A Quick Guide to Gnuplot

Andrea Mignone
Physics Department, University of Torino
AA 2019-2020

What is Gnuplot ?

- Gnuplot is a free, command-driven, interactive, function and data plotting program, providing a relatively simple environment to make simple 2D plots (e.g. f(x) or f(x,y));

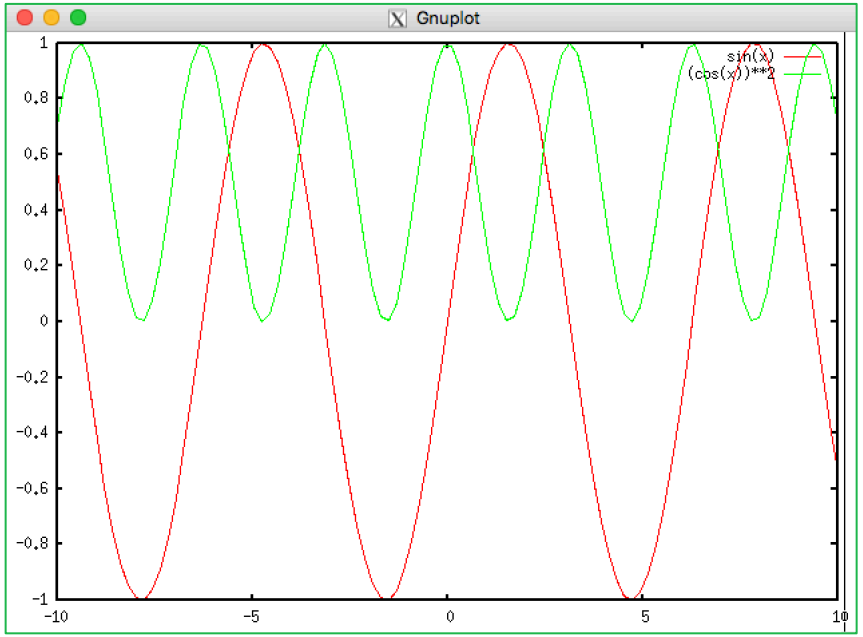
- It is available for all platforms, including Linux, Mac and Windows (`http://www.gnuplot.info`)

- To start gnuplot from the terminal, simply type

```
> gnuplot
```

- To produce a simple plot, e.g. `f(x) = sin(x)` and `f(x) = cos(x)^2`

```
gnuplot> plot sin(x)
gnuplot> replot (cos(x))**2 # Add another plot
```



- By default, gnuplot assumes that the independent, or "dummy", variable for the plot command is "x" (or “t” in parametric mode).

Mathematical Functions

- In general, any mathematical expression accepted by C, FORTRAN, Pascal, or BASIC may be plotted. The precedence of operators is determined by the specifications of the C programming language.
- Gnuplot supports the same operators of the C programming language, except that most operators accept integer, real, and complex arguments.
- Exponentiation is done through the ** operator (as in FORTRAN)

Function	Returns
abs(x)	absolute value of x, |x|
acos(x)	arc-cosine of x
asin(x)	arc-sine of x
atan(x)	arc-tangent of x
cos(x)	cosine of x, x is in radians.
cosh(x)	hyperbolic cosine of x, x is in radians
erf(x)	error function of x
exp(x)	exponential function of x, base e
inverf(x)	inverse error function of x
invnorm(x)	inverse normal distribution of x
log(x)	log of x, base e
log10(x)	log of x, base 10
norm(x)	normal Gaussian distribution function
rand(x)	pseudo-random number generator
sgn(x)	1 if x > 0, -1 if x < 0, 0 if x=0
sin(x)	sine of x, x is in radians
sinh(x)	hyperbolic sine of x, x is in radians
sqrt(x)	the square root of x
tan(x)	tangent of x, x is in radians
tanh(x)	hyperbolic tangent of x, x is in radians

Bessel, gamma, ibeta, igamma, and lgamma functions are also supported. Many functions can take complex arguments. Binary and unary operators are also supported.

