Drawing effective (and beautiful) graphs with TEX*

Jean-luc Doumont
JL Consulting
Achterenberg 2/10
B–3070 Kortenberg, Belgium
JL@JLConsulting.be

Abstract

A standard approach to producing documents that include illustrations consists in typesetting text with specialized typesetting software (such as TEX) and inserting illustrations created with other, equally specialized software. To integrate the illustrations better into the typeset page, though, it would be nice to be able to produce or modify them directly with the typesetting software. Drawing graphs with TEX, for example, would allow one to, say, set them `\hsize` wide and `0.75\hsize` high, position labels exactly `\baselineskip` below the horizontal axis, and, especially, typeset all annotations with the same fonts, sizes, and mathematical beauty as the rest of the document.

The hybrid TEX and PostScript macros presented in this paper take advantage of TEX's powerful approach to graph and annotate data sets in a variety of ways, in order to produce effective, beautiful, well-integrated graphs. They use TEX to draw all horizontal and vertical lines (axes, tick marks, grid lines) and set all annotations, and PostScript to draw the data, as markers, lines, and areas. Though fairly simple, they were successfully harnessed to a wide range of real-life applications, up to logarithmic graphs and (with some patience) complex multipanel displays. Clearly, they are a tool for drawing final graphs, not for exploring or transforming data sets.

Most designers produce documents by assembling components produced with different tools: typically, they typeset text with a text-oriented application and create illustrations with one or several drawing or graphing applications. This approach, it would seem, offers the best of both worlds, by using the best-suited tool for each part of the job. Unfortunately, it often suffers one—in some cases, major—drawback: the poor integration of the illustrations in the typeset page.

Though increasingly sophisticated, graphing applications still offer but limited control over some details of the display, all the more so if their focus is data manipulation, not data visualization. Among the less accessible parameters are typically

- the size of the display, the size and position of the graphing area within the display, etc.;
- the thickness of the axes, the length of the major and minor tick marks, the length and spacing of line segments in dashed lines, etc.;
- the typeface used for and the relative or absolute position of text elements (including subscripts and superscripts).

As a consultant on technical communication, including graphing, I am disappointed at the output of many graphing applications (or at the considerable efforts needed to produce acceptable output). In line with the recommendations of such authors as Jacques Bertin [1], William Cleveland [2], and Edward Tufte [3], I encourage the participants of my training programs to produce simple, intuitive, visually correct representations that favor data over decoration. Yet I find the default output of many applications to be overdecorated or hard to decipher. I was in search of a system that would help one focus on data, not decoration, much in the way TEX can help one focus on logical structuring, not visual rendering.

The idea of harnessing TEX to drawing figures is nothing new, going back at least to Leslie Lamport's original LATEX package [4]. The basic principle is always the same: define a coordinate system and position objects with respect to it with all of TEX's accuracy. The macros presented in this paper follow

[This article has not yet been edited. – Ed.]

this principle, but introduce some new ideas, for example that of using stretchable glue (rather than coordinate calculations) to position tick marks.

My graphing macros were developed in several steps. At the time I was still using dedicated graphing software, I first became frustrated with the poorly set text elements it offered, especially as soon as these involved any level of mathematics (symbols, subscripts or superscripts, etc.). I then began to produce graphs without any annotation, insert them in TEX, and overprint all text elements, including numerical labels along the axes. I soon realized that, if I was able to place numbers along axes properly, I could as easily add a tick mark next to each, so I decided to display the data themselves with graphing software and draw axes and grids with TEX, too. Displaying the data with PostScript code inserted in TEX was the logical next step.

This paper first explains the principles along which the macros draw axes, data, and annotations, then discusses extensions, limitations, and advantages of the approach.

Coordinates and axes

As you might expect, the macros eventually produce the graph as a TEX box of given width and height, which one specifies with the delimiting commands `\draw{`⟨*width*⟩`}{`⟨*height*⟩`}...\enddraw`. The `\draw` command accepts as optional argument (in square brackets à la LATEX) a specification of the graph's background color, the default being transparent (not white).

```
\draw{8pc}{8pc}
\xrange{0}{3}{10}
\yrange{0}{50}{1}
\xaxis{0}{1}{10}{3}{4}
\yaxis{0}{10}{1}{5}{1}
\markerdata ... \enddata
\linedata ... \enddata
\an{.2}{36}{$r^2=0.93$}rb
\enddraw
```

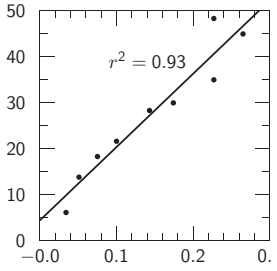

Figure 1: A simple graph and (part of) the code that generated it. Fractionary numbers are specified as fractions; for example, `\xrange0310` specifies the width of the graph to correspond to the range $[0/10, 3/10]$.

