`Gnuplot` Short Course

Tim Langlais
langlais@me.umn.edu

February 21, 1999

Introduction

`Gnuplot` is a powerful *freeware* program for plotting both 2D and 3D data. `Gnuplot` will run under a variety of environments including Linux, IRIX, Solaris, Windows, and DOS. `Gnuplot` requires only minimal graphics capability and will even run under a vt100 terminal. It has a variety of output options flexible enough so that plots generated by `gnuplot` can be inserted into text documents. This document covers `gnuplot` 3.5. `Gnuplot` pre 3.6 release is available on some, but not all, MEnet machines.

You should use this document in conjunction with "gnuplot-course.tar.gz," an archive file that contains all of the scripts and data in this document (as well as the document itself). From MEnet machines,

```
unix% cp ~langlais/gnuplot-course.tar.gz .
unix% gunzip gnuplot-course.tar.gz
unix% tar -xvf gnuplot-course.tar
```

This will create a directory called "gnuplot" with several subdirectories.

Basic 2D Plots

To start `gnuplot` (make sure you are in the "data" directory first), simply type

```
unix% gnuplot
```

and away you go...

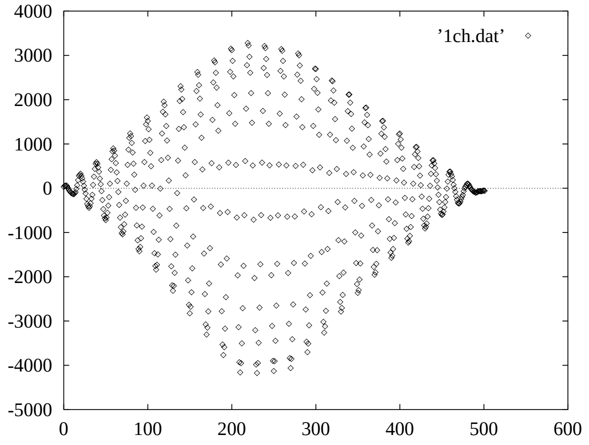
Say you have a single column of data in a file called "1ch.dat" that you would like to plot

28.062
52.172
55.703
64.281
43.438, etc.

This is simple to plot

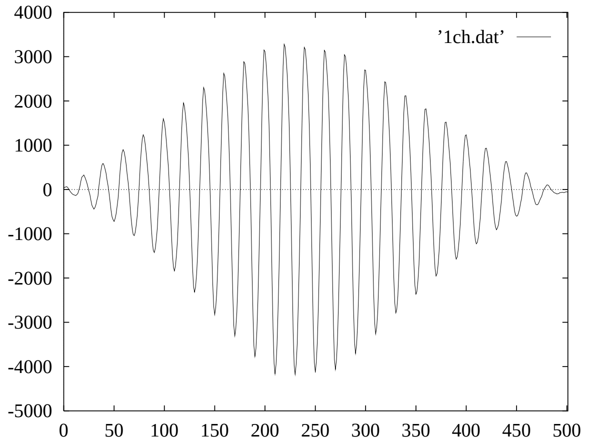
```
gnuplot> plot '1ch.dat'
```

`Gnuplot` will autoscale axes to include all the data. By default, `gnuplot` will plot the data using points. The file '1ch.dat' *must* be in the current working directory that you ran `gnuplot` from, else you will have to specify the path to the file; `..` and `/` are allowed, `~` is not.



Let's plot the data using lines, add more divisions to the x-axis, and rescale using `set xrange`.

```
gnuplot> set data style lines
gnuplot> set xtics 0,50,1000
gnuplot> set xrange [0:500]
gnuplot> plot '../data/1ch.dat'
```



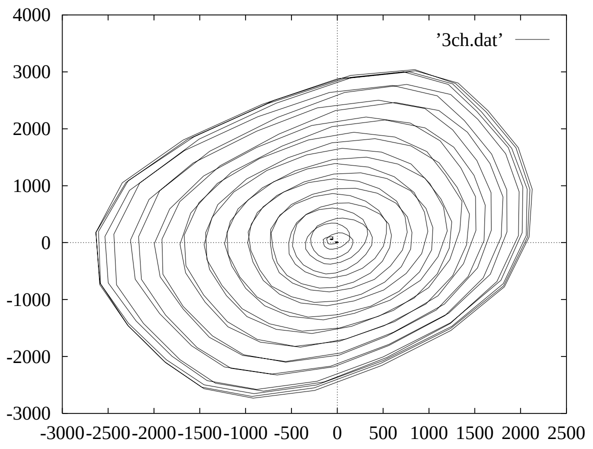
There are several choices for data style, including `lines`, `points`, `linespoints`, or `dots`. The `set xtics` command takes three arguments, `<start>, <increment>, <end>`, and `set xrange`, two, `[<start>:<end>]`.

