of these effects renders mean and variance nearly useless, but the median and quartiles will tend to hold up in these situations. Use them!

13.3 Ranked data

Imagine I give you a list of the countries in the European Union, together with their land area (in square kilometers) and population numbers. How would you represent this information? How would you represent it *graphically*?

The particular challenge here is that *the independent variable has no intrinsic ordering*. What does this mean?

Given the name of a country, the value of the area measure is fixed; hence the name is the independent variable and the area is the dependent variable. We're used to plotting the independent variable along the x axis and observing the behavior of the dependent variable with it. But in this case, there's no natural ordering of the independent variable. Sure, we can order the states alphabetically by their names, but this ordering is entirely arbitrary and bears no relationship on the data. (We wouldn't expect the size of a state to change if we gave it a different name, would we?) Also, the ordering would change if we were to translate the names to a different language. But the information that we want to display depends on the areas, and shouldn't be affected by the spelling of the country names in any way.

For data like this, the only ordering that's intrinsic to the data itself is in the values of the dependent variable. Therefore, a graphical representation of this data should be ordered by the dependent variable, not the independent one. Such plots are often called *dot plots*, but I tend to think of them as *rank-order plots*. Figure 13.11 shows an example.

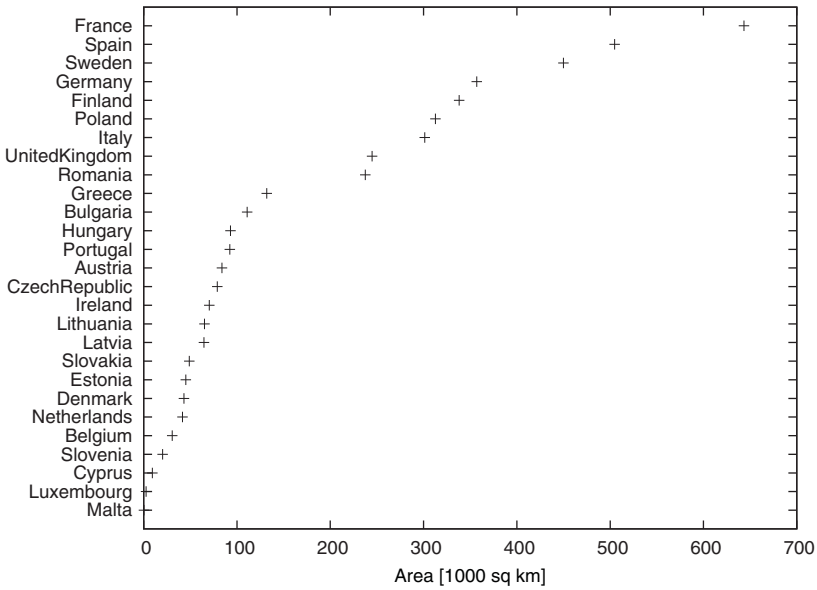

Figure 13.11 A rank-order plot. Because there's no natural ordering in the independent variable (in this case, the country names), we sort the data by the dependent variable to emphasize the structure in the data.

If the input file is sorted by the appropriate quantity, we can generate such plots easily using gnuplot's facility for reading tic labels from the input file. Given an input file containing the names and areas in two columns,[6] such as this:

```
France          643427
Spain           504782
Sweden          449964
Germany         357021
Finland         338145
...
```

the entire plot can be generated using the following command:

```
plot [][26:1] "data" using 2:0:ytic(1)
```

The `ytic(1)` function selects the values in column 1 as tic labels for the y axis (see section 7.3.4), and the pseudocolumn 0, which evaluates to the line number in the input file, is used as the corresponding vertical coordinate (see section 3.4.2). The inverted y range places the first line in the file at the top of the graph instead of the bottom.

This is the basic idea. We could've plotted the state names along the x axis instead, but then we'd need to rotate the labels, to make sure they don't overlap. Unfortunately, rotating the labels by 90 degrees (so that they run vertically) makes them hard to read. A good trick is to rotate them by some angle so that they run diagonally (we'll see an example in figure 13.13). But the initial layout, with the names running down the y axis, is the easiest to read.