The macros then position drawing elements with respect to, but not necessarily within, the reference box, by using a coordinate system defined as a scale and an offset. In the horizontal direction, for example, the position `\x` (a dimension) is calculated from the coordinate ⟨*x*⟩ (a number) by the linear transformation `\x=`⟨*x*⟩`\xsc \advance\x+\xoff`. The scale `\xsc` and offset`\xoff`—two dimensions—are not specified as such by the user but are calculated from the specified graph width (for example, —8pc—) and axis range (for example, from 10 to 50). Unlike the calculation of the position `\x`, which involves *multiplying* a dimension by a (possibly fractionary) number, that of the scale `\xsc` from the graph width and axis range requires *dividing* a dimension—an operation TEX restricts to integers. To allow fractionary ranges nonetheless (say, from 0.0 to 0.3, as in Figure 1), the range specifications foresee a denominator: `\xrange{0}{3}{10}`, for example, specifies the horizontal axis to range from "zero divided by ten" to "three divided by ten" or $[0.0, 0.3]$.

While the `\xrange` and `\yrange` macros specify the range of the horizontal and vertical axes (corresponding to the graph width and height), they do not specify how axes should be drawn; the `\xaxis` and `\yaxis` macros do. These macros take as arguments the coordinate of the first major tick mark, the increment between major tick marks, a denominator (as for `\xrange` and `\yrange`), the number of major tick marks after the first one, and the number of minor tick marks between two major ones. The horizontal axis of Figure 1, for example, was draw with the command `\xaxis{0}{1}{10}{3}{4}`. The denominator (10, in this example) indirectly specifies the number of decimal places for the numerical labels: the command `\xaxis{0}{10}{100}{3}{4}` would have labeled the axis with values "0.00" to "0.30."

Rather than positioning each tick mark and corresponding value with a scale and offset system, the macros place them in a box of the proper width or height and separate them with equal amounts of stretchable glue, so tick marks end up equidistant. By using different amounts of glue (specifically, $\log 2 - \log 1$ = `0.3010fil`, $\log 3 - \log 2$ = `0.1761fil`, and so on), they can even produce logarithmically spaced tick marks without having to calculate any logarithm at all (Figure 2).

The axes can be drawn in different combinations and fine-tuned in various ways. First, optional boolean arguments to the `\xaxis` and `\yaxis` macros enable or disable parts of the axis drawing, such as axis line, numerical labels, major or minor grid lines, and mirror axis at the right or top. Second, many parameters defined as dimensions can be freely specified; among them are the line thickness, the length of both major and minor tick marks, the

