Appendix C

Gnuplot

Gnuplot is a relatively simple tool to plot data and functions. It uses a simple command language rather than a graphical user interface, which has the big advantage that one can write *Gnuplot* script files that do very complex things and store them for later use.

C.1 Basics

C.1.1 Simple examples

(Surprise: we won't plot "Hello World" ;-)

Start gnuplot from a shell (i.e from your *xterm*, *gterm*, *eterm*, *konsole* window, or whatever it may be called):

```
user@asgard:~$  gnuplot
```

Now try the following:

```
Plotting a function:
gnuplot>  plot sin(x)
Changing axis range:
gnuplot>  set xrange [-6:6]
gnuplot>  replot

Plotting several functions:
gnuplot>  plot [x=*:*] [-2:2] sin(x) title 'Sine', tan(x) title 'Tan'

gnuplot>  set yrange [-3:3]
gnuplot>  replot
```

Plotting data from file:

```
gnuplot>  plot "height.dat" using 1:2 title 'stone', \
              "height.dat" using 1:3 title 'bird'
```

2-d plotting:

```
gnuplot>  splot sin(x)*cos(y)
gnuplot>  set isosample 21
gnuplot>  replot
```

Getting help:

```
gnuplot>  help plot
gnuplot>  help plot using
```

Quit gnuplot:

```
gnuplot>  exit
```

In the following, we will mostly omit the **gnuplot>** prompt.

C.1.2 Special characters

Character	*Meaning*
`#`	comment sign (for scripts/commands, as well as data files)
`\`	at end of line: next line is continuation line
`;`	separates commands within one line (as in *F90*)
`[`x`:`y`]`	range [x, y]
`$i`	i-th column of data file
`'text'`	text string
`"text"`	identical to `'text'`

C.1.3 One-dimensional plotting

The '`plot`' command is used for one-dimensional plots.

Basic syntax

The syntax of the '`plot`' command is approximately as follows:

```
plot  [xrange [yrange]]  \
      {function | "filename"}  \
      [using xcol: ycol]  \
```

```
[title "title"]  \
[with style]  \
[, {function | "filename"} [using xcol: ycol] [title "title"] ...]
```

Here square brackets (as in "[*xrange* [*yrange*]]") indicate arguments that are optional, while the curly braces and the vertical bar in "{*function* | *'filename'*}" indicate that either "*function*" or "*'filename'*" should be chosen.

Examples:

```
plot besj0(x)                          # plot Bessel function

plot [0:30] besj0(x)                   # specify x range
plot [x=0:30] besj0(x)                 # same thing
plot [0:30] [-1:1] besj0(x)            # set ordinate range as well

plot besj0(x) title 'Bessel'           # explicitly set title

plot besj0(x), besj1(x)                # plot several functions in one graph

plot "height.dat" using 1:3            # plot data from file 'height.dat',
                                       # column 3 over column 1
plot "height.dat" using 1:3 \
                  with linespoints     # use connected symbols

plot "height.dat" using 1:2,  \        # combine plots of different columns
     "height.dat" using 1:3

plot "height.dat" using 1:2, \         # combine file data and function
     1.3+4.4*(t-1.2)**2 with lines

plot real(exp(0,1*x)) title 'Re exp(i x)', \
     imag(exp(0,1*x)) title 'Im exp(i x)' # complex numbers
```

The *function* can be any Fortran or C expression like "`sin(exp(x**2)+3/cos(1+x))`"; *Gnuplot* knows some more mathematical functions than these languages, see [GPMan]. Note that the name of the independent value *must* be x (or t for parametric plots)for one-dimensional plots. For two-dimensional plots, the independent variables must be x and y (or u and v for parametric plots).

In the simplest case, the column selectors *xcol, ycol* are just the numbers of individual columns (as in the examples above). More generally, they can indicate functions of the column data like in

```
set xlabel 'sinh(t)'                    # see below
plot "height.dat" using (sinh($1)):($2), \
     "height.dat" using (sinh($1)):(sqrt($3)-0.5)
set xlabel                              # clear xlabel
```

Note the use of `$1` for indicating a column; the column selector `1:3` above can be seen as a shorthand for `($1):($3)`.

