mfpic — Pictures in TEX with Metafont — version 0.25

• *Why?*

I got the idea for `mfpic` mostly out of a feeling of frustration. Different output mechanisms for printing or viewing TEX DVI files each have their own ways to include pictures. More often than not, there are provisions for including PostScript data into a DVI file using TEX `\special`'s. However, this technique seems far from TEX's ideal of device-independence, and besides, different TEX output drivers handle these `\special`'s in different ways. The same problems arise with including `tpic \special`'s. LaTEX's `picture` environment has a hopelessly limited supply of available objects to draw — if you want to draw a graph of a polynomial curve, you're out of luck. There is, of course, PICTEX, which is wonderfully flexible and general, but its most obvious feature is its speed — or rather lack of it. Processing a single picture in PICTEX can often take several minutes. It occured to me that it might be possible to take advantage of the fact that METAFONT is *designed* for drawing things. The result of pursuing this idea is `mfpic`, a set of macros for TEX and METAFONT which incorporate Metafont-drawn pictures into a TEX file. The nature of the macros, from the user's point of view, is very much like PICTEX. I do not pretend that `mfpic` has anything like the scope of PICTEX, but it should suit most purposes for drawing small graphs and including them in your TEX documents.

• *Author.*

`mfpic` was written primarily by Tom Leathrum (formerly moth@dartmouth.edu) during the late spring and summer of 1992. Those who helped most in this process are credited in the Acknowledgements.

• *Manifest.*

Seven files are included in the `mfpic` distribution:

`mfpicdoc.tex` (this document, which can be processed in plain TEX),
`mfpic.tex` (the TEX macros),
`graphbase.mf` (the METAFONT macros),
`objects.tex` (a sample file containing at least one picture of each object),
`pictures.tex` (a short sample file with a few more complicated pictures),
`lamfpic.tex` (an outmoded short header file for LaTeX, which now merely inputs mfpic.tex, and remains solely for backward compatibility), and
`lapictures.tex` (the LaTEX version of `pictures.tex`).

Information on how to set up a few specific configurations of computers is also available, separately.

• *Setting Up and Processing.*

Setting up TEX and METAFONT to process these files will, to an extent, depend on your local installation. The biggest problem you are likely to have, regardless of your installation, will be convincing TEX and its output drivers to find METAFONT's output files.

To process the sample file, first run TEX on the file `pictures.tex`. TEX will complain that it can't find the file `pics.tfm`, but will continue processing the file anyway. When the file is finished processing, you will now have a file called `pics.mf`. This is the METAFONT file containing the descriptions of the pictures for `pictures.tex`. You need to run

METAFONT on `pics.mf`, with `\mode=localfont` set up. (Read your METAFONT manual to see how to do this.) Now that you have the font and font metric files generated by METAFONT, reprocess the file `pictures.tex` with TeX. The resulting DVI file should now be complete, and you should be able to print and view it at your computer (assuming your viewer and print driver have been set up to be able to find the font generated by `pics.mf`).

These three steps of processing — processing with TeX, processing with METAFONT, and reprocessing with TeX — may not always be necessary. In particular, if you change the TeX document without making any changes at all to the pictures, then there will be no need to repeat either of the last two steps. There is also a somewhat subtle circumstance under which you can skip the third step — if you change the picture in such a way as *not* to affect the font metric file, then you do not have to reprocess with TeX, because the original metric used for the first step will put the pictures in the right places. The only `mfpic` macros that affect the font metric file are the two macros listed in the *Files and Environments* section below.

• *How It Works.*

When you run TeX on the file `pictures.tex`, TeX generates a file `pics.mf`. This file is formed by `\write` commands in the `mfpic` macros. The user should never have to read or change the file `pics.mf` directly — the `mfpic` macros take care of it. The user familiar with METAFONT will notice, by looking at the `mfpic` macros, that the `mfpic` drawing macros translate almost directly into simple METAFONT `draw` commands. The `\mflabel`'s and `\mfcaption`'s, however, are placed on the graph by TeX, not METAFONT.

• *The Macros.*

Files and Environments.

`\opengraphsfile{`⟨*font*⟩`}`... `\closegraphsfile`

These macros open and close the Metafont file which will contain the pictures to be included in this document. The name of the file will be ⟨*font*⟩`.mf`. If the ⟨*font*⟩ parameter is changed, you will have to reprocess the TeX file after processing ⟨*font*⟩`.mf`.