But what if you have multicolumn data? Is it possible to plot one column of that data? Of course! Here we plot the second column of data from the file "3ch.dat"

```
gnuplot> plot '3ch.dat' using 2
```

The `using` command specifies a certain column of data, or cross-plots between columns. Here we'll plot column 1 on the x-axis and column 2 on the y.

```
gnuplot> plot '3ch.dat' using 1:2
```

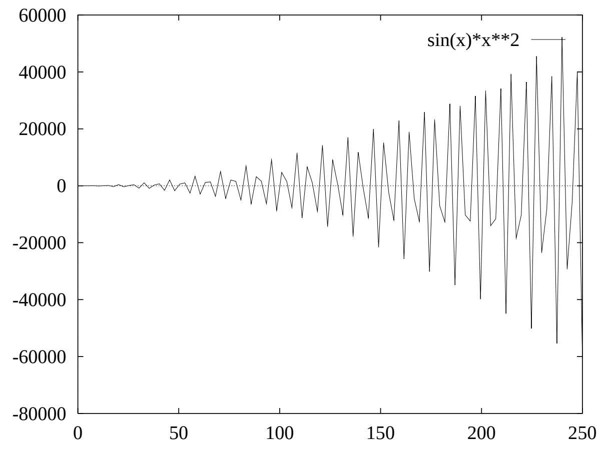


Operators, Constants, and Functions

`Gnuplot` not only reads data from files, it can also plot analytical functions. Toward this end, `gnuplot` provides the usual list of operators `+, -, *, /, **`, etc, and functions, `sin(), cos(), log(), exp()`, etc. A summary of operators and functions is available at the end of this text. Let's plot a simple function based on the above (note that `**` is exponentiation a la `FORTRAN`).

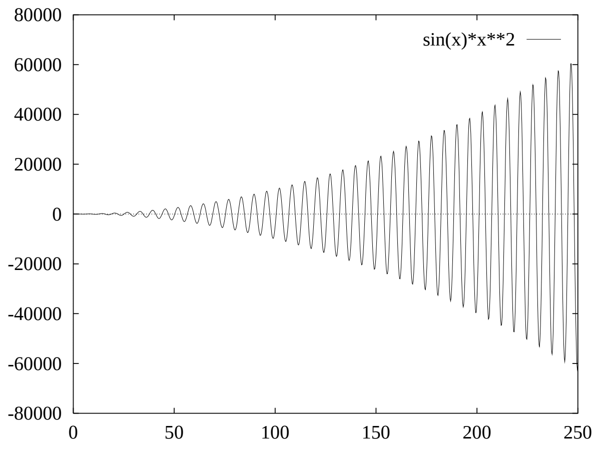
```
gnuplot> set xrange [0:250]
gnuplot> plot sin(x)*(x**2)
```

`Gnuplot` assumes that `x` is the independent variable.



The graph does not look like a smooth function at all. `Gnuplot` evaluates the function at certain points only—the sample rate is too low for this function. To change this, change the number of samples (or evaluations) `gnuplot` performs.

```
gnuplot> set samples 1000
gnuplot> replot
```



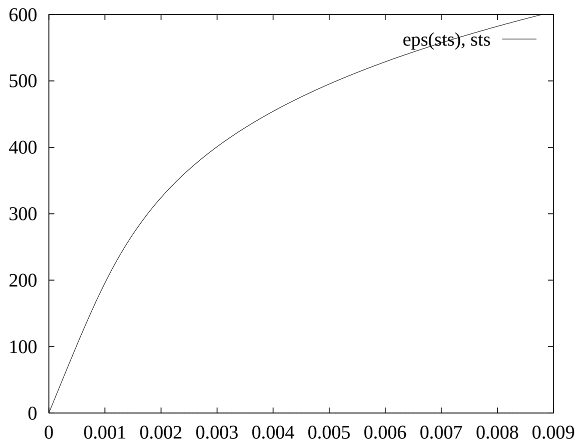
Ahhh! Much better.

`Gnuplot` lets the user define constants and functions as well. Say we have some sort of material property data that is described by the following equation

$$\epsilon = \frac{\sigma}{E} + \left(\frac{\sigma}{K'}\right)^{\left(\frac{1}{n'}\right)}$$

and we want to plot the function with ϵ (eps) on the x-axis and σ (sts) on the y-axis. The equation is valid for $\sigma = [0 : 600]$. First, we set `gnuplot` to plot parametrically, then we change the default parametric dummy variable `t` to `sts`. Since we know the valid range for `sts`, let's change that too. The function is not strictly defined at 0 so we will enter a small value for the lower bound. Then, we define the function `eps(sts)` and the material constants `E, Kp, np`. Finally, we plot the function.

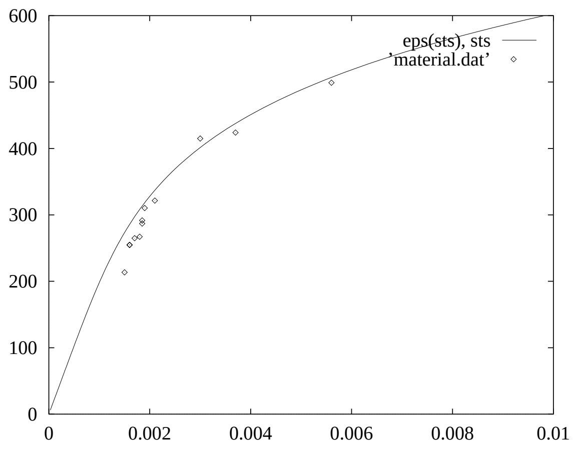
```
gnuplot> set parametric
    dummy variable is t for curves,
    u/v for surfaces
gnuplot> set dummy sts
gnuplot> set trange [1.0e-15:600]
gnuplot> eps(sts)=sts/E+(sts/Kp)**(1.0/np)
gnuplot> E = 206000.0
gnuplot> Kp = 1734.7
gnuplot> np = 0.2134
gnuplot> plot eps(sts), sts
```



Note that `gnuplot` is case sensitive: `E` is different from `e`. For 2D parametric plots, the format of the plot command is `plot <x(t)>, <y(t)>` where `t` is the dummy variable and `x(t)` and `y(t)` can be any functions of `t`. Constants can be entered in decimal format (600.456) or in exponential format (`1.0e-15`$= 1.0 * 10^{-15}$). String constants (e.g. `a='fred'`) are not allowed.