Options

Many options can be used to change the appearance of the graph, e.g.

```
set title 'Bessel functions'
set xlabel 'r'; set ylabel 'J0(r)'  # set axis labels for future plots
plot besj0(x), besj1(x)             # plot Bessel function
set xlabel; set ylabel; set title   # clear labels again

set logscale y                      # future plots are semilogarithmic
plot [-5:5] cosh(x)
set nologscale                      # revert to linear scaling
```

See also §C.1.5 below.

Plotting styles

The most important styles for '`with` style' are

lines: Plot continuous line.

points: Plot individual points using symbols like '+', 'o', etc.

linespoints: Plot points connected by line.

impulses: Plot "impulse" lines from x axis to each data point.

dots: Plot using tiny dots (useful for scatter plots of large data sets).

histeps: Plot histogram-like steps centred at the x coordinates of the data points

errorbars: Plot points with error bars

xerrorbars: Only horizontal error bars

yerrorbars: Only vertical error bars

C.1.4 Combining plots

Combining several functions, etc. in one plot

```
plot [-1:10] [-4:4] sin(x),  \
                    x title 'linear',  \
                    x-x**3/3! title 'cubic',  \
                    x-x**3/3!+x**5/5! title 'quintic'
```

Each of the plots can have its own '`title`' and '`with`' settings:

```
plot [-1:10] [-4:4]  \
    sin(x)             with linespoints  title 'sine function',  \
    x title            with points       title 'linear',  \
    x-x**3/3!          with errorbars    title 'cubic',  \
    x-x**3/3!+x**5/5!  with lines        title 'quintic'
```

The result is shown in Fig. C.1.

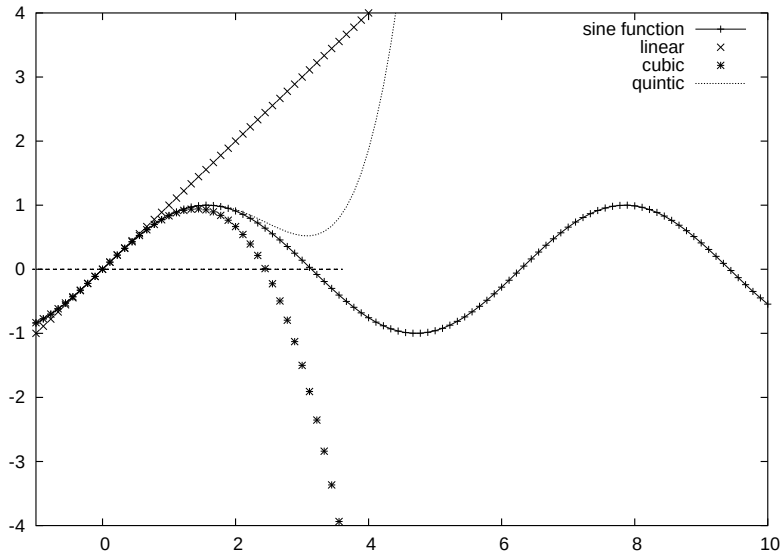

Figure C.1: Four curves in one plot with individual line styles and labels.

Graphs with subplots

The '`set multiplot`' command allows you to combine several subplots in one single plot:

```
gnuplot>    set multiplot
multiplot>  set size 1.0, 0.5  # each graph has full width, half height
```

```
multiplot>  set origin 0.0, 0.5; plot besj0(x)   # top graph
multiplot>  set origin 0.0, 0.0; plot besj1(x)   # bottom graph
multiplot>  set nomultiplot                      # reset to single plot
```

Note how the prompt changes to indicate that you are in subplot mode. The resulting graph is shown in Fig. C.2