`\mfpicture[`⟨*xscale*⟩`][`⟨*yscale*⟩`]{`⟨*xneg*⟩`}{`⟨*xpos*⟩`}{`⟨*yneg*⟩`}{`⟨*ypos*⟩`}` ...`\endmfpicture`

These macros open and close the environment in which the rest of the macros below make sense. The `\mfpicture` macro also sets up the local coordinate system for the picture. The ⟨*xscale*⟩ and ⟨*yscale*⟩ parameters establish the length of the coordinate system units, as multiples of the user-settable dimension `\mfpicunit` which is initially one TeX point (`1pt`). At least one scale parameter must be specified, but if only one is specified, then they are assumed to be equal. The ⟨*xneg*⟩ and ⟨*xpos*⟩ parameters establish the lower and upper (resp.) bounds for the x-axis coordinates; similarly, ⟨*yneg*⟩ and ⟨*ypos*⟩ establish the bounds for the y-axis. These bounds are expressed in local units — in other words, the actual width of the picture will be ⟨*xscale*⟩·(⟨*xpos*⟩−⟨*xneg*⟩) points. These scales and bounds are used primarily to establish the metric for the character containing the picture described within the environment. If any of these parameters are changed, the ⟨*font*⟩`.tfm` file will be affected, so you will have to reprocess the TeX file after processing ⟨*font*⟩`.mf`.

The rest of the `mfpic` macros do not affect the font metric file (⟨*font*⟩`.tfm`), and so if these commands are changed or added in your document, you will not have to repeat the third step of processing (reprocessing with TeX) to complete your TeX document.

For backward compatibility with version `0.2` of `mfpic`, if `\picture` is *not* already defined, then it is let equal to `\mfpicture`, and `\endpicture` is let equal to `\endmfpicture`. Also, the macros `\mfpic` and `\endmfpic` are provided as abbreviations for `\mfpicture` and `\endmfpicture`, respectively.

For the remainder of the macros, the numerical parameters are expressed in the units of the local coordinate system specified by `\mfpicture`, unless otherwise indicated.

Points, Lines, Arrows, and Dotted Lines.

`\point{(`⟨*xval*⟩`,`⟨*yval*⟩`)}`

This macro draws a small filled circle centered at the point `(`⟨*xval*⟩`,`⟨*yval*⟩`)`. The diameter of the circle is determined by the dimension `\pointsize`. The default value of `\pointsize` is 2 points.

`\line{(`⟨*xval0*⟩`,`⟨*yval0*⟩`),(`⟨*xval1*⟩`,`⟨*yval1*⟩`)}`

This macro draws the line segment with endpoints at the points `(`⟨*xval0*⟩`,`⟨*yval0*⟩`)` and `(`⟨*xval1*⟩`,`⟨*yval1*⟩`)`.

`\arrow{(`⟨*xval0*⟩`,`⟨*yval0*⟩`),(`⟨*xval1*⟩`,`⟨*yval1*⟩`)}`

This macro draws an arrow starting from the point `(`⟨*xval0*⟩`,`⟨*yval0*⟩`)` and pointing to the point `(`⟨*xval1*⟩`,`⟨*yval1*⟩`)`. The length of the arrowhead is determined by the dimension `\headlen`. The default value of `\headlen` is 3 points. Each side of the arrowhead is a METAFONT Bezier curve starting from the tip of the arrowhead and heading initially parallel to the arrow line, then bending out to the endpoint.

`\dottedline{(`⟨*xval0*⟩`,`⟨*yval0*⟩`),(`⟨*xval1*⟩`,`⟨*yval1*⟩`)}`

This macro draws the dotted (or, more properly, dashed) line segment specified as in the `\line` macro. The length of the dashes is determined by the dimension `\dashlen`. The default value of `\dashlen` is 4 points. The space between the dashes is determined by the dimension `\dashspace`. The default value of `\dashspace` is 4 points. The space between the dashes may be adjusted by as much as $\frac{dashspace}{n}$, where n is the number of spaces appearing in the line segment, in order not to have partial dashes at the ends.

