The plot functions of **`pst-plot`**[*]

Jana Voß and Herbert Voß

Abstract

Plotting of external data records is one of the standard problems of technical and industrial publications. One common approach is importing the data files into an external program, such as `gnuplot`, provided with axes of coordinates and further references, and finally exported to LaTeX. By contrast, in this article we explain ways to get proper data plotting without using external applications.

1 Introduction

The history and the meaning of PostScript have been covered sufficiently in many articles. For the programming language PostScript have a look at [3, 4]. The package `pst-plot` [8] under consideration here is part of the `pstricks` project. It must be loaded into a LaTeX document as usual with `\usepackage{pst-plot}` or alternatively, for documents written in plain TeX, with `\input pst-plot.tex`.

`pst-plot` provides three plot macros for the representation of external data, with the following syntax:

```
\listplot*[<parameter>]{<data macro>}
\dataplot*[<parameter>]{<data macro>}
\fileplot*[<parameter>]{<file name>}
```

The starred forms have the same meaning as with all macros of `pstricks`: to plot the data in a reversed mode. Thus, for a default black-on-white diagram one produces the negative with the starred command form, namely white-on-black. Additionally, a negative plot implies that PostScript closes the path of the points from the last to the first one and fills all points inside this closed path with the actual fillcolor. For our purposes in this article there will be no real sense in this negative view; therefore, all of the following examples are plotted with the normal (unstarred) form only.

For further information about `pstricks`, have a look at the (more or less) official documentation [6], or the extensive description in the "standard LaTeX" book [2] or in [1, 9]. Altogether, however, these do not fully describe the substantial differences between these three plot macros.

For all of the examples in this article, the complete `pspicture` environment is indicated, so that the examples may be directly copied. The documentation for the `multido` macro used here can be found in the package itself [7]; macros not otherwise mentioned are described in the `pstricks` documentation [6].

[*] Based on "Die Plot-Funktionen von pst-plot," *Die TeXnische Komödie* **2/02**, June 2002, pages 27-34, with permission.

Table 1: Possible options

style option	meaning
`plotstyle=dots`	plot (x,y) as a dot
`=line`	draw a line from a dot to the following one
`=polygon`	nearly the same as `line`, but with a line from the last to the first dot
`=curve`	interpolation between three dots, whereby the curve can go beyond the point of origin and/or termination point
`=ecurve`	like `curve`, but ends at the first/last dot
`=ccurve`	like `curve`, but closed

The general `plotstyle` parameter is particularly important, and can take the values shown in table 1.

By default, the `line` option is selected.

The following general commands are also useful in conjunction with the plot commands. They are also defined by the `pst-plot` package:

```
\readdata{<data macro name>}{<file name>}
\savedata{<data macro name>}{<file name>}
```

In the following examples only the `\readdata` macro is used, but it would be straightforward to create examples with `\savedata`.

2 Examples for `\listplot`

The syntax of `\listplot` is :

```
\listplot{<data macro name>}
```

The data macro may contain any additional (LA)TEX- or PostScript-commands. The (LA)TEX macros are expanded first before they are passed as a **list** of $x|y$ values to PostScript. The data records can be defined inside the document like

```
\newcommand{\dataA}{
 1.00000000  1.00000000
 0.56000000  0.31000000
 0.85841600  0.17360000
 ...
```

or can be read from an external data file with `\readdata`:

```
\readdata{\dataA}{/anyPath/data.dat}
```

The example in figure 1 shows the Henon attractor, a typical graphic of a system with chaotic behavior[5].

Figure 2 is nearly the same as figure 1, with the addition of PostScript code to get the "Draft" watermark. (Some familiarity with PostScript is needed to fully understand its operation.) To save space, listing 2 does not contain the data, which is nothing more than a sequence of pairs of floats, each value separated by a space, as shown above.

