1 Introduction

`matlab.mp` is a MetaPost package for plotting out 2-D data. This is a common task for any scientists or engineers. The most frequently used tool for doing such tasks are probably Gnuplot and Matlab. But I have not been very satisfied with the plotting quality of either. Gnuplot is fast, versatile, and you can choose between several line styles/widths, but the available line styles are largely hard-coded into Gnuplot, and not very beautiful. Matlab generates better plots, but one particular problem with Matlab made me decided to create something that suits me needs: if you need to plot a thick dotted line in Matlab, the seperation distance between dots along the line stays *constant*, so the line looks like a "railroad track", instead of a dotted line.

Another choice for plotting data is the `mpgraph` MetaPost package. The "weakness" of this package is that its default style does not look like Matlab (because I am accustomed to Matlab style). Plus, the usage of `mpgraph` is not as easy as Matlab. Then I decided to integrate the rich plotting ability of `mpgraph` with the ease of use of Matlab.

2 Quick Usage

The process of plotting a data plot consists of roughly 3 steps: (1) read in the data from a data source file. (2) plotting out the data, possibly specifying desired line styles. (3) plotting "decorations", i.e. legends, axis labels, grids, etc. Since we use `mpgraph` package, steps (2) and (3) should appear within the `begingraph`/`endgraph` construct.

Here is a sample file `simple.mp` that uses of `matlab.mp`:

```
input matlab.mp
prologues:=0;
beginfig(1)
%begin a matlab data plot
begingrf(5in,4in);
% read in column 1 of data file "sample_data"
% and store it as the 0th "data vector" in our repository
rdata("sample_data", 1);
% read in column 2 of data file "sample_data"
% and store it as the 1st "data vector" in our repository
rdata("sample_data", 2);
% plot out data, using vector 0 as X values, and vector 1 as Y values
% well, we have to make sure that the two vectors have the same
% lengths
mtplot("0␣1");
ylabel("sample␣Y␣label␣$\alpha$");
xlabel("sample␣X␣label␣{\bf␣Bold}␣and␣{\it␣italic}␣");
% do the post-processing
finishgrf;
```

```
legend(3, 3, "\bf␣column␣1␣and␣2␣of␣sample\_file");
endfig;
end
```

`sample_data` is a text file containing columns of numbers. Both files can be found in the `examples` directory of the package. That directory also contains a Makefile to generate the resulting `simple.1` eps file.

3 More Details

Now we show in more detail how to draw a data plot. First we call `draw begingrf(width,height)`, just as when using `mpgrah`. In fact, `begingrf` just calls `begingraph`, and set a few default style parameters. Then we read in data from external data files by `rdata("file name", column_index)`. So each call of rdata only reads in one column of data, which is stored into an internal vector, indexed from 0 upwards.

Then we actually plot out the data by calling `mtplot("X Y .....")`. The parameters to this macro are all put into the same string, so the implementation can be easier. Parameters are seperated by space in the string. There are two required parameters: the column index of the data vector to be used as X coordinates, and that for Y. The remaining parameters are used to specify line styles. We have **4 categories of line styles**: widths, color, marker shape, and line pattern (dotted, dashed, etc). Widths are specified by "`linewidthn`", where **n** is a number giving the linewidth in `pt`. Color specification is "`cx`", where x is "`k|r|g|b|y|o|p`" for black, red, green, blue, yellow, orange, and purple, respectively. Marker shape is "`mx`", where x is `o|x|*|d|s` for circle, cross, star, diamond, and square. Line patterns are "`solid, dash, dot, dashdot, dddash`" (for dot dot dash) and "`dddot`" (for dash dash dot). If you do not specify line styles, each newly plotted line is automatically assigned a new style.

Finally we can plot out the legends, by calling `legend`. The first parameter gives X position, 1,2,3 corresponds to left, center, and right. Similarly, 1,2,3 for the second parameter corresponds to bottom, middle, and top. Axis labels are printed out by `xlabel(s), ylabel(x)`, where `s` is just a string. Since we use the `latexmp` package for strings, you can use any valid latex constructs for the legends and labels.

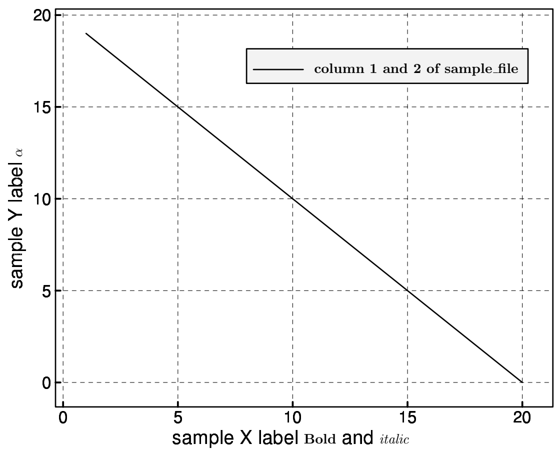

Figure 1: simple graph

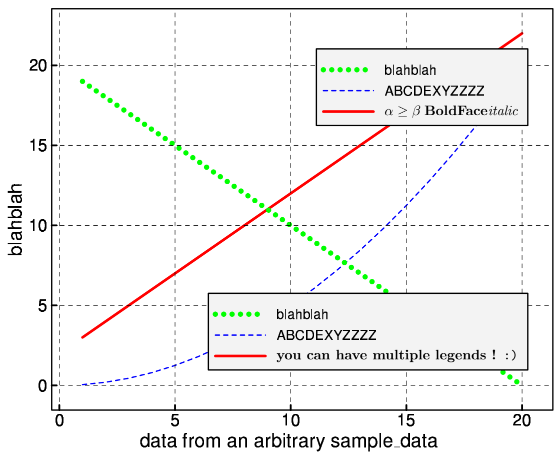

Figure 2: Full feature graph