`\dottedarrow{(`⟨*xval0*⟩`,`⟨*yval0*⟩`),(`⟨*xval1*⟩`,`⟨*yval1*⟩`)}`

This macro draws a dotted arrow specified as in the `\arrow` macro, with the arrowhead the same shape and size as in the `\arrow` command and the dash length and spacing determined as in the `\dottedline` macro.

Axes and Axis Marks.

`\axes`

This macro draws the x and y axes for the coordinate system. The length of the arrowhead is determined by the dimension `\axisheadlen`. The default value of `\axisheadlen` is 5 points. The shape of the arrowhead is determined as in the `\arrow` macro.

`\xmarks{`⟨*value0*⟩`,`⟨*value1*⟩`,...}` and `\ymarks{`⟨*value0*⟩`,`⟨*value1*⟩`,...}`

These macros place hash (tic) marks on the x and y axes (resp.) at the places indicated by the values in the list. The length of the hash marks is determined by the dimension

`\hashlen`. The default value of `\hashlen` is 4 points.

Polygons.

`\rect{(`⟨*xval0*⟩`,`⟨*yval0*⟩`),(`⟨*xval1*⟩`,`⟨*yval1*⟩`)}`

This macro draws the rectangle specified by the given points (⟨*xval0*⟩,⟨*yval0*⟩) and (⟨*xval1*⟩,⟨*yval1*⟩) being any two opposite corners of the rectangle.

`\dottedrect{(`⟨*xval0*⟩`,`⟨*yval0*⟩`),(`⟨*xval1*⟩`,`⟨*yval1*⟩`)}`

This macro draws the dotted line rectangle specified as in the `\rect` macro. The dash length and spacing are determined as in the `\dottedline` macro.

`\block{(`⟨*xval0*⟩`,`⟨*yval0*⟩`),(`⟨*xval1*⟩`,`⟨*yval1*⟩`)}`

This macro draws the filled rectangle specified as in the `\rect` macro.

`\rectshade{(`⟨*xval0*⟩`,`⟨*yval0*⟩`),(`⟨*xval1*⟩`,`⟨*yval1*⟩`)}`

This macro draws dots shading in the rectangle specified as in the `\rect` macro. This macro does *not* draw the outline rectangle. The spacing between the dots is determined by the dimension `\shadespace`. The default value of `\shadespace` is 1 point.

`\polygon{(`⟨*xval0*⟩`,`⟨*yval0*⟩`),(`⟨*xval1*⟩`,`⟨*yval1*⟩`),...}`

This macro draws a polygon with the specified points at the vertices. The resulting polygon can *not* be allowed to cross itself.

`\polyshade{(`⟨*xval0*⟩`,`⟨*yval0*⟩`),(`⟨*xval1*⟩`,`⟨*yval1*⟩`),...}`

This macro shades the region enclosed by the polygon specified as in the `\polygon` macro. The spacing between dots is determined as in the `\shaderect` macro.

`\polyfill{(`⟨*xval0*⟩`,`⟨*yval0*⟩`),(`⟨*xval1*⟩`,`⟨*yval1*⟩`),...}`

This macro fills the region enclosed by the polygon specified as in the `\polygon` macro.

Parameters.

There are many parameters to `mfpic` which the user can modify to obtain different effects, such as different arrowhead size or shape. Most of these parameters have been described already in the context of macros they modify, but they are all described together here. Most of the parameters are stored by TEX as dimensions, and so are available globally, even if there is no METAFONT file open; changes to them are subject to the usual TEX rules of scope. Some parameters, however, are stored by METAFONT, so the macros to change them will have no effect unless a METAFONT file is open, and the changes are subject to METAFONT's rules of scope — to the `mfpic` user, this means that changes inside the `\mfpicture` ... `\endmfpicture` environment are local to that environment, but other TEX groupings have no effect on scope.

`\pointsize`

This dimension stores the diameter of the filled circle drawn by the `\point` macro. The default value is 2 points.

`\pen{`⟨*pensize*⟩`}`

This macro establishes the width of the drawing pen. The default is 0.5 points. This quantity is stored by METAFONT.

\headlen

This dimension stores the length of the arrowhead drawn by the \arrow macro. The default value is 3 points.

