Drawing Graphs with MetaPost

John D. Hobby

Abstract

This paper describes a graph-drawing package that has been implemented as an extension to
the MetaPost graphics language. MetaPost has a powerful macro facility for implementing such
extensions. There are also some new language features that support the graph macros. Existing
features for generating and manipulating pictures allow the user to do things that would be
difficult to achieve in a stand-alone graph package.

Contents

1 Introduction 1
2 Using the Graph Macros 2
   2.1 Basic Graph-Drawing Commands 3
   2.2 Coordinate Systems . . . . . . . 5
   2.3 Explicit Grids and Framing . . . 8
   2.4 Processing Data Files . . . . . . 10
3 Manipulating Big Numbers 11
4 Typesetting Numbers 13
5 Conclusion 14
A Summary of the Graph Package 14
   A.1 Graph Administration . . . . . . 14
   A.2 Drawing and Labeling . . . . . . 15
   A.3 Grids, Tick Marks, and Framing 15
   A.4 Miscellaneous Commands . . . . 16
   A.5 Arithmetic on Numeric Strings . 16
   A.6 Internal Variables and Constants 16

1 Introduction

MetaPost is a batch-oriented graphics language based on Knuth’s \textsc{MetaFont}\textsuperscript{1}, but with PostScript\textsuperscript{2} output and numerous features for integrating text and graphics. The author has tried to make this
paper as independent as possible of the user’s manual \cite{5}, but fully appreciating all the material
requires some knowledge of the MetaPost language.

We concentrate on the mechanics of producing particular kinds of graphs because the question
of what type of graph is best in a given situation is covered elsewhere; e.g., Cleveland \cite{2, 3, 4} and
Tufte \cite{9}. The goal is to provide at least the power of UNIX\textsuperscript{3} \texttt{grip} \cite{1}, but within the MetaPost
language. Hence the package is implemented using MetaPost’s powerful macro facility.

The graph macros provide the following functionality:

1. Automatic scaling
2. Automatic generation and labeling of tick marks or grid lines
3. Multiple coordinate systems
4. Linear and logarithmic scales
5. Separate data files
6. Ability to handle numbers outside the usual range

\textsuperscript{1}\textsc{MetaFont} is a trademark of Addison Wesley Publishing Company.
\textsuperscript{2}\textsc{PostScript} is a registered trademark of Adobe Systems Inc.
\textsuperscript{3}\textsc{UNIX} is a registered trademark of UNIX System Laboratories, Inc.

1
7. Arbitrary plotting symbols
8. Drawing, filling, and labeling commands for graphs

In addition to these items, the user also has access to all the features described in the MetaPost user’s manual [5]. These include access to almost all the features of PostScript, ability to use and manipulate typeset text, ability to solve linear equations, and data types for points, curves, pictures, and coordinate transformations.

Section 2 describes the graph macros from a user’s perspective and presents several examples. Sections 3 and 4 discuss auxiliary packages for manipulating and typesetting numbers and Section 5 gives some concluding remarks. Appendix A summarizes the graph-drawing macros.

2 Using the Graph Macros

A MetaPost input file that uses the graph macros should begin with

    input graph

This reads a macro file graph.mp and defines the graph-drawing commands explained below. The rest of the file should be one or more instances of

    beginfig((figure number));
    (graphics commands) endfig;

followed by end.

The following (graphics commands) suffice to generate the graph in Figure 1 from the data file agepop91.d:

    draw begingraph(3in,2in);
    gdraw "agepop91.d";
    endgraph;

(Each line of agepop91.d gives an age followed the estimated number of Americans of that age in 1991 [8].)

Figure 1: A graph of the 1991 age distribution in the United States
2.1 Basic Graph-Drawing Commands

All graphs should begin with

\begin{verbatim}
begingraph(⟨width⟩,⟨height⟩);
\end{verbatim}

and end with \texttt{endgraph}. This is syntactically a ⟨picture expression⟩, so it should be preceded by \texttt{draw} and followed by a semicolon as in the example.\footnote{See the User’s Manual [5] for explanations of \texttt{draw} commands and syntactic elements like ⟨picture expression⟩.} The ⟨width⟩ and ⟨height⟩ give the dimensions of the graph itself without the axis labels.