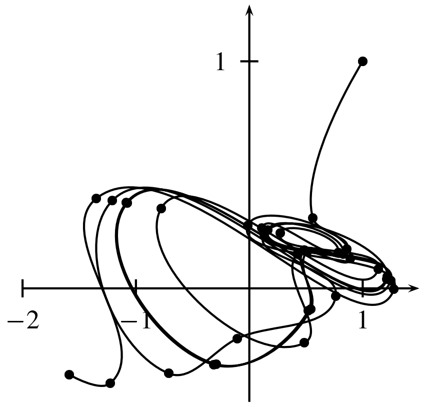

Figure 1: Example for `\listplot`

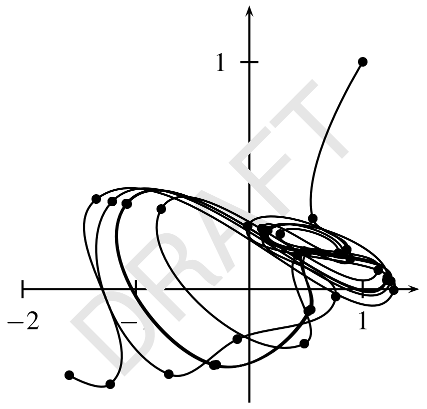

Figure 2: Example for modified `\listplot`

Listing 1: LATEX source for figure 1

```
\readdata{\henon}{henon.dat}
\psset{xunit=1.5cm, yunit=3cm}
\begin{pspicture}(-3,-0.5)(2.25,1.25)
  \psaxes{->}(0,0)(-2,-0.5)(1.5,1.25)
  \listplot[%
    showpoints=true,%
    linecolor=red,%
    plotstyle=curve]{\dataA}
\end{pspicture}
```

Listing 2: LATEX source for figure 2

```
\newcommand{\DataA}{%
  [ ... data ... ]
  gsave          % save graphic status
  /Helvetica findfont 40 scalefont setfont
  45 rotate      % rotate by 45 degrees
  0.9 setgray    % 1 is color white
  -60 10 moveto (DRAFT) show
  grestore
}
\psset{xunit=1.5cm, yunit=3cm}
\begin{pspicture}(-3,-0.5)(2.25,1.25)
  \psaxes{->}(0,0)(-2,-0.5)(1.5,1.25)
  \listplot[%
    showpoints=true,%
    plotstyle=curve]{\dataA}
\end{pspicture}
```

Naturally, `[... data ...]` is replaced by all the $x|y$-values; they're omitted here only to save space.

As an alternative to direct modification of the data set passed to `\listplot`, one can redefine the macro `defScalePoints` from `pst-plot`. For example, to change the $x|y$ values and then rotate the whole plotted graphic (don't ask why!), the redefinition is as shown in listing 3.

Listing 3: LaTeX source for figure 3

```
\makeatletter
\pst@def{ScalePoints}<%
%------------------------------------
  45 rotate % rotate the whole object
%------------------------------------
  /y ED /x ED
  counttomark dup dup cvi eq not { exch pop
        } if
  /m exch def /n m 2 div cvi def
  n {
%------------------------------------
     exch % exchange the last two elements
%------------------------------------
     y mul m 1 roll
     x mul m 1 roll
     /m m 2 sub
     def } repeat>
\makeatother
```

This gives figure 3.

Thus, the advantage of `\listplot` is that one can easily modify the data values without any external program. Here is one more example—suppose you have the following data records:

```
1050, 0.368
1100, 0.371
1200, 0.471
1250, 0.428
1300, 0.391
1350, 0.456
1400, 0.499
1500, 1.712
1550, 0.475
1600, 0.497
```

which perhaps came automatically from a technical device. The unit of the x-values is micrometer but it makes

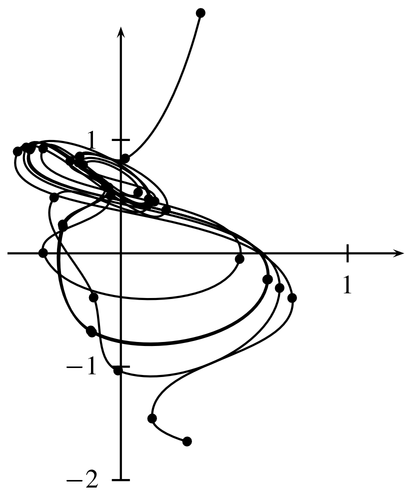

Figure 3: Example for `\listplot` with a redefined `ScalePoints`

more sense to use millimeter for the plot. A redefinition of `ScalePoints` makes it very easy to plot the data with this change of scale:

Listing 4: Rescale all x values

```
  \makeatletter
\pst@def{ScalePoints}<%
  /y ED /x ED
  counttomark dup dup cvi eq not { exch pop
        } if
  /m exch def /n m 2 div cvi def
  n {
     y mul m 1 roll
     x mul 1000 div m 1 roll% <-- divide by
          1000
     /m m 2 sub
     def } repeat>
\makeatother
```

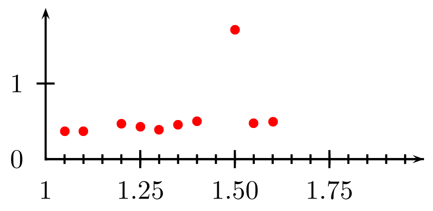

Figure 4: Example for modified data values with a redefined `ScalePoints`

3 Examples for `\dataplot`

`\dataplot` has the same syntax as `\listplot`, so the first question is, what is the difference between the two?

