Mkpic: how Perl can help TEX

Wybo Dekker
wybo@dekkerdocumenten.nl

February 21, 2015

Abstract

Perl may be an easy interface to TEX when it comes to repetitive tasks, like writing letters, creating reports from databases, and many more. This article shows how Perl can be used to generate many similar pictures *via* the *mfpic* style

Keywords: *perl, mfpic, mkpic*

1 Introduction

I recently had to produce about 40 pictures for insertion into a book on elementary mathematics. I decided that the *mfpic* would suite most of my needs. But writing *mfpic* commands is not easy. Figure 1, for example, can be constructed using the following *mfpic* commands:

```
 1 \mftitle{parabola}
   \setlength{\mfpicunit}{1mm}
   \begin{mfpic}[16][5.45]{0}{4}{-6}{5}
   \axes
 5 \hatchwd{2}
   \tlabel[bc](0,5.54){$y$}
   \tlabel[cl](4.21,0){$x$}
   \tlabel[tc](2,-0.18){\strut 2}
   \tlabel[bc](3,0.18){\strut 3}
10 \tlabel[cr](-0.07,-5){\strut -5}
   \tlabel[cr](-0.07,0){\strut 0}
   \tlabel[cr](-0.07,4){\strut 4}
   \rhatch\lclosed\connect
   \lines{(0,0),(0,4)}
15 \function{0,3,.05}{4-x*x}
   \lines{(3,-5),(3,0)}
   \endconnect
   \function{0,3.2,.05}{4-x*x}
   \dotted\arrow\lines{(3,-5),(0,-5)}
20 \dotted\arrow\lines{(3,-5),(3,0)}
   \tlabel[bc](3,4){\parbox[b]{60mm}{%
     \center $f(x)=4-x^2$}}
   \arrow\lines{(3,3.46),(1.7,1.1)}
   \tlabel[bc](2,5){\parbox[b]{60mm}{%
25   \center Area $O_1$}}
   \arrow\lines{(2,4.46),(1,2)}
   \tlabel[bc](4,2){\parbox[b]{60mm}{%
     \center Area $O_2$}}
   \arrow\lines{(4,1.46),(2.8,-2)}
30 \end{mfpic}
```

As you can see, this implies a lot of typing and one has to type many nested [], { }, and () pairs. Also, several floating point numbers, such as those in lines 6–12, depend on the scaling factors defined in line 3. They have

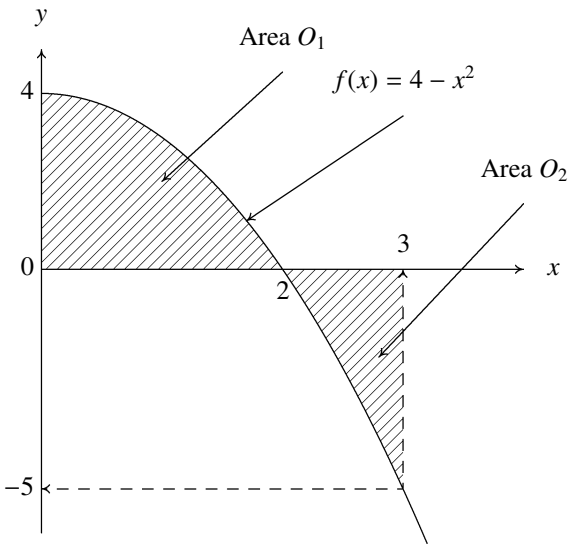

Figure 1: parabola

to be calculated manually, and changing the scale will imply recalculation of those values. The scale itself is set in line 3: I wanted the picture to be 64 mm wide, so I had to calculate $64/(4-0) = 16$ for the scaling factor in the x-direction. It would be much easier if one could type something like:

```
 1 begin parabola 64 64 0 -6 4 5 $x$ $y$
   xmark 2
   Xmark 3
   ymark -5 0 4
 5 bhat
   lines 0 0 0 4
   func 0 3 .05 4-x*x
   lines 3 -5 3 0
   ehat
10 func 0 3.2 .05 4-x*x
   xydrop 3 -5
   arrow 3 4 1.7 1.1 $f(x)=4-x^2$
   arrow 2 5 1 2 Area $O_1$
   arrow 4 2 2.8 -2 Area $O_2$
15 end
```