The command

\begin{verbatim}
gdraw ⟨expression⟩ ⟨option list⟩
\end{verbatim}

draws a graph line. If the ⟨expression⟩ is of type string, it names a data file; otherwise it is a path that gives the function to draw. The ⟨option list⟩ is zero or more drawing options

\texttt{withpen(⟨pen expression⟩) | withcolor(⟨color expression⟩) | dashed(⟨picture expression⟩)}

that give the line width, color, or dash pattern as explained in the User’s Manual [5].

In addition to the standard drawing options, the ⟨option list⟩ in a \texttt{gdraw} statement can contain

\begin{verbatim}
plot ⟨picture expression⟩
\end{verbatim}

The ⟨picture expression⟩ gives a plotting symbol to be drawn at each path knot. The \texttt{plot} option suppresses line drawing so that\footnote{\texttt{Troff} users should replace \texttt{btex \$bullet\ etex} with \texttt{btex \$bu etex}.}

\begin{verbatim}
gdraw "agepop91.d" plot btex \$\bullet\ etex
\end{verbatim}

generates only bullets as shown in Figure 2. (Following the \texttt{plot} option with a \texttt{withpen} option would cause the line to reappear superimposed on the plotting symbols.)

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure2.png}
\caption{The 1991 age distribution plotted with bullets}
\end{figure}

Watch out for the following: the ⟨picture expression⟩ is placed with the lower-left corner at the path knot, not its center. If you want it to be dead-center, you have to correct the placement yourself. For the example above, you need something like this instead:

\begin{verbatim}
def MPbullet =
    btex \lower\fontdimen22\cmsy \hbox to 0pt{\hss\cmsy\char15\hss} etex
enddef;
\end{verbatim}
followed by:

\texttt{gdraw "agepop91.d" plot MPbullet}

The \texttt{glabel} and \texttt{gdotlabel} commands add labels to a graph. The syntax for \texttt{glabel} is

\begin{verbatim}
glabel. (label suffix)(⟨string or picture expression⟩, ⟨location⟩) (option list)
\end{verbatim}

where ⟨location⟩ identifies the location being labeled and ⟨label suffix⟩ tells how the label is offset relative to that location. The \texttt{gdotlabel} command is identical, except it marks the location with a dot. A ⟨label suffix⟩ is as in plain MetaPost: ⟨empty⟩ centers the label on the location; \texttt{lft}, \texttt{rt}, \texttt{top}, \texttt{bot} offset the label horizontally or vertically; and \texttt{ulft}, \texttt{urt}, \texttt{llft}, \texttt{lrt} give diagonal offsets. The ⟨location⟩ can be a pair of graph coordinates, a knot number on the last \texttt{gdraw} path, or the special location \texttt{OUT}. Thus

\begin{verbatim}
gdotlabel.top(btex $(50,0)$ etex, 50,0)
\end{verbatim}

would put a dot at graph coordinates (50,0) and place the typeset text “(50,0)” above it. Alternatively,

\begin{verbatim}
glabel.ulft("Knot3", 3)
\end{verbatim}

otypesets the string "Knot3" and places it above and to the left of Knot 3 of the last \texttt{gdraw} path. (The knot number 3 the path’s “time” parameter [5, Section 8.2].)

The ⟨location⟩ \texttt{OUT} places a label relative to the whole graph. For example, replacing \texttt{"gdraw "agepop91.d""} with

\begin{verbatim}
glabel.lft(btex \vbox{\hbox{Population} \hbox{in millions}} etex, OUT);
glabel.bot(btex Age in years etex, OUT);
gdraw "agepopm.d";
\end{verbatim}

in the input for Figure 1 generates Figure 3. This improves the graph by adding axis labels and using a new data file \texttt{agepopm.d} where the populations have been divided by one million to avoid large numbers. We shall see later that simple transformations such as this can be achieved without generating new data files.

