The PSTricks for R Package

Erik Olofsen

2024-02-07

Contents

Chapter 1

About the Package PSTricks

The package "PSTricks" provides layers above the PSTricks macro package for LaTeX (https://tug.org/PSTricks/main.cgi) by implementing wrapper functions for the macros. The advantage of these wrapper functions is that the arguments can be created with R functions, especially obviously the data. The second advantage is that everything is under your control; graphics are not determined by R's base plotting routines or by for example the "ggplot2" package, which have quite complicated mechanics and the default output may be difficult to change. A third advantage is that when using LaTeX, the fonts used in the graphs are easier to match those in the text.

For the main package there are the two manuals (the original "pst-user.pdf" and "pstricks-doc.pdf"), there are manuals for additional packages, and there is the book by Herbert Voß. Furthermore, there are chapters dedicated to it in the book "The LaTeX Graphics Companion". Finally, there is information on Wikipedia: https://en.wikipedia.org/wiki/PSTricks .

"tikz" would be an alternative. The package "tikzDevice" outputs TeX code, but does not provide wrappers for the tikz language; interfacing with the tikz syntax might be difficult to match with elegance.

The present interface consists of R function calls which are not too different from the macro calls; but unfortunately the nice PSTricks syntax with parentheses, brackets, and curly braces is unavoidably lost.

Mainly the main PSTricks macros, where the original manual served as the main guide (also for examples), have been layered at present. Adding functions for additional packages is not difficult; adding documentation and tests takes most of the effort (see the last section of this chapter).

The default output of an R script with PSTricks calls is a .pdf, generated by calling a LaTeX engine and cleaning up automatically.

1.1 Higher level functions

Functions have been added to produce plots. The first letters of the function names are usually "pp" to differentiate them from the PSTricks macro names which usually start with "ps". "pp" was chosen because the function names of the "ggplot2" package also usually start with two indentical letters. Here it could indicate that it is a second layer above the PostScript language. Also many functions starting with "geom_" have been added which a similar, but not completely identical, functionality as those in "ggplot2", see Chapter 9.

1.2 Installation

The package may be installed as usual, for example in the following two ways.

- devtools

In the directory where PSTricks_0.1.0.tar.gz (or a later version) has been untarred, within R:

```
install.packages("devtools")
devtools::install(build_vignettes=TRUE)
```

Building the vignettes needs a working system with LaTeX and "pandoc", so if that is not (yet) the case `build_vignettes=FALSE` (the default) may be useful.

- handling the dependencies by hand

```
R CMD INSTALL PSTricks_0.1.0.tar.gz
```

This will generate messages about additional R packages that need to be installed.

1.2.1 System packages

The following system packages need to be installed:

- texlive-latex-base
- texlive-pstricks
- texlive-xelatex if the "xelatex" engine is desired
- pandoc

1.3 The configuration file

Overrides for defaults may be given, in a file "~/pstricks.yml", for

- engine
- paper
- packages
- pstpkgs
- familydefault
- tmpdir

These may be likely to have relatively stable values across various scripts for a project.

The file is load using the "rconfig" package. If the environment variable "R_RCONFIG_FILE" is used, it will override the default "~/pstricks.yml".

For example, the sample "pstricks.yml" contains

```
engine: pdflatex
packages: tgheros
```

See the documentation of the `PSTricks()` function for the available settings.

Four engines may be used, "latex", "pdflatex", "xelatex", and "lualatex". The outputs of the engines are not identical. There is a script "examples/engines.R" to test the engines, in particular the handling of opacity.

1.4 Using the library

The library is loaded as usual:

```
library(PSTricks)
```

1.5 Examples, vignette, and tests

The packages comes with default documentation of all functions. The vignette provides a first look and serve as a test during "R CMD check", and is included as the next chapter. Many examples in the source files also function tests during the check. (see Chapter 12). With the "testthat" tests mostly the lower level functions are tested.

1.6 Extending for unlayered macro packages

If a macro has not been wrapped, a new function may be added where `ppbuild()` is the main building block:

```
ppbuild("psbuild",1,2,"opt","arg","arg1","arg2","arg3","arg4","arg0",TRUE)
#> [1] "\\psbuild*{arg0}[opt]{arg}(1,2){arg1}{arg2}{arg3}{arg4}"
```

A version for 3D plotting is available although 3D plotting is otherwise not implemented:

```
ppbuild3D("psbuild",1,2,3,"opt","arg","arg1","arg2","arg3","arg4","arg0",TRUE)
#> [1] "\\psbuild*{arg0}[opt]{arg}(1,2,3){arg1}{arg2}{arg3}{arg4}"
```

See "examples/3d.R" for an example on how this can be done.

See also functions like `pparg()`, `ppopt()`, and `ppcoords()` that construct macro parts.

Chapter 2

The Vignette

The introductory vignette is included here so that the information is also available in .pdf format.

2.1 Using the library

```
library(PSTricks)
```

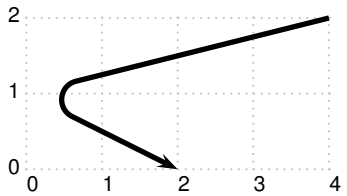
2.2 The original line drawing example

Drawing is done by creating a PSTricks object, a picture, and a line or another graphical object:

```
p <- PSTricks() %>%
    pppicture(4,2,par="showgrid=true") %>%
    psline(c(4,0,2), c(2,1,0), "linewidth=2pt,linearc=.25", "->")
```

Calling `print` will create a .tex file with the PStricks line macro, run LaTeX to create a .pdf and a .png to be included in the present vignette:

```
print(p, "intro-psline", topng=TRUE)
```



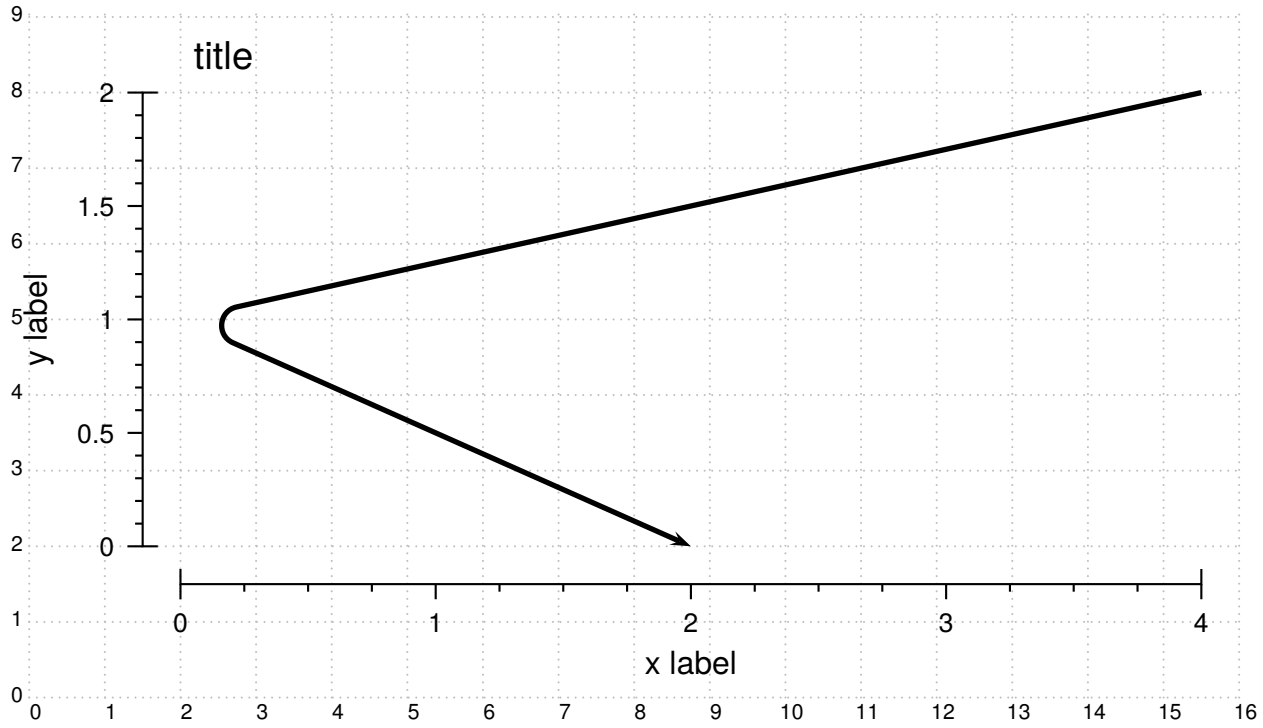
Note the name `pppicture` instead of the original name `pspicture`.

