GPLOT & XYPLOT
a library and interactive plotting program

Germán Prieto

February 2, 2010

1 Introduction

The library *libgprieto.a* contains a simple plotting library that can be used to plot using `gnuplot` and can be called inside any fortran program. It has many of the capabilities of *gnuplot* but is called in a similar way as you do in matlab. This means you don't need to save all your variables in a file, and then use gnuplot, or read them in matlab to plot. You would rather type

```
call gnuplot(x,y)
```

and you will have a ghostview window popping up with your plot of x *vs* y.

The library creates a file (`gnu.exe` that is read by *gnuplot*, and plots y as a function of x, interpolating with straight lines. Axis are automatically assigned (inside *gnuplot*) with reasonable limits and annotations. The data filename created is called *gnu_data_X.dat* and is saved in the directory. Also the Encapsulated PostScript figure is saved as *newplot.eps*. This file can be viewed with *ghostview* and edited in *Adobe Illustrator* for example.

Some of the embellishments available in *gnuplot* can also be used with this library, for example plotting many series (the *hold* command), plotting symbols, logarithmic scales, titles, axis labels, etc. Some features not yet available are for example filled in areas, error bars and other.

The great advantage of this library is that you only have to enter two parameters for calling the plotting subroutine, all other parameters are optional. Almost every option has a default value in *gnuplot* so you don't have to specify all of the parameters. You can also switch the order of the parameters as you like (as long as the two first parameters are in place). That means you can specify an x and y labels

```
call gnuplot(x,y,xlabel='time',ylabel='amp')
```

or if you don't want to label the x axis, you don't have to leave the option blank, you simply don't even mention it

```
call gnuplot(x,y,ylabel='amp')
```

The order of the parameters does not matter, similar to what you have in Matlab.

The idea of creating this subroutine is not to replace other plotting programs that may be a lot better, but rather to allow you to display your plots in different points of your program and *see* what it looks like. Or see if the variables seem reasonable, without having to either `print *, x`, or save the variable, and then plot it.

Recently I added an interactive plotting program `xyplot`. This program uses the GPLOT library to create the figures, but may be run from the command line. The program is interactive in the sense that ia runs as a command interpreter. The user enters options from a catalog of available commands. Only some commands are requiered, all others may be omitted and default values will be chosen by the program.
A simple program would be

```
> xyplot
Enter commands for plotting data (? for help)
file signal.dat
exec
quit
```

This will create a file *plot.eps* with the signal plotted with unit interval $dt = 1.0$. The user does not need to specify the length of the series. But if only a section of the data is desired, some options are available. More on the *xyplot* program will come later.

2 Requirements

The first requirement is pretty obvious, you need to have *gnuplot* installed and running. You can search for *gnuplot* at http://www.gnuplot.info/. GNUPLOT can usually be installed in any operating system.

The second and most important requirement is to have a F90 (or F95) compiler available. This subroutine uses many of the new features of Fortran 90, such as optional arguments, modules, allocatable arrays, etc. Thus, a F77 compiler won't be useful. If you are still with F77, I would suggest (and this is my personal opinion), switch to F90. You can do the same things and maybe more, but not less. And you could use this subroutine.

A final requirement is that you have to edit the Makefile a little bit to adjust to your specific compiler options. There are small differences between compilers, especially the option to look for modules, and the Unix library. The main problem here, is that this library uses a non-standard subroutine (`system`), that is not available from all F90 compilers. But usually is available as a library, called the Unix Library. In the Absoft Compiler the library is called `libU77.a`. The Sun compiler in many places has this library ready in the main compiler.

3 Compiling

I assume you have unzipped the files, and we will call the main directory $PLOT. The compilation will create a library file, a module and the command line program xyplot

```
libgplot.a
plot.mod
xyplot
```

Compiling the library is rather simple. Run the Makefile in $PLOT/

```
prompt > make
```

this will create both the object file, the module file and the executable. The only adjustment you might have to make is to edit the compiler section to adjust to your computer settings.

For easier compiling, I created a `make.inc` file where the user has to define the location of the outputs. The user needs to change to the appropriate locations

```
LIB =      /usr/local/lib/
MOD =      /usr/local/mod/
PROG =     /usr/local/bin/
```

where the three lines represent where the modules, libraries and executables are to be saved. Note that you need to have writing permission in these places for the compilation to run smoothly.