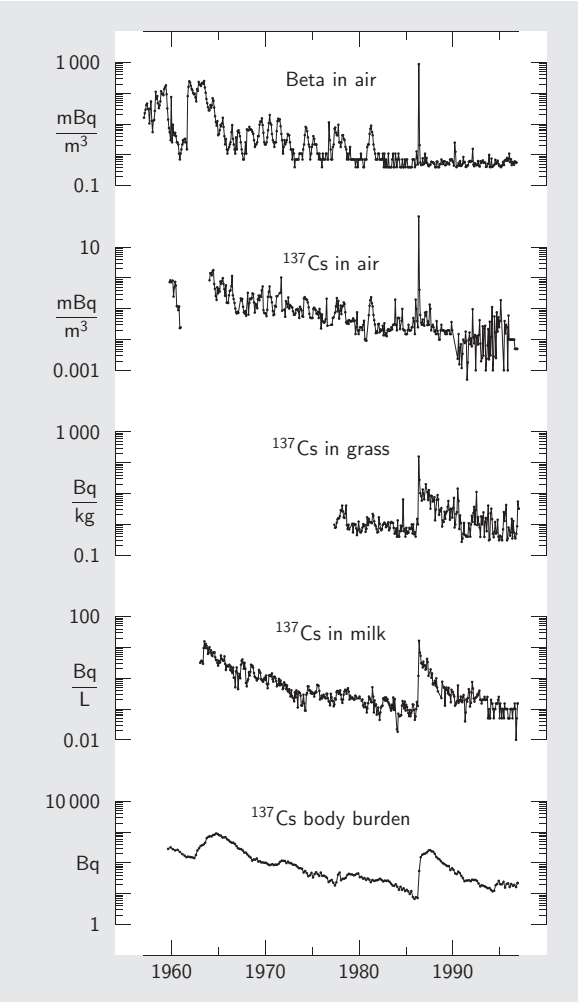

Figure 2: A more complex display, using several offset vertical scales [5]. With the data entered as logarithmic values, the apparently complex problem of logarithmic display boils down to setting tick marks right (with appropriate amounts of stretchable glue) and replacing the corresponding numerical labels x by 10^x (here, this default labeling was replaced by more readable annotations).

distance between axis and numerical labels, and the possible dash pattern used for the grid lines.

The macros actually offer a fairly general way of drawing straight lines. The fully TEX macros `\hl` and `\vl` draw horizontal and vertical lines, respectively, with length and starting points specified in drawing coordinates. They use the current value of line thickness and dash pattern (if any), and draw "projecting square caps" (matching a PostScript "linecap" value of 2), so perpendicular lines originating from the same point connect nicely. They are complemented by the hybrid TEX/PostScript macro `\rl`, drawing sloped lines in a similar way.

The macros for setting the range and those for drawing the axes are very flexible: they are largely independent, can be used several times in a given graphs, and can appear before or after other drawing commands (though some commands obviously require the proper range to be set first). The range may thus be redefined to allow the correct display of data expressed in other units. Moreover, the axis need not extend to the full range. It can be drawn with several `\xaxis` or `\yaxis` commands having different arguments and boolean flags; in this way, it may for example show tick marks along its full length, but numerical labels for some of the tick marks only (Figure 2). It can also be drawn before the data and be overprinted by them, or after the data and overprint them (Figure 3). Finally, it can be usefully complemented by horizontal or vertical lines (Figures 4 and 5).

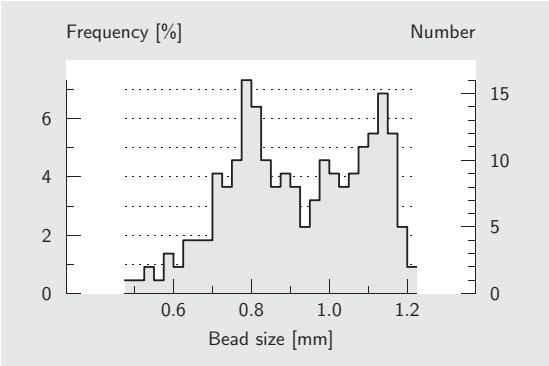

Figure 3: A histogram with different axes left and right [6]. The command sequence specifies the drawing sequence: here, first the vertical axis with the corresponding grid lines (background), then the data, then the horizontal axis (foreground).

Data

Though they could have used TEX code for some, the macros consistently use PostScript code for all forms of data representation: markers, lines, boxes, error bars, areas, etc. In a rather straightforward way, constructs such as `\linedata`⟨*data*⟩`\enddata` push the data onto the PostScript stack, then invoke a PostScript routine to handle them. This routine accesses some TEX parameters, most importantly the vertical and horizontal scales and offsets.

The data set is coded as a simple space-delimited list of space-separated pairs of values—or occasionally triplets, as when specifying error bars (Figure 4). Such an encoding can typically be obtained

by a simple copy-and-paste operation from the tabular representation of a spreadsheet. In a prior step, the spreadsheet application may help put the data in the desired form; it may, for example, compute their logarithm, or maybe display them with a limited number of significant digits, so as not to make the subsequent TEX file unnecessarily large.