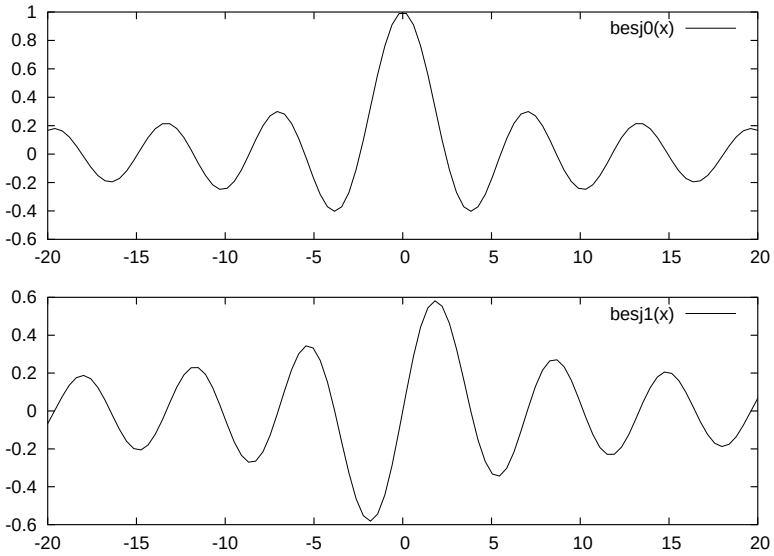

Figure C.2: Combining subplots using '`multiplot`'.

C.1.5 Setting options

Many aspects of graphs can be modified by setting *options*. An option will keep its value until it is set again (or until the are reset collectively). Thus, if you set "`xrange`", "`yrange`" and "`zrange`", these settings will stick, while specifying the ranges in the command line ('`plot [x=1:5] [-2:2] ...`') will act only for that plot.

Many options have a '`no-`' form, like e.g. "`key`", which can be used like '`set nokey`'.

Many options can be reset to their default value using '`set` option' without a value. To reset all plotting-related options collectively, use the '`reset`' command.

For a complete list of options, see '`help set`'; for help on one option, use '`help set` *option*' or '`help` *option*'. To see the current value of an option, use '`show` *option*'.

Annotation

xlabel, ylabel, zlabel: Set labels for the axis

title: Set title of whole plot (appears above top of box)

key: Set, position, or disable labels ('keys') for individual plots

Example (Fig. C.3):

```
set xlabel 't [Myr]'
set ylabel 'R [Mpc]'
set title 'Weird cosmology'
set nokey    # don't need this, since we have titles and labels
plot [x=0:30] x**0.5 + 0.015*x**1.7
set xlabel; set ylabel; set title    # clear labels
```

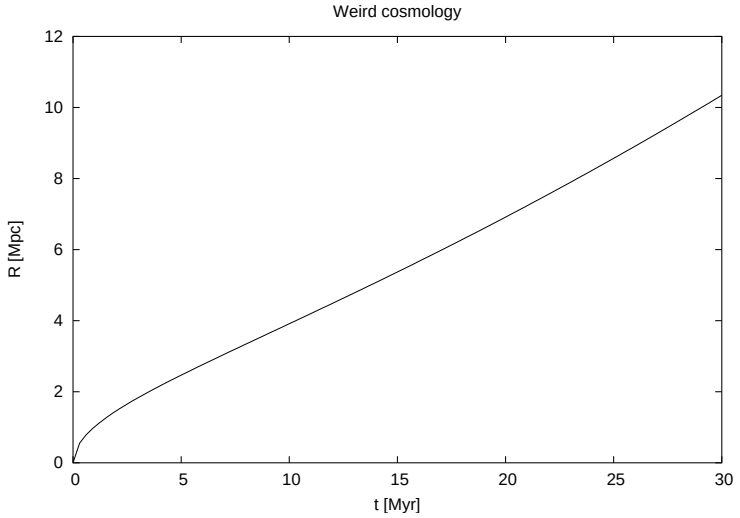

Figure C.3: Setting labels.

Axis scaling

size: Set various aspects of the graph size. Particularly useful are '`set size ratio 1`' to set the aspect ratio of the graph to 1 (so its box will be a square), and '`set size ratio -1`' which makes the axis scaling isotropic (so circles will really be circles, etc.).

origin: Set position of plot

autoscale: Automatically set axes range to accommodate all data points.

logscale: (Semi-)logarithmic plotting. '`set logscale y`' makes the y axis logarithmic, '`set nologscale`' switches back to linear scaling.