Now, the difficult part comes when compiling the program that is going to use our plotting library. This is very important for the right use of the plotting subroutine. The makefile for the latest version is

```
# Makefile to compile the plotting library

SHELL = /bin/tcsh

#-----------------------------------------------------
# make.inc included
#-----------------------------------------------------

include ./make.inc

#-----------------------------------------------------
# Compiler
#-----------------------------------------------------

# gfortran Compiler
FC = gfortran
```

```
#-----------------------------------------------------
# Compiler flags
#-----------------------------------------------------

#   Optimize
#FFLAGS = -O

#-----------------------------------------------------
# Location of files
#-----------------------------------------------------

DIR     =   ./
PLSRC   =   ./src/
PLPROG  =   ./programs/

#-----------------------------------------------------
# Module flag
#-----------------------------------------------------

# Sun Compiler
#MFLAG = -M
# Nag Compiler
#MFLAG = -i
#MFLAG = -I
# Absoft Compiler
#MFLAG = -p

# Intel or g95 compiler
MFLAG = -I

MODULE = $(MFLAG)$(MOD)

#-----------------------------------------------------
# Fortran Files
#-----------------------------------------------------

PLFILES = gplot.f90 gnuplot.f90
PLPRFILES = xyplot.f90

#-----------------------------------------------------
# Objects Files
#-----------------------------------------------------

PLOBJS =  $(PLFILES:.f90=.o)

#-----------------------------------------------------
```

```
# Compile libraries
#-----------------------------------------------

all : libgplot.a organize pl_progs

libgplot.a : $(PLOBJS)
ar cr $@ $(PLOBJS)
ranlib $@
rm *.o

%.o : $(PLSRC)%.f90
$(FC) $(FFLAGS) -c $< -o $@

organize : libgplot.a
mv *.mod $(MOD)
mv *.a $(LIB)

#-----------------------------------------------
# Compile programs
#-----------------------------------------------

pl_progs : xyplot

%:      $(PLPROG)%.f90 $(LIB)libgplot.a
$(FC) $(FFLAGS) $(MODULE) $< $(LIB)libgplot.a -o $(PROG)$@
mv *.mod $(MOD)
```

There are two things to note here. First, the flag for looking at modules may be different for different compilers. I have some compilers referenced here if need be. The Absoft compiler uses `-p`, meaning path, to look for the modules. The Sun compiler uses `-M` meaning modules and other possibilities such as `-I` or `-i` for the NAG compiler. You have to find out what your compiler options are.

Second, some compilers need to use the Unix Library (`libU77`), while the others have it already inside the compiler. So, if your compiler has the Unix Library already, this part is not needed. Note however that this library is not F90 standard.

Finally you will need to `use` the module in your program. The way you use a module in a program is simply to type `use mod_name` before defining any variables. So, as an example, we have

```
program test_plot

!
! This is a test program to get the plotting routine to work
! and plot inside any fortran code.
!
```

```
!************************************************

   use plot

   implicit none

   real(8), dimension(5) :: x, y

!************************************************

...

end program test_plot
```

4 Basics

Here I will explain the way the subroutine is called. The argument list that is needed and the way to input the different arguments. A list of the arguments supported i available in the next section.

```
gnuplot( x, y, hold, logxy, title, xlabel, ylabel, affine,
    char, dash, frame,
    weight, output,
    symbol, xlimit, ylimit, color)
```

This is the whole list of supported arguments. You could also type

```
call gnuplot( x, y )
```

and it would work as well. This means that all other arguments (except x, y) are optional arguments, and the *plotxy* default values are used instead.

Be aware that all arguments starting from `hold` are character arguments. So a good way to call the subroutine could be

```
call gnuplot( x, y, 'hold', 'loglog' )
```

which means you will plot x,y in the loglog domain and hold the figure for plotting another thing. If you call the subroutine this way, the order of the arguments has to be exact. That means you have to have a hold before defining the type of plot you want. This would be somehow annoying, if you want to have limits in your y axis, you would need to define all the rest of the arguments.

