LEARNING Gnuplot

#gnuplot

Table of Contents

About

Chapter 1: Getting started with Gnuplot

Remarks

This section provides an overview of what gnuplot is, and why a developer might want to use it.

It should also mention any large subjects within gnuplot, and link out to the related topics. Since the Documentation for gnuplot is new, you may need to create initial versions of those related topics.

Versions

Version	Last *patchlevel*	Last Release Date
5.0.x	5.0.5	2016-10-09
4.6.x	4.6.7	2015-04-28
4.4.x	4.4.4	2011-11-13
4.2.x	4.2.6	2007-07-01
4.0.x	4.0.0	2004-04-01

Examples

Installation or Setup

Gnuplot is a portable command-line driven graphing utility. This example will show how to setup gnuplot in the various platforms.

Windows

1. Download the latest version of the installer from gnuplot site.
2. Run the downloaded file and allow it to run as administrator if requested
3. On the setup window select the language and follow the instructions on screen.
4. (optional) During the installation you may select the gnuplot to be added to the PATH that will allow you to run commands from anywhere on the command line. If you choose not to do so you may add it manually later or `cd` to the gnuplot installed directory prior to running commands.

The default installation location of gnuplot on Windows is `C:\Program Files (x86)\gnuplot`

NOTE: the filename will be of the format: `gp<version>-win32-mingw.exe`

Linux

The installation on Linux can be done through the different package managers as follows.

Arch

```
$ sudo pacman -S gnuplot
```

Debian and Ubuntu

```
$ sudo apt-get update
$ sudo apt-get install gnuplot
```

CentOS / RedHat

```
$ sudo yum check-update
$ sudo yum install gnuplot
```

Fedora

```
$ sudo dnf check-update
$ sudo dns install gnuplot
```

Mac OSX

Using Homebrew

```
$ brew install gnuplot
```

Using MacPorts

```
$ sudo port install gnuplot
```

Test the installation

After installing gnuplot it's a good idea to run a simple example to ensure all is working fine.

1. Open your terminal
2. Type `gnuplot`.

3. Your prompt should now change to `gnuplot>`
4. Type: `plot sin(x)`

If all is well you should see now a sin(x) graphic generated by gnuplot.

Note: if you are on Windows and have not added `gnuplot` to your `PATH` you' ll need to navigate to the `<gnuplot_install_path>\bin` folder. The default location is: `C:\Program Files (x86)\gnuplot\bin`

Basic introduction to programming language's rules

From the gnuplot 5.0 official online documentation:

> The command language of gnuplot is **case sensitive**, i.e. commands and function names written in *lowercase* are not the same as those written in *capitals*. All command names may be abbreviated as long as the abbreviation is not ambiguous. Any number of commands may appear on a line, separated by semicolons `;`. (T. Williams, C. Kelley - *gnuplot 5.0, An Interactive Plotting Program*)

Some examples of these basic rules are

1. A case sensitive language

Typing *lowercase*-defined commands in *uppercase* will generate an `invalid command` warning.

```
gnuplot> set xlabel "x"
gnuplot> Set xlabel "x"
         ^
         invalid command
```

Also the `N` variable will be different from the `n` one.

2. Abbreviations

You can find an almost complete list of abbreviations here. Anyway the first three letters of any command in *gnuplot* work always as abbreviations. Some commands allows also a more powerful contraction. A little example is given below.

```
gnuplot> p sin(x)
gnuplot> rep
gnuplot> q
```

where `p` stands for `plot`, `rep` for `replot` and `q` for `quit`.

3. Separators

The symbol used to separate commands on a singe line is `;`

```
set title "My First Plot"; plot 'data'; print "all done!"
```

5. Comments

> Comments are supported as follows: a `#` may appear in most places in a line and gnuplot will ignore the rest of the line. It will not have this effect inside quotes, inside numbers (including complex numbers), inside command substitutions, etc. In short, it works anywhere it makes sense to work. (*Ibidem*)

Just remember the simple *"anywhere it makes sense to work"* rule.

```
gnuplot> # this is a comment, nothing will happen
gnuplot> plot sin(x) # another valid comment
gnuplot> plot sin(#x)
                     ^
         invalid expression
```