What if we want to show and compare multiple data sets, such as the land area and the population? The best strategy is to declare a primary data set, which determines the ordering for all others. In figure 13.12, we can see an example. The points of the secondary data set (the population in millions) have been connected by lines to make them stand out more. Additionally, the x axis has been scaled logarithmically, which is often useful with dot-plots of this sort. We can see that overall the population count follows the area, but there are some notable outliers: the northern Scandinavian countries Sweden and Finland are thinly populated, whereas the so-called Benelux countries (Belgium, Netherlands, and Luxembourg) have an exceptionally high population density.

Dot- or rank-order plots are useful whenever the dependent variable has no natural ordering. On the other hand, if the dependent variable can be ordered, even if it's nonnumeric (such as the categories *Strong Dislike*, *Dislike*, *Neutral*, *Like*, *Strong Like*), we should use the information in our graphs and order data points by the independent variable.

---

[6] All data in this section comes from the *CIA World Factbook*, because I was unable to find this data in a suitable format on the European Union's official web site.

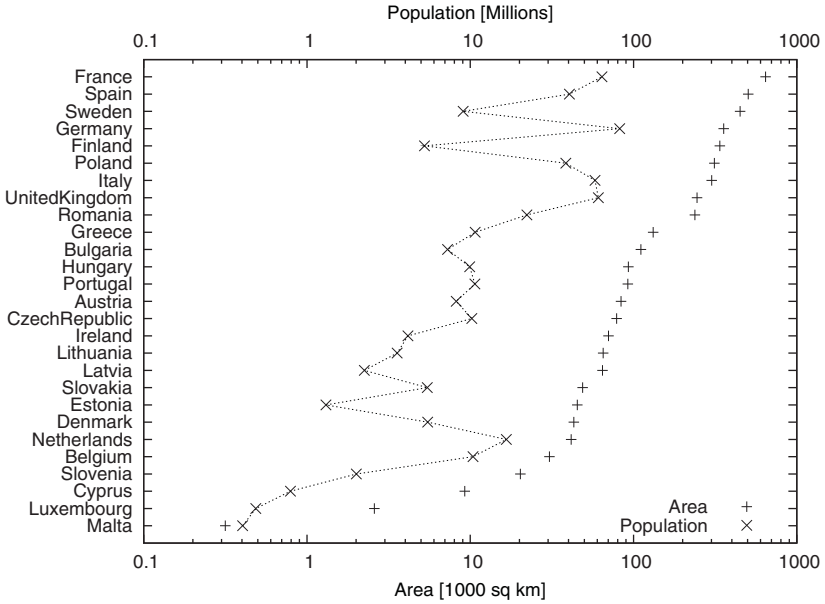

Figure 13.12 A rank-order plot displaying a primary and a secondary data set for comparison. The country names are sorted according to the primary data set (the area); the points in the secondary data are connected by lines to make them easier to distinguish. Note the logarithmic scale for the horizontal axes.

13.4 Multivariate data

Sometimes we don't even know what to look *at*, much less what to look *for*. This problem typically arises for large, somewhat disparate sets of data. For example, later in this section we'll look at a data set containing measurements for more than 200 individual samples of glass. For each bit of glass, nine different quantities have been measured. In such a situation, it's not at all clear where to begin. What quantity should we plot as a function of which other? Which one will tell us the most about the data in the sample? Our first task is therefore to find which quantities are the most relevant. We can then study how the other quantities vary with them. It would also be nice to be able to break the original data set up into a handful of groups, so that the records within each group are somehow similar. But first we'd have to find the criteria by which the data could be classified!

In this section, we study two different graphical methods that have been suggested for problems of this kind: parallel coordinate plots and star plots.

13.4.1 Parallel coordinate plots

The purpose of a parallel coordinate plot is to visualize *all* measurements for a large number of records *simultaneously*. The price we pay is a highly unintuitive and not very pretty graph. A parallel coordinate plot is strictly a tool for graphic discovery, not for presentation or communication.

Let's imagine we have a data set of n records, where each record consists of k measurements of different quantities. Each record is therefore a point in a k-dimensional space: each measured quantity spans a separate *dimension*. To construct a scatter plot, we'd have to pick two (or at most three) of these dimensions as axes of the plot. In a parallel coordinate plot, we instead assign a fixed location along the x axis to each of the measured quantities. For each record, the value of this quantity is taken as the y coordinate at the corresponding x location. All points from a single record are then connected with straight lines.