![Figure 3: An improved version of the 1991 age distribution graph](image)

All flavors of \TeX can handle multi-line labels via the \texttt{\vbox} within \texttt{\hbox} arrangement used above, but \LaTeX users will find it more natural to use the \texttt{tabular} environment [7]. Troff user’s can use nofill mode:

\begin{verbatim}
btex .nf
Population
in millions etex
\end{verbatim}
2.2 Coordinate Systems

The graph macros automatically shift and rescale coordinates from data files, gdraw paths, and glabel locations to fit the graph. Whether the range of \( y \) coordinates is 0.64 to 4.6 or 640,000 to 4,600,000, they get scaled to fill about 88% of the height specified in the begingraph statement. Of course line widths, labels, and plotting symbols are not rescaled.

The setrange command controls the shifting and rescaling process by specifying the minimum and maximum graph coordinates:

\[
\text{setrange}(\langle \text{coordinates} \rangle, \langle \text{coordinates} \rangle)
\]

where

\[
\langle \text{coordinates} \rangle \rightarrow \langle \text{pair expression} \rangle \\
| \langle \text{numeric or string expression} \rangle, \langle \text{numeric or string expression} \rangle
\]

The first \( \langle \text{coordinates} \rangle \) give \((x_{\min}, y_{\min})\) and the second give \((x_{\max}, y_{\max})\). The lines \( x = x_{\min}, \)

\[
x = x_{\max}, y = y_{\min}, \text{ and } y = y_{\max}
\]

define the rectangular frame around the graph in Figures 1–3.

For example, an adding a statement

\[
\text{setrange(origin, whatever, whatever)}
\]

to the input for Figure 3 yields Figure 4. The first \( \langle \text{coordinates} \rangle \) are given by the predefined pair constant origin, and the other coordinates are left unspecified. Any unknown value would work as well, but whatever is the standard MetaPost representation for an anonymous unknown value.

\[
\text{draw begingraph(3in,2in);} \\
\text{glabel.lft(btex \vbox{\hbox{Population} \hbox{in millions}} etex, OUT);} \\
\text{glabel.bot(btex Age in years etex, OUT);} \\
\text{setrange(origin, whatever,whatever);} \\
\text{gdraw "agepopm.d";} \\
\text{endgraph;}
\]

Figure 4: The 1991 age distribution graph and the input that creates it.

Notice that the syntax for setrange allows coordinate values to be given as strings. Many commands in the graph package allow this option. It is provided because the MetaPost language uses fixed point numbers that must be less than 32768. This limitation is not as serious as it sounds.
because good graph design dictates that coordinate values should be “of reasonable magnitude” [2, 9]. If you really want $x$ and $y$ to range from 0 to 1,000,000,

```
setrange(origin, "1e6", "1e6")
```
does the job. Any fixed or floating point representation is acceptable as long as the exponent is introduced by the letter “e”.

Coordinate systems need not be linear. The `setcoords` command allows either or both axes to have logarithmic spacing:

```
⟨coordinate setting⟩ → setcoords(⟨coordinate type⟩, ⟨coordinate type⟩)
⟨coordinate type⟩ → log | linear | -log | -linear
```

A negative ⟨coordinate type⟩ makes $x$ (or $y$) run backwards so it is largest on the left side (or bottom) of the graph.

Figure 5 graphs execution times for two matrix multiplication algorithms using

```
setcoords(log, log)
```

to specify logarithmic spacing on both axes. The data file `matmul.d` gives timings for both algorithms:

```
20 .007861 standard MM: size, seconds
30 .022051
40 .060391
60 .15922
80 .4031
120 1.53
160 3.915
240 18.55
320 78.28
480 279.24
20 .006611 Strassen: size, seconds
30 .020820
40 .049219
60 .163281
80 .3975
120 1.3125
160 3.04
240 9.95
320 22.17
480 72.60
```

A blank line in a data file ends a data set. Subsequent `gdraw` commands access additional data sets by just naming the same data file again. Since each line gives one $x$ coordinate and one $y$ coordinate, commentary material after the second data field on a line is ignored.

Placing a `setcoords` command between two `gdraw` commands graphs two functions in different coordinate systems as shown in Figure 6. Whenever you give a `setcoords` command, the interpreter examines what has been drawn, selects appropriate $x$ and $y$ ranges, and scales everything to fit. Everything drawn afterward is in a new coordinate system that need not have anything in common with the old coordinates unless `setrange` commands enforce similar coordinate ranges. For instance, the two `setrange` commands force both coordinate systems to have $x$ ranging from 80 to 90 and $y$ starting at 0.