4. Extending commands

> Commands may extend over several input lines by ending each line but the last with a backslash (`\`). The backslash must be the last character on each line. The effect is as if the backslash and newline were not there. That is, no white space is implied, nor is a comment terminated. Therefore, commenting out a continued line comments out the entire command. (*Ibidem*)

For example, to split `plot` command on multiple lines,

```
plot\
    sin(x),\
    cos(x)
```

will plot the same as

```
plot sin(x), cos(x)
```

A little note on *"commenting out a continued line comments out the entire command"*. If you type the command

```
plot\
    sin(x),\ # I would like to comment here
    cos(x)
```

an error will occur:

```
gnuplot> plot\
>        sin(x),\ # I would like to comment here
                  ^
         invalid character \
```

So it's better to be careful and respect the rule *"anywhere it makes sense to work"* while using `#` comments.

Read Getting started with Gnuplot online: https://riptutorial.com/gnuplot/topic/3284/getting-started-with-gnuplot

Chapter 2: 2D Plotting Styles

Examples

Selecting a plotting style

Explicite selection

A plotting style is usually selected using the `with` keyword, like

```
plot x with points
```

This allows to use different plotting styles for every `plot`:

```
plot x with points, 2*x with lines
```

Typing `help with` in the gnuplot command window gives a list of all available plotting styles.

Global plotting style selection

Plotting styles can also be set globally for all plot commands. Here, gnuplot distinguishes between function and data plots, for which different default styles can be set.

For functions use `set style function`:

```
set style function linespoints
plot x, 2*x
```

For data files use `set style data`:

```
set style data lines
plot 'file.dat', 'other-file.dat'
```

Note, that for functions the default style is `lines`, and for data files it is `points`. With `show style data` and `show style function` you can inspect the currently selected plotting styles.

Read 2D Plotting Styles online: https://riptutorial.com/gnuplot/topic/4302/2d-plotting-styles

Chapter 3: Basic plotting of data files

Introduction

One of the main useful features of *gnuplot* is the possibility of plotting **data files**. Plotting a data file is really simple with *gnuplot*, actually, once you have opened the software from the terminal, you only need to digit the command `plot 'file'` to get an automatic plot.

First of all, before plotting, you must be sure to be under the same directory where the data file is, otherwise you'll eventually get a `warning`.

Syntax

- plot *datafile* using *column_expression* with *style*

Examples

Plot a single data file

The default *gnuplot* command `plot` (also only `p`) plot dataset with columns, of the form of the **`data_set.dat`** file below.

```
# Prototype of a gnuplot data set
# data_set.dat
# X -    X^2 -    2*X -    Random
0        0        0        5
1        1        2        15
1.4142   2        2.8284   1
2        4        4        30
3        9        6        26.46
3.1415   9.8696   6.2832   39.11
4        16       8        20
4.5627   20.8182  9.1254   17
5.0      25.0     10.0     25.50
6        36       12       0.908
```

As you can see you can write in your data set in floating point notation. Now everything is ready to make the data plot: by typing only

```
plot "data_set.dat"
```

gnuplot will produce a graph in your `output` destination. The default settings will use the first two columns of your data file, respectively x and y. To specify the columns to be plotted use the **`using`** specifier

```
plot "data_set.dat" using 2:4
```

which means "plot the file using column 2 as X and column 4 as Y". In the case your data set is a tridimensional file just use `splot` ad add the z-column

```
splot "data_set.dat" using 1:2:3
```

There are also different style (see gnuplot documentation or Selecting a plotting style for further infos) for plotting points. As said before, the default style is `point`

```
plot "data_set.dat" using 1:4 with point
```

which will plot the same as if you do not type `with point`. An useful style for data plotting is `linespoint` which is, obviously, "lines + points". **E.G.:**

```
plot "data_set.dat" using 1:4 with linespoint
# the abbreviated form is completely equivalent:
# p "data_set.dat" u 1:4 w lp
```