Using set/unset

- The set/unset commands can be used to controls many features, including axis range and type, title, fonts, etc…
- Here are some examples:

Command	Description
`set xrange[0:2*pi]`	Limit the x-axis range from 0 to 2*pi,
`set ylabel "f(x)"`	Sets the label on the y-axis (same as "`set xlabel`")
`set title "My Plot"`	Sets the plot title
`set log y`	Set logarithmic scale on the y-axis (same as "`set log x`")
`unset log y`	Disable log scale on the y-axis
`set key bottom left`	Position the legend in the bottom left part of the plot
`set xlabel font ",18"`	Change font size for the x-axis label (same as "`set ylabel`")
`set tic font ",18"`	Change the major (labelled) tics font size on all axes.
`set samples 2500`	Set the number of points used to draw a function.

- Immediate help is available inside gnuplot via the "help" command.

Plotting Datafiles

- Gnuplot can also plot ASCII datafile in multicolumn format;

file.dat

```
# Comments can be placed here
x0  y0  z0   ...
x1  y1  z1   ...
.    .   .   ...
xN  yN  zN   …
```

- To plot a multi-column datafile using the 1st column for the abscissa and the 2nd column as the ordinate, use

```
gnuplot> plot “file.dat” using 1:2
```

- Add a second plot using 1st (=x) and 3rd (=y) columns:

```
gnuplot> replot “file.dat” using 1:3
```

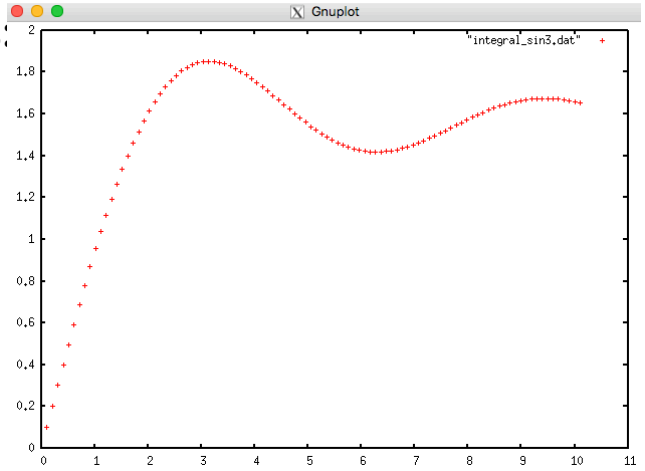
- If the “using” keyword is not specified, 1st and 2nd columns are assumed:

```
gnuplot> plot “file.dat”
```

Example of Plotting Styles

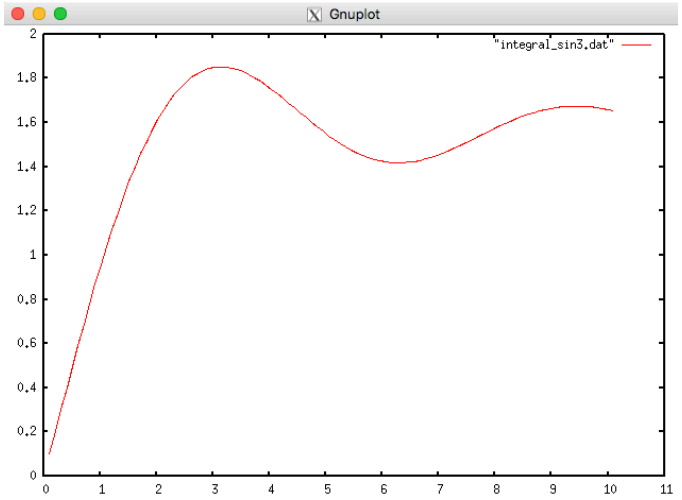
- When plotting datafiles, Gnuplot uses symbols:

```
gnuplot> plot “file.dat”
```