2.3 Axes and scaling

One main advantage of using R instead of plain LaTex is the fact that variables may be created in R. Furthermore, for general plots, there is a `ppaxis` command to create axes which allows for more control than the original `psaxis` macro.

```
x <- c(4,0,2)
y <- c(2,1,0)

p <- PSTricks() %>%
    pppicture(16,9,par="showgrid=true") %>%
```

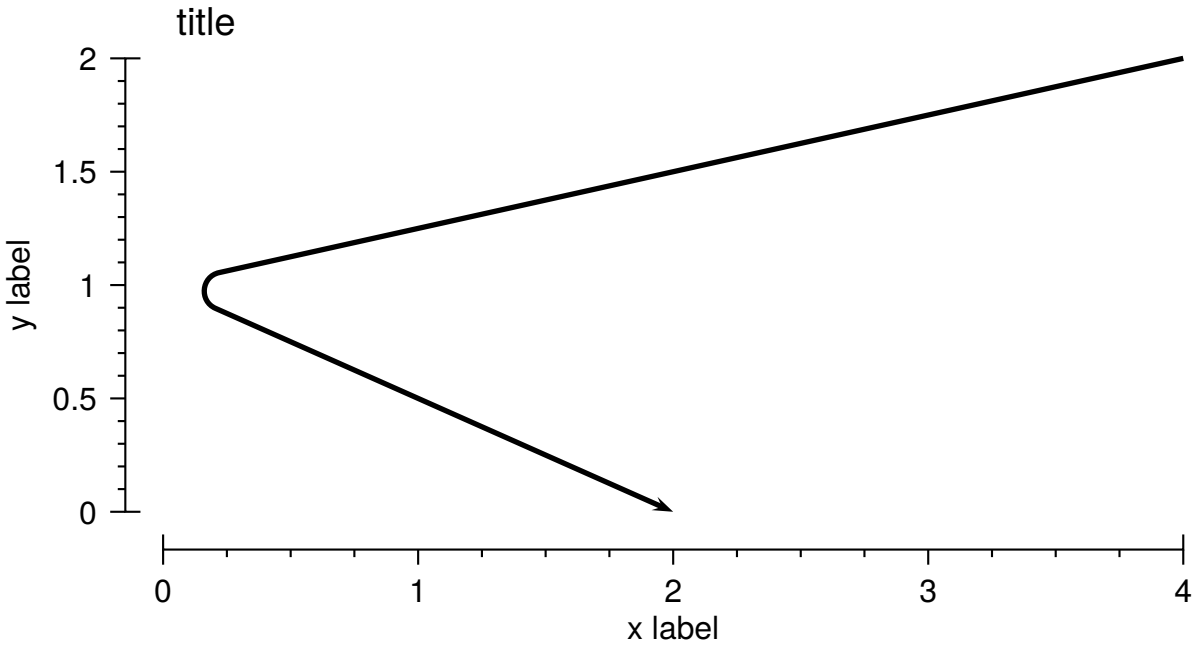
```
    ppaxis('x', c(0,4), label='\\large x label') %>%
    ppaxis('y', c(0,2), label='\\large y label')

# p, needed for the scaling conversion functions `cx` and `cy` below,
# contains only the the necessary information after p has been assigned
# as in the previous statement

p <- p %>%
    psline(cx(p,x), cy(p,y), "linewidth=2pt,linearc=.25", "->") %>%
    pptitle("\\Large title")

print(p, "intro-basic", topng=TRUE)
```



2.4 ggplot-like geoms

"geoms" like `geom_line` (similar but not identical to the one in the "ggplot2" package) perform implicit conversions from the `x` and `y` items in `data` to coordinates on the paper, and determine the limits of the data in these variables. Axes are drawn automatically. The non-geom functions in the sequence do not depend on `data`.

```
data <- data.frame(x=c(4,0,2),y=c(2,1,0))

PSTricks() %>%
    pppicture(16,9, data=data) %>%
    everypsbox("\\large") %>%
    geom_line(par="linewidth=2pt,linearc=.25,arrows=->") %>%
    xlab("x label") %>%
    ylab("y label") %>%
    pptitle("\\Large title") %>%
    print("intro-newstyle", topng=TRUE)
```


title
2
1.5
1
0.5
0
y label
0
1
2
3
4
x label

Chapter 3

The `PSTricks` object

After installation, a first test could be, within R:

```
names(PSTricks::PSTricks())
#>  [1] "doc0pened" "pic0pened" "paperx"    "papery"    "x"         "y"
#>  [7] "landscape" "center"    "config"    "lines"
```

`PSTricks()` returns an (S3) object with various attributes. The "lines" attribute will contain the LaTeX code lines. The attributes are manipulated by the various PSTricks functions and normally need not be accessed directly. See the documentation of the `PSTricks()` function for a short description of the shown attributes of the `PSTricks` object.

Next, to see if running LaTeX works:

```
PSTricks::PSTricks()
```

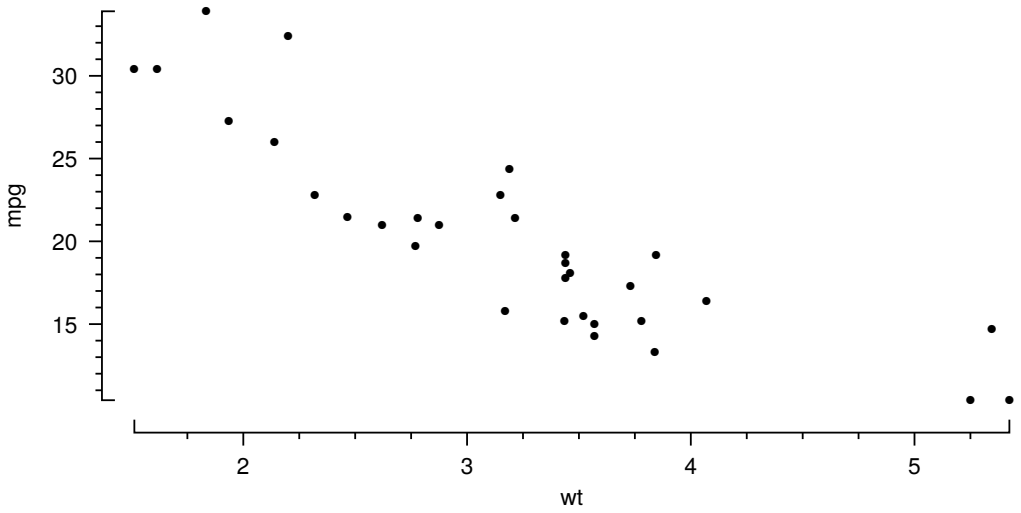
However, because the document is empty, no .pdf will be generated.