40
35

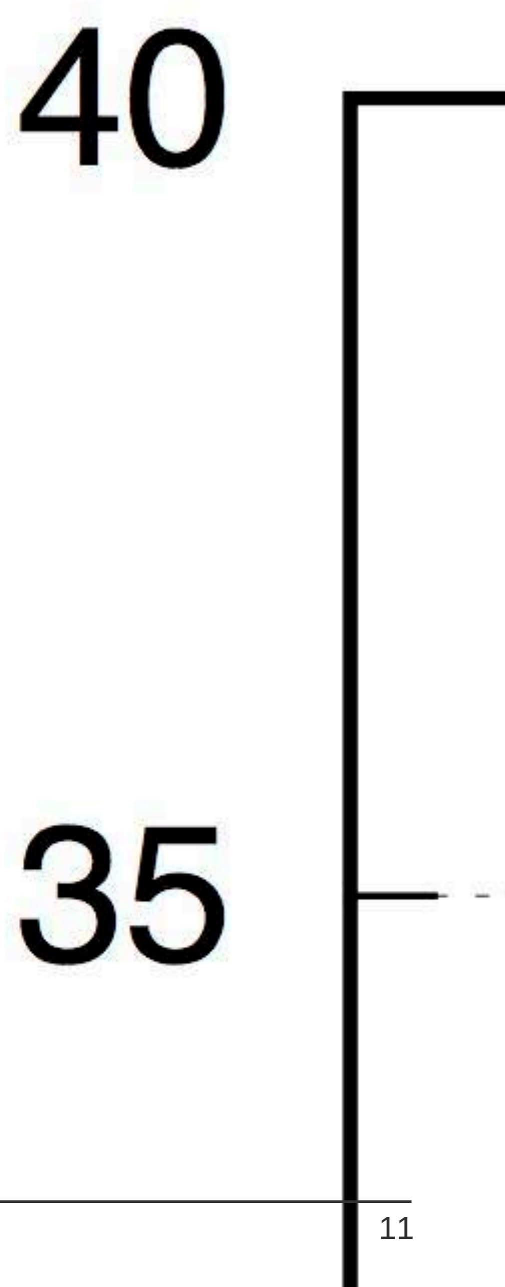
`plot` function any argument you prefer, by separating them with a `,`:

```
p "data_set.dat" u 1:2 w lp,\
    "data_set.dat" u 1:3 w lp,\
    "data_set.dat" u 1:4 w lp
```

Anyway sometimes there could be too much columns to write one by one. In these case the `for` iteration loop results very useful:

```
p for [col = 2:4] "data_set.dat" using 1:col w lp
```

which gives the output


40
35

, with a decided steps (if not specified = 1). For example `for [i = 0:6:2]` will increment `i` from 0 to 6 in 2 steps: `i = 0, 2, 4, 6`. All values (start, stop and increment) are casted to integer values.

***Grid**

The grid is often useful when plotting a data set. To add a grid type

```
set grid
```

Plotting multiple data files

First method - Concatenation of strings

The simplest method to plot multiple data files is to insert a `for` loop inside the `plot` command of gnuplot. Assuming you have `N` files named sequently, *i.e.*

```
file_1.dat
file_2.dat
file_3.dat
...
file_N.dat
```

Executing the command

```
plot for[i = 1:N] "file_".i.".dat"
```

will plot all the files between `file_1.dat` and `file_N.dat` in the same graph.

Example with three data files

Table of datasets

X-Axes	Y-Axe file_1.dat	Y-Axe file_2.dat	Y-Axe file_3.dat
1	1	1	1
2	2	4	2
3	3	9	6
4	4	16	24
5	5	25	120

Commands

```
set terminal postscript color noenhanced ##setting the term
set output "multiple_files.ps"
set key center ##legend placement

plot [1:5][1:120] \
    for [i = 1:3] "file_".i.".dat" \
    pointsize 1.3 linecolor i+4 \
    title "file\_".i.".dat" \
    with linespoint
```

The loop starts with `for [i = 1:3] "file_".i.".dat"` and execute the `plot` command until it reaches `i = 3`. The `.i.` is the concatenated number.

`title "file\_".i.".dat"` has been written with the `\` in order to make the _ symbol in the name of the files appears as an *underscore* rather than a *subscript*, and `noenhanced` specifier is fundamental to obtain this result.