When you use multiple coordinate systems, you have to specify where the axis labels go. The default is to put tick marks on the bottom and the left side of the frame using the coordinate system in effect when the `endgraph` command is interpreted. Figure 6 uses the

```
autogrid(,otick.lft)
```
Figure 5: Timings for two matrix multiplication algorithms with the corresponding MetaPost input.

Figure 6: Annual lead emissions and average level at atmospheric monitoring stations in the United States. The MetaPost input is shown above the graph.
to label the left side of the graph with the $y$ coordinates in effect before the `setcoords` command. This suppresses the default axis labels, so another `autogrid` command is needed to label the bottom and right sides of the graph using the new coordinate system. The general syntax is

\[ \text{autogrid}(\langle\text{axis label command}\rangle, \langle\text{axis label command}\rangle) \langle\text{option list}\rangle \]

where

\[ \langle\text{axis label command}\rangle \rightarrow \langle\text{empty}\rangle | \langle\text{grid or tick}\rangle \langle\text{label suffix}\rangle \]

\[ \langle\text{grid or tick}\rangle \rightarrow \text{grid} | \text{itick} | \text{otick} \]

The \langle\text{label suffix}\rangle should be \text{lft}, \text{rt}, \text{top}, or \text{bot}.

The first argument to `autogrid` tells how to label the $x$ axis and the second argument does the same for $y$. An (empty) argument suppresses labeling for that axis. Otherwise, the \langle\text{label suffix}\rangle tells which side of the graph gets the numeric label. Be careful to use \text{bot} or \text{top} for the $x$ axis and \text{lft} or \text{rt} for the $y$ axis. Use \text{otick} for outward tick marks, \text{itick} for inward tick marks, and \text{grid} for grid lines. The \langle\text{option list}\rangle tells how to draw the tick marks or grid lines. Grid lines tend to be a little overpowering, so it is a good idea to give a \text{withcolor} option to make them light gray so they do not make the graph too busy.

### 2.3 Explicit Grids and Framing

In case `autogrid` is not flexible enough, axis label commands generate grid lines or tick marks one at a time. The syntax is

\[ \langle\text{grid or tick}\rangle, \langle\text{label suffix}\rangle(\langle\text{label format}\rangle, \langle\text{numeric or string expression}\rangle) \langle\text{option list}\rangle \]

where \langle\text{grid or tick}\rangle and \langle\text{label suffix}\rangle are as in `autogrid`, and \langle\text{label format}\rangle is either a format string like "$g$" or a picture containing the typeset numeric label.

The axis label commands use a macro

\[ \text{format}((\text{format string}), \langle\text{numeric or string expression}\rangle) \]

to typeset numeric labels. Full details appear in Section 4, but when the \langle\text{format string}\rangle is "$g$", it uses decimal notation unless the number is large enough or small enough to require scientific notation.

The example in Figure 7 invokes

\[ \text{format}("g", y) \]

explicitly so that grid lines can be placed at transformed coordinates. It defines the transformation

\[ \text{newy}(y) = y/75 + \ln y \]

and shows that this function increases almost linearly.\(^6\) This is a little like using logarithmic $y$-coordinates, except that $y$ is mapped to $y/75 + \ln y$ instead of just $\ln y$.

Figure 7 uses the command

\[ \text{frame}.(\langle\text{label suffix}\rangle \langle\text{option list}\rangle) \]

to draw a special frame around the graph. In this case the \langle\text{label suffix}\rangle is \text{lft} to draw just the bottom and left sides of the frame. Suffixes \text{lrt}, \text{ulft}, and \text{urt} draw other combinations of two sides; suffixes \text{lft}, \text{rt}, \text{top}, \text{bot} draw one side, and \langle\text{empty}\rangle draws the whole frame. For example

\[ \text{frame dashed evenly} \]

draws all four sides with dashed lines. The default four-sided frame is drawn only when there is no explicit `frame` command.