To be able to view the generated "pp.tex" file:

```
print(PSTricks(),"test",topdf=FALSE)
cat(system("cat test.tex",intern=TRUE),sep='\n')
#> \documentclass[a4paper]{article}
#> \usepackage[margin=0pt]{geometry}
#> \usepackage[T1]{fontenc}
#> \usepackage{tgheros,graphicx}
#> \usepackage{auto-pst-pdf}
#> \renewcommand{\familydefault}{\sfdefault}
#> \parskip 0 cm
#> \parindent 0 cm
#> \pagestyle{empty}
#> \begin{document}
#> \end{document}
```

A test that should give a .pdf (called "pp.pdf" if run in an R session rather than a script):

```
geom_dots(PSTricks(),aes(x=wt,y=mpg),mtcars)
```


mpg
30
25
20
15
2
3
4
5
wt

The output is one or more whole pages by default, that is, a picture is not "cropped" (see Chapter 11).

3.1 Flow of PSTricks objects

- Creating an object:

```
p <- PSTricks()
```

See the main documentation for a description of the arguments of `PSTricks()`.

- Modifying attributes:

```
p <- function(p, ...)
```

- Modifying using pipes (automatically imported from the "magritrr" package):

```
p <- p %>%
    functionA(...) %>%
    functionB(...)
```

- Printing the assembled information:

```
print(p)
```

See Chapter 11 for the latter.

3.2 Messages

Errors with piping the PSTricks object may cause confusing error messages; for example, with

```
p <- p %>%
    functionA(...) %>%
    functionB(...) %>%

print(p)
```

`p` will be the first and second argument of `ppwrite()`, and the latter function expects a string as the second argument.

See also directory "examples/errors/".

3.3 Returning a string from a wrapped PSTricks macro

Most functions add a line to the "lines" attribute of the PSTricks object, and to access that attribute, the object variable ("p" in the above) has to be available.

Some PSTricks macro calls are nested, so that sometimes the macro call as a string is needed and that string should not be appended to the lines attribute. To allow this, the object may be specified as NULL, which is the default if possible, so for example:

```
psline(,c(1,2),c(3,4))
#> [1] "\\psline(1,3)(2,4)"
```

See Chapter 14 for a complex example.

Chapter 4

The Document

The last example of the previous chapter creates the following LaTeX document:

```
print(geom_dots(PSTricks(),aes(x=wt,y=mpg),mtcars),"test",topdf=FALSE)
cat(system("cat test.tex",intern=TRUE),sep='\n')
#> \documentclass[a4paper]{article}
#> \usepackage[margin=0pt]{geometry}
#> \usepackage[T1]{fontenc}
#> \usepackage{tgheros,graphicx}
#> \usepackage{auto-pst-pdf}
#> \renewcommand{\familydefault}{\sfdefault}
#> \parskip 0 cm
#> \parindent 0 cm
#> \pagestyle{empty}
#> \begin{document}
#> \setlength{\hoffset}{2.5cm}
#> \setlength{\voffset}{10cm}
#> \pspicture(16,9)
#> \psline(2,1.7)(2,1.5)(15.5,1.5)(15.5,1.7)
#> \psline(3.681,1.5)(3.681,1.3)
#> \uput[d](3.681,1.3){2}
#> \psline(7.133,1.5)(7.133,1.3)
#> \uput[d](7.133,1.3){3}
#> \psline(10.585,1.5)(10.585,1.3)
#> \uput[d](10.585,1.3){4}
#> \psline(14.036,1.5)(14.036,1.3)
#> \uput[d](14.036,1.3){5}
#> \psline(2.818,1.5)(2.818,1.4)
#> \psline(4.544,1.5)(4.544,1.4)
#> \psline(5.407,1.5)(5.407,1.4)
#> \psline(6.27,1.5)(6.27,1.4)
#> \psline(7.996,1.5)(7.996,1.4)
#> \psline(8.859,1.5)(8.859,1.4)
#> \psline(9.722,1.5)(9.722,1.4)
#> \psline(11.448,1.5)(11.448,1.4)
#> \psline(12.311,1.5)(12.311,1.4)
#> \psline(13.173,1.5)(13.173,1.4)
#> \psline(14.899,1.5)(14.899,1.4)
#> \uput[d](8.75,0.8){wt}
```

```
#> \psline(1.7,2)(1.5,2)(1.5,8)(1.7,8)
#> \psline(1.5,3.174)(1.3,3.174)
#> \uput[l](1.3,3.174){15}
#> \psline(1.5,4.451)(1.3,4.451)
#> \uput[l](1.3,4.451){20}
#> \psline(1.5,5.728)(1.3,5.728)
#> \uput[l](1.3,5.728){25}
#> \psline(1.5,7.004)(1.3,7.004)
#> \uput[l](1.3,7.004){30}
#> \psline(1.5,2.153)(1.4,2.153)
#> \psline(1.5,2.409)(1.4,2.409)
#> \psline(1.5,2.664)(1.4,2.664)
#> \psline(1.5,2.919)(1.4,2.919)
#> \psline(1.5,3.43)(1.4,3.43)
#> \psline(1.5,3.685)(1.4,3.685)
#> \psline(1.5,3.94)(1.4,3.94)
#> \psline(1.5,4.196)(1.4,4.196)
#> \psline(1.5,4.706)(1.4,4.706)
#> \psline(1.5,4.962)(1.4,4.962)
#> \psline(1.5,5.217)(1.4,5.217)
#> \psline(1.5,5.472)(1.4,5.472)
#> \psline(1.5,5.983)(1.4,5.983)
#> \psline(1.5,6.238)(1.4,6.238)
#> \psline(1.5,6.494)(1.4,6.494)
#> \psline(1.5,6.749)(1.4,6.749)
#> \psline(1.5,7.26)(1.4,7.26)
#> \psline(1.5,7.515)(1.4,7.515)
#> \psline(1.5,7.77)(1.4,7.77)
#> \uput[l]{90}(0.5,5){mpg}
#> \psdots(5.821,4.706)(6.701,4.706)(4.786,5.166)(7.875,4.809)(8.652,4.119)(8.721,3.966)(9
#> \endpspicture
#> \end{document}
```

The document lines are held in the “lines” attribute of a PSTricks object, and written to a .tex file when the object is printed.

The package provides functions to open and close a document (`ppopendoc()` and `ppclosedoc()`), but these are called automatically when needed.

The arguments of the `PSTricks()` function allow for some control of the preamble.

By default, a picture is centered by using horizontal and vertical offset parameters.

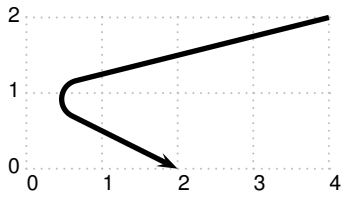
A “pspicture” environment is created (see the next Chapter) automatically in this case, with a default size, and the `geom_dots()` function is responsible for the necessary basic PSTricks macro calls.

Chapter 5

Basic PSTricks Objects

So drawing is done by creating a PSTricks object, a picture, and a line or another graphical object:

```
PSTricks() %>%
    pspicture(4,2, par="showgrid=true") %>%
    psline(c(4,0,2), c(2,1,0), "linewidth=2pt,linearc=.25", "->") %>%
    endpspicture() %>%
    print("pstricks-psline", topng=TRUE)
```



From a puristic viewpoint, `pspicture()` with `endpspicture()` are used with basic PSTricks macros. In most examples with the present package, `pppicture()` is used to avoid all `endppictures()` in the example code. Also, in this example, `pspicture()` is not strictly mandatory because with the "topng" option, the output will be cropped (otherwise the size will be a whole page).

5.1 Arguments

With `psline` as an example, the original syntax is

```
\psline*[par]{arrows}(x0,y0)(x1,y1)... (xn,yn)
```

In R, the definition is:

```
psline <- function(p=NULL, x, y, par=NULL, arrows=NULL, star=FALSE) { ... }
```

So the nice delimiters with the macros ('{arg}', '[arg]', '(x,y)') could unfortunately not be retained.