Example (Fig. C.4):

```
set xlabel 't [Myr]'; set ylabel 'R [Mpc]'
set title 'Weird cosmology'; set nokey    # don't need this
```

```
set logscale x
set logscale y
plot [x=0.1:30] x**0.5 + 0.015*x**1.7
set xlabel; set ylabel; set title          # clear labels
set nologscale                             # back to linear
```

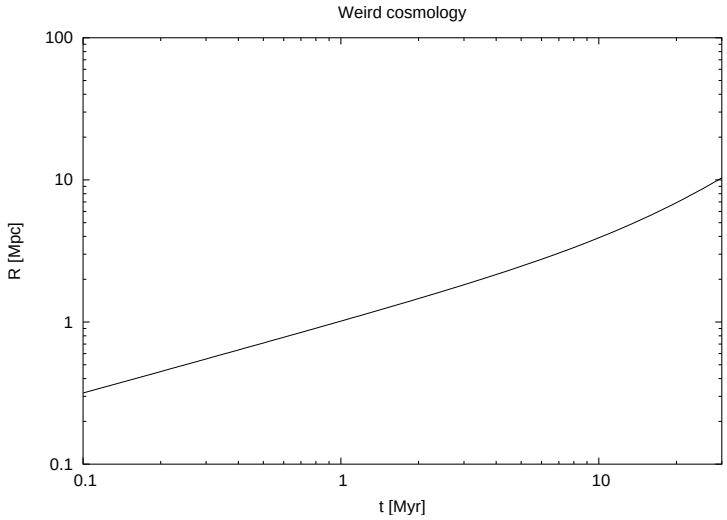

Figure C.4: (Double) logarithmic plot.

Plot type

parametric: Do parametric plot

polar: Do polar plot

Example (Fig. C.5):

```
set size ratio -1                  # isotropic scaling
set parametric                     # independent variable is now $t$
plot [t=0:2*pi] sin(3*t), cos(5*t)
set size noratio                   # reset
set noparametric                   # don't forget to reset
```

Function sampling

samples: Set sampling rate (number of points) for function plotting. The default value is 100.

isosamples: Set sampling rate (number of lines) for function surface plotting. The default value is 10 for both directions.

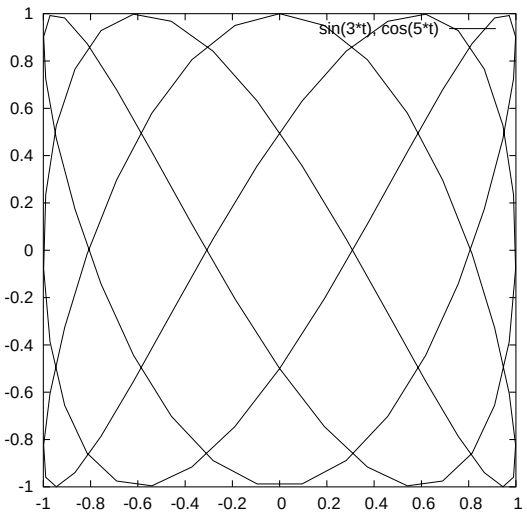

Figure C.5: Parametric plot.

Other options

border: Customize (or switch off) graph borders

contour: See §C.1.7 below.

data style: Set default way of data plotting.

function style: Set default way of function plotting.

grid: Plot a grid over the graph.

surface: See §C.1.7 below.

terminal: Select output device; see §C.1.6 below.

view: Set viewing direction for surface plots.

C.1.6 Selecting the output device

```
set term                    # list available output devices

set term dumb               # switch to ASCII art for emailing
plot besselj0(x)
set term x11                # switch back to separate graphics window
```

While *ASCII* plots (shown in Fig. C.6) naturally has low resolution, they are sometimes quite useful for getting a quick overview, e.g. when running gnuplot on a remote computer over a slow connection or one that does not permit remote graphics.