An example will make this more clear. Let's look at a data set of more than 200 individual samples of glass taken from crime scenes.[7] For each glass sample, nine quantities were measured: the refractive index and the content of substances such as sodium, silicon, iron, and so on. Each of these quantities is assigned a position along the x axis, and the reported value is used as y coordinate. Figure 13.13 shows the resulting plot after only a single record has been plotted; figure 13.14 shows it after all records have been added, with the original record highlighted.

We can now examine this plot for possible structure in the data set. We look for clusters or gaps along one of the marked x values. Those can be used to classify data sets into groups. We can also look for outliers and for correlations among data sets: positively correlated quantities show up as parallel (or nearly parallel) lines, whereas negative correlation is indicated by lines crossing each other.

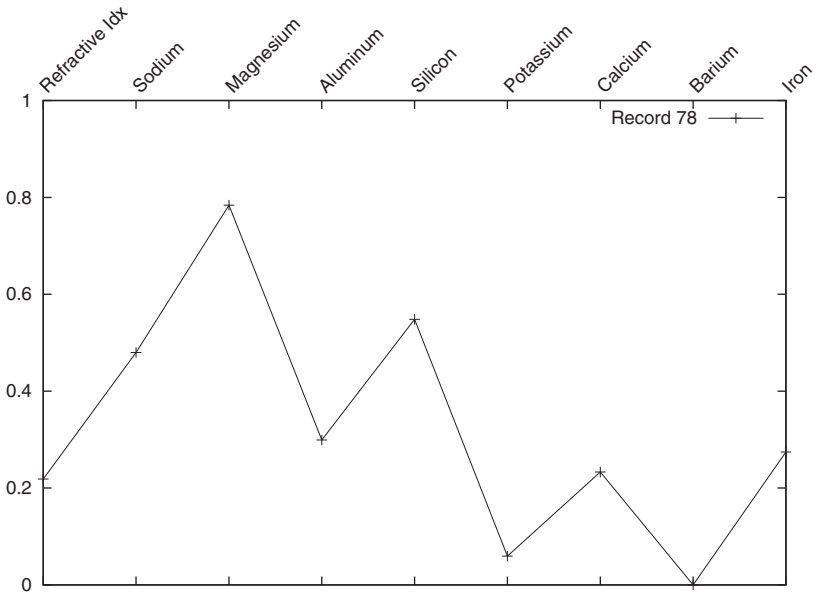

Figure 13.13 A single record in a parallel coordinates plot

[7] This example comes from the "Glass Identification" data set, available from the UCI Machine Learning Repository: Asuncion, A. and Newman, D.J. (2007). UCI Machine Learning Repository [http://www.ics.uci.edu/~mlearn/MLRepository.html]. Irvine, CA: University of California, School of Information and Computer Science.

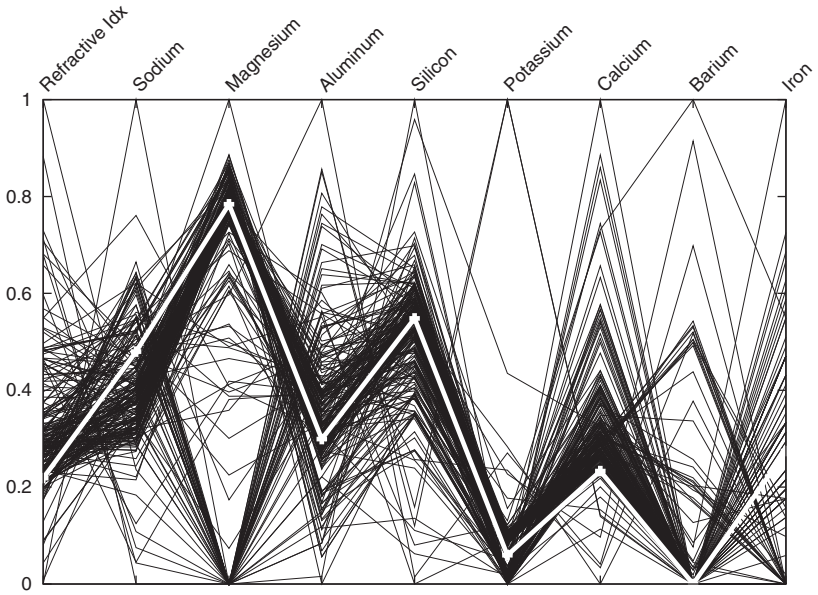

Figure 13.14 All records in a parallel coordinates plot. The record from figure 13.13 is highlighted.