Importantly, all `x` and `y` values are combined in vectors, the most often used argument "par" follows, and the less used "arrows" and "star" are at the end. This often saves the typing of "par=...". If an argument is NULL, it will not be added to the macro call. The unit of a number is usually "cm", but sometimes parameters, usually those that are not `x` or `y` and are passed unmodified to the PSTricks macros, may contain a unit and for example "pt" or "mm" may be specified.

As may be expected, `?psline` at the R prompt shows the function definition.

5.2 Wrappers

Almost all basic PSTricks macros have been wrapped. Most additional PSTricks packages have not been layered. Documentation "pst-user.pdf" has been used as the main reference. All the information about `par` is as is, as this it is just a string. Some comments regarding specific sections:

- 2: additional function `ppnewrgbcolor()` which handles R color specifications
- 4: `pssettolength` and `psaddtolength` not wrapped
- 6: later macro `\psTextFrame` wrapped
- 7: later macro `\pscircleOA` wrapped
- 11: "pst-plot" not layered (except `psaxes()`), although `\fileplot` is used by `*P2E(fileplot=TRUE)`
- 23: `\KillGlue` and `\DontKillGlue` not wrapped
- 25: "multido" not layered
- 30: "pst-node" layered (see example below)
- 35: `\psmatrix` not layered
- 37: "pst-tree" not layered
- 46: "pst-fill" not layered
- 47: "pst-3d" not layered
- 48: "pst-coil" layered
- 49: `\SpecialCoor` not layered, but see Chapter 10.
- 50: overlays not layered
- 51: "pst-grad" not layered
- 52: "pst-text" not layered
- 53: "pst-char" not layered
- 55: "pst-eps" partly layered, see Chapter 10.
- A: `everypsbox()` wrapped

5.3 ppappend

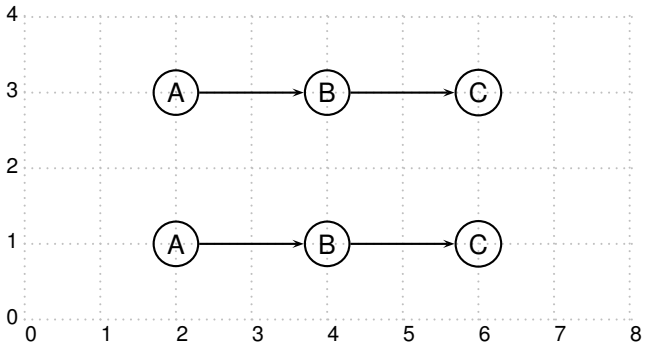
If a wrapper function is not available, the macro or any other LaTEX command may be supplied as a string to the `ppappend()` function which adds the argument string to the "lines" attribute of the `PSTricks` object.

5.4 pst-node

Also for the additional PSTricks package "pst-node" calling the macros from R may be useful, because even though node names replace coordinates, calculations on the node defining coordinates are possible, as in the following example. Not the addition of "pst-node" to the "pstpkgs" argument of `PSTricks()`.

```
p <- pppicture(PSTricks(pstpkgs="pst-node"),8,4,par="showgrid=true")

f <- function(p,x,y)
{
    p %>%
        cnodeput(x,y,"A","A") %>%
        cnodeput(x+2,y,"B","B") %>%
        cnodeput(x+4,y,"C","C") %>%
        ncline("A","B",arrows="->") %>%
        ncline("B","C",arrows="->")
}
```

```
p <- p %>% f(2,3) %>% f(2,1) %>%
    print("nodes", topng=TRUE)
```



Chapter 6

Pictures

The `pspicture` environment creates space for a picture in the LaTeX document. The `pspicture()` wrapper function is available, as is a higher level function `pppicture()`, which sets many additional PSTricks attributes used by higher level plotting functions to initial values. `endpppicture()` will be automatically called when needed; `endpspicture()` will not. See the `pspicture()` documentation, and the previous chapter, for examples.

By default, `pppicture()` allocates the whole page; if a size is specified, the picture will be centered on the page (unless specified otherwise). `ppnewpage()` may be called to start a new page; a new pppicture will be centered the same as the first one. A new pppicture will reset the lower and higher level option values to default ones; see "examples/resetoptions.R".

The `x` and `y` arguments of `pppicture()` may be vectors with length two, and the first elements may be negative, although this is not very useful with the higher level functions

`pppicture()` adds a "geoms" attribute to the PSTricks object, where `geom_*()` calls will be collected, see Chapter 9.

Chapter 7

Subplots

By using subplots, the available space on the paper, as determined by `pppicture()`, is divided in a number plot columns and a number of plot rows.

`ppsubplot()` determines some parameters for axes that are to be drawn (manually or automatically).

See "examples/subplot.R" and "examples/subplotl.R".

`ppmansubplot()` may be used to place a subplot manually anywhere on the paper, rather than being determined by numbers of columns and rows.

`ppsubplot()` without specifying the numbers of columns and rows will automatically move to the next subplot. If the page is full, `ppnewpage()` will be called automatically.

The length of the axes are determined by the space within a subplot. The functions `xaspect()`, `yaspect()`, and `xyaspect()` may be used to calculate the dimensions of the picture to have a certain aspect ratio of the axes.

Chapter 8

Axes

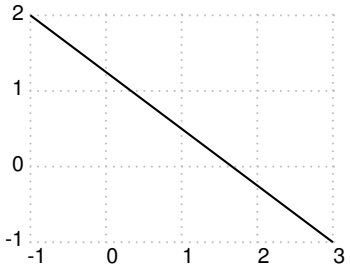
8.1 Axes and grids

Grids are usually not shown in final graphs, but they may help finding values attained by for example line objects.

8.1.1 showgrid

This option is very useful for positioning the various PSTricks objects on the paper.

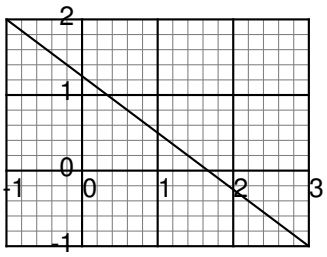
```
p <- pppicture(PSTricks(),c(-1,3),c(-1,2),par="showgrid=true") %>%
    psline(c(-1,3),c(2,-1))

print(p, "showgrid", topng=TRUE)
```



8.1.2 psgrid

While the package provides its own grid drawing functions, the original macro is available.

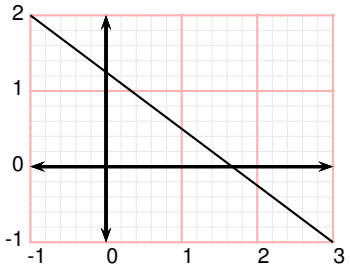
```
p <- pppicture(PSTricks(),c(-1,3),c(-1,2)) %>%
    psgrid(c(0,-1,3),c(0,-1,2)) %>%
    psline(c(-1,3),c(2,-1))

print(p, "psgrid", topng=TRUE)
```



8.1.3 psaxes

Likewise, the original macro for drawing axes is available.