\(^6\)The manual \cite{5} explains how \text{vardef} defines functions and \text{mlog} computes logarithms.
vardef newy(expr y) = (256/75)*y + mlog y enddef;
draw begingraph(3in,2in);
glabel.lft(btex \vbox{\hbox{Population} \hbox{in millions}} etex, OUT);
path p;
gdata("timepop.d", $, augment.p($1, newy(Scvnum $2))); 
gdraw p withpen nullpen;
for y=5,10,20,50,100,150,200,250:
    grid.lft(format("%g",y), newy(y)) withcolor .85white;
endfor
autogrid(grid.bot,) withcolor .85white;
gdraw p;
frame.llft;
endgraph;

Figure 7: Population of the United States in millions versus time with the population re-expressed as $p/75 + \ln p$. The MetaPost input shown above the graph assumes a data file `timepop.d` that gives (year, $p/75 + \ln p$) pairs.
To label an axis as `autogrid` does but with the labels transformed somehow, use

```plaintext
auto.x` or `auto.y
```

for positioning tick marks or grid lines. These macros produce comma-separated lists for use in `for` loops. Any `x` or `y` values in these lists that cannot be represented accurately within MetaPost’s fixed-point number system are given as strings. A standard macro package that is loaded via

```plaintext
input sarith
```

defines arithmetic operators that work on numbers or strings. Binary operators `Sadd`, `Ssub`, `Smul`, and `Sdiv` do addition, subtraction, multiplication, and division.

One possible application is rescaling data. Figure 4 used a special data file `agepopm.d` that had `y` values divided by one million. This could be avoided by replacing “`gdraw "agepopm.d"`” by

```plaintext
gdraw "agepop91.d";
for u=auto.y: otick.lft(format("%g",u Sdiv "1e6"), u); endfor
autogrid(otick.bot,)
```

### 2.4 Processing Data Files

The most general tool for processing data files is the `gdata` command:

```plaintext
gdata((string expression), (variable), (commands))
```

It takes a file name, a variable `v`, and a list of commands to be executed for each line of the data file. The commands are executed with `i` set to the input line number and strings `v1`, `v2`, `v3`, ... set to the input fields on the current line. A null string marks the end of the `v` array.

Using a `glabel` command inside of `gdata` generates a scatter plot as shown in Figure 8. The data file `countries.d` begins

```plaintext
20.910 75.7 US
1.831 66.7 Alg
```

where the last field in each line gives the label to be plotted. Setting `defaultfont` in the first line of input selects a small font for these labels. Without these labels, no `gdata` command would be needed. Replacing the `gdata` command with

```plaintext
gdraw "countries.d" plot btex\circ etex
```

would change the abbreviated country names to open circles.

Both `gdraw` and `gdata` ignore an optional initial `%` on each input line, parse data fields separated by white space, and stop if they encounter an input line with no data fields. Leading percent signs make graph data look like MetaPost comments so that numeric data can be placed at the beginning of a MetaPost input file.

It is often useful to construct one or more paths when reading a data file with `gdata`. The `augment` command is designed for this:

```plaintext
augment.(path variable)((coordinates))
```

If the path variable does not have a known value, it becomes a path of length zero at the given coordinates; otherwise a line segment to the given coordinates is appended to the path. The `(coordinates)` may be a pair expression or any combination of strings and numerics as explained at the beginning of Section 2.2.

If a file `timepop.d` gives `t`, `p` pairs, `augment` can be used like this to graph `newy(p)` versus `t`:

```plaintext
path p;
gdata("timepop.d", s, augment.p(s1, newy(scantokens s2))); 

gdraw p;
```
defaultfont:="cmr7";
draw begingraph(3in,2in);
glabel.lft(btex \vbox{\hbox{Life}\hbox{expectancy}} etex, OUT);
glabel.bot(btex Per capita G.N.P. (thousands of dollars) etex, OUT);
setcoords(log,linear);
gdata("countries.d", s,
    glabel(s3, s1, s2);
)
endgraph;

Figure 8: A scatter plot and the commands that generated it

(MetaPost’s scantokens primitive interprets a string as if it were the contents of an input file. This
finds the numeric value of data field \texttt{s2}.)