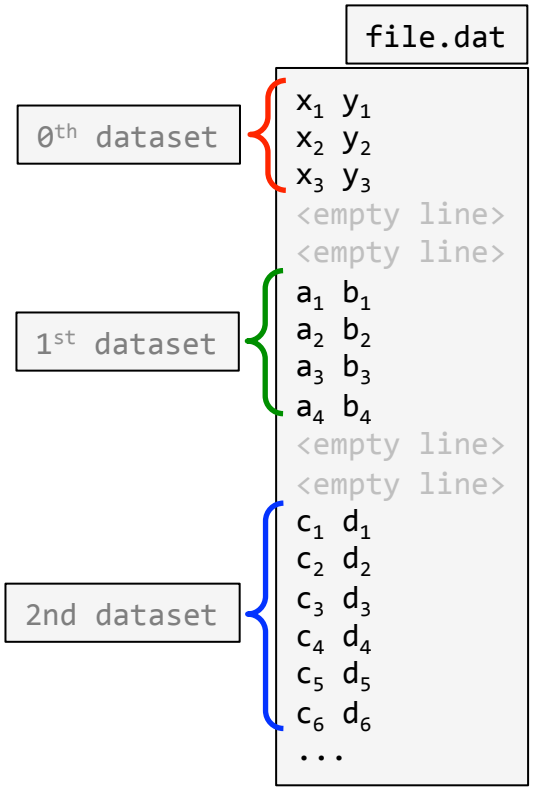
- To join symbols with lines, use

```
gnuplot> plot “file.dat” with lines
```



Datafiles containing multiple datasets

- A single datafile may also include more than one data set, which must be separated by a pair of empty lines, e.g.



- In this case you can tell gnuplot which dataset should be read using the '`index`' keyword. For instance,

```
gnuplot> plot "file.dat" using 1:2 index 0
```

will plot the 3 points (x_1, y_1), (x_2, y_2), (x_3, y_3).

- Likewise

```
gnuplot> plot "file.dat" using 1:2 index 2
```

will plot the 6 points (c_1, d_1), (c_2, d_2) ... (c_6, d_6).

Creating Scripts for Gnuplot

- A Gnuplot script is a simple text file (with the extension ".gp") containing a set of instructions to produce the desired plot.
- Consider the following file, "`myscript.gp`"

```
reset                    # force all graph-related options to default values
fname = "myfile.dat"     # file name

set autoscale xfixmin    # axis range automatically scaled to include the range
set autoscale xfixmax    # of data to be plotted

set tics font   ",18"
set xlabel "x"  font ",18"
set ylabel "y"  font ",18"

set lmargin at screen 0.1    # set size of left margin
set rmargin at screen 0.82   # set size of right margin
set bmargin at screen 0.12   # set size of bottom margin
set tmargin at screen 0.95   # set size of top margin
plot fname using 1:3
```

- Comments are preceded with a "#" symbol.

- From gnuplot, you can now invoke this script using the "`load`" command:

```
gnuplot> load "myscript.gp"
```

Producing Datafiles from C++

- There're basically two ways to produce a multicolumn ASCII datafile from the output of a C++ program:
  1. [Simple, not very general] By redirecting the output of a program to file:

```
./myprogram > myprogram.dat
```

The ">" sign is used for redirecting the output of a program to something other than stdout (standard output, which is the terminal by default). Similarly, the >> appends to a file or creates the file if it doesn't exist.

2. [Clever, more general] By creating the file using the `ofstream` (or similar) class in C++

```
#include <fstream>
...
ofstream fdata;         // declare Output stream class to operate on files
fdata.open("decay.dat"); // open output file
...
for (...){
  fdata << x << "  " << fx << "  "  << .. << endl; // write to file
}
fdata.close();                                     // close file
```