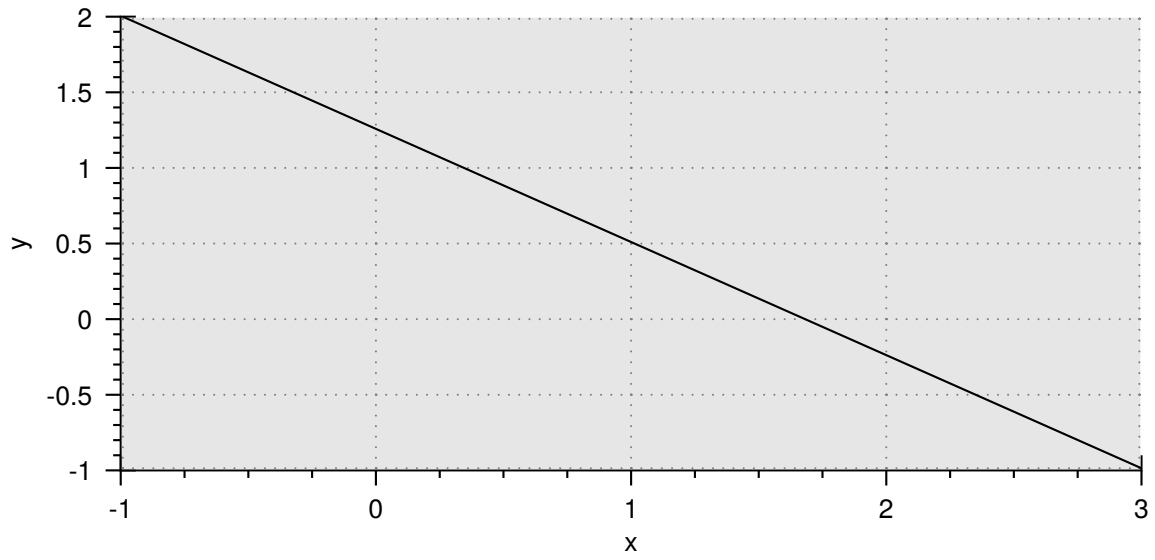
```
p <- pppicture(PSTricks(pstpkgs="pst-plot"),c(-1,3),c(-1,2),
        par="showgrid=true") %>%
    psaxes(c(0,-1,3), c(0,-1,2),
        "linewidth=1.2pt,labels=none,ticks=none", "<->") %>%
    psline(c(-1,3),c(2,-1))

print(p, "psaxes", topng=TRUE)
```



8.1.4 ppaxis and ppgrid

When axes with scaling are drawn manually, a grid may be added as follows. This looks best when there is no space between the axes.

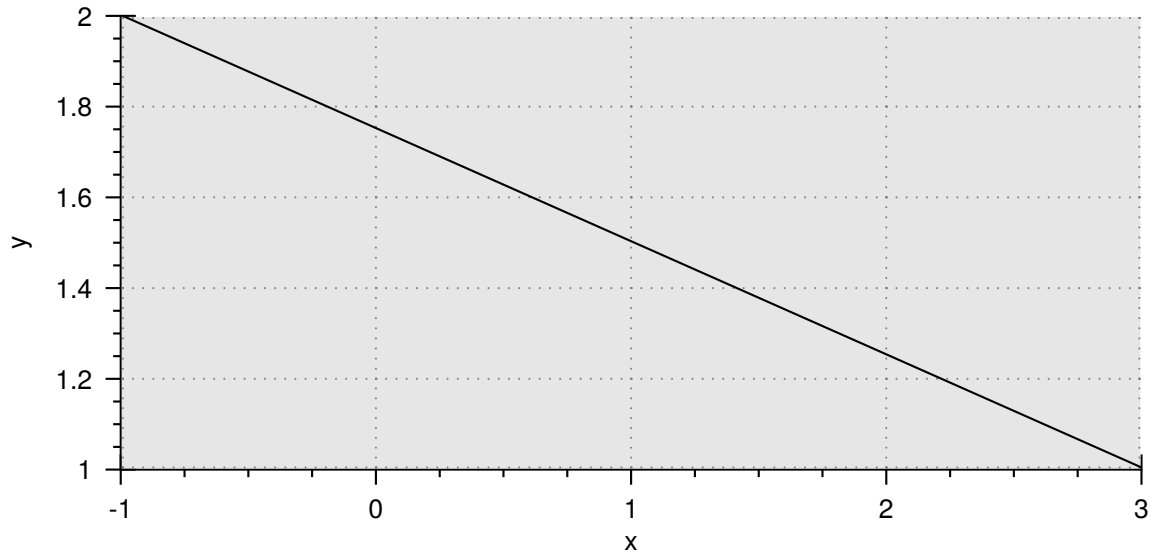
```
p <- pppicture(PSTricks(),16,9) %>%
    newrgbcolor("verylightgray",.9,.9,.9) %>%
    ppsetmargins(mrgaxes=0) %>%
    ppaxis('x',c(-1,3),"x") %>%
    ppaxis('y',c(-1,2),"y") %>%
    ppgrid("linestyle=dotted,linecolor=gray",background="verylightgray")

p <- psline(p,cx(p,c(-1,3)),cy(p,c(2,-1)))

print(p, "ppgrid", topng=TRUE)
```



8.1.5 geoms with automatic axes and geom_grid

The same graph as in the previous subsection may be achieved with “geoms”:

```
p <- PSTricks() %>%
    newrgbcolor("verylightgray",.9,.9,.9) %>%
    ppsetmargins(mrgaxes=0) %>%
    geom_grid("linestyle=dotted,linecolor=gray",
        background="verylightgray") %>%
    geom_line(data=data.frame(x=c(-1,3),y=c(2,1)))

print(p, "geom_grid", topng=TRUE)
```



See also “examples/ggstyle.R”

Note that subdivision grid lines are not implemented. Also, it is currently quite difficult to match the positions of the dots with the minor tickmarks.

8.2 Scaling and clipping

So axes may be drawn manually or automatically by geoms (see also Chapter 9). When data are plotted manually, the data have to be scaled to coordinates on the paper by the functions `cx()` and `cy()`; geoms call these functions when needed. The drawing of the axes may be suppressed by the option “noshow” of `ppaxis()` but of course almost always the axes are a desired part of the plot.

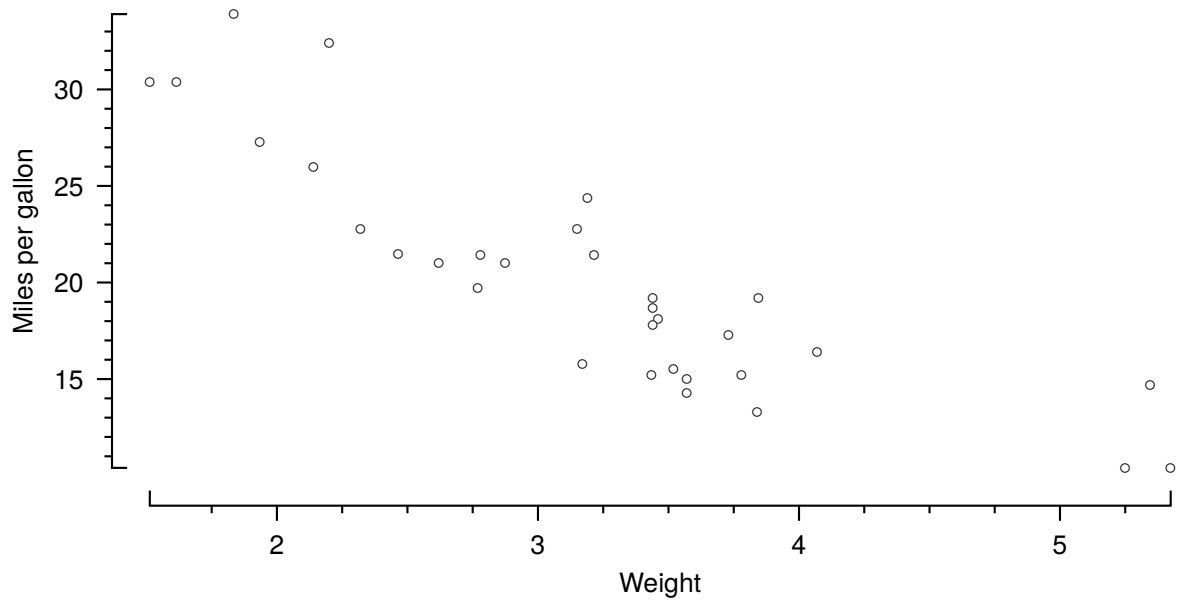
Clipping is also done by `cx()` and `cy()`, unless the starred version of the picture environment is used (see `pppicture()` and the original PSTricks documentation). `cx()` and `cy()` clip the data to the axes area of the subplot - this is only marginally larger than the limits of the axes, so sometimes this is not very conspicuous. Inf and -Inf in the data have a special use: these values will be clipped - see Section 13.2. See also “examples/pictures.R”.