Here we see no brackets, braces or parentheses anymore, width and height are set straightforwardly to 64 mm and the labels along the axes are redefined as xmarks and ymarks, for which nothing has to be given but the x- and y-values, respectively. The corresponding y- and x-values are supposed to be calculated automati-

cally.

Another construction that frequently occurs in my pictures is a label with an arrow starting from the center of its baseline, such as the one in lines 21–23 in the long listing. This is replaced in the short listing with line 12, where the starting position of the arrow is supposed to be calculated automatically. As a result, if I want to move the label, the arrow is moved with it automatically.

All this is possible by using a *Perl* interface that converts the short command file into an *mfpic* source file.

2 The Perl interface

In the *Perl* script *mkpic*, the available commands are all defined in the subroutine `parse_input`. My initial version defined only a few commands, and while using the script, new commands were inserted when they were needed. It's easy to insert your own new commands here, just look at what's already there and create new commands by analogy. The `__DATA__` section of the script contains the picture needed for this documentation (and some more).

The first was the *begin* command, of course, which has also the most complex definition, as it defines many scale-dependent variables and TEX commands that might be useful for any command defined later.

2.1 How to use mkpic

First of all, read the manpage of the *Perl*-script, generated from the script using pod2latex, which is shown in section 4.

The easiest way to use the script is to append your own commands to the `__DATA__` section of the script, maybe after removing what's already there, and run it. This will produce a file mkpic.sty, which provides LATEX-commands named `\Fig<name>`, where `<name>` stands for every name you use in the *begin* command. Finally, you can use those `\Fig<name>` commands in a LATEX document.

3 Some more examples

Here are a few more examples illustrating some features of the *mkpic* script:

The following commands will produce figure 2:

```
 1 begin droparrows 64 64 0 3 12 8 $x$ $y$
   xmark $a$ 2 $x_1$ 4 $x_2$ 8 $b$ 10
   ydrop 2 4.414
   ydrop 4 5
 5 ydrop 8 5.828
   ydrop 10 6.162
   label cc 4 4 $f(x_1)$
   label cc 8 4 $f(x_2)$
   label cc 7 8 $f(x)$ increases on $[a,b]$
10 label cc 7 7.5 $x_1<x_2\Rightarrow f(x_1)<F(x_2)$
   func 1 11 .1 x**(.5)+3
   end
```