But what if we have actual discrete data that we'd like to compare to the analytical curve? No problem! Let's draw the analytical curve with lines and the actual data with points.

```
gnuplot> set xrange [0:0.01]
gnuplot> plot eps(sts), sts with lines, \
> 'material.dat' with points
```



Note too how we can break input lines using `\` just like in a UNIX shell. The `\` at the end of the `plot` line tells `gnuplot` to continue to the next line and treat it and the preceding line as one. You can even split lines within a string like a title or file name.

Formatting

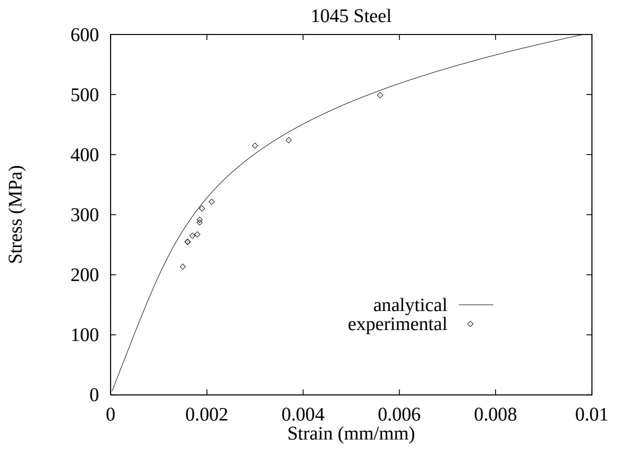
`Gnuplot` has several formatting parameters that can be used to change the appearance of a plot. These parameters are accessed using the `set` command. A list of these formatting parameters (for `gnuplot` 3.5) appears at the end of this document.

`Gnuplot` automatically locates the line identifiers—called the key—in the upper right corner of the plot. In our example, the analytical curve passes through the line labels, making the labels difficult to read. We can change this using `set key <x>, <y>`. Let's also change the line labels themselves.

```
gnuplot> set key 0.007, 150
gnuplot> plot eps(sts), sts \
> title 'analytical' with lines,\
> 'material.dat' title 'experimental' \
> with points
```

The `<x>` and `<y>` parameters of the `set key` command refer to the local coordinate system of the plot. The `title` command within `plot` *must* come before the `with` command. Adding a plot title and axis labels is easy too.

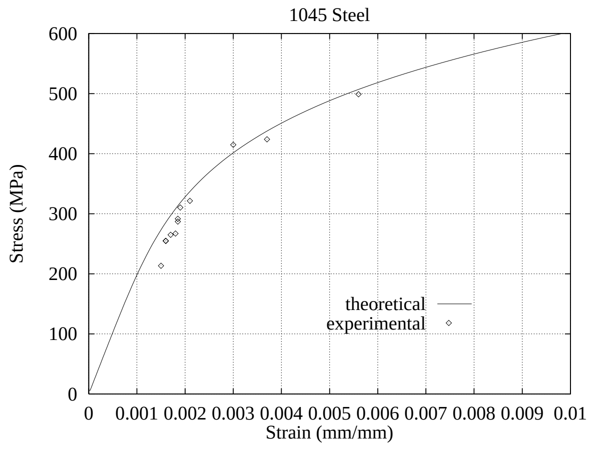
```
gnuplot> set title '1045 Steel'
gnuplot> set ylabel 'Stress (MPa)'
gnuplot> set xlabel 'Strain (mm/mm)'
gnuplot> replot
```



The `replot` command repeats the last `plot`.

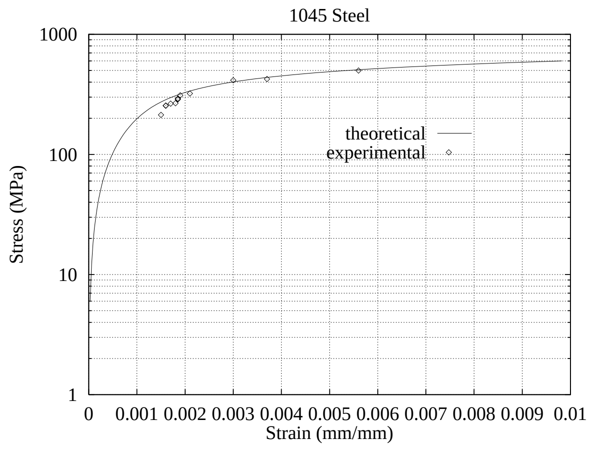
The `set xlabel` and `set ylabel` commands also take optional arguments: `set xlabel 'string' <xoffset> <yoffset>`. The offsets are measured in characters. Let's add more tics to the x-axis, set up a grid, and move the ylabel closer to the actual graph.

```
gnuplot> set xtics -1, 0.001, 1
gnuplot> set grid
gnuplot> set ylabel 'Stress (MPa)' 2, 0
gnuplot> replot
```



There are three basic axis parameters: 1) `set xrange [<x>:<y>]`, which allows the user to specify the viewable range, 2) `set autoscale`, which forces `gnuplot` to set the viewable range, and 3) `set logscale <x|y>`, which sets logarithmic scaling. Let's logscale the y-axis on the above plot,

```
gnuplot> set logscale y
gnuplot> replot
```