8.3 Axes and tickmarks

With manually drawn axes, their limits are specified in calls to `ppaxis()`; with geoms these are determined automatically:

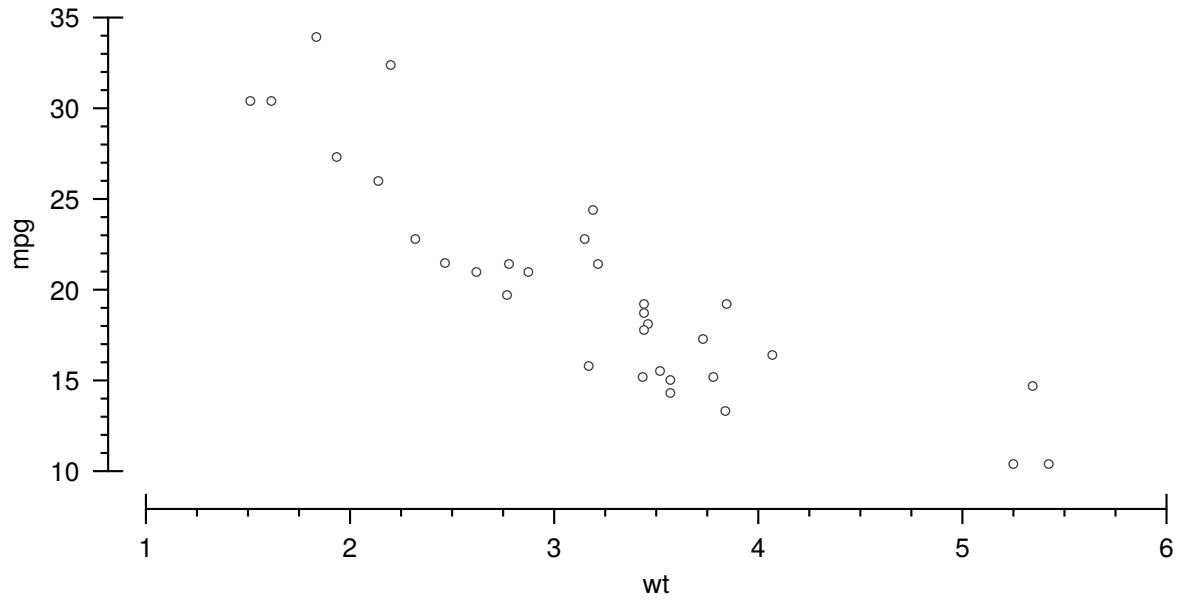
```
p <- PSTricks() %>%
    pppicture(16,9,mtcars,aes(x=wt,y=mpg)) %>%
    ppylabsep(0.8) %>%
    psset("dotstyle=Bo") %>%
    geom_dots() %>%
```

```
    xlab("Weight") %>%
    ylab("Miles per gallon")

print(p, "axis2def", topng=TRUE)
```



While the axes are such that there is a good view on the data, manual limits may be desirable:

```
p <- geom_dots(p) %>%
    xlim(1,6) %>%
    ylim(10,35)

print(p, "axis2manlim", topng=TRUE)
```



For automatic or manually selection of the number of major and minor tickmarks there are the functions `ppxticks()` (alias `xticks()`), `ppyticks()` (alias `yticks()`), `ticks()` and `ppticks()` - see the main documentation.

For further examples, see "examples/ticks.R".

8.3.1 Strings as tick labels

Sometimes the ticks need to have label strings other than their automatically converted numerical value - see for example “examples/mtcars.R”.

Chapter 9

Geoms

Geoms may be used to have automatic scaling of data values to coordinates on the paper. Limits of `x` and `y` data are updated each time a `geom` is called and when next axes are drawn, these should accommodate all data (ranges). In subsequent steps the axes may be optimized by specifying nicer values for the limits.

The source data is specified by the "data" argument of `pppicture()`, `ppsubplot()`, or any geom.

Apart from the "data" argument, the "mapping" argument is a characteristic element of drawing with geoms. The mapping is specified (as with "ggplot2") using the `aes()` function.

This section lists the available geoms and associated functions; their actual usage may be found in their example sections in the main documentation.

The "arrows" argument is not implemented, but these may be set using "par".

9.1 `aes()`

The geoms try to find variables in the data with names such as `x` and `y`. These may be given different names using for mappings such as for example `aes(x=time)`. Like with the data, a mapping may be specified at the geom, or globally at `pppicture()` or `ppsubplot()`.

So if `x` is sought, it will be checked it if can be obtained via the mapping specified with the geom. If that fails, the global mapping will be checked. If that fails, it will be checked if the variable `x` is present in the dataset without a mapping.

The arguments of `aes()` can only be simple name mappings, functions such as in `aes(x=fun(x))` will not be evaluated. Also, factors as arguments, for example as a way to determine line colors will not be handled.

On the other hand, for some functions arguments like `par` may have a single value, or `par` may be available as a variable in the data set with the same length as the `x` data. This is possible if such arguments are given in the geoms listed below.

It is not available for functions like `geom_line()`; multiple lines with for example different colors need to be specified via different columns in the data set (or different data sets).

9.2 Settings

Geom calls are collected in a list which are called at the end of a plot (by `ppgeoms()`, see below). Therefore, if for example `psset()` is called a few times, only the last one would have effect

when the list is evaluated, and it could affect the drawing of the axes. If settings should be synchronized with geom calls, the following functions may be used: `geom_set()`, `geom_linewidth()`, `geom_everypsbox()`.

9.3 Line geoms

The following functions are available that call the underlying macros: `geom_line()`, `geom_polygon`, `geom_frame(,par,star)`, `geom_curve()`, `geom_ecurve()`, `geom_ccurve()`.

Geoms that plot `y` versus `x` data usually remove NAs en negative numbers when using a logarithmic axis, see "examples/nainf.R".

9.4 Dot geoms

Available is: `geom_dots(,par,star)`.

9.5 Text geoms

Available are: `geom_framebox(,par,refpoint,rotation,star)`,`geom_rput(,refpoint,rotation,star)`, `geom_uput(,refangle,rotation,labelsep,star)`.

9.6 Expansions

In addition, the following "ggplot2" style functions are available:

- `geom_abline()`,`geom_hline()`,`geom_vline()`,`geom_errorbar()`,`geom_hist()`,`geom_legend()`, `geom_grid()`.
- Setting axis limits

See `xlim()`, `ylim()`, `lims()`.

- Setting an axis label

See `xlab()`, `ylab()`, `labs()`.

9.6.1 A note about `geom_errorbar()`

`geom_errorbar` has the option of one-sided intervals. Normally `ymin` and `ymax` will be located, but other options are `y` and `ymax`, and `ymin` and `y`. These may be specified in the mapping. However if the missing variable of these three possibilities is present in the data, for example `ymin` is present although `y` and `ymax` are specified, `ymin` will be found and a two-sided interval will be plotted. This may be prevented by explicitly writing "aes(ymin=NA)".

9.7 Explicitly finishing a plot

A plot is finished by the `ppgeoms()` function, which determines data limits, draws the axes, and data visualizations. The function is called automatically when a plot is printed or when a new subplot is started.

If something really weird happens, this may be caused by the geoms being evaluated later than expected. Calling `ppgeoms()` explicitly at the end of a subplot may help. One case where this is

needed is when data with multiple (primary and secondary) axes are plotted. See the example in the main documentation.