`\listplot` builds a list of all the data and then multiplies all values with the length unit. This takes some time, so you may prefer a so-called “quick plot”, where the data can be passed more quickly to PostScript, depending on the plotstyle and especially the option `showpoints`. Table 2 shows whether this is possible. A quick plot is not possible with `\listplot`, whereas `\dataplot` uses it whenever possible. When it is not possible, `\dataplot` simply calls `\listplot`.

Table 2: Possible options for a ”quick plot“

plotstyle	options	macro
`line`	all, except	quick plot
	`linearc, showpoints, arrows,`	`\listplot`
`polygon`	all, except	quick plot
	`linearc, showpoints`	`\listplot`
`dots`	all	quick plot
`bezier`	all, except	quick plot
	`arrows, showpoints`	`\listplot`
`cbezier`	all, except	`\listplot`
	`showpoints`	quick plot
`curve`	all	`\listplot`
`ecurve`	all	`\listplot`
`ccurve`	all	`\listplot`

`\dataplot` needs to be passed a macro holding the data. The data is typically saved in an external file, which can be read by (for instance) the `\readdata` macro, as follows:

```
\readdata{<object name>}{<data file>}
```

For example:

```
\readdata{\feigenbaum}{feigenbaum.data}
```

The amount of data is limited only by TeX’s memory. The above example can be plotted with:

```
\dataplot{\feigenbaum}
```

Overlays with different data files are also possible. For example, figure 5 shows the use of two different data files which are plotted using one coordinate system. It shows the sorting time for “Bubble-Sort” and “Select-Sort” as a function of the number of the elements.

Listing 5: LaTeX source for figure 5

```
\psset{xunit=0.0005cm,yunit=0.005cm}
\begin{pspicture}(0,-50)(10000,1100)
  \readdata{\bubble}{bubble.data}
  \readdata{\select}{select.data}
  \dataplot[%
        plotstyle=line,%
        linecolor=blue]{\bubble}
  \dataplot[%
        plotstyle=line,%
        linecolor=red,%
        linewidth=2pt]{\select}
  \psline{->}(0,0)(10000,0)
  \psline{->}(0,0)(0,1000)
  \rput[l](20,995){time}
  \rput[r](9990,-20){elements}
  \rput[l](4500,800){Bubble-Sort}
  \rput[l](7500,200){Select-Sort}
\end{pspicture}
```

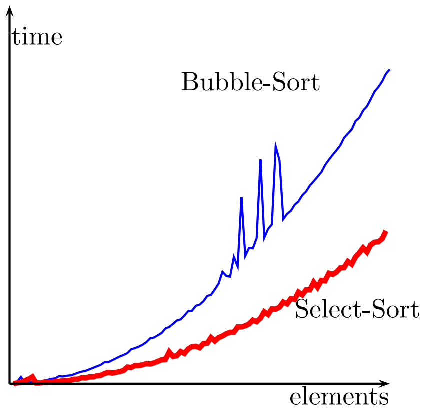

Figure 5: Example for `\dataplot`

In short, the advantage of `\dataplot` is the possibility of a ”quick plot“, and the advantage of `\listplot` is that it is easy to manipulate the data values before they are plotted.

4 Examples for `\fileplot`

`\fileplot` can be used whenever $(x|y)$ data that is saved in a file is to be plotted. The values must be given as pure numerical values in pairs, one pair on each line, and may have spaces, commas, parentheses, and braces as punctuation, as follows:

```
x y
x,y
(x,y)
{x,y}
```

The tab character (`\t` or ASCII `\009`) is often used as a separator, but tab is *not* valid here. Tabs may be converted to spaces in many ways, for example with the standard Unix utility `tr`:

```
tr '\t' ' ' <inFile >outFile
```

The data files may also contain % characters, but no other characters are allowed.

Our first example for `\fileplot` is shown in figure 6, which is an UV/VIS absorber spectrum $A = \lg \frac{I_0}{I}$ as a function of the wavelength. The second example (figure 7) shows the evolution of a population as a function of the spawn factor (Feigenbaum diagram [5]). The source code for these images is shown in listings 6 and 7.