\headshape{$\langle hdwdr \rangle$}{$\langle hdten \rangle$}

This macro establishes the shape of the arrowhead drawn by the \arrow and \axes macros. The value of $\langle hdwdr \rangle$ is the ratio of the width of the arrowhead to its length, and $\langle hdten \rangle$ is the tension of the Bezier curves. The default values are both 1. These values are stored by METAFONT. Setting \headshape{1}{infinity} will make the sides of the arrowheads straight lines.

\dashlen, \dashspace

These dimensions store, respectively, the length of dashes and the length of spaces between dashes for lines drawn by the \dottedline macro. The space between the dashes may be adjusted by as much as $\frac{dashspace}{n}$, where n is the number of spaces appearing in the line segment, in order not to have partial dashes at the ends. The default values are both 4 points.

\dashlineset, \dotlineset

These macros provide convenient standard settings for the \dashlen and \dashspace dimensions. The macro \dashlineset sets both values to 4 points; the macro \dotlineset sets \dashlen to 1 point and \dashspace to 2 points.

\axisheadlen

This dimension stores the length of the arrowhead drawn by the \axes macro. The default value is 5 points.

\hashlen

This dimension stores the length of the axis hash (tic) marks drawn by the \xmarks and \ymarks macros. The default value is 4 points.

\shadespace

This dimension establishes the spacing between dots drawn by the \rectshade and other shading macros. The default value is 1 point.

\darkershade, \lightershade

These macros both multiply the \shadespace dimension by constant factors, $\frac{5}{6}$ and $\frac{6}{5}$ respectively, to provide convenient standard settings for several levels of shading.

Circles, Ellipses, and Disks.

\circle{($\langle xval \rangle$,$\langle yval \rangle$),$\langle rad \rangle$}

This macro draws a circle centered at the point ($\langle xval \rangle$,$\langle yval \rangle$) and with radius $\langle rad \rangle$.

\circshade{($\langle xval \rangle$,$\langle yval \rangle$),$\langle rad \rangle$}

This macro draws a shaded circle, determined as in the \circle macro. The spacing between dots is determined as in the \shaderect macro.

\cdisk{($\langle xval \rangle$,$\langle yval \rangle$),$\langle rad \rangle$}

This macro draws a filled circle, determined as in the \circle macro.

`\ellipse{(`$\langle \mathit{xval}\rangle$`,`$\langle \mathit{yval}\rangle$`),`$\langle \mathit{xrad}\rangle$`,`$\langle \mathit{yrad}\rangle$`}`

This macro draws an ellipse with the x radius $\langle \mathit{xrad}\rangle$ and y radius $\langle \mathit{yrad}\rangle$, centered at the point `(`$\langle \mathit{xval}\rangle$`,`$\langle \mathit{yval}\rangle$`)`.

`\ellshade{(`$\langle \mathit{xval}\rangle$`,`$\langle \mathit{yval}\rangle$`),`$\langle \mathit{xrad}\rangle$`,`$\langle \mathit{yrad}\rangle$`}`

This macro draws a shaded ellipse, determined as in the `\ellipse` macro. The spacing between dots is determined as in the `\shaderect` macro.

`\edisk{(`$\langle \mathit{xval}\rangle$`,`$\langle \mathit{yval}\rangle$`),`$\langle \mathit{xrad}\rangle$`,`$\langle \mathit{yrad}\rangle$`}`

This macro draws a filled ellipse, determined as in the `\ellipse` macro.

`\rotatedellipse{(`$\langle \mathit{xval}\rangle$`,`$\langle \mathit{yval}\rangle$`),`$\langle \mathit{xrad}\rangle$`,`$\langle \mathit{yrad}\rangle$`,`$\langle \mathit{theta}\rangle$`}`

This macro draws an ellipse with the x radius $\langle \mathit{xrad}\rangle$ and y radius $\langle \mathit{yrad}\rangle$ (before rotation), centered at the point `(`$\langle \mathit{xval}\rangle$`,`$\langle \mathit{yval}\rangle$`)`, and rotated around its center by the angle $\langle \mathit{theta}\rangle$ (expressed in degrees).