Figure 9 shows how to use \texttt{augment} to read multiple column data and make multiple paths.
Paths \texttt{p2}, \texttt{p3}, \texttt{p4}, \texttt{p5} give cumulative totals for columns 2 through 5 and pictures \texttt{lab2} through \texttt{lab5}
give corresponding labels. The expression

\begin{verbatim}
image(unfill bbox lab[j]; draw lab[j])
\end{verbatim}

executes the given drawing commands and returns the resulting picture: “unfill bbox lab[j]”
puts down a white background and “\texttt{draw lab[j]}” puts the label on the background. The \texttt{gfill}
command is just like \texttt{gdraw}, except it takes a cyclic path and fills the interior with a solid color. The
color is black unless a \texttt{withcolor} clause specifies another color. See the manual \cite{5} for explanations
of \texttt{for} loops, arrays, colors, and path construction operators like --, cycle, and reverse.

3 Manipulating Big Numbers

MetaPost inherits a fixed-point number system from Knuth’s \texttt{METAfont} \cite{6}. Numbers are expressed
in multiples of \(2^{-16}\) and they must have absolute value less than 32768. Knuth chose this system
because it is perfectly adequate for font design, and it guaranteed to give identical results on all types
of computers. Fixed-point numbers are seldom a problem in MetaPost because all computations are
based on coordinates that are limited by the size the paper on which the output is to be printed. This
does not hold for the input data in a graph-drawing application. Although graphs look best when
coordinate axes are labeled with numbers of reasonable magnitude, the strict limits of fixed-point
arithmetic would be inconvenient.
draw begingraph(3in,2in);
    glabel.lft(btex \hbox{Quadrillions of BTU} etex, OUT);
    path p[];
    numeric t;
    gdata("energy.d", $,
        t:=0; augment.p1($1,0);
        for j=2 upto 5:
            t:=t+scantokens $[j]$; augment.p[j]($1,t);
        endfor
    )
    picture lab[];
    lab2=btex coal etex; lab3=btex crude oil etex;
    lab4=btex natural gas etex; lab5=btex hydroelectric etex;
    for j=5 downto 2:
        gfill p[j]--reverse p[j-1]--cycle withcolor .16j*white;
        glabel.lft(image(unfill bbox lab[j]; draw lab[j]), .7+length p[j]);
    endfor
endgraph;

Figure 9: A graph of U.S. annual energy production and the commands that generated it
A simple way to handle large numbers is to include the line

```
input sarith
```

and then use binary operators `Sadd`, `Ssub`, `Smul`, and `Sdiv` in place of `+`, `-`, `*`, and `/`. These operators are inefficient but very flexible. They accept numbers or strings and return strings in exponential notation with the exponent marked by “e”; e.g., "6.7e-11" means $6.7 \times 10^{-11}$.

The unary operator

```
Sabs ⟨string⟩
```

finds a string the represents the absolute value. Binary operators `Sleq` and `Sneq` perform numeric comparisons on strings and return boolean results.

The operation

```
Scvnum ⟨string⟩
```

finds the numeric value for a string if this can be done without overflowing MetaPost’s fixed-point number system. If the string does not contain “e”, it is much more efficient to use the primitive operation

```
scantokens ⟨string⟩
```

The above operators are based on a low-level package that manipulates numbers in “Mlog form.” A number $x$ in Mlog form represents

$$
\mu^{2^{16}x}, \quad \text{where } \mu = -e^{2^{-24}}.
$$

Any value between $1.61 \times 10^{-28}$ and $3.88 \times 10^{55}$ can be represented this way. (There is a constant `Mten` such that $k \times Mten$ represents $10^k$ for any integer $k$ in the interval $[-29, 55]$.)

The main reason for mentioning Mlog form is that it allows graph data to be manipulated as a MetaPost path. The function

```
Mreadpath(⟨file name⟩)
```

reads a data file and returns a path where all the coordinates are in Mlog form. An internal variable `Gpaths` determines whether `gdraw` and `gfill` expect paths to be given in Mlog form. For example, this graphs the data in `agepop91.d` with $y$ coordinates divided by one million:

```
interim Gpaths:=log;
gdraw Mreadpath("agepop91.d") shifted (0,-6*Mten);
```

### 4 Typesetting Numbers

The graph package needs to compute axis labels and then typeset them. The macro

```
format(⟨string expression⟩, ⟨numeric or string expression⟩)
```

does this. You must first `input graph` or `input format` to load the macro file. The macro takes a format string and a number to typeset and returns a picture containing the typeset result. Thus

```
format("%g",2+2)    yields  4
```

and

```
format("%3g","6.022e23")    yields  6.02 \times 10^{23}
```

A format string consists of

- an optional initial string not containing a percent sign,

---

*The argument to a unary operator need not be parenthesized unless it is an expression involving binary operators.*
• a percent sign,
• an optional numeric precision \( p \),
• one of the conversion letters \( e, f, g, G \),
• an optional final string \( \beta \).