Chapter 10

Options

- Line width

  Sometimes the R code needs to know the line width for proper alignment, therefore it is recommended to use `pplinewidth()` rather than 'psset("linewidth=...").

- Label separation

  The default distance between the tick labels and the axis label is usually reasonable for the `x` axis, but for the `y` axis it depends on the space taken up by the tick labels, which is not taken into account automatically. See `ppsetxlabsep()` (alias `ppxlabsep()`), `ppsetylabsep()` (alias `ppylabsep()`) to specify such distances.

- Log scales

  Various functions are available to specify logarithmic axes, see `ppsetlogx()`, `ppsetlogy()`, `ppsetlogxy()`, `ppsetnologx()`, `ppssetnology()`, `ppsetnologxy()`. For examples, see "examples/nainf.R" and "examples/ticks.R".

- Secondary axes

  Secondary axes (at the right side or at the top, or primary axes at the left side or the bottom) may be selected by using `ppsetprimaryx()`, `ppsetprimaryy()`, `ppsetprimary()`, `ppsetsecondaryx()`, `ppsetsecondaryy()`, `ppsetsecondary()`. See "examples/secondary.R" for an example.

- Special coordinates

  PSTricks' macro `SpecialCoor` has not been wrapped, but with the functions `ppsetpolar()`, `ppsetcartesian()`, and `degrees()`, its functionality is partly available.

- Margins

  The location of axes in a subplot depends on an overall parameter "margin" and a factor "mrgaxes", see `ppsetmargins()`. Note that with a small margin, axis ticks and labels may be drawn outside of the subplot.

- PSTtoEPS

  The "PSTtoEPS" feature may be used to handle a large amounts of graphic objects (see Section 55 of the original documentation). The functions that handle this are `startP2E()` and `endP2E()` that may be called from `ppgeoms()` when this is requested by `ppsetpsttoeps()`. See "examples/psttoeps*.R" for examples and `PSTricks()` for a description of the "tmpdir" attribute. "tmpdir" is ideally "/tmp", but TeX is not allowed to write there by default; it needs "openout_any = a" in "texmf.cnf".

- `ppaxis()`

  Sometimes the geoms provide insufficient flexibility and the axes need to be drawn manually. See the main documentation for the options of `ppaxis()`. An obscure option is "noshow": the data will be scaled as if an axis is present, but the axis is not drawn - see "examples/noshow.R".

Chapter 11

Printing

If the `PSTricks` object, manipulated or not, is not assigned to a variable, it will be `printed`, that is, the `print` function for the object will be called with the object as its argument.

So with

```
PSTricks()
```

`print.PSTricks()` will be called, which calls `ppwrite()`, and then exits silently. The latter function does the following:

- closes a `pspicture` if one was open;
- closes the LaTeX document constructed in the `lines` attribute of the `PSTricks` object;
- writes an actual .tex file;
- calls a LaTeX engine to create a .pdf unless requested not to do so;
- crops the .pdf if requested;
- creates a .png if requested;
- creates a .eps if requested (see "examples/toeps.R");
- cleans up unless requested not to do so.

So by default, a .pdf is generated. And by default, the name is the same as the name of the running R script.

See the main documentation for the options of `ppwrite()`.

Chapter 12

Example Files

- 3d.R - Shows how to add PSTricks macro calls (see Section 1.6).
- basic.R - An example without using geoms.
- dots.R - Shows available dots in different styles.
- engines.R - Tests the engines that produce a .pdf from a .tex file. The "pdflatex" engine gives a two page .pdf when using landscape (could be prevented by using a smaller pspicture size). Also tests opacity. See Section 1.3.
- example.R - A simple example with a geom.
- ggstyle.R - Tries to generate a plot with a ggplot style (see Section 8.1.5).
- mtcars.R - Demonstrates how to print text labels (see Section 8.3.1).
- nainf.R - Shows handling of NaNs and Infs (see Section 9.3).
- noshow.R - Demonstrates scaling without an axis (see Section 10).
- oneliner.R - A plot generated by just one line.
- pictures.R - Two pictures showing the difference in clipping (see Section 8.2).
- psttoeps.R and psttoeps2.R - Tests of the PSTtoEps functionality (see Section 10).
- resetoptions.R - Shows that a new picture resets most options (see Chapter 6).
- secondary.R - Tests of secondary axes (see Section 10).
- subplot.R - Tests of subplot options (see Chapter 6).
- subplotl.R - Tests of subplot options on landscape paper (see Chapter 6).
- ticks.R - Examples of setting tickmarks.
- tiger.R - Tests including the "tiger.eps" file.
- toeps.R - Test of generating a .eps file (possibly for including in other documents).

12.1 Running all examples

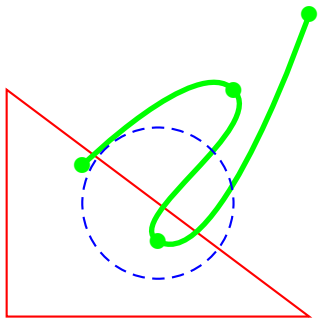
The script "examples/examples.sh" runs the above scripts, as well as runs all examples given with the documentation (so that these are not only tested, but their output may be viewed), with the output in directories named after the R script where these are located. Script "examples/clean.sh" may be used to clean up.

Chapter 13

The Examples on Wikipedia

13.1 Example 1

```
p <- PSTricks() %>%
    pppicture(5, 5) %>%
    pspolygon(c(1,5,1), c(1,1,4), "linecolor=red") %>%
    pscurve(c(5,3,4,2), c(5,2,4,3), "linecolor=green,linewidth=2pt,
                                     showpoints=true") %>%
    pscircle(3, 2.5, 1, "linecolor=blue,linestyle=dashed")

print(p, "wiki-example1", topng=TRUE)
```