`\rotatedellshade{(`$\langle \mathit{xval}\rangle$`,`$\langle \mathit{yval}\rangle$`),`$\langle \mathit{xrad}\rangle$`,`$\langle \mathit{yrad}\rangle$`,`$\langle \mathit{theta}\rangle$`}`

This macro draws a shaded ellipse, determined as in the `\rotatedellipse` macro. The spacing between dots is determined as in the `\shaderect` macro.

`\rotatededisk{(`$\langle \mathit{xval}\rangle$`,`$\langle \mathit{yval}\rangle$`),`$\langle \mathit{xrad}\rangle$`,`$\langle \mathit{yrad}\rangle$`,`$\langle \mathit{theta}\rangle$`}`

This macro draws a filled ellipse, determined as in the `\rotatedellipse` macro.

Circular Arcs.

`\arc{(`$\langle \mathit{xval0}\rangle$`,`$\langle \mathit{yval0}\rangle$`),(`$\langle \mathit{xval1}\rangle$`,`$\langle \mathit{yval1}\rangle$`),`$\langle \mathit{sweep}\rangle$`}`

This macro draws a circular arc starting from the point `(`$\langle \mathit{xval0}\rangle$`,`$\langle \mathit{yval0}\rangle$`)`, ending at the point `(`$\langle \mathit{xval1}\rangle$`,`$\langle \mathit{yval1}\rangle$`)`, and covering an arc angle of $\langle \mathit{sweep}\rangle$ degrees, measured counterclockwise around the center of the circle. If, for example, the points `(`$\langle \mathit{xval0}\rangle$`,`$\langle \mathit{yval0}\rangle$`)` and `(`$\langle \mathit{xval1}\rangle$`,`$\langle \mathit{yval1}\rangle$`)` lie on a horizontal line with `(`$\langle \mathit{xval0}\rangle$`,`$\langle \mathit{yval0}\rangle$`)` to the left, and $\langle \mathit{sweep}\rangle$ is between 0 and 180 (degrees), then the center of the circle will be above the horizontal line (in order for the angle to be counterclockwise). Negative values of $\langle \mathit{sweep}\rangle$ give arcs curving in the other direction.

`\arcarrow{(`$\langle \mathit{xval0}\rangle$`,`$\langle \mathit{yval0}\rangle$`),(`$\langle \mathit{xval1}\rangle$`,`$\langle \mathit{yval1}\rangle$`),`$\langle \mathit{sweep}\rangle$`}`

This marco draws an arc, specified as in the `\arc` macro, with an arrowhead placed at the `(`$\langle \mathit{xval1}\rangle$`,`$\langle \mathit{yval1}\rangle$`)` end. The shape and size of the arrowhead are the same as in the `\arrow` macro.

`\arcshade{(`$\langle \mathit{xval0}\rangle$`,`$\langle \mathit{yval0}\rangle$`),(`$\langle \mathit{xval1}\rangle$`,`$\langle \mathit{yval1}\rangle$`),`$\langle \mathit{sweep}\rangle$`}`

This macro shades the region enclosed by the arc, specified as in the `\arc` macro, and the line segment between `(`$\langle \mathit{xval0}\rangle$`,`$\langle \mathit{yval0}\rangle$`)` and `(`$\langle \mathit{xval1}\rangle$`,`$\langle \mathit{yval1}\rangle$`)`. The spacing between dots is determined as in the `\shaderect` macro.

`\arcfill{(`$\langle \mathit{xval0}\rangle$`,`$\langle \mathit{yval0}\rangle$`),(`$\langle \mathit{xval1}\rangle$`,`$\langle \mathit{yval1}\rangle$`),`$\langle \mathit{sweep}\rangle$`}`

This macro fills the region specified as in the `\arcshade` macro.

Polar Coordinates.

`\linedir{(`$\langle \mathit{xval}\rangle$`,`$\langle \mathit{yval}\rangle$`),`$\langle \mathit{theta}\rangle$`,`$\langle \mathit{length}\rangle$`}`

This macro draws a line segment starting from the point `(`$\langle \mathit{xval}\rangle$`,`$\langle \mathit{yval}\rangle$`)` and going in the direction specified by ⟨*theta*⟩, where ⟨*theta*⟩ is an angle measured in degrees counterclockwise from the direction parallel to the positive x axis, and with length ⟨*length*⟩.