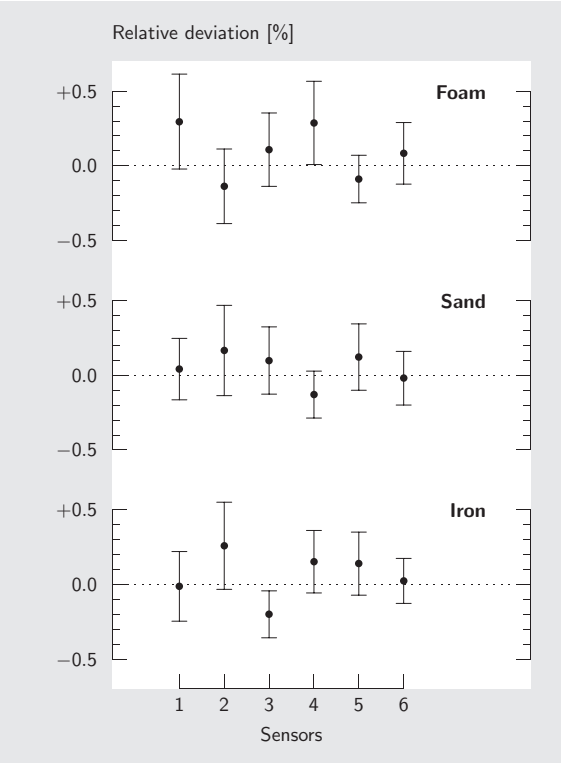

Figure 4: A graph with data and error bars [5]. The vertical axis was complemented by a zero line. The error bars are actually entered separately, as a list of triplets. Like other graphical elements, they may be drawn before or after the data points themselves—an issue when markers are of a different color (white-filled, for example).

The data-graphing macros (`\linedata`, `\markerdata`, etc.) accept as optional argument a further specification of the way the data must be rendered. This argument, in the form of either direct PostScript code or TEX macros expanding to PostScript code, may thus specify the width and color of the line, and the size and shape of markers, but also the clipping area and any other PostScript parameter such as line caps, line joints, or dash patterns. Default parameters may be specified via an `\everywave` macro.

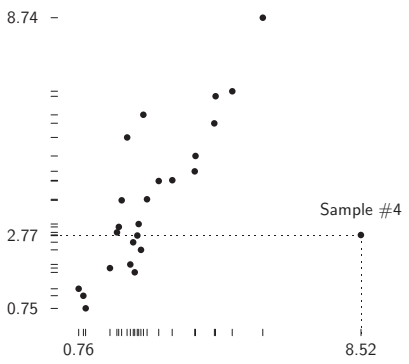

Figure 5: A dot-dash-plot à la Edward Tufte [3]. Ranges were specified, but axes were not. Data were displayed as three `\markerdata...\enddata` series (dots, horizontal marks, vertical marks). One horizontal line, one vertical line, and six annotations completed this sober display.

Annotations

The macros provide two forms of annotations. The numerical labels along the axes are positioned automatically by the `\xaxis` and `\yaxis` commands. All other annotations, including the labels of the axes, are positioned by the `\an` macro or variations of it.

In a way somewhat similar to LATEX's `\framebox` construct, the `\an` macro positions annotations by any corner of their enclosing box. The command `\an{0.2}{35}{$r^2=0.93$}rb`, for example, places the annotation $r^2 = 0.93$ such that its right (`r`) and bottom (`b`) corner be positioned at coordinates $(0.2, 35)$. More exactly, it places the annotation so that its right bottom corner be positioned a distance `\dx` left and `\dy` up from the point of specified coordinates. The offset (`\dx`,`\dy`) allows the user to position annotations "a little away" from the object they annotate, such as a data point. Like the text itself, the specified offset does not scale with the dimensions of the graph; coordinates computed to be "a little away," by contrast, would have the distance from the annotated object scale with the graph.

Though the numerical labels are usually set automatically by the `\xaxis` and `\yaxis` commands, they can also be set manually, for example, when they are nonequidistant or should otherwise be rendered differently (Figures 2 and 5). In this case, the offset (`\dx`,`\dy`) can be set equal to the offset specified for numerical labels, so `\an` sets the annotation exactly where `\xaxis` or `\yaxis` would.

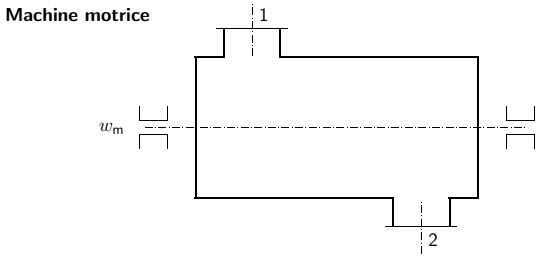

Figure 6: A simple drawing based on the graphing macros [7]. All elements are easily drawn with TEX commands (no PostScript).