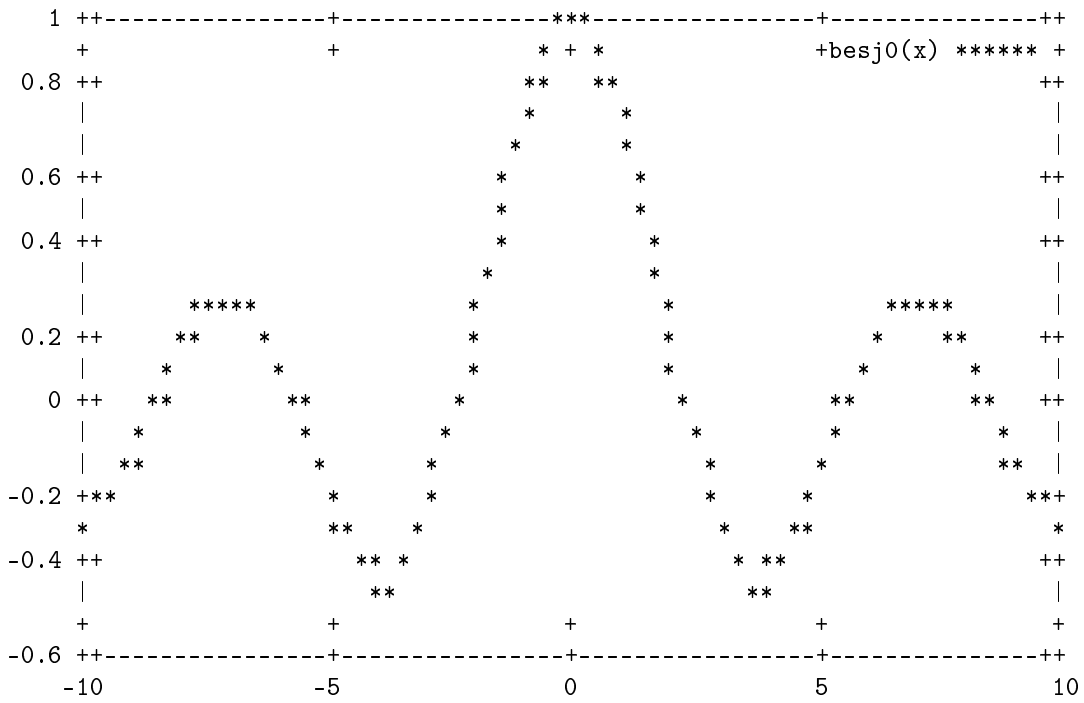

Figure C.6: ASCII version of the Bessel function $J_0(x)$.

```
help term postscript
set term postscript color       # switch to color postscript output
set output "gnuplot.ps"         # write output to file 'gnuplot.ps'
plot cos(x)
set term x11                    # switch back to separate graphics window
set output                      # close file 'gnuplot.ps'

help term postscript            # get help on options for postscript
help term postscript enhanced   # advanced options for postscript

set term postscript enhanced; set output "gnuplot.ps" # switch to postscript
set xlabel '{/TimesItalic x} [10^6 {/Symbol m}m]'
plot [x=-5:5] sinh(x)  \
              title '{/Symbol G}^{/Times-Italic ik}_{/Times-Italic l}'
set term x11; set output        # switch device back
```

The last example produces a PostScript file like Fig. C.7. As you can see, you can use super- and subscripts and even Greek letters. This is however cumbersome (it will be cumbersome with *any* plotting software), and if you need to produce fancy axis labels and titles, *Gnuplot* is probably not the best tool.

If you have interactively created a nice plot on your screen and want to print it, use

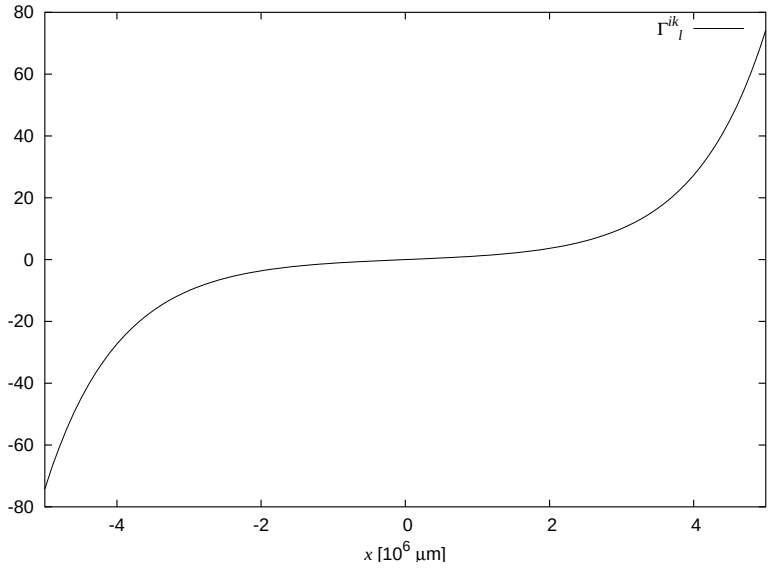

Figure C.7: Advanced text formatting in Gnuplot PostScript plots

```
set term postscript          # black/white postscript for printing
set output "gnuplot.ps"
replot
set term x11; set output     # switch back to separate graphics window
```

Note: To view the PostScript file, use '`gv`' or '`ghostview`'. To print it, you can type "`p`" or "`P`" from *gv*, or type '`lpr` *filename.ps*' from the shell.