`\arrowdir{(`⟨*xval*⟩`,`⟨*yval*⟩`),`⟨*theta*⟩`,`⟨*length*⟩`}`

This macro draws an arrow starting from the point `(`⟨*xval*⟩`,`⟨*yval*⟩`)` and going in the direction specified by ⟨*theta*⟩, where ⟨*theta*⟩ is an angle measured in degrees counterclockwise from the direction parallel to the positive x axis, with length ⟨*length*⟩, and with an arrowhead at the end opposite the point `(`⟨*xval*⟩`,`⟨*yval*⟩`)`. The size and shape of the arrowhead are determined as in the `\arrow` macro.

`\arcth{(`⟨*xval*⟩`,`⟨*yval*⟩`),`⟨*theta1*⟩`,`⟨*theta2*⟩`,`⟨*rad*⟩`}`

This macro draws the arc with center at the point `(`⟨*xval*⟩`,`⟨*yval*⟩`)`, of radius ⟨*rad*⟩, from the angle ⟨*theta1*⟩ to the angle ⟨*theta2*⟩, where both angles are measured in degrees counterclockwise from the direction parallel to the x axis.

`\arctharrow{(`⟨*xval*⟩`,`⟨*yval*⟩`),`⟨*theta1*⟩`,`⟨*theta2*⟩`,`⟨*rad*⟩`}`

This macro draws an arc arrow, with the arc specified as in the `\arcth` macro.

`\wedgeshade{(`⟨*xval*⟩`,`⟨*yval*⟩`),`⟨*theta1*⟩`,`⟨*theta2*⟩`,`⟨*rad*⟩`}`

This macro shades the wedge-shaped region, from the angle ⟨*theta1*⟩ to the angle ⟨*theta2*⟩ inside the circle with center at the point `(`⟨*xval*⟩`,`⟨*yval*⟩`)` and radius ⟨*rad*⟩, where both angles are measured in degrees counterclockwise from the direction parallel to the x axis. The spacing between dots is determined as in the `\shaderect` macro.

`\wedgefill{(`⟨*xval*⟩`,`⟨*yval*⟩`),`⟨*theta1*⟩`,`⟨*theta2*⟩`,`⟨*rad*⟩`}`

This macro fills the wedge-shaped region specified as in the `\wedgeshade` macro.

Curves.

`\curve{(`⟨*xval0*⟩`,`⟨*yval0*⟩`),(`⟨*xval1*⟩`,`⟨*yval1*⟩`),...}`

This macro draws a METAFONT Bezier path through the points specified, in the order specified.

`\polyline{(`⟨*xval0*⟩`,`⟨*yval0*⟩`),(`⟨*xval1*⟩`,`⟨*yval1*⟩`),...}`

This macro draws a piecewise linear path through the points specified, in the order specified. For backward compatibility with version `0.2`, `\polycurve` is let equal to `\polyline`, but `polyline` is recommended as a more descriptive name.

`\curvedarrow{(`⟨*xval0*⟩`,`⟨*yval0*⟩`),(`⟨*xval1*⟩`,`⟨*yval1*⟩`),...}`

This macro draws a METAFONT Bezier curve, specified as in the `\curve` macro, then puts an arrowhead at the end. The arrowhead is the same size and shape as the arrowhead in the `\arrow` macro.

`\polyarrow{(`⟨*xval0*⟩`,`⟨*yval0*⟩`),(`⟨*xval1*⟩`,`⟨*yval1*⟩`),...}`

This macro draws a polygonal path, specified as in the `\polycurve` macro, then puts an arrowhead at the end. The arrowhead is the same size and shape as the arrowhead in the `\arrow` macro.

Cyclic Curves.

`\cyclic{(`⟨*xval0*⟩`,`⟨*yval0*⟩`),(`⟨*xval1*⟩`,`⟨*yval1*⟩`),...}`

This macro draws a cyclic METAFONT Bezier curve through the points specified. The curve can *not* be allowed to cross itself.

`\cycleshade{(`⟨*xval0*⟩`,`⟨*yval0*⟩`),(`⟨*xval1*⟩`,`⟨*yval1*⟩`),...}`

This macro shades the region enclosed by the cyclic curve specified as in the `\cyclic` macro. The spacing between dots is determined as in the `\shaderect` macro.