The final result is shown below

120
100

as the C-language `sprintf`. The right syntax, from the gnuplot 5.1 documentation is

```
sprintf("format", x, y, ...)
```

A brief example will clarify every doubt.

```
file_name(n) = sprintf("file_%d.dat", n)
plot for[i = 1:N] file_name(i) title file_name(i)
```

Read Basic plotting of data files online: https://riptutorial.com/gnuplot/topic/3591/basic-plotting-of-data-files

Chapter 4: Fit data with gnuplot

Introduction

The fit command can fit a user-defined function to a set of data points `(x,y)` or `(x,y,z)`, using an implementation of the nonlinear least-squares (**NLLS**) Marquardt-Levenberg algorithm.

Any user-defined variable occurring in the function body may serve as a fit parameter, but the return type of the function must be real.

Syntax

- ***fit*** [*xrange*][*yrange*] *function* "*datafile*" ***using*** *modifier* ***via*** *parameter_file*

Parameters

Parameters	Detail
Fitting parameters `a`, `b`, `c` and any letter that had not been used previously	Use letters to represent parameters that will be used to fit a function. E.g.: `f(x) = a * exp(b * x) + c`, `g(x,y) = a*x**2 + b*y**2 + c*x*y`
File parameters `start.par`	Instead using uninitialised parameters (the Marquardt-Levenberg will automatically initialise for you `a=b=c=...=1`) you can put them in a file `start.par` and them call with in the *parameter_file* section. E.g.: `fit f(x) 'data.dat' u 1:2 via 'start.par'`. An example for the `start.par` file is shown below

Remarks

Short introduction

`fit` is used to find a set of parameters that ’best’ fits your data to your user-defined function. The fit is judged on the basis of the sum of the squared differences or ’residuals’ (SSR) between the input data points and the function values, evaluated at the same places. This quantity is often called ’chisquare’ (i.e., the Greek letter chi, to the power of 2). The algorithm attempts to minimize SSR, or more precisely, WSSR, as the residuals are ’weighted’ by the input data errors (or 1.0) before being squared. (*Ibidem*)

The `fit.log` file

After each iteration step a detailed info is given about the fit's state both on the screen and on a so-called log-file `fit.log`. This file will never be erased but always appended so that the fit's history isn't lost.

Examples

Fitting data with errors

> There can be up to 12 independent variables, there is always 1 dependent variable, and any number of parameters can be fitted. Optionally, error estimates can be input for weighting the data points. (T. Williams, C. Kelley - *gnuplot 5.0, An Interactive Plotting Program*)

If you have a data set and want to fit if the command is very simple and natural:

```
fit f(x) "data_set.dat" using 1:2 via par1, par2, par3
```

where instead `f(x)` could be also `f(x, y)`. In the case you also have data error estimates just add the `{y | xy | z}errors` (`{ | }` represent the possible choices) in the *modifier* option (see **Syntax)**. For example

```
fit f(x) "data_set.dat" using 1:2:3 yerrors via par1, par2, par3
```

where the `{y | xy | z}errors` option require respectively 1 (`y`), 2 (`xy`), 1 (`z`) column that specify the value of the error estimate.

Exponential fitting with `xyerrors` of a file

> Data error estimates are used to calculate the relative weight of each data point when determining the weighted sum of squared residuals, WSSR or chisquare. They can affect the parameter estimates, since they determine how much influence the deviation of each data point from the fitted function has on the final values. Some of the fit output information, including the parameter error estimates, is more meaningful if accurate data error estimates have been provided.. (*Ibidem*)

We'll take a sample data set **`measured.dat`**, made up by 4 columns: the x-axis coordinates (`Temperature (K)`), the y-axis coordinates (`Pressure (kPa)`), the x-error estimates (`T_err (K)`) and the y-error estimates (`P_err (kPa)`).

```
#### 'measured.dat' ####
### Dependence of boiling water from Temperature and Pressure
##Temperature (K) - Pressure (kPa) - T_err (K) - P_err (kPa)

368.5     73.332        0.66    1.5
364.2     62.668        0.66    1.0
359.2     52.004        0.66    0.8
354.5     44.006        0.66    0.7
348.7     34.675        0.66    1.2
343.7     28.010        0.66    1.6
```

```
338.7     22.678        0.66    1.2
334.2     17.346        0.66    1.5
329.0     14.680        0.66    1.6
324.0     10.681        0.66    1.2
319.1     8.015         0.66    0.8
314.6     6.682         0.66    1.0
308.7     5.349         0.66    1.5
```