Writing 2D Arrays

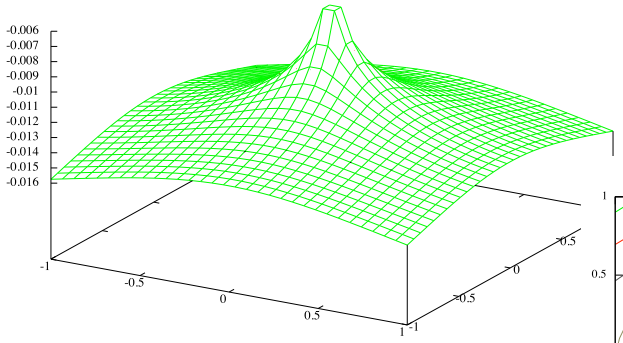
- Two-dimensional arrays (such as `f[i][j]`) can be written in multi-column ASCII format with the index *j* changing faster and a blank records separating blocks with different index *i*:

```
x0   y0      f(0,0)
x1   y0      f(1,0)
. . .
xN   y0      f(N,0)
                          ← <empty line>
x0   y1      f(0,1)
. . .
xN   y1      f(N,1)
                          ← <empty line>
.
.
.
                          ← <empty line>
x0   yN      f(0,N)
. . .
xN   yN      f(N,N)
```

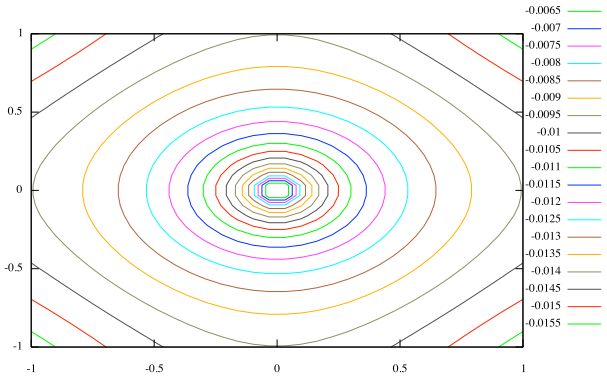
Visualizing 2D Arrays

- Gnuplot can be used to display 2D arrays using the "`splot`" command instead of "*`plot`*".

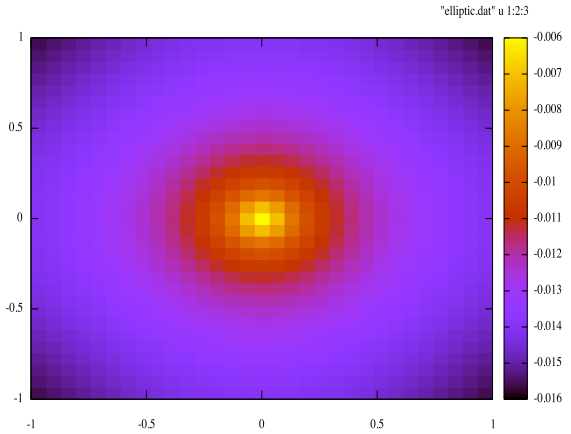
- Different visualizations are possible:



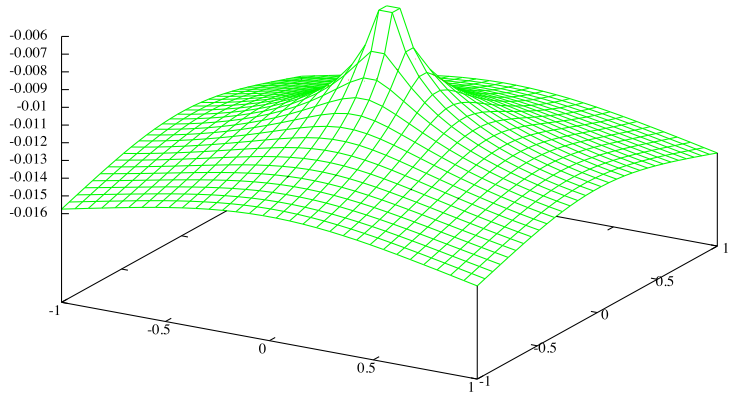
Surface plot



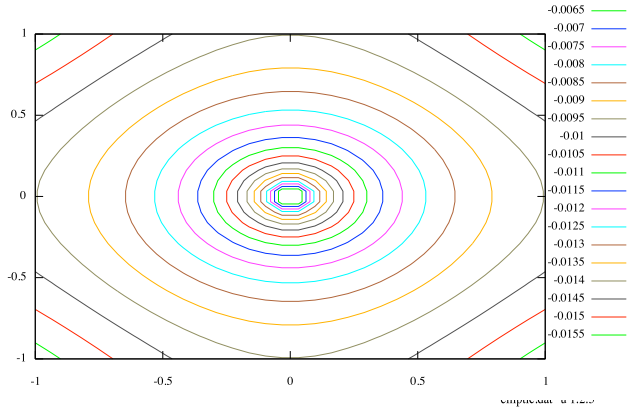
Contour plot



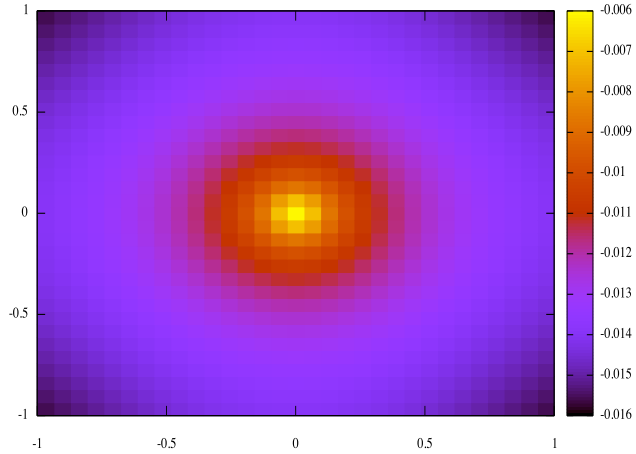
Colored maps




```
gnuplot> set surface
gnuplot> set hidden3d
gnuplot> splot “data.dat”  u 1:2:3  w lines
```




```
gnuplot> set contour
gnuplot> unset surface
gnuplot> set view map
gnuplot> set cntrparam level 20
gnuplot> splot "elliptic.dat" u 1:2:3 w lines
```

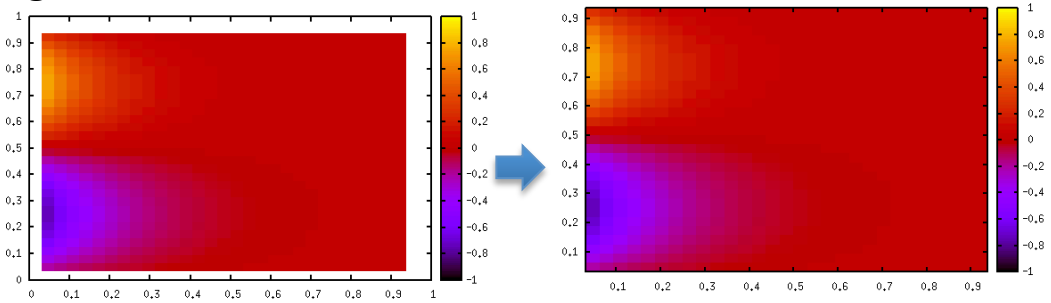



```
gnuplot> set pm3d map
gnuplot> splot “data.dat”  u 1:2:3
```

More on pm3d map

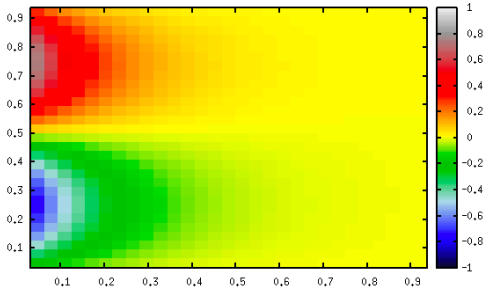
- Pm3D map is a useful plotting style for function of 2D variables. Some tips:
  - Exact axis range can be forced using

```
gnuplot> set autoscale xfixmin
gnuplot> set autoscale xfixmax
gnuplot> set autoscale yfixmin
gnuplot> set autoscale yfixmax
gnuplot> splot “file.dat”
```