C.1.7 Two-dimensional plotting

The '`splot`' commands plots two-dimensional functions and data.

Use '`splot`' for plotting a surface

```
splot sin(x)*cos(y)

set isosample 21             # use more lines
set hidden3d                 # show hidden lines
set xlabel 'x'               # set axes labels
set ylabel 'y'
set zlabel 'sin(x)*cos(y)'
set nokey                    # don't write extra label
```

```
splot [x=-3:3] [y=-3:3] sin(x)*cos(y)
```

The result is shown in Fig. C.8.

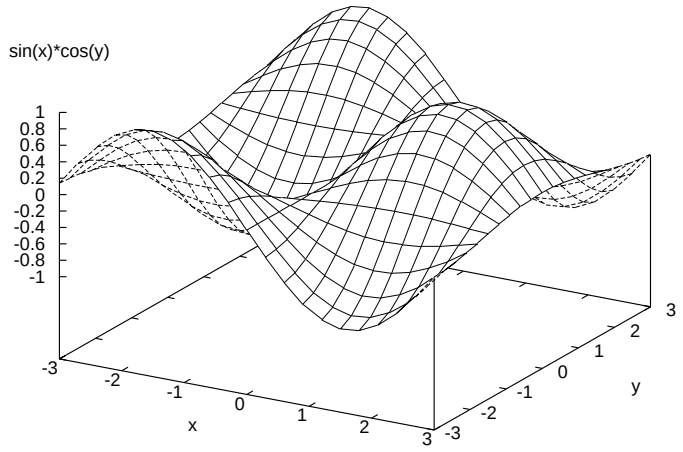

Figure C.8: Plotting surface using '`splot`'.

```
set contour
set nosurface
set size ratio -2
set view , 0, 1, 1              # set viewing direction (phi, theta)
splot [x=-3:3] [y=-3:3] sin(x)*cos(y)
```

C.1.8 Functions

You can set parameters and define functions using a straight-forward syntax.

```
n = 5

binom(n,k) = n!/(k!*(n-k)!)

sinc(x) = (x!=0) ? sin(x)/x : 1
```

C.1.9 Writing scripts

You will often want to be able to re-create a graphic with slightly different features or data. This is easy if you write gnuplot scripts, rather than use gnuplot from the command line.

In fact, you can use the command line and, after you have the plot you want, save the commands that lead to it with '`save` *filename.gp*'; this will only save the last plot command. You should then edit the file '`filename.gp`', remove settings that are irrelevant and comment those that are.

Bessel.gp

```
#                                   -*-gnuplot-*- (set mode for Emacs)
#  Bessel.gp
#
# Date: 24-Jan-2005
# Description:
#  A simple gnuplot script that plots some quantities related to
#  Bessel functions

# Calculate modulus and argument for Bessel functions. For
# trigonometric functions, this would just give mod=1 and arg=phi
modJ(x) = sqrt(besj0(x)**2 + besj1(x)**2)
argJ(x) = atan2(besj1(x), besj0(x))
dx = 0.01
dJ0(x)  = (besj0(x+dx)-besj0(x-dx)) / (2*dx)

set xrange [0:50]
set samples 200           # need a few more points for this range
set xlabel 'x'

clear
set multiplot             # four subplots in this graph
set size 0.5, 0.5

set origin 0, 0.5         # top left
set title 'Bessel functions J0, J1'
plot besj0(x), besj1(x)

set origin 0, 0           # bottom left
set title 'Modulus'
plot modJ(x)

set origin 0.5, 0.5       # top right
set title 'Arg'
plot argJ(x)
```

```
set origin 0.5, 0         # bottom right
set title 'Derivative'
plot -dJ0(x) title '-dJ0(x)/dx',         besj1(x) with points

set nomultiplot

# pause 5 "Waiting 5 seconds before quitting"
# quit
```

From gnuplot, you use this with

```
gnuplot>  load "Bessel.gp"
```

Alternatively, you can do

```
user@asgard:~$  gnuplot Bessel.gp
```

directly from the shell; in this case, you will probably want to put

```
pause -1 "Press Return to quit"
```

so you have time to look at the plot

The result from '`Bessel.gp`' is shown in Fig. C.9.