Now, just compose the prototype of the function that from the theory should approximate our datas. In this case:

```
Z = 0.001
f(x) = W * exp(x * Z)
```

where we have initialised the parameter `Z` because otherwise evaluating the exponential function `exp(x * Z)` results in huge values, which leads to (floating point) Infinity and NaN in the Marquardt-Levenberg fitting algorithm, usually you would not need to initialise the variables - have a look here , if you want to know more about Marquardt-Levenberg.

It is time to fit the data!

```
fit f(x) "measured.dat" u 1:2:3:4 xyerrors via W, Z
```

The result will look like

```
After 360 iterations the fit converged.
final sum of squares of residuals : 10.4163
rel. change during last iteration : -5.83931e-07

degrees of freedom    (FIT_NDF)                        : 11
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 0.973105
variance of residuals (reduced chisquare) = WSSR/ndf   : 0.946933
p-value of the Chisq distribution (FIT_P)              : 0.493377

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
W               = 1.13381e-05      +/- 4.249e-06    (37.47%)
Z               = 0.0426853        +/- 0.001047     (2.453%)

correlation matrix of the fit parameters:
                W      Z
W               1.000
Z              -0.999  1.000
```

Where now `W` and `Z` are filled with the desired parameters and errors estimates on those one.

The code below produce the following graph.

```
set term pos col
set out 'PvsT.ps'

set grid
set key center
set xlabel 'T (K)'
set ylabel 'P (kPa)'


Z = 0.001
f(x) = W * exp(x * Z)
fit f(x) "measured.dat" u 1:2:3:4 xyerrors via W, Z

p [305:] 'measured.dat' u 1:2:3:4 ps 1.3 pt 2 t 'Data' w xyerrorbars,\
f(x) t 'Fit'
```

Plot with fit of `measured.dat` Using the command `with xyerrorbars` will display errors estimates on the x and on the y. `set grid` will place a dashed grid on the major tics.

80

70

, which includes the square meters of a house in a certain city and its price in $1000.

```
### 'house_price.dat'
## X-Axis: House price (in $1000) - Y-Axis: Square meters (m^2)

245    426.72
312    601.68
279    518.16
308    571.50
199    335.28
219    472.44
405    716.28
324    546.76
319    534.34
255    518.16
```

Let's fit those parameters with *gnuplot* The command itself is very simple, as you can notice from the syntax, just define your fitting prototype, and then use the `fit` command to get the result:

```
## m, q will be our fitting parameters
f(x) = m * x + q
fit f(x) 'data_set.dat' using 1:2 via m, q
```

But it could be interesting also using the obtained parameters in the plot itself. The code below will fit the **`house_price.dat`** file and then plot the `m` and `q` parameters to obtain the best curve approximation of the data set. Once you have the parameters you can calculate the `y-value`, in this case the *House price*, from any given `x-vaule` (*Square meters* of the house) just substituting in the formula

```
y = m * x + q
```

the appropriate `x-value`. Let's comment the code.

0. Setting the term

```
set term pos col
set out 'house_price_fit.ps'
```

1. Ordinary administration to embellish graph

```
set title 'Linear Regression Example Scatterplot'
set ylabel 'House price (k$ = $1000)'
set xlabel 'Square meters (m^2)'
set style line 1 ps 1.5 pt 7 lc 'red'
set style line 2 lw 1.5 lc 'blue'

set grid
set key bottom center box height 1.4

set xrange [0:450]
set yrange [0:]
```

2. The proper fit

For this, we will only need to type the commands:

```
f(x) = m * x + q
fit f(x) 'house_price.dat' via m, q
```