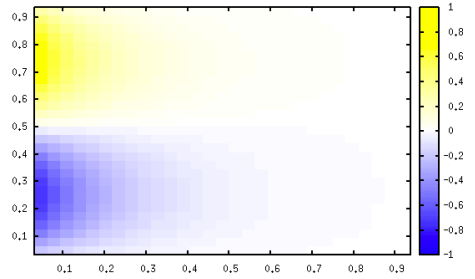
  - Gray-to-rgb mapping can be set through

```
gnuplot> set palette defined
```



  - A color gradient can be defined and used to give the rgb values.

```
gnuplot> set palette defined (0 "blue", 1 "white", 2 "yellow")
```



Slicing Datasets: the "every" keyword

- The keyword "every" specifies which datalines (subsets) within a single data set are to be plotted. It has the following syntax:

```
plot 'file' every I:J:K:L:M:N
```

where

I	J	K	L	M	N
Line increment	Data block increment	First line	First data block	Last line	Last data block

- Examples:

```
plot 'file' every 2        # Plot every 2 lines
plot 'file' every ::3      # Plot starting from the 3rd line
plot 'file' every ::3::15 # Plot lines 3-15
```

- Note: the increments default is set to unity, the start values to the first point or block, and the end values to the last point or block.

Slicing Datasets: taking x- and y- slices

- In a 2D datasets (see "Writing 2D Arrays"), we can use plot with the every keyword to produce 1D cuts along a given direction.

- To take an *x*-slice (a plot at constant *y* along the *x*-direction), you may use

```
j = 2  # Fix the vertical index j = 2 (= 3rd block)
plot fname using 1:3 every :::(j)::(j) with linespoint

# This is equivalent to (expliciting writing the increment and starting indices):
plot fname using 1:3 every 1:1:0:(j)::(j) with linespoint pt 4
```

- To take an *y*-slice (a plot at constant *x* along the *y*-direction), you may use

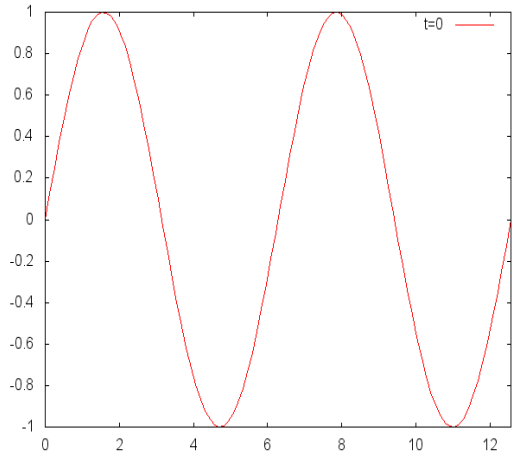
```
i = 1   # Fix the horizontal index i = 1 (= 2nd block)
plot fname using 2:3 every ::(i)::(i)

# Note: the previous command does not allow data points to be connected by lines
# If you wish to connect data points with lines, you may "cheat"
# using the splot command:

set view map
splot fname using 2:2:3 every ::(i)::(i) w lp
```

Creating Animations

- Animations can be built using the `do for[]{..}` in gnuplot (v ≥ 4.6).

- Consider the following example (`simple_animation1.gp`):

```
omega = 2.0*pi;
ntot  = 250             # Number of frames in one period
dt    = 1.0/ntot        # The increment between frames
do for [n=0:2*ntot]{
  t   = n*dt            # Time
  plot sin(x - omega*t)
  pause 0.1             # pause in seconds
}
```