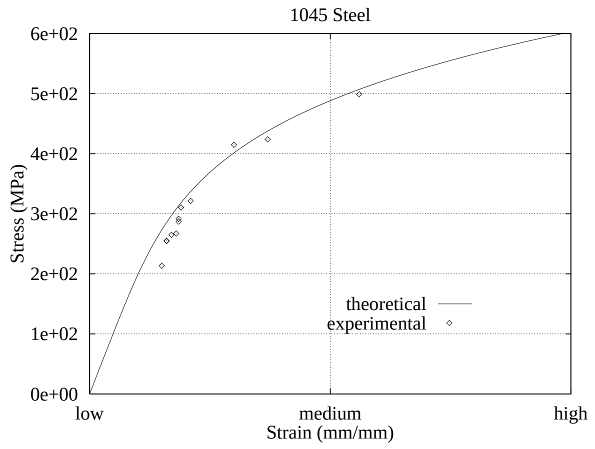
Grids are especially useful for plots with logarithmic scales.

It is also possible to customize the appearance of the tic labels in `gnuplot`. First, let's turn off the y-axis logscale, then set the y-axis tics to read in exponential notation with no digits after the decimal point.

```
gnuplot> set nologscale y
gnuplot> set format y '%.0e'
gnuplot> replot
```

The format command uses `C printf()` syntax, so `set format x '\%d wombats'` is a valid command. It is also possible to specify no-numeric tic labels.

```
gnuplot> set xtics ('low' 0, \
> 'medium' 0.005, 'high' 0.01)
gnuplot> replot
```

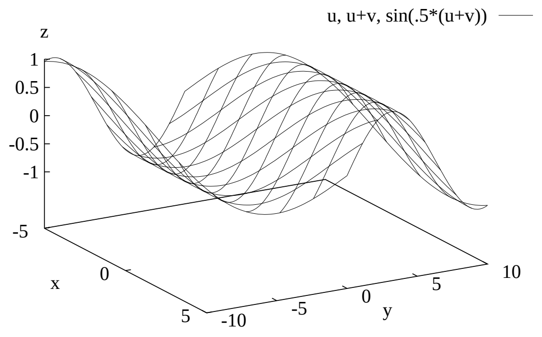


The parentheses are necessary when specifying non-numeric labels.

3D Plots

Besides 2D graphs, `gnuplot` can also generate simple 3D plots like

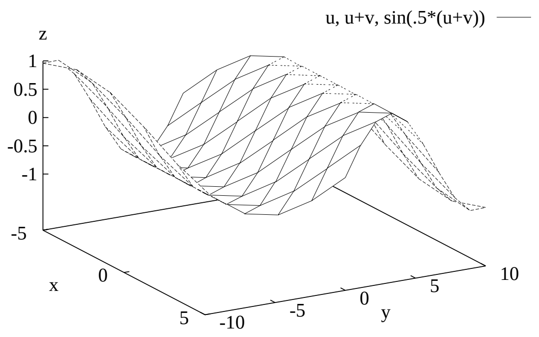
```
gnuplot> set data style lines
gnuplot> set parametric
    dummy variable is t for curves,
    u/v for surfaces
gnuplot> set view 60, 60, 1, 1
gnuplot> set xlabel 'x'
gnuplot> set ylabel 'y'
gnuplot> set zlabel 'z'
gnuplot> splot u,u+v,sin(0.5*(u+v))
```



Plots in 3D must use `splot` instead of `plot`. You can set the perspective view using `set view <rot_x>, <rot_z>, <scale>, <scale_z>`, which allows the user to set rotations about the x- and z-axis as well as scale the graph.

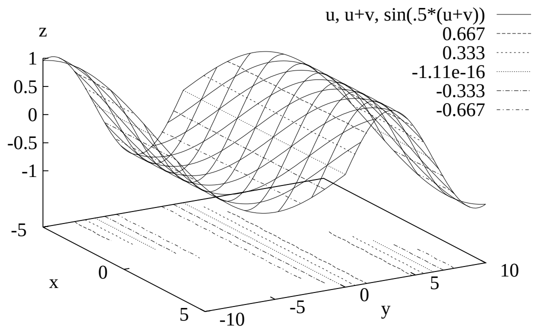
`Gnuplot` will also do hidden line removal

```
gnuplot> set hidden3d
gnuplot> splot u,u+v,sin(0.5*(u+v))
```



and draw contours based on the z-axis value (sorry, gnuplot cannot handle 4D data—3 spatial dimensions plus 1 data channel). You can choose to have the contours drawn on the surface, on the base, or both.

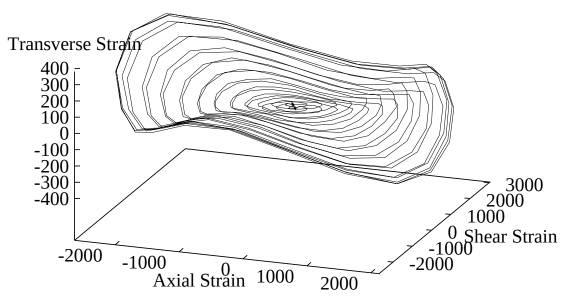
```
gnuplot> set contour both
gnuplot> splot u,u+v,sin(0.5*(u+v))
```



Unfortunately, the mess of lines is often confusing rather than enlightening.