`\cyclefill{(`⟨*xval0*⟩`,`⟨*yval0*⟩`),(`⟨*xval1*⟩`,`⟨*yval1*⟩`),...}`

This macro fills the region enclosed by the cyclic curve specified as in the `\cyclic` macro.

Functions.

`\function{`⟨*xmin*⟩`,`⟨*xmax*⟩`,`⟨*step*⟩`,`⟨*expr*⟩`}`

This macro plots the function determined by $f(x)=$⟨*expr*⟩, where ⟨*expr*⟩ is written with the only unknown being `x`, using as x values the points starting with ⟨*xmin*⟩ and stepping by ⟨*step*⟩ until reaching ⟨*xmax*⟩. The function is interpolated with a METAFONT Bezier path. The expression *expr* is passed directly into the corresponding METAFONT macro and interpreted there, so METAFONT's rules for algebraic expressions apply. Operations available include `+`, `-`, `*`, `/`, and `**` (`x**y`$=x^y$), with `(` and `)` for grouping. Functions available include `round`, `floor`, `ceiling`, `abs`, `sqrt`, `sind`, `cosd`, `mlog`, and `mexp`. (*Notes:* the trigonometric functions `sind` and `cosd` take arguments in degrees; `mlog(x)`$=256\ln x$, and `mexp` is its inverse. There are other operations and functions available, but these are the most useful for plotting purposes.) You can also define the function $f(x)$ by cases using the Metafont conditional expression `if` ⟨*boolean*⟩`:` ⟨*expr*⟩ `else:` ⟨*expr*⟩ `fi`. Relations available for the ⟨*boolean*⟩ part of the expression include `=`, `<`, `>`, `<=`, and `>=`.

`\polyfunction{`⟨*xmin*⟩`,`⟨*xmax*⟩`,`⟨*step*⟩`,`⟨*expr*⟩`}`

This macro plots the function determined by $f(x)=$⟨*expr*⟩, as in the `\function` macro, but using a polygonal path instead.

`\parafcn{`⟨*tmin*⟩`,`⟨*tmax*⟩`,`⟨*step*⟩`,(`⟨*expr0*⟩`,`⟨*expr1*⟩`)}`

This macro plots the parametric path determined by $(x(t),y(t))=$`(`⟨*expr0*⟩`,`⟨*expr1*⟩`)`, where ⟨*expr0*⟩ and ⟨*expr1*⟩ are written with the only unknown being `t`, using as t values the points starting with ⟨*tmin*⟩ and stepping by ⟨*step*⟩ until reaching ⟨*tmax*⟩. The function is interpolated with a METAFONT Bezier path. The expressions ⟨*expr0*⟩ and ⟨*expr1*⟩ are passed directly into the corresponding METAFONT macro, as in the `\function` macro, so the same operations are available.

`\polyparafcn{`⟨*tmin*⟩`,`⟨*tmax*⟩`,`⟨*step*⟩`,(`⟨*expr0*⟩`,`⟨*expr1*⟩`)}`

This macro plots the parametric path specified as in the `\parafcn` macro, but using a polygonal path instead.

`\shadefcn{`⟨*xmin*⟩`,`⟨*xmax*⟩`}{`⟨*expr0*⟩`}{`⟨*expr1*⟩`}`

This macro draws the shaded region between the two functions $f(x)=$⟨*expr0*⟩ and $g(x)=$⟨*expr1*⟩, where ⟨*expr0*⟩ and ⟨*expr1*⟩ are written with the only unknown being `x`, bounded also by the vertical lines at ⟨*xmin*⟩ and ⟨*xmax*⟩. The functions f and g can *not* be allowed to cross between ⟨*xmin*⟩ and ⟨*xmax*⟩. The expressions ⟨*expr0*⟩ and ⟨*expr1*⟩ are passed directly into the corresponding METAFONT macro, as in the `\function` macro, so the same operations are available. The spacing between dots is determined as in the `\shaderect` macro; the function step size is the same distance.

Labels and Captions.

The next two macros do not affect the METAFONT file (⟨*font*⟩`.mf`) at all, but are added to the picture by TeX. Therefore, if these macros are changed or added in your document, there is no need to repeat either of the last two steps (processing with METAFONT or reprocessing with TeX) in order to complete your TeX document.