The initial and final strings are typeset in the default font (usually \texttt{cmr10}), and the typeset number is placed between them. For the \( e \) and \( g \) formats, the precision \( p \) is the number of significant digits allowed after rounding; for \( f \) and \( G \), the number is rounded to the nearest multiple of \( 10^{-p} \). If the precision is not specified, the default is \( p = 3 \). The \( e \) format always uses scientific notation and the \( f \) format uses ordinary decimal notation but reverts to scientific notation if the number is at least 10000. The \( g \) and \( G \) formats also revert to scientific notation for non-zero numbers of magnitude less than 0.001.

The \texttt{format} macro needs a set of templates to determine what font to use, how to position the exponent, etc. The templates are normally initialized automatically, but it is possible to set them explicitly by passing five picture expressions to \texttt{init_numbers}. For instance, the default definition for \TeX{} users is

\begin{verbatim}
init_numbers(btex$-$etex, btex$1$etex, btex$\times10$etex,
btex$^-$etex, btex$^2$etex)
\end{verbatim}

The first argument tells how to typeset a leading minus sign; the second argument is an example of a 1-digit mantissa; third comes whatever to put after the mantissa in scientific notation; next comes a leading minus sign for the exponent and a sample 1-digit exponent.

Picture variable \texttt{Fe_plus} gives a leading plus sign for positive numbers, and \texttt{Fe_base} gives whatever should precede the exponent when typesetting a power of ten. Calling \texttt{init_numbers} initializes \texttt{Fe_plus} to an empty picture and constructs \texttt{Fe_base} from its second and third arguments.

5 Conclusion

The graph package makes it convenient to generate graphs from within the MetaPost language. The primary benefits are the power of the MetaPost language and its ability to interact with \TeX{} or troff for typesetting labels. Typeset labels can be stored in picture variables and manipulated in various ways such measuring the bounding box and providing a white background.

We have seen how to generate shaded regions and control line width, color, and styles of dashed lines. Numerous other variations are possible. The full MetaPost language \cite{metafont} provides many other potentially useful features. It also has enough computing power to be useful for generating and processing data.

A Summary of the Graph Package

In the following descriptions, italic letters such as \( w \) and \( h \) denote expression parameters and words in angle brackets denote other syntactic elements. Unless specified otherwise, expression parameters can be either numerics or strings. An \langle option list \rangle is a list of drawing options such as \texttt{withcolor .5white} or \texttt{dashed evenly}; a \langle label suffix \rangle is one of \texttt{lft}, \texttt{rt}, \texttt{top}, \texttt{bot}, \texttt{ulft}, \texttt{urt}, \texttt{llft}, \texttt{lrt}.

A.1 Graph Administration

\texttt{begingraph}(\( w, h \)) Begin a new graph with the frame width and height given by numeric parameters \( w \) and \( h \).

\texttt{endgraph} End a graph and return the resulting picture.
setcoords($t_x, t_y$) Set up a new coordinate system as specified by numeric flags $t_x$, $t_y$. Flag values are $\pm \text{linear}$ and $\pm \text{log}$.

setrange($\langle \text{coordinates} \rangle$, $\langle \text{coordinates} \rangle$) Set the lower and upper limits for the current coordinate system. Each $\langle \text{coordinates} \rangle$ can be a single pair expression or two numeric or string expressions.

A.2 Drawing and Labeling

All of the drawing and labeling commands can be followed by an $\langle \text{option list} \rangle$. In addition to the usual MetaPost drawing options, the list can contain a $\text{plot}$ (picture) clause to plot a specified picture at each data point.