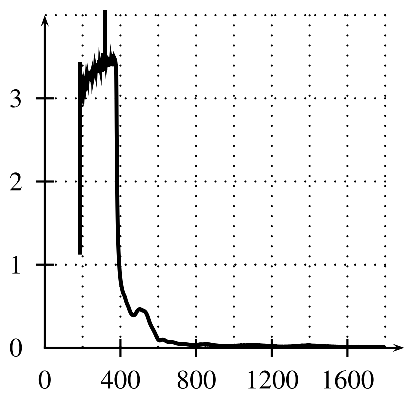

Figure 6: Example for `\fileplot`

Listing 6: LaTeX source for figure 6

```
\psset{xunit=0.0025cm,yunit=1.1cm}
\begin{pspicture}(-25,-.25)(1950,4)
  \fileplot[plotstyle=line]{fileplot.data}
  \psaxes[dx=400,Dx=400]{->}(1900,4)
  \multido{\n=200+200}{9}{%
    \psline[linestyle=dotted](\n,0)(\n,4)%
  }
  \multido{\n=+1}{5}{%
    \psline[linestyle=dotted]%
       (0,\n)(1800,\n)%
  }
\end{pspicture}
```

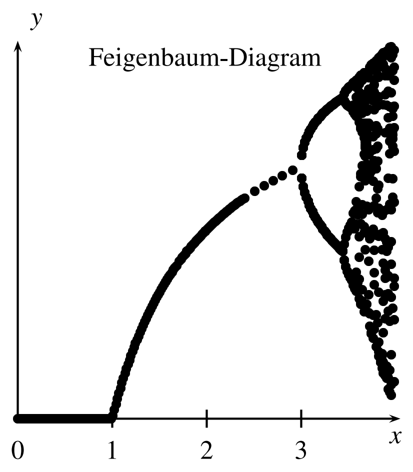

Figure 7: Another example for `\fileplot`

Listing 7: LaTeX source for figure 7

```
\psset{xunit=1.5cm,yunit=6cm}
\begin{pspicture}(-0.25,-0.05)(4.25,1)
  \fileplot[plotstyle=dots]{%
     feigenbaum.data}
  \psaxes{->}(0,0)(4.05,1)
  \rput(4,-0.05){$x$}
  \rput(0.2,1.05){$y$}
  \rput[l](0.2,3.75){Feigenbaum-Diagram}
\end{pspicture}
```

As you may see in listing 8, `\fileplot` does little of its own. It first calls `\readdata` to read the data, and then, depending on the kind of data and specified options, `\fileplot` uses `\dataplot` for a quick plot if possible. Otherwise, it falls back to `\listplot`.

Listing 8: The source of the `\fileplot` macro

```
\def\fileplot{\def\pst@par{}\pst@object{
    fileplot}}
\def\fileplot@i#1{%
  \pst@killglue
  \begingroup
    \use@par
    \@pstfalse
    \@nameuse{testqp@\psplotstyle}%
    \if@pst
      \dataplot@ii{\pst@readfile{#1}}%
    \else
      \listplot@ii{\pst@altreadfile{#1}}%
    \fi
  \endgroup
  \ignorespaces%
}
```

`\fileplot` has the advantage of being easy to use, but the disadvantage of needing a lot of memory: TeX has to read the all the data values before it can process anything. As a rule of thumb, when there are more than 1000 data entries TeX's main menory must be increased. Furthermore, the running time may be enormous, especially on slow machines.

To prevent such problems, one can use the macro `\PSTtoEPS` to create an `eps` file. For more information, see the documentation of `pstricks` [6].

References

[1] Denis Girou. Présentation de PSTricks. *Cahier GUTenberg*, 16:21–70, April 1994.

[2] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Graphics Companion*. Addison-Wesley Publishing Company, Reading, Mass., 1997.

[3] Nikolai G. Kollock. *PostScript richtig eingesetzt: vom Konzept zum praktischen Einsatz*. IWT, Vaterstetten, 1989.

[4] Glenn C. Reid. *Thinking in PostScript*. Addison-Wesley, Boston, 1990.

[5] Herbert Voß. *Chaos und Fraktale selbst programmieren: von Mandelbrotmengen über Farbmanipulationen zur perfekten Darstellung*. Franzis Verlag, Poing, 1994.

[6] Timothy van Zandt. *PSTricks - PostScript macros for generic TEX*. http://www.tug.org/applications/PSTricks, 1993.

[7] Timothy van Zandt. *multido.tex - a loop macro, that supports fixed-point addition*. http://ctan.tug.org/tex-archive/graphics/pstricks/generic/multido.tex, 1997.

[8] Timothy van Zandt. *pst-plot: Plotting two dimensional functions and data*. http://ctan.tug.org/tex-archive/graphics/pstricks/generic/pst-plot.tex, 1999.

[9] Timothy van Zandt and Denis Girou. Inside PSTricks. *TUGboat*, 15:239–246, September 1994.

◇ Jana Voß
Wasgenstr. 21
14129 Berlin GERMANY
Jana@perce.de

◇ Herbert Voß
Wasgenstr. 21
14129 Berlin GERMANY
voss@perce.de
http://www.perce.de