## 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-industrial publications. Very often the data files are imported into `gnuplot`, provided with axes of coordinates and further references and finally exported to LaTeX. 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 (Kollock, 1989; Reid, 1990). The package `pst-plot` (Zandt, 1999), which is part of the `pstricks` project has to be loaded into a LaTeX document as usual with `\usepackage{pst-plot}` or alternatively, if it is a TeX document 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 star versions have the same meaning as all macros of `pstricks`, to plot the data in a reversed mode, i.e. for a default black-on-white diagram one receives the negative with the star version which is white-on-black. Addidtional a negativ plot means 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 most examples which are described in this article there will be no real sense for this negative view, therefore all of the following examples are plotted without the star version.

For further information about `pstricks` have a look at the more or less official documentation (Zandt, 1993) or the extensive description in the "standard LaTeX" book (Goosens, Mittelbach, and Samarin, 1997) or in (Girou, 1994; Zandt and Girou, 1994). Altogether do not really describe the substantial differences between the three plot macros. For all examples the complete `pspicture` environment is indicated in each case, so that a direct assumption of the examples is possible. A documentation of the `multido` macro could be find in the package itself (Zandt, 1997), all other here far not mentioned in the documentation too `pstricks`.(Zandt, 1993)

In particular the plot style is a parameter of importance and can take the following values, which are shown in table 1.

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 two dots, whereby the curve can go beyond the point of origin and/or terminator point |
| `=ecurve` | like `curve`, but it ends at the first/last dot |
| `=ccurve` | like `curve` but a closed one |

With a missing indication in principle the `line` option is selected. Further the still following instructions are interesting in the connection, which likewise by the package `pst plot` are defined:

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

In the following examples only the `\readdata` macro is used, but it is not a real problem to create some more 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 the `\readdata` macro:

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

The given example of figure 1 shows the Henon attractor, a typical graphic of a system with a chaotic behaviour.(Voß, 1994)

The figure 2 is nearly the same than figure 1 only with some additional PostScript code to get the "Draft" watermark. There is some knowledge needed to understand all these PostScript code. To save some space the listing 2 does not contain the data, which is nothing more

than a sequence of pairs of floats, each value separated by a space.

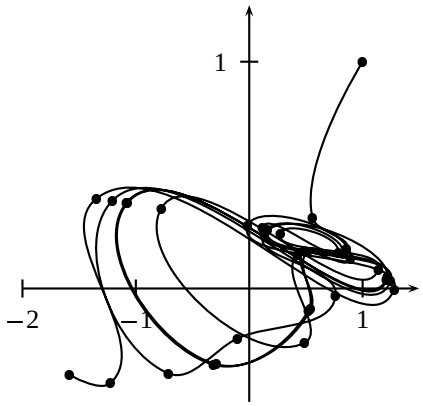


**Figure 1**: Example for `\listplot`

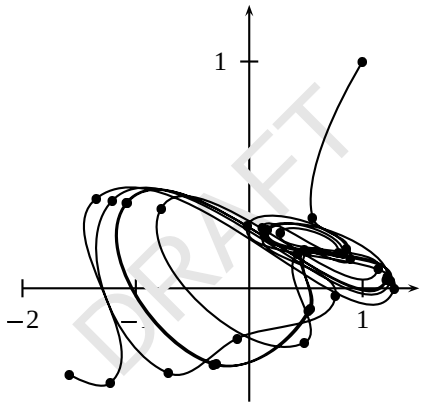


**Figure 2**: Example for modified `\listplot`

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 grafic 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}
```

`[... data ...]` has to be replaced by the list of all data records of the $x|y$-values; the only reason why they are not printed here is to save some space.

Alternative to the modification of the data set from `\listplot` one can redefine `defScalePoints` from `pst-plot`. For example: if someone wants to change the $x/y$ values and then rotate the whole plotted grafic (don't ask for the sense), so the redefinition has to be like 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.

The advantage of listplot is that one can easily modify the data values without any external program. One more example: if 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 may come automatically from a technical system. The unit of the x-values is micrometer but it makes

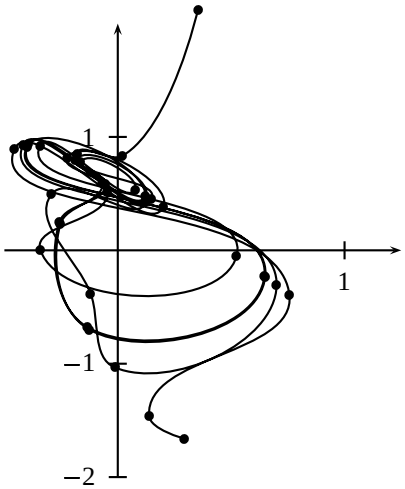


**Figure 3**: Example for `\listplot` with a redefined `ScalePoints`

more sense to use millimeter for the plot. A redefinition of `ScalePoints` is very easy to plot the data with other x-values.

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
```

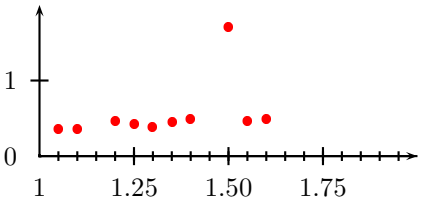


**Figure 4**: Example for modified data values with a redefined `ScalePoints`

## 3 Examples for `\dataplot`

`\dataplot` has the same syntax as `\listplot` so that there is the question where is the difference? `\listplot` builds a list of all data records and then multiplies all values with the length unit. This takes some time, so that it may be better to choose a so-called ”quick plot“. Depending to the plotstyle and especially the option `showpoints` it is possible to pass the data records in another quicker way to PostScript. Table 2 shows whether this is possible. A ”quick plot“ is not possible with the forgoing `\listplot`, whereas the `\dataplot` uses whenever it is possible the ”quick plot“. Otherwise `\dataplot` uses internally the `\listplot` macro.