Gnuplot can read in data files of gridded and scattered data. Let's make a 3D plot of scattered data.

```
gnuplot> set data style lines
gnuplot> set view 60,20,1,1
gnuplot> set xtics -3000,1000
gnuplot> set xlabel 'Axial Strain'
gnuplot> set ylabel 'Shear Strain' 2
gnuplot> set zlabel 'Transverse Strain'
gnuplot> set nokey
gnuplot> splot '3ch.dat'
```



Saving Your Work

The gnuplot save command is an easy way to save all of parameter settings, constant definitions, function definitions, and the last plot command. You can even elect to save only certain variables or functions. Gnuplot stores the information in an ASCII text file that can be read into gnuplot using load <file>. An excerpt of what gnuplot saves

```
set title "1045 Steel" 0,0
set notime
set rrange [-0 : 10]
set trange [0 : 600]
set urange [-5 : 5]
set vrange [-5 : 5]
set xlabel "Strain (mm/mm)" 0,0
set xrange [0 : 0.01]
set ylabel "Stress (MPa)" 0,0
set yrange [0 : 600]
set zlabel "" 0,0
set zrange [-10 : 10]
set autoscale r
set noautoscale t
set autoscale  y
set autoscale z
set zero 1e-08
eps(sts)=sts/E+(sts/Kp)**(1.0/np)
E = 206000.0
Kp = 1734.7
np = 0.2134
```

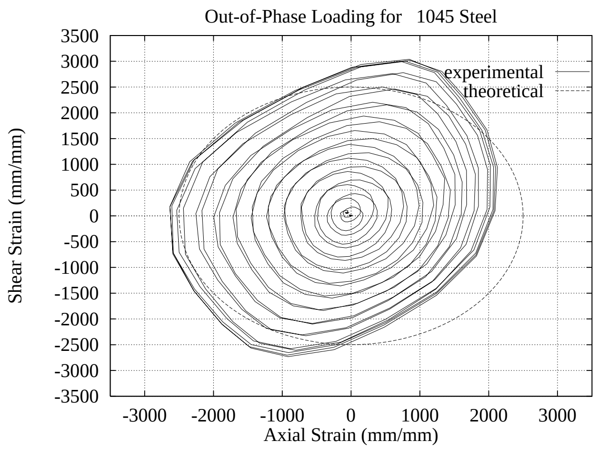
Notice that gnuplot simply saves a list of gnuplot commands.

To this point, we have interacted with gnuplot by typing in commands at the prompt. The easier, more efficient way to interact with gnuplot—especially if you are creating complex plots that reset parameters and define variables—is to edit a text file list of commands, much like a MATLAB or ANSYS batch file. Using your favorite editor, create a file called 'script2.gp' that looks like

```
# set square ranges
set xrange [-3500:3500]
set yrange [-3500:3500]

# leave plenty of tics
set xtics -10000,1000
set ytics -10000,500

# set the format of the tic-mark labels
set format '%g'

# set the plot title and axis labels
set title 'Out-of-Phase Loading \
  for 1045 Steel'
set ylabel 'Shear Strain (mm/mm)'
set xlabel 'Axial Strain (mm/mm)'
set key 2800,2800

# set up a grid
set grid

# enable parametric plotting
set parametric
set trange [0:2.0*pi]

# finally plot the graph
plot '3ch.dat' u 1:2 \
  title 'experimental', \
  2500.0*cos(t), 2500.0*sin(t) \
  title 'theoretical'
```

Save the file and load

```
gnuplot> load 'script2.gp'
```



Gnuplot ignores empty lines and lines that begin with #. Everything else gnuplot interprets as a command. If you mistyped format as frmat in the above file, gnuplot would raise an error

```
gnuplot> load 'script2.gp'
gnuplot> set frmat '%g'
                    ^
         "script2.gp", line 11: valid ...
```

Using files like 'script2.gp' to interact with gnuplot is a good idea because the files serve as records of what you have done. In a later section we will discuss how gnuplot can be automated using shell scripts and these batch files.

Output Options

While gnuplot has several output options, Postscript is the most useful. You can send Postscript files directly to the printer using `lpr -P<printername> <filename>`. Generally, it is best to pipe Postscript output to a file. The following will generate a landscape 7 inch by 10 inch plot.

```
gnuplot> load 'script2.gp'
gnuplot> set output 'plot.ps'
gnuplot> set terminal postscript
gnuplot> replot
gnuplot> set term x11; replot
```

Once the terminal has been set to postscript, you need to replot in order to produce output. Remember to reset the terminal to x11 to display the graph on the screen. Note that gnuplot can accept several commands on the same line when they are separated by ;, just like in the UNIX shell.

The `set terminal postscript` command has many options. Let's generate a color portrait Postscript file that uses the Times 12-point font.

```
gnuplot> set output 'plot2.ps'
gnuplot> set terminal postscript \
> portrait color "Times-Roman" 12
gnuplot> replot
gnuplot> set term x11; replot
```

You can substitute any valid Postscript font for Times-Roman.

If you want to change the size or aspect ratio of your plot, you can use `set size`. For instance,

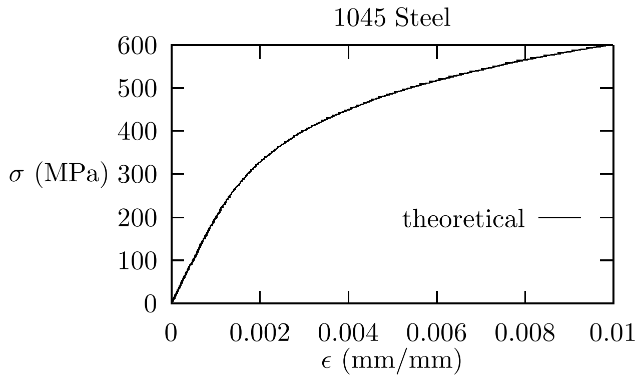
```
gnuplot> set size .721,1
```

will create a square graph.