The data set in figure 13.14, for instance, exhibits clustering along the third axis (measuring magnesium content). Taking this as a hint, I separate the records into two sets: one with high magnesium content and one with low magnesium content. In figure 13.15, I show only the records with high magnesium content, which allows us to identify additional characteristics. For example, the records shown in figure 13.15 can be partitioned again based on the potassium content. It also appears as if records with high potassium content have a low calcium concentration. On the other hand, iron exhibits no clustering whatsoever. In this way, we can proceed and detect those criteria (such as high or low magnesium content) that can be used to classify records.

There are a few technical points that need to be discussed. The first concerns the best data input format for this kind of plot. Listing 13.3 shows the first few lines of the original data set. Each row contains one record; the individual measurements are separated by commas. The first entry in a line is the index of that record, followed by the nine measurements. The last entry is a check digit, which we'll ignore.

Listing 13.3 The beginning of the Glass Identification data set

```
1,1.52101,13.64,4.49,1.10,71.78,0.06,8.75,0.00,0.00,1
2,1.51761,13.89,3.60,1.36,72.73,0.48,7.83,0.00,0.00,1
3,1.51618,13.53,3.55,1.54,72.99,0.39,7.78,0.00,0.00,1
4,1.51766,13.21,3.69,1.29,72.61,0.57,8.22,0.00,0.00,1
5,1.51742,13.27,3.62,1.24,73.08,0.55,8.07,0.00,0.00,1
...
```

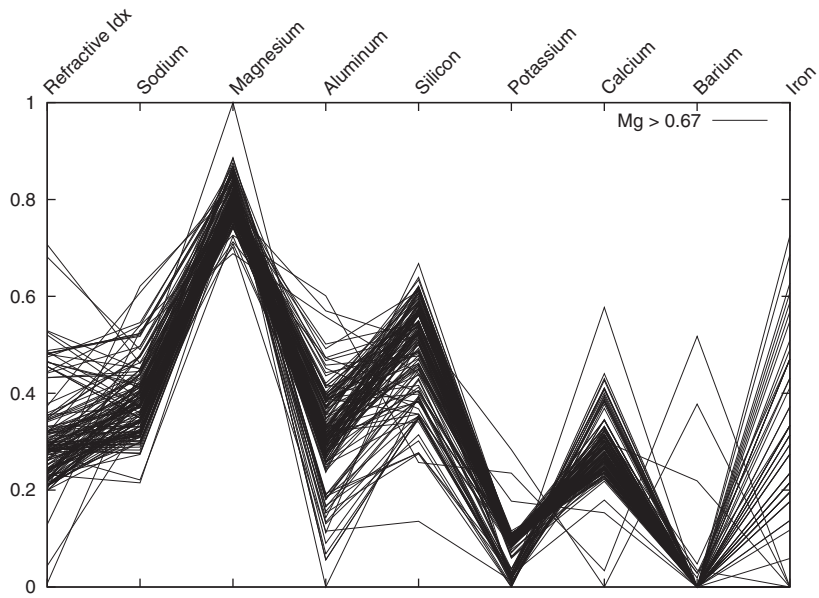

Figure 13.15 A subset of records: only those records from figure 13.14 are shown in which the magnesium concentration exceeds 0.67. Note the secondary structure in this subset: there are two distinct clusters of data characterized by their concentration of potassium and calcium.

This format isn't ideal for the kind of plot we have in mind. It would be much better if all measurements for a single record would form a column instead of a row. I therefore transform the original data set using the short Perl program shown in listing 13.4 to a more suitable format. The first few rows of the transformed data set are shown in listing 13.5.

In the transformed data set, each record has been turned into a data block, with individual data blocks separated from each other using two blank lines. This means that we can now plot each data block individually using the `plot ... index` facility (see section 3.1.1).

The transformation script also rescales the original values to lie in the unit interval [0:1]. Although not strictly necessary, this is usually a good idea to make different measured quantities comparable. Finally, the appearance of a parallel coordinates plot depends on the specific ordering of the dimensions along the x axis. You may want to try out different permutations to see how this changes the image.