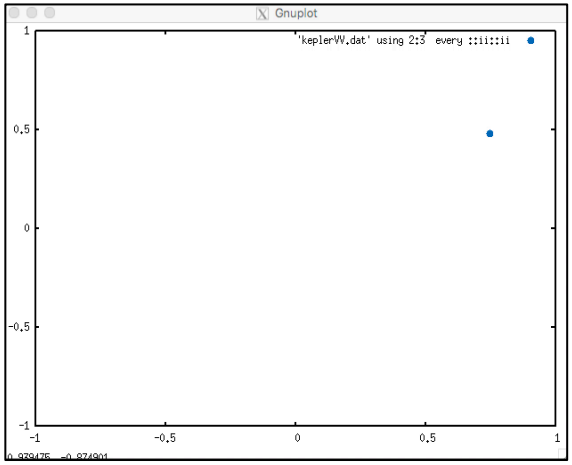
- If your gnuplot support .png, .gif or .jpeg terminal, images can be saved to disk:

```
set term png    # From now on, plots will be done on png terminal
                # and not on screen
omega = 2.0*pi;
ntot  = 250             # Number of frames in one period
dt    = 1.0/ntot        # The increment between frames
do for [n=0:2*ntot]{
 fname = sprintf ("sin_%04d.png",n)   # File name
  set output fname                    # Redirect output to file
  t   = n*dt            # Time
  plot sin(x - omega*t)
}
```

Trajectory: 2D Animation

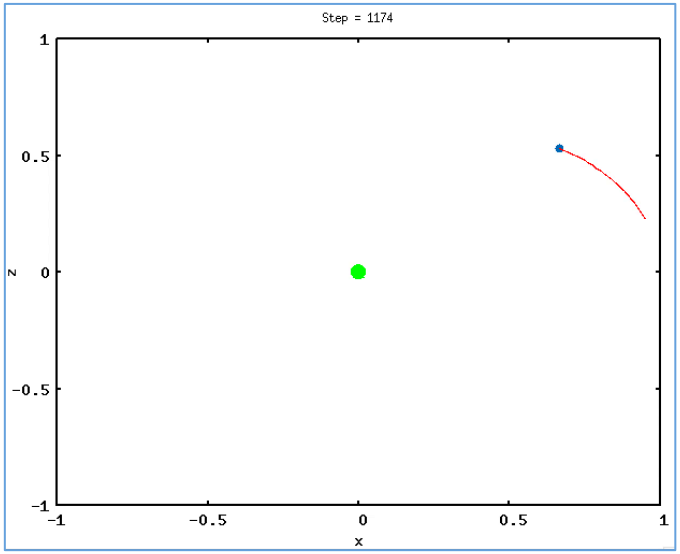
- The following script demonstrate how a trajectory can be animated:

```
set xrange [-1:1]   # Always a good idea to
set yrange [-1:1]   # fix the axis range

set pointsize 2                              # symbol size
set style line 2 lc rgb '#0060ad' pt 7   # circle
do for [ii=1:3762] {   # Start plotting
   plot 'keplerVV.dat' using 2:3 every ::ii::ii linestyle 2
   pause 0.02
}
```



- An improved version adds the Sun (in green) and a red wake (taken from `Animations/kepler*.*`):

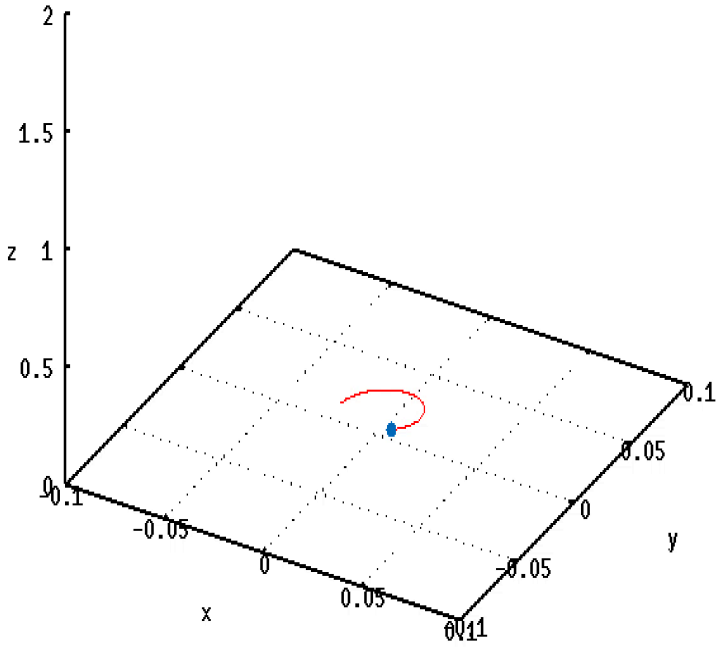
```
...
ntail = 50  # number of points to draw in the tail
ninc  = 3   # increment between frames

# Add the sun in the center as a green filled circle
set object circle at first 0,0 size scr 0.01 \
    fillcolor rgb 'green' fillstyle solid

do for [ii=1:3762:ninc] {
   im  = ((ii - ntail) < 0 ? 1:ii-ntail)
   title = sprintf ("Step = %d",ii)
   set title title
   plot 'keplerVV.dat' using 2:3  every ::ii::ii linestyle 2, \
        'keplerVV.dat' using 2:3  every ::im::ii with lines  lt 1
}
```