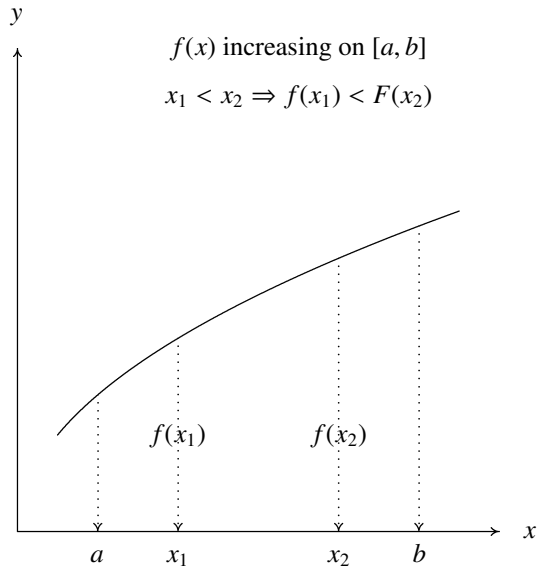

Figure 2: droparrows

These commands illustrate how valid *mfpic* commands can be interspersed between *mkpic* commands (see figure 3:

```
 1 begin asymptotes 64 64 0 0 10 10 $x$ $y$
   curve 1 1 2 3 4 5.7 7 8.1 9 9
   \shift{(-.05,.05)}
   point 2 3 4 5.7 7 8.1
 5 \shift{(-.05,.05)}
   func 1.4 2.6 .1 1.65*x-.3
   func 3.2 4.8 .1 1.025*x+1.6
   func 6.1 7.9 .1 .62*x+3.76
   label cl 9.5 9 $f(x)$
10 label tl 5 5 $f^\prime(x)>0$
   label tl 5 4 $f^\prime(x)$ decreasing
   end
```

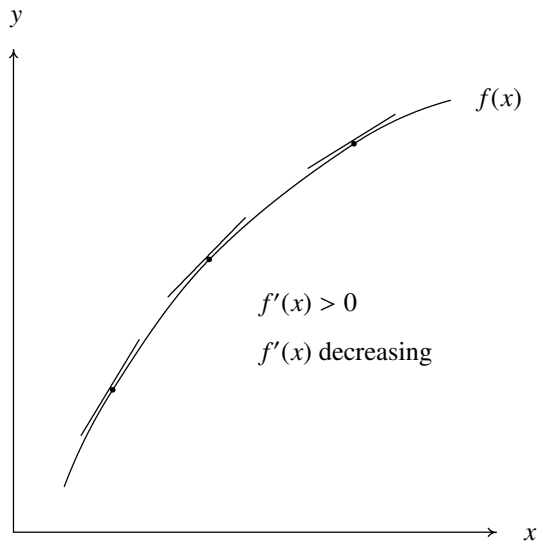

Figure 3: asymptotes

Figure 4 is produced by:

```
begin inflections 64 64 -.85 -1.5 1.5 5 $x$ $y$
func -.6 1.5 .05 6*(x**4)-8*(x**3)+1
lines -.2 1 .2 1
label cr -.25 1 horizontal
arrow .5 2 0 1 inflection point
arrow .8 1 .65 0 inflection point
arrow 1 5 1.5 4.375 $f(x)=6x^4-8x^3+1$
Xmark 1
ymark \raisebox{-3.5mm}{0} 0 -1
xydrop 1 -1
end
```