Listing 13.4 Perl script to transform listing 13.3 to the format in listing 13.5

```
while( <> ) {
  chomp;
  push @r, [ split "," ];
}
```

```
for $i ( 1..scalar( @{ $r[0] } )-1 ) { # For each column...
  ( $min, $max ) = ( 100000, 0 );      # ... find min and max
  for $j ( 0..scalar @r-1 ) {          # ... over all rows.
    $min = $r[$j][$i] < $min ? $r[$j][$i] : $min;
    $max = $r[$j][$i] > $max ? $r[$j][$i] : $max;
  }
  for $j ( 0..scalar @r-1 ) {     # Rescale this column in all rows
    $r[$j][$i] = ($r[$j][$i] - $min)/($max-$min);
  }
}

for $r ( @r ) {
# unless( $r->[3] > 0.67 ) { next; } # Optional filter logic
  for $i ( 1..scalar( @{ $r[0] } )-1 ) {
    print "$i\t", $r->[$i], "\n";
  }
  print "\n\n";
}
```

Listing 13.5 The transformed data set, ready for plotting

```
# ColumnIndex   RescaledValue
1               0.43
2               0.43
3               1
4               0.25
5               0.35
6               0.00
7               0.30
8               0
9               0
10              0

1               0.28
2               0.47
3               0.80
4               0.33
5               0.52
6               0.07
7               0.22
8               0
9               0
10              0

...
```

Finally, I've collected the most pertinent commands I used to create figure 13.13 in listing 13.6. The most interesting aspect is the way x tic labels are handled. I use explicit text labels rotated by 45 degrees. I also use the secondary x axis (at the top), rather than the primary axis, because the textual labels align better along the top than at the bottom. Additionally, the size of the top margin has been adjusted manually to make room for the rotated text labels.

Listing 13.6 Commands to generate figure 13.13 from the data in listing 13.5

```
unset xtics
set tmargin 5
set x2tics rotate by 45 offset 0 mirror
➥ ( 'Refractive Idx' 1, 'Sodium' 2, 'Magnesium' 3, 'Aluminum' 4,
➥   'Silicon' 5, 'Potassium' 6, 'Calcium' 7, 'Barium' 8, 'Iron' 9 )
plot [1:9][0:1] "data" i 78 u 1:2 w linesp t 'Record 78'
```

13.4.2 Multivariate analysis

The study of problems involving the simultaneous consideration of several related statistical quantities is called *multivariate analysis*. The first, and often the most important, goal of multivariate analysis is to find any kind of structure in the data, and thereby simplify the problem. This leads to classification, clustering, or projection techniques, some of them quite sophisticated.[8]

Multivariate analysis is hard, and graphical methods quickly seem to reach their limits. (I think parallel coordinate plots can't be regarded as anything but a kludge, but much weirder techniques have been suggested. The idea behind Chernoff-faces, for example, is to encode each quantity as a facial feature in a stylized human face: size of the mouth or distance between eyes, and so on. The observer then tries to find the faces that are "most similar" or "least similar" to one another.) For much larger data sets, one may resort to computationally intensive methods, which go under the name of *data mining* or more specifically *pattern recognition* and *machine learning*.[9] The latter set of methods is a highly active area of research.

I must say that I experience a certain degree of discomfort with the "random search" character of some multivariate methods. The purpose of data analysis is to gain insight into the problem domain that the data came from, but any brute-force method that isn't guided by intuition about the problem domain runs the risk of being about the numbers only, not about the actual system that the data came from originally.

The analysis we did earlier on the glass data set (see listing 13.3) is a case in point: we found that records can be classified according to their magnesium content—but what does that tell us about the original problem? This isn't at all clear at this point. We'll have to go back and understand more about the context in which this data was collected.

Multivariate classification methods, such as the parallel coordinates technique introduced here, can be a useful starting point when faced with large and unsystematic data sets, or any time we don't have good intuition about the actual problem

---

[8] Two short and accessible introductory texts are *Multivariate Statistical Methods: A Primer* by Bryan F.J. Manly (Chapman & Hall, 3rd ed., 2004) and *Introduction to Multivariate Analysis* by Chris Chatfield and A. Collins (Chapman & Hall, 1981).

[9] Three introductory texts, in approximate order of increasing sophistication, are: *Pattern Recognition and Machine Learning* by Christopher M. Bishop (Springer, 2007); *Pattern Classification* by Richard O. Duda, Peter E. Hart, David G. Stork (Wiley-Interscience, 2nd ed., 2000); and *The Elements of Statistical Learning* by T. Hastie, R. Tibshirani, J. H. Friedman (Springer, 2003).

domain. We can use these methods to develop strategies for more detailed analysis, but we must make sure to tie the results back to the original problem. The purpose of data analysis is insight into the problem domain, not insight into the data!