As third and main argument, the `\an` macro accepts about anything, not just text: a framed box, a `\special` command, why, even another graph. It provides, therefore, a very general way of positioning objects with respect to a coordinate system. Because it was meant for short text snippets, it considers the "bottom" of the box enclosing the annotation to be its baseline, not its real bottom (⟨*depth*⟩ below the baseline), so text with and without descenders would be positioned in the same way.

Because the coordinate system can easily be changed at any time (by calling the `\xrange` and `\yrange` macros or even by assigning new values to the scales and offsets directly), the annotations may be aligned to elements other than the graph data. For example, if the vertical scale is set so that one unit corresponds to one `\baselineskip`, annotations can easily be set to align to text elements outside the graph; such an alignment may be nice for integrating the graph in a multicolumn page designed on a strict underlying grid.

Drawings other than graphs

Though designed with graphs in mind, the macros can easily accommodate other types of illustrations based on the accurate positioning of graphical elements according to a coordinate system (Figure 6). With a macro or two for framing text, they allow organizational chart—no PostScript required. With additional PostScript arrow heads and nonrectangular shapes (such as circles or diamonds), they allow flow diagrams (Figure 7) or more complex drawings.

Limitations

Like most macros, those presented in this paper are limited in scope: they are a TEX/PostScript hybrid, rely on TEX's limited arithmetic capabilities, and are oriented towards producing final graphs, not playing around with the data.

First of all, the macros show the limitations of all TEX macros relying on PostScript: their use of `\special`'s makes them implementation-dependent and their output is not readily visible in a dvi viewer. To see the graphs, one must typically convert the dvi file into a PostScript one, then print this file or view it on screen. This possibly slow process may render the visual optimization of a graph somewhat laborious (the legible positioning of annotations in complex graphs comes to mind). Still, the PostScript code manipulates graphical elements only; it need not manipulate text.

Second, the macros rely on TEX's limited arithmetic capabilities for calculating the scales, so they suffer round-off errors when handling large numbers, for example, when an axis is specified to range from 0 to, say, 10 000 000. (Of course, such large numbers read poorly and are best replaced by smaller numbers in larger units.)

Third, the macros are designed to display data sets, but clearly not to explore or transform them. In this respect, they do not replace dedicated graphing or data-processing applications, allowing one to select which data to represent and in which graphical form to represent them. (Similarly, TEX is meant as a tool for typesetting text, not writing it, though many of its features facilitate the writing process.)

Of course, the macros may been seen as having many more limitations, though these usually reflect deliberate design choices more than constraints: they are essentially restricted to two-dimensional representations and certainly do not favor decorative depth effects on either axes or data, they assume direct labeling of the data and thus provide no immediate way of creating legends, they foresee no mechanism for drawing pie charts or for setting text other than horizontally, etc. (Additional effects can of course be added, sometimes quite simply, by anyone familiar with TEX and maybe PostScript.)

Advantages

Despite the above limitations, typical of many TEX macros, the macros presented in this paper offer indisputable advantages for harmonizing graphs with the rest of a document, for optimizing a collection of graphs, or for producing unusual displays.

By having access to TEX's parameters, the macros allow a harmonious integration of graphs in a document. The graphing area can, for example, be specified to be exactly `\hsize` wide and, say, `15\baselineskip` high, so it aligns well to other elements on the page and of course automatically rescales if the document is typeset in a different width or interline spacing. For documents based