C.2 Appendix

C.2.1 Lab exercises

Include your Makefile, Gnuplot script, your Fortran code and all of the figures properly annotated; comment your files.

Question 42 *Simple Harmonic Oscillator*

The simplest example of an oscillating system is a mass connected to a rigid foundation by way of a spring. The spring constant k provides the elastic restoring force, and the inertia of the mass m provides the overshoot.

(a) By applying Newton's second law $F = ma$ to the mass,

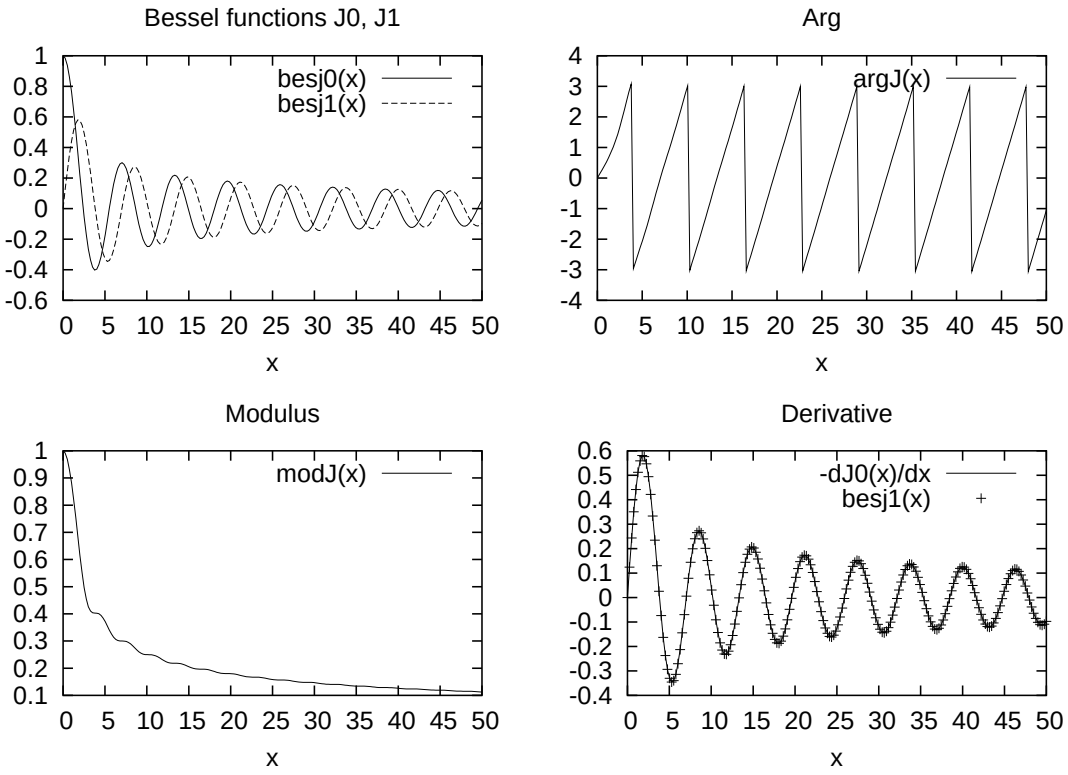

Figure C.9: Output from '`Bessel.gp`'.

(i) Derive the equation of motion for the system. The amplitude of the oscillation is $x_m = 2$ while the phase constant of the oscillation is $\phi = 0$, as determined by the initial condition (initial displacement and velocity) at time $t = 0$ when one begins observing the oscillatory motion.