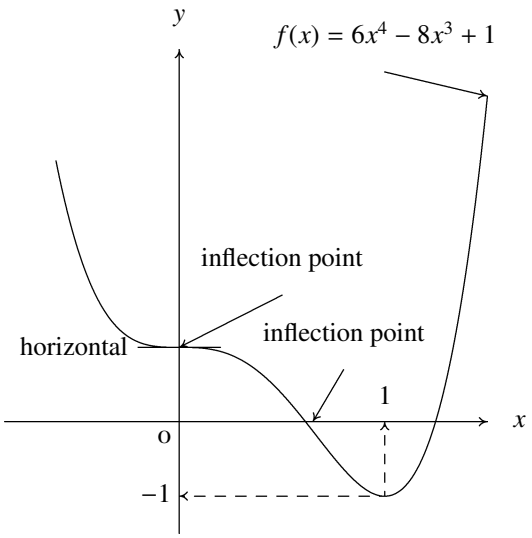

Figure 4: inflections

And here is an elaborate quasi 3D picture. It shows how comments can be inserted. Standard axes are suppressed because they need special treatment (see figure 5):

```
begin paraboloid 64 64 -4 -4 4 4 - -
\dashed
lines -4 0 0 0 0 0 0 -4 # neg z and neg y
\dashed
lines 0 0 3 1.73        # neg x
\arrow
lines 0 0 0 4           # pos y
\arrow[5]
lines 0 0 4 0           # pos z
\arrow[5]
lines 0 0 -3 -1.73      # pos x
\dotted
lines -1 4 -1 0 -4 -1.73  # intersections y=-1 plane
\dotted
lines -1 -.577 -4 -.577 # intersection x=2 plane with xy-plane
% extra helplines
\dotted
lines -2 .423 -2 -.577 0 0 0 1 -2 .423
Ymark 3
% end of extras
\dotted
lines 0 3 -1 3 -4 1.27
\dashed\sclosed
curve -3 2.42 -1.5 2.711 -1 2.42 -2.5 2.134
label bc 0 \yhi $z$
label cl \xhi 0 $y$
label tr -3.1 -1.8 $x$
label cl -.85 -.577 2
label tc 0 5.5 $f(x,y)=x^2-4x+2y^2+4y+7$
xmark -1
Ymark 1
\shift{(-2,.42)}
\dashed
func 0 .5 .1 9*x*x
func -.5 0 .1 7*x*x
\dashed
func -1 0 .1 2*x*x
func 0 1 .1 2*x*x
end
```

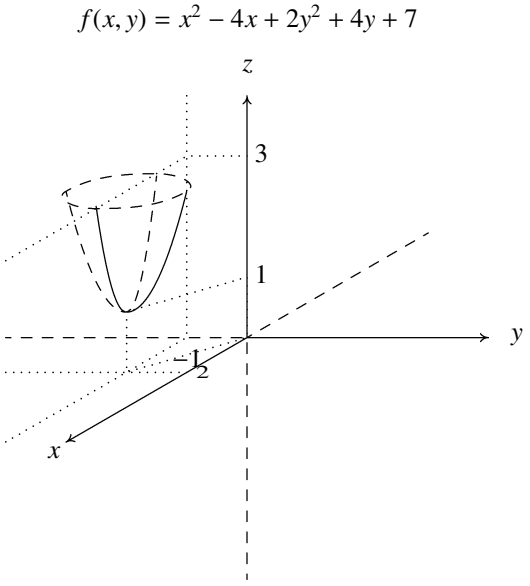

Figure 5: paraboloid

4 The mkpic manpage

mkpic - interface for making pictures with mfpic

Synopsis

mkpic [options] [picfile]

options:

```
--clean               remove all but input file and die
--pdfsample           create pdf file with sample images
--font=<font command> set default font for labels
--[no]box             produce framed boxes
--version             report version number and die
--help                display help info and die
--[no]debug           display debugging information
--log=<logfile>       file for warning messages
```

commands:

```
begin    name xl yl xmin ymin xmax ymax xlabel ylabel
end
stop
<var>=<value>
#        comment

arccst   xcenter ycenter xstart ystart theta
arcset   xstart ystart xend yend theta
arccrtt  xcenter ycenter radius theta1 theta2
arc3     x1 y1 x2 y2 x3 y3

xmark    [label1] x1 [label2] x2 ...
Xmark    [label1] x1 [label2] x2 ...
ymark    [label1] y1 [label2] y2 ...
Ymark    [label1] y1 [label2] y2 ...

xdrop    x y
ydrop    x y
xydrop   x y

arrow    x1 y1 x2 y2 label
label    YX x y label
xlabels  YX x y dx label ...
ylabels  YX x y dy label ...

point    x1 y1 x2 y2 ...
dpoint   x1 y1 dx1 dy1 ...
lines    x1 y1 x2 y2 ...
dlines   x1 y1 dx1 dy1 ...
curve    x1 y1 x2 y2 ...
dcurve   x1 y1 dx1 dy1 ...

rect     x1 y1 x2 y2
drect    x y dx dy
dcrect   x y dx dy
crect    x1 y1 x2 y2
arect    xc yc width height theta
bar      x xdev height

func     xmin xmax step expression-in-x

grid     dx dy xgap ygap
hatch
bhat
ehat
```

Description

mkpic provides an easy interface for making small pictures with mfpic. To this end you create an input file has to be created consisting of commands, one per line, with space separated parameters (or you modify the `__DATA__` section of the **mkpic** script, which is used if you run it without an input file). **mkpic** produces two files. Assuming an input file named *picfile* defaulting to *mkpic* these are:

picfile.mac a macro file which will contain **TeX** commands for every picture

picfile.sty a style file for latex, defining the same **TeX** commands for every picture.