Trajectory: 3D Animations

- If the particle's trajectory is not confined to a plane, then you can modify the script by using set parametric and splot (taken from `Animations/spiral_anim.*`)

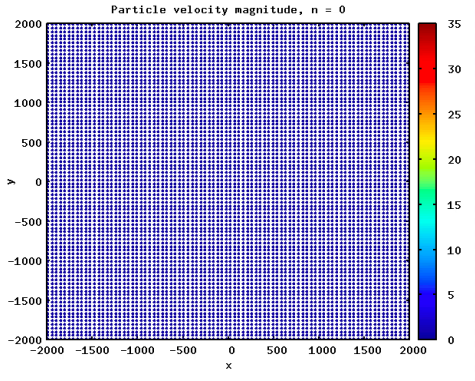
```
set parametric
set xyplane at 0
set grid

set pointsize 2                          # symbol size
set style line 2 lc rgb '#0060ad' pt 7   # circle

# -- Plot setting --
set xrange[-0.1:0.1]
set yrange[-0.1:0.1]
set zrange[0:2]

nstop = 990
ntail = 70
ninc  = 3  # increment between frames

set view 60,30
set hidden3d
fname = "spiral_anim.dat"   # datafile name
do for [ii=1:nstop:ninc] {
   print ii
   im = ((ii - ntail) < 0 ? 1:ii-ntail)
   splot fname using 2:3:($4) every ::ii::ii linestyle 2,\
         fname using 2:3:($4) every ::im::ii with lines lt 1

   # Add shadow on the xy plane
   replot fname using 2:3:(0*$4) every ::im::ii with lines lt 3
}
```



Many Particles Animation

- If you have many particles travelling at different energies, you may have several datafiles, one for each time t.
- In this case a different input data-file is read at each loop cycle:

```
set cbrange [0:35]    # Fix the colorbar range

set pointsize 1
set style line 2 lc rgb '#0060ad' pt 7   # circle

set xlabel "x" font ",18"
set ylabel "y" font ",18"
set tics font ",18"

vmag(vx,vy,vz) = sqrt(vx*vx + vy*vy + vz*vz) # Define useful column-function

do for [n=0:100] {
  title = sprintf ("Particle velocity magnitude, n = %d",n) # Title string
  set title title_string font ",18”
  fname = sprintf ('particles.%04d.tab',n) # Datafile string

  plot fname using 2:3:(vx=$5, vy=$6, vz=$7, vmag(vx,vy,vz)) \
       every 1 with points ls 2 palette
}
```



- See `Animations/nparts_anim.*`.

References on the Web

- Many tutorials on Gnuplot are available online.

- http://www.gnuplotting.org - This website gives many useful examples on how to create nice looking plots. The section Gnuplot basics → Plotting data explains many different ways to plot datafiles.

- http://lowrank.net/gnuplot/index-e.html - Here you can find a nice tutorial, explaining Legend, tics, label, 2D and 3D plotting and much more.