Using the capabilities of Fortran 90, you don't have to define all the variables anymore, nor you have to maintain the order (only the first two which have to be always defined). The only trick is that you have to define what each argument is. Thus, the previous example can also be called

```
call gnuplot( x, y, hold = 'hold', logxy='loglog' )
call gnuplot( x, y, logxy='loglog', hold='hold' )
```

Both this two calls would have the same result. If you would want to add a x axis label

```
call gnuplot( x, y, xlabel='Time (s)' )
```

which would output a linear plot (no hold) with the x label accordingly. All the argument names in this list are the arguments used by *plotxy*, and for the most part the input for each argument has to follow *plotxy* rules. That means, if you would type

```
call gnuplot( x, y, logxy='hold' )
```

a problem would be generated. This would depend on *plotxy* response rather than the subroutines response. Since *plotxy* is pretty stable, has been around for a while and many bugs have been corrected during the years, the result in this case is simply a linear plot of x and y. The program won't crash, it would simply skip that line and use the default linear plot.

5 Commands

The commands here are taken from the original *plotxy* documentation. The whole name is needed in order for the program to understand the command name. For a much better explanation of the commands and command fields go to the *plotxy* documentation. Here I just try a brief and simply explanation of the available commands for *gnuplot*.

hold 'hold' ' '

Does not display the figure yet, waiting for a next plot to be added to the particular figure. Works similar to the Matlab hold command. Be careful that if you put a hold, there is no output until a call to *gnuplot* without a hold is done.

```
call gnuplot( x, y, 'hold' )
call gnuplot( x, y, ' ' )
call gnuplot( x, y, ....,hold='hold' )
```

All posibilities puts a hold on the current figure. One can have as many holds as *plotxy* can sustain, but a final command without a hold is needed, to obtain an output.

logxy 'linlin' 'loglin' 'linlog' 'loglog' 'equilin'

logxy '1' '2' '3' '4'

Specifies the type of scales for the plot. The numeric values can also be used, respectively. *equilin* has a similar scaling for the x and y axis, thus circles remain circles.

Default *linlin*

Blank *loglog*

```
call gnuplot( x, y, ..., logxy='loglog' )
call gnuplot( x, y, ..., logxy='3', ... )
```

title text

A title for the plot. It may be up to 115 characters.

```
call gnuplot( x, y, ..., title='A plot', ... )
```

xlabel text

Specifies a label to be written centered below the x axis.

```
call gnuplot( x, y, ..., xlabel='X var', ... )
```

ylabel text

Specifies a label to be written centered below the y axis.

```
call gnuplot( x, y, ..., ylabel='Y var', ... )
```

affine a b c d

Transforms the x and y coordinates of the subsequent data series to be read according to $new(x) = a * x + b$, $new(y) = c * y + d$. This is an affine transformation.

Default a = 1, b = 0, c= 1, d = 0

char h [angle]

Change the height of the lettering everywhere to h inches. It applies to the text after the command, so that different texts can letter heights can be achieved. An optional parameter angle can also be used. See *plotxy* documentation for better understanding.

Default h = 0.11, angle = 0

```
call gnuplot( x, y, ..., char='0.15', ... )
call gnuplot( x, y, ..., char='0.15 45', ... )
```

dash [s1 s2] or '–', '..'

Plot data series as dashed line, with visible segment s1 inches long, and missing segment s2 in inches. I created two defaults, of dashed lines '–' and almost dots '..'.

Default s1 = 0, s2 = 0 continuos line

Blank s1 = 0.2, s2 = 0.1

```
call gnuplot( x, y, ..., dash='0.3 0.1', ... )
call gnuplot( x, y, ..., dash='--', ... )
call gnuplot( x, y, ..., dash='..', ... )
```

frame '+box' '-box' and others

This command lets you decide whether to put a box on your plot (complete the box) or not. Other posibilities are available, see *plotxy* documentation.

Default -box

Blank +box

```
call gnuplot( x, y, ..., frame='+box', ... )
call gnuplot( x, y, ..., frame=' ', ... )
```

in both cases a box is applied to the figure.

weight w [wlines]

This command controls the weight of plotted lines by giving the integer w, the line thickness in one-thousandths of an inch. Optionally, lines and points on the graph can be given a different (usually heavier) weight from text and other material; this is set in wlines. Invoking weight with only one parameter implicitly sets wlines=w for subsequent input data.

Default w = 6, wlines = w

```
call gnuplot( x, y, ..., weight='8', ... )
call gnuplot( x, y, ..., weight='6 10', ... )
```

output 'filename'

Defines the name of the plot to be filename, with a maximum of 64 characters. Be careful, is field is blank, *plotxy* will restart plot in current file.