Gnuplot and LaTeX

If you want to include gnuplot output in a LaTeX document, there are several options available. One gnuplot terminal type is latex. In this case, the output is raw LaTeX code that can be included directly into a document. This has the advantage that LaTeXish commands can be used in titles, etc.

```
gnuplot> set xlabel '$\epsilon$ (mm/mm)'
gnuplot> set ylabel '$\sigma$ (MPa)'
gnuplot> set output 'plot.tex'
gnuplot> set terminal latex
gnuplot> load 'script1.gp'
gnuplot> set term x11; replot
```



The simplest way to include this in a LaTeX document is

```
\begin{center}
\input{plot.tex}
\end{center}
```

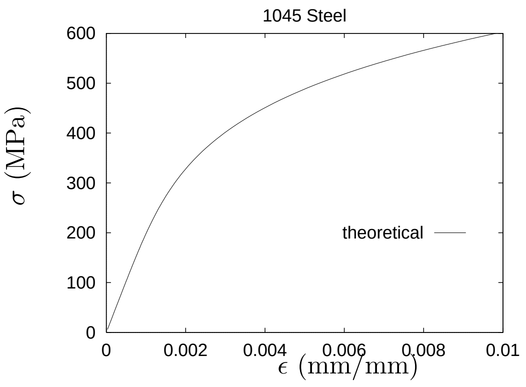
This method is not robust. Sometimes, the output LaTeX code is too big for LaTeX to handle without some work. Furthermore, LaTeX cannot draw smooth curved lines without aliasing so that the theoretical curve above looks jagged.

Selecting `postscript eps` is another way to get similar results. It is easy to include `eps` documents in LaTeX. But since `gnuplot` *cannot* selectively output Symbol font in Postscript mode, you have to use `psfrag` to substitute LaTeX commands for simple text (`psfrag` actually searches the Postscript for the text string).

```
gnuplot> set xlabel 'epsilon'
gnuplot> set ylabel 'sigma'
gnuplot> set output 'plot.eps'
gnuplot> set terminal postscript eps \
> "Helvetica" 24
gnuplot> load 'script1.gp'
gnuplot> set term x11; replot
```

The companion LaTeXcode is

```
\usepackage{graphicx}
\usepackage{psfrag}
.
.
.
\begin{center}
\psfrag{epsilon}{$\epsilon$ (mm/mm)}
\psfrag{sigma}{$\sigma$ (MPa)}
\includegraphics{plot.eps}
\end{center}
```



Fonts specified for `postscript eps` are halved when the output is generated. Thus, `"Helvetica" 24` actually appears as 12 point Helvetica.

Interacting with the Shell

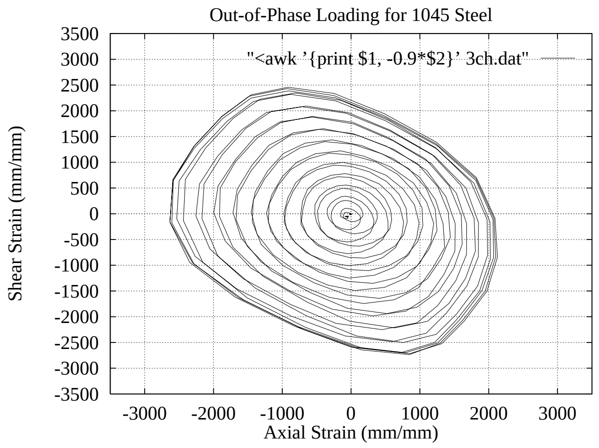
`Gnuplot` allows the user to escape to the shell using `!`. If we wanted to know the contents of the current working directory, we would type

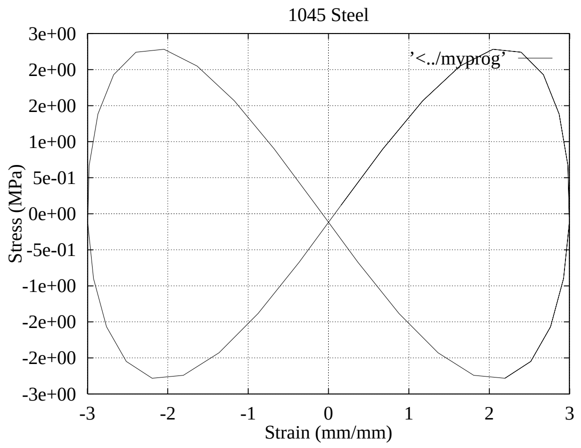
```
gnuplot> !ls
1ch.dat      junk.gp       script1.gp
3ch.dat      material.dat  script2.gp
!
gnuplot>
```

The `gnuplot>` prompt will return *only* when the UNIX process spawned by `!` has finished. The `!` shell escape is an easy way to list files, look at file contents, etc. Rather unfortunately, `!cd ../directory` does not move the user to another directory.