With the **–pdfsample** option, two other files are produced:

picfile.pdf a PDF file containing all pictures. This lets you easily check the results of your designs.

picfile.tex the **TeX** source used for creating this PDF file.

In **LaTeX**, you have to include `\usepackage{picfile}` and to include commands like `\Fig<name>` in your source, where `name` is the name you gave one of your pictures in an **mkpic** `begin` command.

In **TeX** and **ConTeXt**, you have to `\input` *picfile.mac* and to include commands like `\Fig<name>` in your source, where `name` is the name you gave one of your pictures in an **mkpic** `begin` command.

In **TeX**, you must use the `\bye` command (**not** `\end` to finish your **TeX** source

See the RUNNING section for how to run **mkpic** and **TeX/LaTeX/ConTeXt**.

COMMANDS

The source is set up so that it is easy to add your own commands,

Currently the following commands have been implemented:

begin end Every picture begins with the **begin** command and ends with the **end** command. The **begin** command defines a name for the picture and defines a tex command with that name, prefixed with 'Fig'. The resulting command is written to a *.mac* file. Thus the command

```
begin aa ...
```

starts writing `\def\Figaa{...` to the *.mac* file, and the picture can be reproduced in a **TeX** document by importing the *.mac* file and using the `\Figaa` command.

xl and yl are the lengths of the x- and y-axes. xlabel and ylabel are the label that are placed at the ends of those axes. Use a space to suppress labeling, or "-" to suppress drawing the axes at all.

stop stops further reading of the input. Useful if you have many pictures, but want to see only the first few for testing purposes.

var=value sets the variable var to value. This variable, or an expression containing it, can be used instead of any numerical parameter. Variable names may contain lower and uppercase letters, digits or underscores, with the restriction that they must start with a letter and may not end in an underscore.

`#` denotes a comment

xmark ymark Xmark Ymark These commands place one or more labels along the x- or y-axes, either below (**xmark** and **ymark**) of above (**Xmark** and **Ymark**) the axis.

For the **[xXyY]mark** commands a parameter containing any character other than [-.0-9] is interpreted as the label (this implies that you cannot use expressions here!) to be placed and its position is expected in the next parameter. If a parameter is just a number, it is placed at that x-position. If you want a number to be interpreted as a label, put it in braces: *{1950}*.

arccst (Mnemonic: **c**enter **s**tart **t**heta.) Draws an arc with its center in *xcenter,ycenter*, starting in *xstart,ystart* and with an arc length of *theta* degrees.

arcset (Mnemonic: **s**tart **e**nd **t**heta.) Draws an arc starting in *xstart,ystart*end ending in *xend,yend* and with an arc length of *theta* degrees.

arccrtt (Mnemonic: **c**enter **r**adius **t**heta1 **t**heta2.) Draws an arc with its center in *xcenter,ycenter*, a radius *radius* starting at *theta1* degrees Ãęnd ending at *theta2* degrees.

arc3 (Mnemonic: **3** points.) Draws an arc starting at *(x1,y1), through (x2,y2) and ending in (x3,y3).*

xdrop ydrop xydrop These commands draw dotted arrows perpendicularly to the x-axis, the y-axis and both axes, respectively, ending on the axes with the arrow head.

arrow draws an arrow from *(x1,y1)* to *(x2,y2)* labeled on its tail with *label*

label draws a label at *(x,y). YX* tells how it will be adjusted: for Y=t,b,c *(x,y)* will be, in the y-direction, on top, bottom or center of the label respectively, for X=l,r,c it will be, in the x-direction, left, right or center adjusted on *(x,y)*. Thus

```
label tl 0 0 Hello World!
```

will draw the string "Hello World" with its lower left corner at (0,0)

xlabels draws many labels, starting at *(x,y)*, and incrementing *x* with *dx* after every label. *YX*: see **label**. Labels may not contain spaces; if you need spaces, use ~ instead.