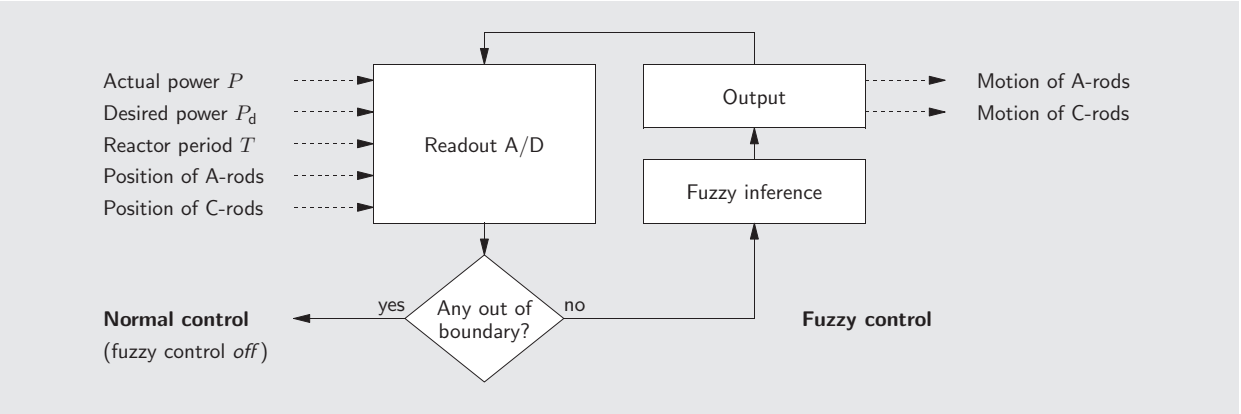

Figure 7: A flow sheet based on the graphing macros [5]. The only PostScript elements are the arrow heads and diamond outline.

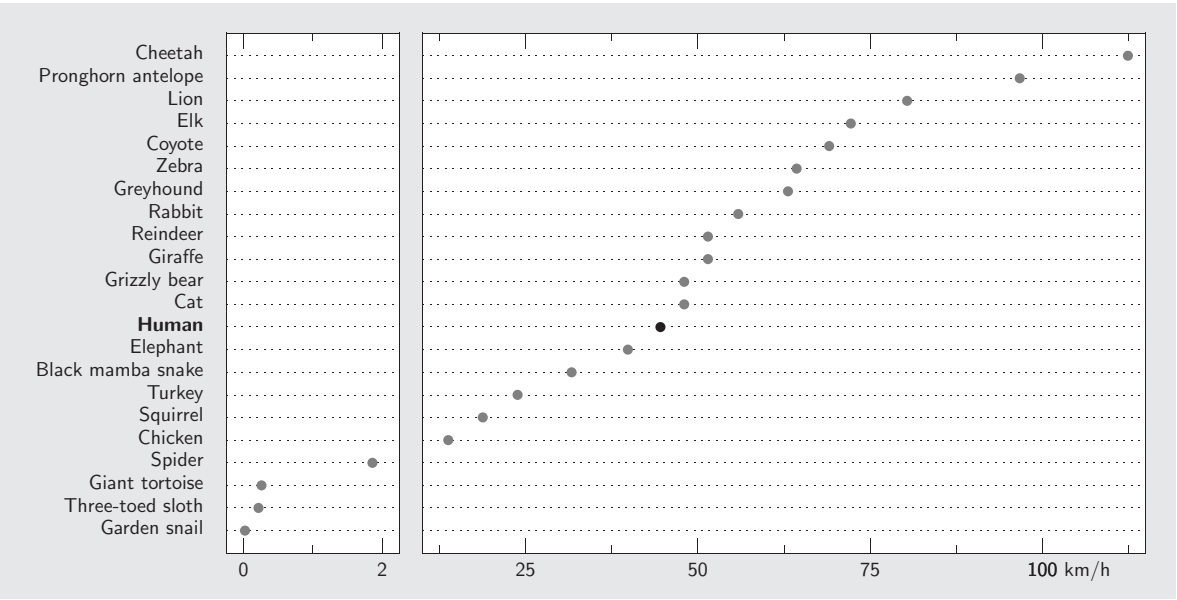

Figure 8: A dot chart à la William Cleveland [2]. The left panel presents a view ten times larger than the right panel. The vertical axis is labeled with annotations, not numerical values. All elements but the dots themselves are drawn with TEX commands.

on a strict grid, elements of the graph can easily be specified to align to it: labels of the horizontal axis can line up on an adjacent baseline, text annotations can be left-aligned to a grid position, etc.

As an important part of a smooth graph integration, the macros can typeset all annotations with the same typefaces, sizes, and mathematical beauty as the rest of the document. For example, one can easily add a formula to the graph, set an annotation on two lines exactly `\baselineskip` apart, or specify units in ISO notation, such as $\mathrm{J \cdot m^{-2} \cdot s^{-1}}$ (with my macro `\[J.m-2.s-1]` [8]). In contrast to imported illustrations (in, say, encapsulated format), the graphs can scale the position and size of components independently, by changing the corresponding TEX settings: an "enlarged" graph will thus not have thicker axes, bigger markers, or larger annotations.