The drawing and labeling commands are closely related to a set of similarly named commands in plain MetaPost. The $\text{gdrawarrow}$ and $\text{gdrawdblarrow}$ commands are included to maintain this relationship.

gdotlabel.($\langle \text{label suffix} \rangle(p, \langle \text{location} \rangle)$) This is like $\text{glabel}$ except it also puts a dot at the location being labeled.

gdraw $p$ Draw path $p$, or if $p$ is a string, read coordinate pairs from file $p$ and draw a polygonal line through them.

gdrawarrow $p$ This is like $\text{dgraw}$ $p$ except it adds an arrowhead at the end of the path.

gdrawdblarrow $p$ This is like $\text{dgraw}$ $p$ except it adds an arrowheads at each end of the path.

gfill $p$ Fill cyclic path $p$ or read coordinates from the file named by string $p$ and fill the resulting polygonal outline.

glabel.($\langle \text{label suffix} \rangle(p, \langle \text{location} \rangle)$) If $p$ is not a picture, it should be a string. Typeset it using $\text{defaultfont}$, then place it near the given location and offset as specified by the $\langle \text{label suffix} \rangle$.

The $\langle \text{location} \rangle$ can be $x$ and $y$ coordinates, a pair giving $x$ and $y$, a numeric value giving a time on the last path drawn, or $\text{OUT}$ to label the outside of the graph.

A.3 Grids, Tick Marks, and Framing

auto.($x$ or $y$) Generate default $x$ or $y$ coordinates for tick marks.

autogrid($\langle \text{axis label command} \rangle$, $\langle \text{axis label command} \rangle$) Draw default axis labels using the specified commands for the $x$ and $y$ axes. An $\langle \text{axis label command} \rangle$ may be $\langle \text{empty} \rangle$ or it may be $\text{itick}$, $\text{otick}$, or $\text{grid}$ followed by a $\langle \text{label suffix} \rangle$.

frame.($\langle \text{label suffix} \rangle \langle \text{option list} \rangle$) Draw a frame around the graph, or draw the part of the frame specified by the $\langle \text{label suffix} \rangle$.

grid.($\langle \text{label suffix} \rangle(f,z)$) Draw a grid line across the graph from the side specified by the $\langle \text{label suffix} \rangle$, and label it there using format string $f$ and coordinate value $z$. If $f$ is a picture, it gives the label.

itick.($\langle \text{label suffix} \rangle(f,z)$) This is like $\text{grid}$ except it draws an inward tick mark.

otick.($\langle \text{label suffix} \rangle(f,z)$) This is like $\text{grid}$ except it draws an outward tick mark.
A.4 Miscellaneous Commands

`augment (variable) ⟨(coordinates)⟩` Append ⟨coordinates⟩ to the path stored in ⟨variable⟩.

`format(f, x)` Typeset x according to format string f and return the resulting picture.

`gdata(f, (variable), (commands))` Read the file named by string f and execute ⟨commands⟩ for each input line using the ⟨variable⟩ as an array to store data fields.

`init_numbers(s, m, x, t, e)` Provide five pictures as templates for future format operations: s is a leading minus; m is a sample mantissa; x follows the mantissa; t is a leading minus for the exponent e.

`Mreadpath(f) ` Read a path for the data file named by string f and return it in “Mlog form”.

A.5 Arithmetic on Numeric Strings

It is necessary to input sarith before using the following macros:

`Sabs x` Compute |x| and return a numeric string.

`x Sadd y` Compute x + y and return a numeric string.

`Scvnum x` Return the numeric value for string x.

`x Sdiv y` Compute x/y and return a numeric string.

`x Sleq y` Return the boolean result of the comparison x ≤ y.

`x Smul y` Compute x * y and return a numeric string.

`x Sneq y` Return the boolean result of the comparison x ≠ y.

`x Ssub y` Compute x − y and return a numeric string.

A.6 Internal Variables and Constants

`Autoform` Format string used by autogrid. Default: "%g".

`Fe_base` What precedes the exponent when typesetting a power of ten.

`Fe_plus` Picture of the leading plus sign for positive exponents.

`Gmarks` Minimum number of tick marks per axis for auto and autogrid. Default: 4.

`Gminlog` Minimum largest/smallest ratio for logarithmic spacing with auto and autogrid. Default: 3.0.

`Gpaths` Code for coordinates used in gdraw and gfill paths: linear for standard form, log for “Mlog form”.

`Mten` The “Mlog form” for 10.0
References