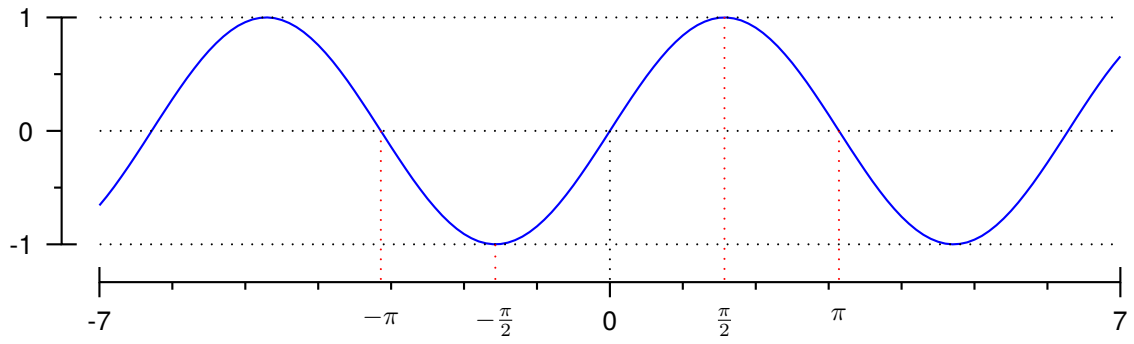


13.2 Example 2

pst-plot is, by design, not supported. The main purpose of the PSTricks package for R is that data generated in R can be visualized, rather that functions are plotted using PostScript itself.

```
data <- data.frame(x=seq(-7,7,0.1))
data$y <- sin(data$x)

p <- PSTricks() %>%
    pppicture(16, 6, data) %>%
    geom_line(par="linecolor=blue") %>%
    geom_abline(0, -1, "linestyle=dotted") %>%
    geom_abline(0, 0, "linestyle=dotted") %>%
    geom_abline(0, 1, "linestyle=dotted") %>%
    xlim(-7,7) %>% ylim(-1,1) %>%
    xticks(3,6) %>% yticks(3,1) %>%
    xlab("") %>% ylab("") %>%
```

```
    ppgeoms()

y <- cy(p, -Inf)
yt <- y - p$xticks$ticklength

x <- cx(p, -pi)
p <- psline(p, c(x,x), c(y,cy(p,0)), "linestyle=dotted,linecolor=red") %>%
     uput(x, yt, "$-\\pi$", 'd')

x <- cx(p, -pi/2)
p <- psline(p, c(x,x), c(y,cy(p,-1)), "linestyle=dotted,linecolor=red") %>%
     uput(x, yt, "$-\\frac{\\pi}{2}$", 'd')

x <- cx(p, 0)
p <- psline(p, c(x,x), c(y,cy(p,0)), "linestyle=dotted")

x <- cx(p, pi/2)
p <- psline(p, c(x,x), c(y,cy(p,1)), "linestyle=dotted,linecolor=red") %>%
     uput(x, yt, "$\\frac{\\pi}{2}$", 'd')

x <- cx(p, pi)
p <- psline(p, c(x,x), c(y,cy(p,0)), "linestyle=dotted,linecolor=red") %>%
     uput(x, yt, "$\\pi$", 'd')

print(p, "wiki-example2", topng=TRUE)
```

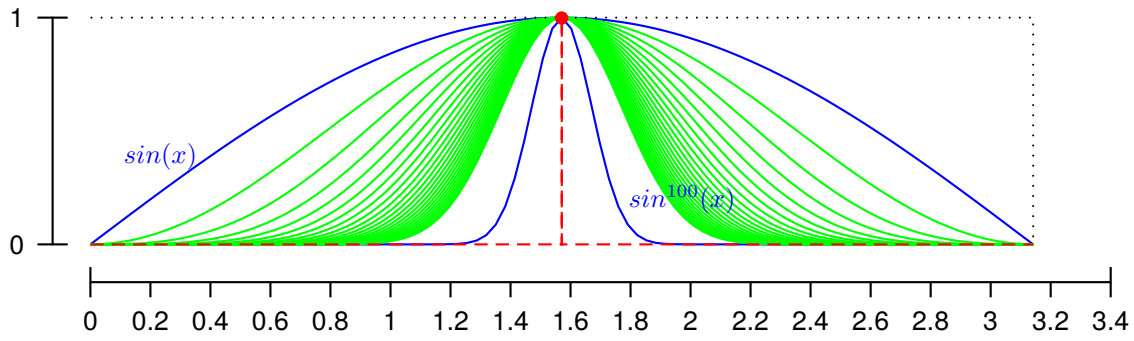


13.3 Example 3

Also `multido` is not implemented, because R's looping commands are available.

```
data <- data.frame(x=pi*seq(0,1,0.01))
data$y <- sin(data$x)

p <- PSTricks() %>%
    pppicture(16, 6, data) %>%
    lims(c(0,3.4), c(0,1)) %>%
    ticks(18, 2) %>%
    labs("", "")

p <- geom_line(p, par="linecolor=blue")

data$yp <- data$y^100
```

```
p <- geom_line(p, aes(x=x,y=yp), data=data, "linecolor=blue")

for (i in 2:25) {
    data$yp <- data$y^i
    p <- geom_line(p, aes(x=x,y=yp), data=data, "linecolor=green")
}

p <- geom_line(p, data=list(x=c(0,pi,pi),y=c(1,1,0)), par="linestyle=dotted")

p <- geom_line(p, data=list(x=c(0,rep(pi/2,3),pi),y=c(0,0,1,0,0)),
               par="linestyle=dashed,linecolor=red")

p <- geom_dots(p, data=list(x=pi/2,y=1), par="linecolor=red,dotsize=5pt")

labels <- list(x=c(0.4,1.75), y=c(sin(0.4),sin(1.75)^100),
               stuff=c("\\blue $sin(x)$","\\blue $sin^{100}(x)$"),
               refangle=c('l','r'))

p <- geom_uput(p, data=labels)

print(p, "wiki-example3", topng=TRUE)
```



Chapter 14

The “pst-user” Introductory Picture

It seems fitting to end this document with the “hello world” of the original PSTricks user’s guide (pst-user.pdf).

```
p <- PSTricks()

p <- pppicture(p, 31*0.4,c(-2*0.4,12*0.4))

p <- psset(p, "unit=.4cm") # works within pspicture

## The frame

p <- p %>%
    psframe(c(0,31), c(-2.5,12), "linewidth=2pt,framearc=.05,linecolor=gray") %>%
    uput(0,4.75, "leecheng", 'l', 90)

## The faucet

p <- p %>%
    psellipse(c(8,1), c(7,3), "linewidth=1pt") %>%
    psframe(c(6.4,8), c(6.5,7.5), "linecolor=white,fillstyle=solid,fillcolor=white") %>%
    psline(c(8,8,4), c(8,7.5,7.5), "linearc=.3,linewidth=1pt") %>%
    psbezier(c(4,3,3,3), c(7.5,7.5,6.5,5.5), "linewidth=1pt") %>%
    psline(c(8,8,5), c(6,6.5,6.5), "linearc=.3,linewidth=1pt") %>%
    psbezier(c(5,4,4,4), c(6.5,6.5,6.5,5.5), "linewidth=1pt") %>%
    psline(c(3,4), c(5.5,5.5), "linewidth=1pt") %>%
    psline(c(5,5,6,6), c(7.5,8,8,7.5), "linearc=.3,linewidth=1pt") %>%
    psframe(c(5.3,5.7), c(8,8.7), "linewidth=1pt") %>%
    psframe(c(4,7), c(8.7,9), "linewidth=1pt,framearc=1,fillstyle=solid,fillcolor=white")

## The droplets

s <- psbezier(, c(0,.25,-0.25,0), c(0,-.4,-.4,0), "linewidth=.5pt")
p <- multirput(p, c(3.5,0), c(4.8,-1), 4, s)

p <- rput(p, 5.5,0, "Dripping Faucet", 't')

## The center arrow

s <- paste0(psset(, "linewidth=2pt"),
            psline(, c(0,2), c(.5,.5)),
```

```
            psline(, c(0,2), c(-.5,-.5)),
            psline(, c(1.5,2.5,1.5), c(1,0,-1)))

## The model

s <- paste0(pspolygon(, c(1,1,4,4), c(4.5,4,4,4.5), "linecolor=white,fillstyle=vlines,
                                                     fillcolor=darkgray,hatchsep=.2"),
            psline(, c(1,4), c(4,4), "linewidth=2pt"),
            psline(, c(2.5,2.5,2.9,2.1,2.9,2.1,2.9,2.1,2.5,2.5),
                     c(4,3.5,3.3,2.9,2.5,2.1,1.7,1.3,1.1,.6),
                     "linewidth=1.5pt"),
            psframe(, c(1.8,3.2), c(-1,.6), "linecolor=black,linewidth=1.5pt,
                                             fillstyle=solid,fillcolor=lightgray"),
            rput(, 2.5,-.2, "$M$"),
            psline(, c(3.7,3.7), c(-.9,.5), arrows="<->"),
            psframe(, c(1.8,3.2), c(-3.5,-1.9), "linecolor=black,linewidth=1.5pt,
                                                 fillstyle=solid,fillcolor=lightgray"),
            rput(, 2.5,-2.7, "$m$"),
            psline(, c(5,5), c(1,-1), arrows="->"),
            rput(, 5.5,0, "$g$", 'l'),
            psline(, c(3.7,3.7), c(-2,-3.4), arrows="->"),
            rput(, 2.5,-4, "Mathematical Model for", 't'),
            rput(, 2.5,-5, "a Dripping Faucet", 't'),
            rput(, -6,-2, s))

p <- rput(p, 20,5, s)

print(p, "leecheng", topng=TRUE)
```