Table 2: Possible Options for a ”quick plot“

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

In fact of the same syntax `\dataplot` needs a macro which holds the data records. In most cases the data refords are saved in an external file. This can be read by `\readdata` or any other macro which is able to save the data in the specific format. The syntax is very easy:

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

For example:

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

Only the amount of memory may reduce the number of the read data entries. Overlays with different data files are also possible. The above example can be plotted with

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

Figure 5 shows the use of two different data files which are plotted into 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[%
```

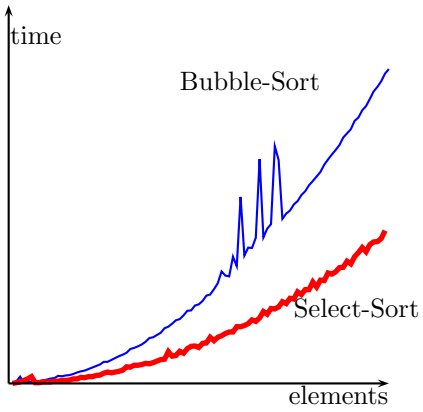


**Figure 5**: Example for `\dataplot`

```
        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}
```

In general it has to be consider, that the user does not ”see“ an important difference between `\dataplot` and `\fileplot`. If the ”quick plot“ is not used then there is as mentioned anyway no difference. The advantage of `\dataplot` is the possibility of a ”quick plot“ and the one from `\listplot` that it is easy to manipulate the data values before they are plotted.

## 4 Examples for `\fileplot`

`\fileplot` can be used whenever in a file saved data records ($(x|y)$) are to be plotted. These are to be arranged as pure numerical values in pairs in one or more lines and may only be separated in four different kinds (letter blank, comma or rounds and/or curved clips):

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

The tab character (`\t` or `\009`)is often choosed as separator, but not valid here. It is not a problem to change these tab characters with any texteditor or for example with a special command like

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

The data files may contain numbers, a seperator and the TEX-command character „%“, but not other characters.

The first example 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 während the evolution of a population as a function of the spawn factor (Feigenbaum diagram (Voß, 1994)). The sourcecode for these images is shown in listing 6 and 7.

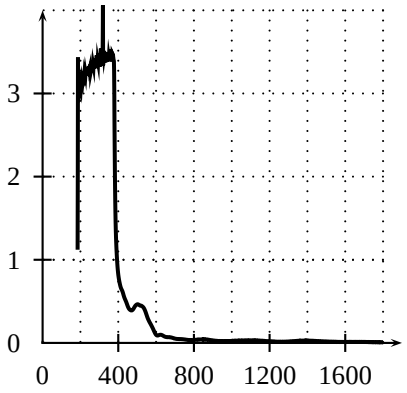


**Figure 6**: Example for `\fileplot`

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}
```

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)
\end{pspicture}
```

As listing 8 shows, `\fileplot` itself does nothing of its own: the data is read by `\readdata`, then the macro decides what to do. Depending to the kind of data and options it tries to use a ”quick plot“, which is a synonym for `\dataplot`, otherwise it switches to `\listplot`.

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

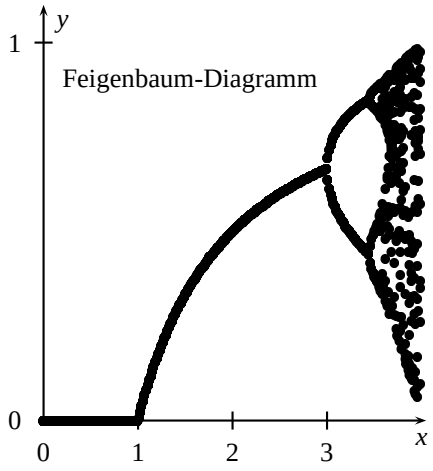


**Figure 7**: Another example for `\fileplot`

```
\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 an easy use but the disadvantage that it needs a lot of memory, because TEX has to read the whole data values before it can do anything. It can be expect that when there are more than 1000 data entries that the default TEX main menory must be increased. Furthermore the compilation time may be eormous, especially on slow machines.

To prevent such memory problems one can use the `\PSTtoEPS` macro to create an EPS file. For more Information have a look at the documentation of `pstricks`. (Zandt, 1993).

## References

Girou, Denis. “Présentation de PSTricks”. *Cahier GUTenberg* **16**, 21–70, 1994.

Goosens, Michel, F. Mittelbach, and A. Samarin. *The LaTeX Graphics Companion*. Addison-Wesley Publishing Company, Reading, Mass., 1997.

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

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

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

Zandt, Timothy van. *PSTricks - PostScript macros for generic TeX*. http://www.tug.org/application/PSTricks, 1993.

Zandt, Timothy van. *multido.tex - a loop macro, that supports fixed-point addition*. CTAN:/graphics/pstricks/generic/multido.tex, 1997.

Zandt, Timothy van. *pst-plot: Plotting two dimensional functions and data*. CTAN:graphics/pstricks/generic/pst-plot.tex, 1999.

Zandt, Timothy van and D. Girou. “Inside PSTricks”. *TUGboat* **15**, 239–246, 1994.