`\mflabel[`⟨*orient*⟩`]{`⟨*x*⟩`}{`⟨*y*⟩`}{`⟨*label text*⟩`}`

This macro places a TeX label on the graph. Without the `[`⟨*orient*⟩`]` parameter, the label is placed with the lower left-hand corner of the label at the point `(`⟨*x*⟩`,`⟨*y*⟩`)`. Note the different format of the point specification from the other `mfpic` macros — this is because `\mflabel` does not use METAFONT to place the label, but instead uses TeX `\kern` commands. The optional parameter `[`⟨*orient*⟩`]` specifies the relative placement of the label with respect to the point `(`⟨*x*⟩`,`⟨*y*⟩`)` — ⟨*orient*⟩ is a two-character sequence where the first character is one of `t` (top), `c` (center), or `b` (bottom), to specify vertical placement, and the second character is one of tt l (left), `c` (center), or `r` (right), to specify horizontal placement. The default orientation is equivalent to specifying `[bl]`. The macro `\label` has been let equal to `\mflabel` for backward compatibility with version `0.2`.

`\mfcaption{`⟨*caption text*⟩`}`

This macro places a caption at the bottom of the graph. The caption text can also include the macro `\\`, which forces a line break in the caption. The macro `\caption` has been let equal to `\mfcaption` for backward compatibility.

For Power Users Only.

`\mfcmd{`⟨*command text*⟩`}`

This macro writes the ⟨*command text*⟩ directly to the Metafont file, using a TeX `\write` command. This can have some rather bizarre consequences, though, so using it is not recommended.

- *Using* `mfpic` *with* LaTeX.

`mfpic.tex` is now LaTeX compatible. Since LaTeX uses `picture` for *its* `picture` environment, the name `mfpicture` is now preferred for an mfpic picture:

`\begin{mfpicture}...\end{mfpicture}`

An abbreviation '`mfpic`' is provided for '`mfpicture`'. The file `lamfpic.tex` remains solely for backward compatibility, and it now consists merely of the command `\input mfpic`. A LaTeX version of the sample file, `lapictures.tex`, has also been provided. Note that the `\opengraphsfile` and `\closegraphsfile` macros have *not* been converted into a LaTeX environment.

- *In the* LaTeX `center` *environment.*

To center an `mfpic` picture within a LaTeX `center` environment, go into horizontal mode after the `center`. For example, using `\noindent`:

```
\begin{center}
\noindent
\begin{mfpic}...
...
\end{mfpic}
\end{center}
```

Instead of the TEX primitive `\noindent`, the plain TEX macro `\leavevmode` may be used. The drawing is positioned in the same place in both cases. Unfortunately, experiments showed that the following device:

```
\ifvmode
      \leavevmode
\fi
```

within `mfpic.tex` to accommodate a possible enclosing LaTeX `center` environment, creates an hbox of excessive width, and that foils the use of `mfpic` in such valuable situations as placing two pictures side-by-side. So mfpic users must deal with the LaTeX `center` environment by inserting `\noindent` or `\leavevmode` themselves.

• *Acknowledgements.*

I would like to thank all of the people here at Dartmouth as well as out in the network world for testing `mfpic` and sending me back comments. I would particularly like to thank Geoffrey Tobin (ecsgrt@luxor.latrobe.edu.au) for his many suggestions, especially about cleaning up the METAFONT code, enforcing dimensions, fixing the dotted line computations, and speeding up the shading routines (through this process, Geoffrey and I managed to teach each other many of the subtleties of METAFONT); and Bryan Green (bgreen@sanjuan.uvic.ca) for his many suggestions, some of which (including his rewriting the `\mfcaption` macro) ultimately led to the current version's ability to put graphs in-line or side-by-side.

• *Changes History.*

Version 0.0.x: Development versions. Never released.

Version 0.1: First version released to network, Tue 30 June 1992.

Version 0.1.x: Test versions for new features, fixes, etc. None of them were released for general distribution. Many new features added, bugs worked out, code cleaned up.

Version 0.2: Second version released to network, Mon 7 September 1992.

Version 0.25: Bug fixes: Picture TFM detection, LaTeX picture environment respected, fractional scales, @ catcode restored, sharp character for MF file, bad mflabel options detected, MF char codes out of range (0 to 255) detected.

Tue 12 October 1993.