`Gnuplot` can also interact with the shell by reading from `stdout`. For instance, we can use the UNIX utility `awk` to plot '3ch.dat' with the shear strain in column 2 multiplied by −0.9. The `awk` utility reads files line by line and by default will print the entire line. The first column is denoted by `$1`, the second by `$2`, etc.

```
gnuplot> plot "<awk '{print $1,-0.9*$2}' \
> 3ch.dat"
```





Normally, `gnuplot` does not differentiate between `"` and `'` when quoting files, etc. However, the UNIX shell *does* differentiate between the two. Using `"` to enclose `{print $1,-0.9*$2}` rather than `'` causes the shell to interpret `$1` rather than allowing `awk` to interpret it. If you are not familiar with these concepts, or with `awk`, use `man awk` to find out more.

`Gnuplot` can read standard output from any program, including your own. Say you have a code called "myprog" that prints three columns of numbers.

```
unix% myprog
0.157008 2.300000 0.120373
0.674853 2.265058 0.898681
1.172192 2.161293 1.568595
1.633916 1.991859 2.049314
.
.
.
```

You can plot these numbers without ever having to create an intermediate file. Notice that we can reset plot parameters to their defaults by entering them without arguments.

```
gnuplot> set autoscale x
gnuplot> set autoscale y
gnuplot> set xtics
gnuplot> set ytics
gnuplot> set key
gnuplot> plot '<../myprog' u 1:3
```

The ability to read from `stdout` is a powerful feature of `gnuplot`. This feature, when used in conjuction with shell scripts, can save disk space and time when generating plots of data.

Automating `Gnuplot`

Unlike many programs that rely on graphical interfaces, `gnuplot` can be run directly from the UNIX command-line. Users can generate plots from shell scripts automatically. This feature may not seem very important until your adviser says of the 37 graphs in your thesis, "I think you should have more tics on the x-axis and you should probably change the units on the y-axis."

First, let's see how we can generate `gnuplot` output directly from the UNIX command-line. Consider the following `gnuplot` script, "script3.gp."

```
set parametric
set dummy sts
set trange [1.0e-15:600]
set key 0.007,150
eps(sts)=sts/E+(sts/Kp)**(1.0/np)
E = 206000.0
Kp = 1734.7
np = 0.2134
set xrange [0:0.01]
set title '1045 Steel'
set ylabel 'Stress (MPa)'
set xlabel 'Strain (mm/mm)'
set term postscript
set output 'plot3.ps'
plot eps(sts), sts \
  title 'theoretical' with lines, \
  'material.dat' title 'experimental' \
  with points
```

We could start an interactive `gnuplot` session and use `load script3.gp` to generate "plot3.ps." But since "script3.gp" is self-contained from input to output, the following will also generate "plot3.ps."

```
unix% gnuplot script3.gp
```

Any number of files can be specified on the command-line; `gnuplot` will execute them in order using `load`.

A shell script is a UNIX batch file. It consists of an initial call to a UNIX shell (`csh`, `tcsh`, `bash`, `sh`, etc.) followed by a list of UNIX commands. The file that contains the shell script needs to be executable.

```
unix% chmod +x shell.sh
```

Here is a simple script, "shell.sh," that *creates* a `gnuplot` script file and generates a Postscript file designated on the command-line.

```
#! /usr/local/bin/bash

( echo 'set yrange [-2:2]'
  echo 'set xrange [0:10]'
  echo 'set samples 1000'
  echo 'set term postscript'
  echo "set output \"$1\""
  echo 'plot sin(x)' ) > temp.gp
gnuplot temp.gp
```

The `echo` command is a primitive `print`. Note that there is a difference between `"` (allows variable substitution) and `'` (does not allow variable substitution). The first UNIX command-line argument is the variable `$1`. The `\"` tells the shell to interpret `"` literally, rather than as the end of the `echo` string. To create a Postscript file called "trial.ps,"

```
unix% shell.sh trial.ps
```

You can use ghostview to see the Postscript file.

Now what if we had a directory full of data files that we wanted to plot? Let's assume that the plots all have the same axis labels and roughly the same range. The data is located in "data/lotsodata." All file names have a ".stn" extension. The script will send the output to a new directory called "post/."

```
#! /usr/local/bin/bash

# makes a directory to put the
#  output in
mkdir post

# assign each file in data/lotsodata
#  to the shell variable $i
for i in `ls data/lotsodata/*`
do
  # strip the .stn extension
  g=`basename $i .stn`

  # working....
  echo 'processing' $i

  # for each file generate a .gp
  #  script that is temporary
  ( echo 'set yrange [-3500:3500]'
    echo 'set xrange [-3500:3500]'
    echo 'set data style lines'
    echo 'set xlabel "Axial Strain"'
    echo 'set ylabel "Shear Strain"'
    echo 'set term postscript'
    echo "set output \"post/$g.ps\""
    echo "plot \"$i\" u 1:2" ) > temp.gp

  # generate output
  gnuplot temp.gp
  \rm temp.gp
done
```

Now try it out and look at the Postscript output.

```
unix% shell2.sh
unix% ghostview post/g1-2-3.ps &
```

Using a shell script this way saves time in generating and re-generating plots. Should you wish to change the x-axis label, you only need to edit the "shell2.sh" and re-run.

Perl has all of the features of shell scripting and more. If you have a problem more complicated than the above examples, you are encouraged to use Perl. MEnet has online documentation for Perl at `http://www.menet.umn.edu/docs/perl/perl.html`.