Default = newplot.eps

Blank restart plot in current file

```
call gnuplot( x, y, ..., output='myfig.eps', ... )
```

symbol 'n [s]' or 'name [s]'

Defines the input series to be a set of discrete points instead of a curve. It can be defined by a number n or the appropriate *name*. The height s in inches might be given optionally. For numbers and kind of symbols available see the *plotxy* documentation.

Default n = -1, s = 0.15

```
call gnuplot( x, y, ..., symbol='3', ... )
call gnuplot( x, y, ..., symbol='diamond', ... )
```

where both give the same output with diamonds.

xlimit xlength [x1 x2 [dx]]

Defines the length of the x axis, xlength, in inches and the lower and upper limits of x: x1, x2. This two are optional. A third optional parameter

dx defines tick marks and numbers written at integer multiples of dx, provided the result is reasonable, if not, the default is used. See *plotxy* documentation for more.

Default xlength = 5, x1 = x2 = 0

```
call gnuplot( x, y, ..., xlimit='7 0 5', ... )
call gnuplot( x, y, ..., xlimit='7 0 5 0.5', ... )
```

ylimit h [angle]

Same as **xlimit**, but for the y axis.

Default ylength = 6, y1 = y2 = 0

```
call gnuplot( x, y, ..., ylimit='7 0 5', ... )
call gnuplot( x, y, ..., ylimit='7 0 5 0.5', ... )
```

color 'color', or 'n'

On devices color capable, sets a new color given by the color name *color* or the color number *n*. See *plotxy* documentation for available colors and numbers. In general also the axes and frames are drawn with the desired color, which to me seems not so nice.

Default black

```
call gnuplot( x, y, ..., color='blue', ... )
call gnuplot( x, y, ..., color='3', ... )
```

where both lead to blue colored lines.

6 xyplot program

This is an interactive program that allows plotting of data in *ascii* format directly from the command line. There is only one mandatory command, all the rest are optional and if omitted a default value will be used. The mandatory command is the file to open. So, the simplest program would be

```
> xyplot
Enter commands for plotting data (? for help)
file signal.dat
exec
quit
```

If you need information on the available commands, simply type ? and a list of all commands will be printed. Most of the commands were explained in the previous section. The nice feature about this kind of program is that you can for help anytime. The list of commands is

```
 ?:                          Remind me of the command list again
 execute:             With given parameters start xyplot program
 file filename:        Enter filename of data series (mandatory)
 interval dt:                          Sampling interval of data
 nterms n:                    Number of data points to be read
 skip k:                   Skip over k lines before reading data
 column k:                    Read data from column k in a table
 review:                           Display current command stack
 clear command:     Delete most recent occurrence of the command
 logxy text:   Desired scale of plot linlin,loglin,linlog,loglog
 title title:                                  Title of the graph
 xlabel/ylabel text:                             xlabel or ylabel
 color number/name:                        color name of number
 affine affine:                    affine transformation a b c d
 achar size [angle]:         height of the characters, and angle
 dash dash:               s1 s2, line and white space, or --, ..
 frame +box -box:      put a box, other plotxy commands accepted
 weight w [wlines]:                    weight of lines in plot
 output file:           filename to output figure (def plot.eps)
 symbol n [h]:               symbol # or name, optional height
 xlimit/ylimit x1 x2 [dx]:      limits. No xlength supported.
 fill:
```

Typing *review* lets you see what options you have in the input. Most of the options are self-explanatory.

To execute the program with the latest options, type *exec* (execute). The default file name for the plot, is *plot.eps*, but this can be changed with the appropriate option.

7 Example for calling the library

Here is a basic example of one single program, with some of the most important commands used. The output figures are shown in the bottom of this file.

```
program test_plot

! This is a test program to get the plotting routine to work

!***************************************

   use plot

   implicit none

   real(8), dimension(5) :: x, y
```

```
!****************************************

   x = (/1, 20, 3, 4, 5/)

   y = (/1, 3, 2, 4, 5/)

!  plot

   call gnuplot(x,y,'',title='Time',xlabel='time')

   x = (/1, 2, 3, 4, 5/)
   y = (/1, 3, 5, 3, 1/)

   call gnuplot(x,y,'hold',dash='--')

   x = (/1, 8, 12, 16, 20/)
   y = (/1, 2, 3, 4, 5/)

   call gnuplot(x,y,dash='..',xlimit='0 20',    &
                   output='gap.eps')


   call gnuplot(x,y,logxy='loglin',title='',   &
          ylabel='Amp^2',output='gap2.eps')

   call gnuplot(x,y,logxy='loglog',output='gap3.eps')

end program test_plot
```