13.4.3 Star plots

Star plots are basically parallel coordinate plots in polar coordinates. Instead of showing many records in a single plot, it's more common to draw a single star plot for each record and to compare the resulting images. They're therefore more suited for smaller data sets (fewer records). Figure 13.16 shows a star plot for some of the records of the glass samples data set. The commands (involving both polar coordinates and multiplot mode) are shown in listing 13.7.

The advantage of star plots over parallel coordinate plots is that they give the viewer more of a sense of a recognizable shape. In figure 13.16, for instance, we can easily distinguish the three records of the top row as being similar to one another, while the three records in the bottom row are clearly different from both the top row and from each other.

Listing 13.7 Commands for a star plot array—see figure 13.16

```
set polar
set size square

unset border

set format x ""; set format y ""
set xtics scale 0; set ytics scale 0

set grid polar 2.0*pi/9.0
set xtics 0.25

unset key

set style data linesp

set multiplot layout 2,3

set label 1 '77' at graph 0,0.95
plot [0:2*pi][-1:1][-1:1] "stardata" i 77 u (2*pi*($1-1)/9.):2

set label 1 '78' at graph 0,0.95
plot [0:2*pi][-1:1][-1:1] "stardata" i 78 u (2*pi*($1-1)/9.):2

set label 1 '79' at graph 0,0.95
plot [0:2*pi][-1:1][-1:1] "stardata" i 79 u (2*pi*($1-1)/9.):2

set label 1 '105' at graph 0,0.95
plot [0:2*pi][-1:1][-1:1] "stardata" i 105 u (2*pi*($1-1)/9.):2

set label 1 '174' at graph 0,0.95
plot [0:2*pi][-1:1][-1:1] "stardata" i 174 u (2*pi*($1-1)/9.):2

set label 1 '184' at graph 0,0.95
plot [0:2*pi][-1:1][-1:1] "stardata" i 184 u (2*pi*($1-1)/9.):2

unset multiplot
reset
```

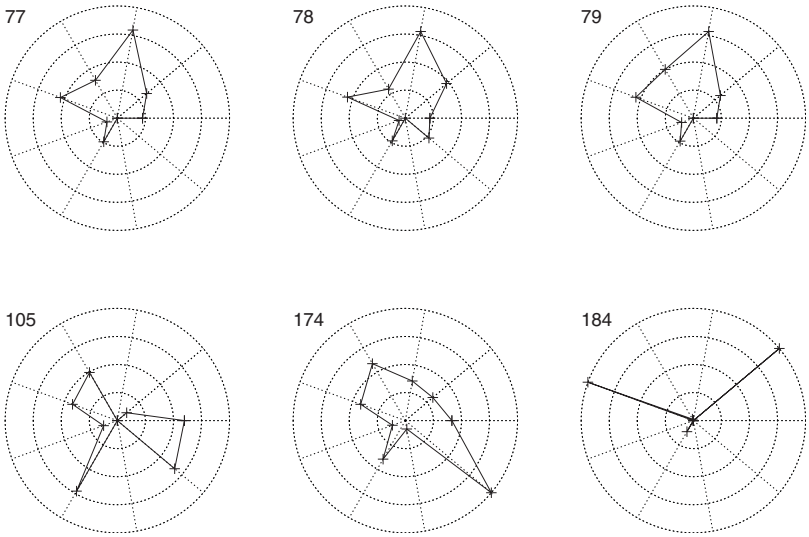

Figure 13.16 Star plot of six individual records from listing 13.3. The records in the top row are more or less similar to one another, but the records in the bottom row belong in very different categories.

13.4.4 Historical perspective: computer-aided data analysis

In this chapter, we talked about some more modern techniques for data analysis: using the median (instead of the mean), kernel density estimates (instead of histograms), parallel coordinate plots (for multivariate data). All these techniques have something in common that sets them apart from their "classical" counterparts: they *require* a computer to be practical.

I already commented on this when discussing the median (which requires sorting the entire data set, compared to the mean, which only requires a running sum of totals). Similar considerations apply to the kernel density estimate: a histogram only requires counting the number of events in each bin, whereas the kernel method requires an evaluation of the kernel function for each data point *and* for each sample point at which the curve should be drawn. And the parallel-coordinate plot is intended for data sets that are too large for manual techniques, anyway.