(ii) Show that the natural oscillating frequency is

$$\omega_0 = \sqrt{\frac{k}{m}} \, . \tag{3.1}$$

(b) Write a Fortran 90 code that outputs into the same file the displacement (x) and time (t) for 3 different masses $m = 1, m = 1/4, m = 1/9$ (keep k constant equal to 1 and take $x_m = 2$ and $\phi = 0$ for all 3 cases). *Make sure you cover at least 2 periods*[1] *and use at least* 100 *time steps per period*.

Using gnuplot, read the data from you output file and plot the time evolution of the displacement for the 3 masses (all in the same graph). On the plot indicate the period for each of the three oscillators?

(c) Write out into a different file, the potential energy stored in the spring, the kinetic energy of the mass, and the total energy of the system for the 3 cases and for the same time steps as those used in the first data file.

[1]The period of the oscillatory motion is defined as the time required for the system to start one position, complete a cycle of motion and return to the starting position.

Using multiplot command in gnuplot (in other words one panel for each graph this time – *no separate figures*), plot the potential energy vs time, the kinetic energy vs time, and finally the total energy vs time. Discuss your results.

Question 43 *Damped oscillator*

A more realistic physical model is one that includes dissipative forces from air resistance to the motion of the mass. For simplicity it is often assumed this dissipative force is directly proportional to the velocity of the mass and in the opposite direction.

This is a good approximation of the behaviour of air resistance and produces another differential equation with an exact solution. In fact, it is the only type of dissipative force for which the differential equation of motion has the following exact solution

$$x = x_m e^{-\gamma t} \cos(\omega_1 t + \phi) \tag{3.2}$$

where the damped frequency is $\omega_1 = \sqrt{\omega_0^2 - \gamma^2}$ and depends on the damping factor $\gamma = b/2m$ (b being the drag coefficient related to air resistance on the mass-spring system.

Here we will only deal with the $m = 1$ mass (recall that $k = 1$, $x_m = 2$ and $\phi = 0$).

(a) Using the same Fortran program as in Question 42, write out to a file the displacement x at similar time steps as in the non-damped case. Plot x versus t for three values of the drag coefficient, $b = 1, 2, 3$. Comments on your plots/results and in particular on the $b = 3$ case.

(b) For the $m = 1$ case only, compare the time evolution of the displacement x for the two cases of non-damped versus the $b = 2$ damped case.

Question 44 *A Wave Packet*

The function $f(x,t) = \exp(-(x - 3\ t)^2) \times \sin(3\ \pi\ (x - t))$ describes for a fixed value of t a wave localized in space.

a) Make a program (a gnuplot script) that visualizes this function as a function of x on the interval [-4,4] when t=0.

b) Display an animation of the function $f(x,t)$ by plotting f as a function of x on [-6,6] for a set of t values in [-1,1].

Also make an animated GIF file.

A suitable resolution can be 1000 intervals (1001 points) along the x axis, 60 intervals (61 points) in time, and 6 frames per second in the animated GIF file.

(You will see that f(x,t) models waves that are moving to the right (when x is a space coordinate and t is time). The velocity of the individual waves and the packet is different, demonstrating a case in physics when the phase velocity of waves is different from the group velocity.)

Making Animations

How to make fast and efficient movies with Linux (web-display purposes)

You can convert the entire series of frames (jpeg, tiff etc ..)
into an animated GIF file using the ImageMagick convert command:

> convert -delay 15 -loop 0 frame *.jpeg movie.gif

The *-delay* option sets a minimum display time per frame: 0.01 seconds.
The *-loop 0* option sets a flag requesting a display program to loop
over the frames repeatedly.

Depending on which method of animation you chose, the command to play
back your new movie is either

> animate movie.gif

or (if you generated an mpeg instead of a gif movie)

> mpeg_play -quiet -dither color movie.mpeg

N.B.: I often USE: convert -delay 30 frame *.jpeg movie.gif

ImageMagick

You will be generating mostly ‘‘PS" and ‘‘EPS" figures with Gnuplot.
You can then convert them to other formats using **ImageMagick** which
is installed in your machines.

If you have a postscript ’PS’ or encapsulated postscript ’EPS’ you can
always convert to jpeg using the following 2 steps:

> pstopnm myfile.ps myfile.pnm

> pnmtojpeg myfile.pnm myfile.jpeg