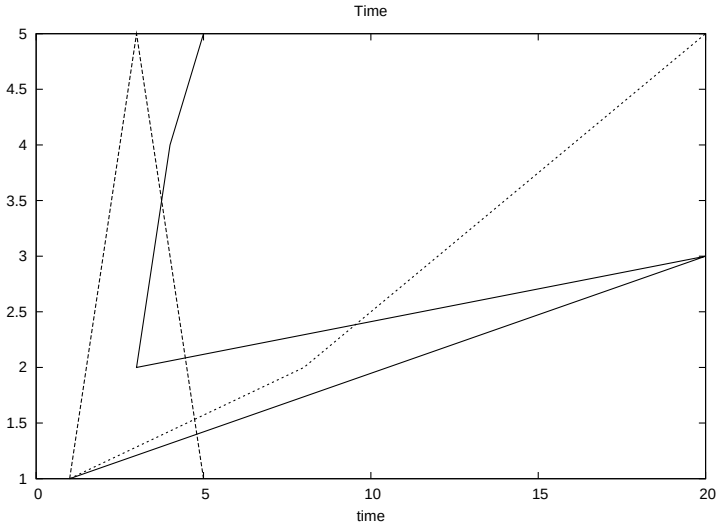

Figure 1: gap.eps

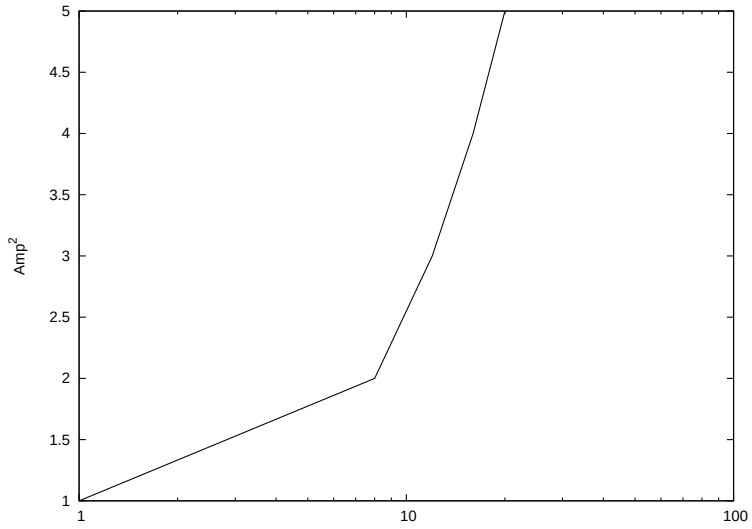

Figure 2: gap2.eps

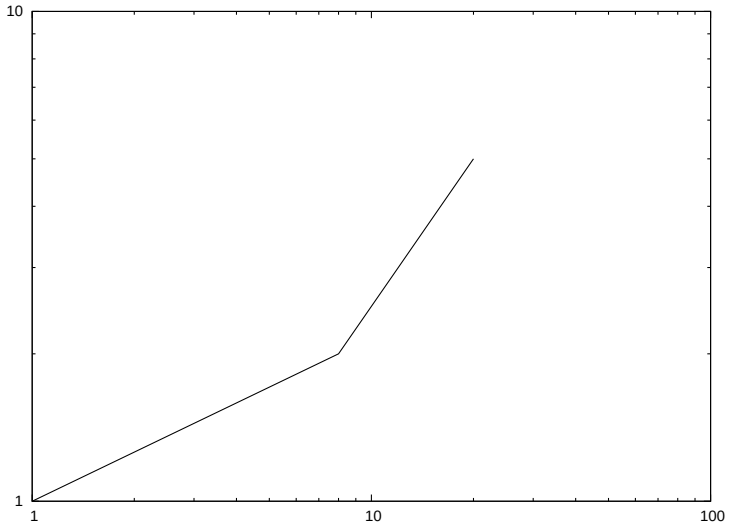

Figure 3: gap3.eps