But this is only the beginning. Once we fully embrace the computer as a readily available and fully legitimate tool, what other methods for visual exploration become possible? The short answer is: we don't know yet. There are some new ideas that have started to come out of research in computer-assisted data visualization, some good, some certainly misguided. Time and experience will tell which is which.

One possible direction for the development of new visualization techniques is the ability to interact dynamically with a plot. For instance, a concept known as *brushing* involves two different views on a single, multivariate data set. When selecting a subset of points with the mouse in one view, the corresponding points in the other view are

highlighted simultaneously. This technique can be used to investigate structure in multivariate data sets. (I can also imagine applications to the parallel-coordinate plots we discussed.)

Gnuplot isn't suitable for such applications, but some academic software systems are freely available for experimentation. If you're interested, you might want to check out GGobi (www.ggobi.org) or Mondrian (www.rosuda.org/mondrian). The book *Graphics of Large Datasets* by A. Unwin, M. Theus, and H. Hofmann (Springer, 2006) also contains many useful pointers in this regard.

13.5 Summary

In this chapter, we started with the most fundamental questions we may pose to a data set, and discussed ways to answer them using graphical methods:

- For questions about the functional relationships between two quantities, we'll usually use a scatter plot of some form. We also discussed how a smooth curve approximation can help to detect structure in noisy data.
- For data sets of random points, we're mostly interested in the distribution of the points. Questions of this sort quickly take us into the territory of statistics, and I introduced jitter plots, histograms and kernel density estimates, and cumulative distribution functions.
- Data sets for which the independent variable has no natural sort order pose particular challenges. I suggested rank-order plots, in which we utilize the sort order of the *dependent* variable as the best way to visualize such data sets.
- Finally, we looked at unstructured, multivariate data sets. Here, our ambition was much more modest: rather than making definitive statements about the data, we were satisfied merely to find some form of structure in the data, which can help partition the data set into smaller and more uniform fragments.

In passing, I mentioned several mathematical concepts that are of particular usefulness during analysis. First we discussed logarithms, which are always helpful with data spanning many orders of magnitude, and which can reveal exponential and power-law behavior in data. Although I first discussed logarithms in connection with scatter plots, they're more generally useful, for example for rank-order plots. I also discussed the problems that may arise from the uncritical use of classical summary statistics (such as mean and standard deviation) and recommended instead statistics based on the median and percentiles. And finally, I tried to put the purpose and challenges of multivariate statistics into perspective.

In the next chapter, we'll get more specific and work through some particular problems in much more detail. Stay tuned.

OPEN SOURCE/DATA VISUALIZATION

Gnuplot IN ACTION

Philipp K. Janert

FOREWORDS BY COLIN D. KELLEY AND THOMAS WILLIAMS

Gnuplot in Action is a comprehensive tutorial written for all gnuplot users: data analysts, computer professionals, scientists, researchers, and others. It shows how to apply gnuplot to data analysis problems. It gets into tricky and poorly documented areas. You'll quickly move from basic charts to advanced graphics, mastering powerful techniques like multi-dimensional and false-color plots. You'll also learn scripting techniques for unattended batch jobs or how to use gnuplot to generate web graphics on demand.

This book does not require programming skills, nor previous knowledge of gnuplot.

What's Inside

- Generate simple and complex graphics
- Graphic methods to understand data
- Scripting and advanced visualization

A programmer and data analyst, Philipp K. Janert has been a gnuplot power user for over 15 years, in business and academic environments. He holds a Ph.D. in theoretical physics.

For online access to the author and a free ebook for owners of this book, go to manning.com/GnuplotinAction

"Knee-deep in data? This is your guidebook to exploring it with gnuplot."
—Austin King
Senior Web Developer, Mozilla

"Sparkles with insight about visualization, image perception, and data exploration."
—Richard B. Kreckel, Hacker and Physicist, GiNaC.de

"Incredibly useful for beginners—indispensible for advanced users."
—Mark Pruett, Systems Architect Dominion

"Bridges the gap between gnuplot's reference manual and real-world problems."
—Mitchell Johnson
Software Developer, Border Stylo

"A Swiss Army knife for plotting data."
—Nishanth Sastry, Computer Laboratory, University of Cambridge/IBM

MANNING $34.99 / Can $43.99 [INCLUDING eBOOK]