ylabels Same as **xlabels**, but incrementing *y* with *dy* instead.

point draws points (dots) at *(x1,y1)*, *(x2,y2)* et cetera.

dpoint draws points (dots) starting at *(x1,y1)* and then moving by *(dx1,dy1)*, *(dx2,dy2)* et cetera.

lines draws line segments from *(x1,y1)* to *(x2,y2)*, *(x3,y3)* et cetera.

dlines draws line segments starting at *(x1,y1)* and then moving by *(dx1,dy1)*, *(dx2,dy2)* et cetera.

curve draws a bezier curve from *(x1,y1)* to *(x2,y2)*, *(x3,y3)* et cetera.

dcurve draws a bezier curve starting at *(x1,y1)* and then moving by *(dx1,dy1)*, *(dx2,dy2)* et cetera.

rect draws a rectangle with diagonal points at *(x1,y1)* and *(x2,y2)*.

drect draws a rectangle with diagonal points at *(x,y)* and *(x+dx,y+dy)*.

crect clears a rectangle with diagonal points at *(x1,y1)* and *(x2,y2)*.

dcrect clears a rectangle with diagonal points at *(x,y)* and *(x+dx,y+dy)*.

arect draws a rectangle with a widdh *width* and Ãę height *height*; the middle of the bottom is at *(xc,yc)* and the centerline through *(xc,yc)* makes an angle *theta* with the x-axis.

bar draws a equivalent with *rect x-xdev o x+xdev height*

func draws the function given by *expression-in-x* between *xmin* and *xmax*, stepping with *step* units in the x-direction. Note that the *expression-in-x* will be evaluated by **metafont**, so you will have to use metafont syntax.

grid draw dotted grid lines at distances dx and dy in the x- and y directions; the gaps between the dots are set to *xgap* an *ygap* respectively. For an esthetic appearance, be sure to use integer *dx/xgap* Ãęnd *dy/ygap* ratios.

hatch hatch the closed curve that follows.

bhat starts a path that will eventually be closed, and then hatched.

ehat ends a path started with **bhat**, closes it and then hatches it.

anything else will be inserted as is in the macro file, and therefore should be a valid *mfpic* statement. You use this when you need such a statement only once, or a few times and therefore see no need to define a proper command for it.

Running mkpic/TeX

The difficult way

The effect of running **mkpic picfile** is the creation of *picfile.mac*, which you can `\input` into a **TeX** or **ConTeXt** source, and *picfile.sty* which can be input into a **LaTeX** source using the `\usepackage` command.

After running **TeX** (or **LaTeX** or **ConTeXt**), you will find a file *picfile.mf* and you will have to run **metafont** on it, which (assuming you configured **TeX** for 600 dpi) produces picfile.600gf. This file will have to be converted to a pk file with **gftopk**. Finally, you need to run **TeX** again. So the sequence is:

$ mkpic picfile

$ tex file.tex

$ mf picfile

$ gftopk picfile.600gf

$ tex file

The easy way

You can also include this line into your **TeX** or **ConTeXt** source (before `\input`ing picfile.mac) or into your **LaTeX** source (before `\usepackage{picfile}`:

```
\immediate\write18{mkpic picfile}
```

and **TeX** (**LaTeX**, **ConTeXt**) will do everything for you, except that you will have to run **TeX** at least twice. You need, however, to 1) finish your texjob with `\bye`, **not** `\end`, and 2) enable the `\write18` command by setting, in *texmf.cnf*, the `shell_escape` variable to true (`t`) (or ask your system administrator to do so).

Bug

Currently only up to 256 pictures can be generated. In the future this problem will probably be solved by introducing more than one font and generating tex-command names that have the font name in front.

Author

Wybo Dekker (wybo@dekkerdocumenten.nl)