Though TEX macros may not allow the visual optimization of a given graph as easily as a direct-manipulation application, they do allow the consistent optimization of a collection of graphs. In the declarative spirit of markup languages, the macros presented in this paper allow one to retypeset all graphs in a document at once by changing parameters (say, the axis thickness, tick length, or text size) at the top of the file. For this paper,

for example, the length of tick marks and size of dot markers was adjusted globally after all graphs were drawn. Direct-manipulation software, by contrast, would typically oblige the user to change these parameters for each graph separately—when they allow such parameters to be changed at all.

By giving access to all parameters in TEX-like fashion, the macros also allow one to produce unusual graphs or combinations of graphs—sometimes admittedly at the cost of some work. Custom axes (Figures 5 and 8) are a typical example; small multiples (Figures 2 and 4) and scatterplot matrices (Figure 9) are others. With elementary PostScript programming, one can easily extend the range of available markers or visual representations in general, and even draw functions described by an analytical expression rather than a set of points.

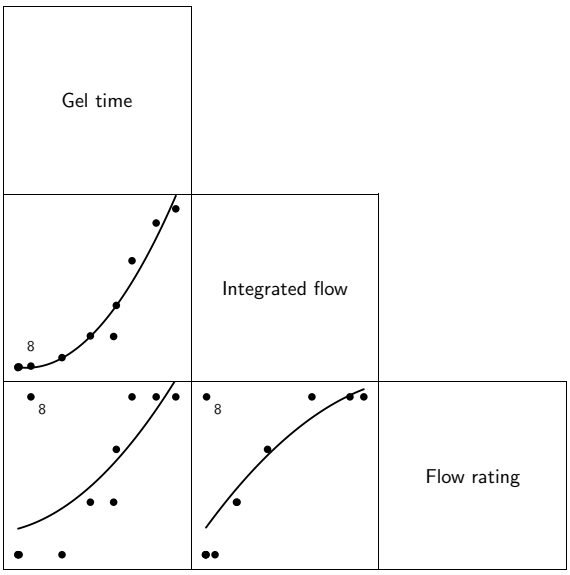

Figure 9: A scatterplot matrix, showing three views of the same data set, with an outlier [6]. The display is an assembly of six graphs, three of which contain nothing but a central annotation. The three fit lines are not entered as a list of data pairs; instead, they are specified as an analytical expression, to be evaluated by the PostScript interpreter.

Though it matters less and less with today's personal computers, the whole is also nicely compact. The macro package is below 32 KB, and so is the file containing the nine displays of this paper. (As a comparison, the spreadsheet file containing the data for Figure 2 amounts to over 75 KB.)

Conclusion

The macros, refined progressively over the years, reached their final form about three years ago, when I co-authored a manual on communicating numerical data for Shell [6]. Since then, I used them successfully to produce numerous real-life graphs and illustrations for my customers and for myself. Though designing an effective graph still requires careful thinking—something software will (probably) never take over—the macros help me focus on data, not rendering details, by allowing me to specify many of these details separately, and for all graphs at once. Most of all, by accessing the same dimensions and using the same positioning mechanisms as for the rest of the document, they allow me to integrate graphs and other drawings harmoniously, thus producing not merely beautiful graphs but also beautiful pages.

The macros have never been released before—so far, they have never been used by anyone but myself—but the positive feedback I received on their output has encouraged me to do so: they are now available under the name `JLdraw`. I hope they can be as useful to others as they are to me.

References

[1] Jacques Bertin, *Sémiologie graphique*, Flammarion, Paris, France (1973).

[2] William S. Cleveland, *The Elements of Graphing Data*, Wadsworth & Brooks, Pacific Grove, California (1985).

[3] Edward R. Tufte, *The Visual Display of Quantitative Information*, Graphics Press, Cheshire, Connecticut (1983).

[4] Leslie Lamport, *LATEX—User's Guide & Reference manual*, Reading, Massachusetts (1986).

[5] *Scientific Report 1997*, Belgian Nuclear Research Center, Mol, Belgium (1998).

[6] Philippe Vandenbroeck, Jean-luc Doumont, and Sophie Rubbers, *Communicating Numerical Data—Guidelines and Examples*, Shell Research (internal document), Louvain-la-Neuve, Belgium (1996).

[7] Michel Giot and Jean-Marie Streydio, *Chimie physique*, Université catholique de Louvain, Louvain-la-Neuve, Belgium (1995).

[8] Jean-luc Doumont, "Pascal pretty-printing: an example of preprocessing within TEX," *TUGboat* **15**:3, 302–307 (1994).