Getting Help

Typing `help` accesses `gnuplot`'s online help. The online help is text-based. It provides command and parameter descriptions as well as examples. Let's try it,

```
gnuplot> help
 GNUPLOT is a command-driven interactive
    function plotting program.

 For help on any topic, type `help`
    followed by the name of the topic.

 The new GNUPLOT user should begin by
 reading the `introduction` topic (type
 `help introduction`) and about the `plot`
 command (type `help plot`). Additional
 help can be obtained from the USENET
 newsgroup comp.graphics.gnuplot.

Help topics available:
    autoscale      binary-data    bugs
    cd             clear          comments
    copyright      environment    exit
    expressions    fit            help
    introduction   line-editing   load
    pause          plot           print
    pwd            quit           replot
```

```
        reread          save            set
        shell           show            splot
        startup         substitution    update
        userdefined
    Help topic:
```

At the help prompt, `Help topic:`, you can type any of the listed topics. In this way you will be directed down a hierarchy until you finally reach information on the specific command you need. If you want information on a specific command, say `set data style`, you can type

```
    gnuplot> help set data style
```

MEnet maintains the `gnuplot` manual on the web at `http://www.menet.umn.edu/docs/gnuplot.html`. This manual consists of the online help in HTML format. The folks at Dartmouth (where `gnuplot` got its start) also maintain a web-page with assorted `gnuplot` information, `http://www.cs.dartmouth.edu/gnuplot_info.html`. There you will find mailing lists, FAQ's, and tutorials. As mentioned in the `gnuplot` help, `gnuplot` also has its own USENET group, `comp.graphics.gnuplot`.

Summary of Commands

Operators

Symbol	Example	Explanation
-	-a	unary minus
~	~a	one's complement
!	!a	logical negation
!	a!	factorial
**	a**b	exponentiation
*	a*b	multiplication
/	a/b	division
%	a%b	modulo
+	a+b	addition
-	a-b	subtraction
==	a==b	equality
!=	a!=b	inequality
&	a&b	bitwise AND
^	a^b	bitwise exclusive OR
|	a|b	bitwise inclusive OR
&&	a&&b	logical AND
||	a||b	logical OR
?:	a?b:c	ternary operation

Functions

abs	absolute value
arg	phase of a complex number
floor	largest integer not greater
ceil	smallest integer that is not less
int	truncation to form integer
rand	random number [0:1], using a seed
real	real part of complex number
imag	imaginary part of a complex number
sin	sine (argument in radians)
cos	cosine (argument in radians)
tan	tangent (argument in radians)
asin	inverse sine in radians
acos	inverse cosine in radians
atan	inverse tangent in radians
sinh	hyperbolic sine (arg in radians)
cosh	hyperbolic cosine (arg in radians)
tanh	hyperbolic tangent (arg in radians)
sqrt	square root
log	log base e
log10	log base 10
exp	exponential function, e**(x)
sgn	sign of the argument (-1,0,1)
erf	error function
erfc	1.0 - erf(x)
inverf	inverse error function
gamma	gamma function
igamma	incomplete gamma function
ibeta	incomplete beta function
norm	the normal distribution function
inorm	
besj0	j0th Bessel function of its arg
besj1	j1st Bessel function of its arg
besy0	y0th Bessel function of its arg
besy1	y1st Bessel function of its arg

Format Parameters

Use the `set <parameter> ...` to access these parameters.

Axis Ranges and Scales

xrange	the x, y, z axis ranges
yrange	
zrange	
autoscale	(un)sets autoscaling for axis
noautoscale	
logscale	(un)set log scaling for axis
nologscale	

Labels

xlabel	x, y and z axis labeling
ylabel	
zlabel	
label	arbitrary label anywhere on plot
nolabel	turns off labeling

title	plot title (top)
time	plot time (lower left)
arrow	draw arrow on plot
noarrow	remove arrow from plot
key	position of line labels
nokey	no line labels

Parametric Plots

parametric	(un)set parametric plots
noparametric	
dummy	rename parametric dummy variable
trange	range for 2D parametric
urange	ranges for 3D parametric
vrange	

Polar Plots

polar	(un)set polar plotting
nopolar	
rrange	radial range for polar plots
angles	angular range

3D Plots

ticslevel	relative scaling of z-axis
surface	3D mesh plotting
nosurface	no 3D mesh
hidden3d	hidden line removal for surfacing
contour	plot contours on surfaces
cntrparam	contouring parameters
isosamples	isoline density on surface
dgrid3d	(no) gridded data
view	3D viewpoint
clabel	(un)label contours
noclabel	
mapping	coordinate system (cartesian, etc.)

Reference Lines

xzeroaxis	marks zero of axis with dotted line
yzeroaxis	
zeroaxis	
noxzeroaxis	unsets marking of zero axes
noyzeroaxis	
nozeroaxis	
grid	grid lines at each tic

Tics and Scales

tics	direction of tics (in or out)
xtics	set number of tics on axis
ytics	
ztics	
noxtics	remove tics (scales)
noytics	
noztics	
format	set format of tic-mark labels
xmtics	label tics with months rather than numbers
ymtics	
zmtics	
noxmtics	unset month tic labeling
noymtics	
nozmtics	
xdtics	label tics with days rather than numbers
ydtics	
zdtics	
noxdtics	unset day tic labeling
noydtics	
nozdtics	

General

border	rectangular border around plot
noborder	eliminate border
data style	set line/point-style for data
function style	set line/point-style for functions
boxwidth	width of boxes in data style
clip	(un)set clipping near borders
noclip	
size	changes physical plot dimensions
offset	offset plot in window
samples	number of samples for functions
output	set output display or device
terminal	set graphics device