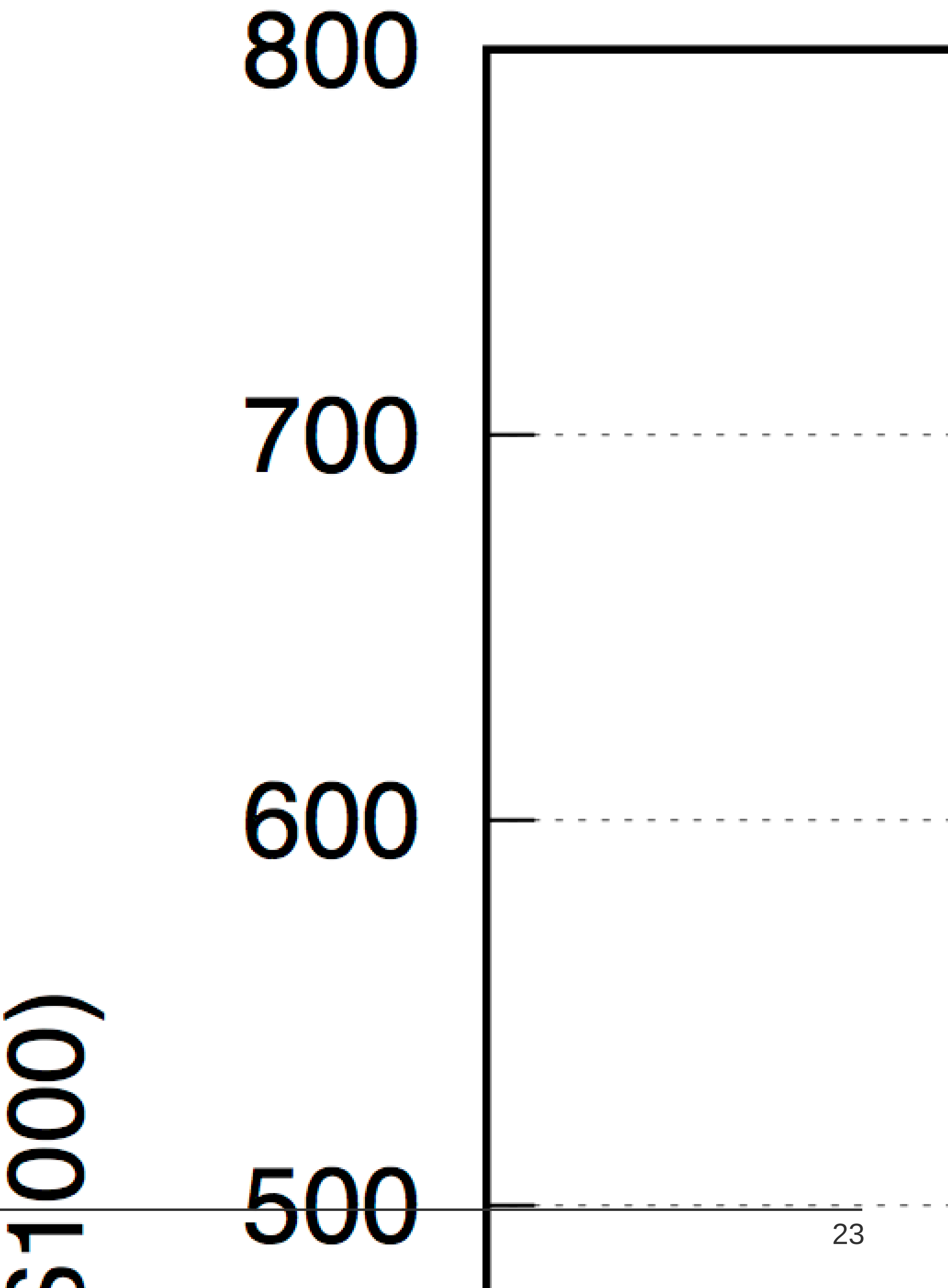
3. Saving `m` and `q` values in a string and plotting

Here we use the `sprintf` function to prepare the label (boxed in the `object rectangle`) in which we are going to print the result of the fit. Finally we plot the entire graph.

```
mq_value = sprintf("Parameters values\nm = %f k$/m^2\nq = %f k$", m, q)
set object 1 rect from 90,725 to 200, 650 fc rgb "white"
set label 1 at 100,700 mq_value


p 'house_price.dat' ls 1 t 'House price', f(x) ls 2 t 'Linear regression'
set out
```

The output will look like this.


800
700
600
500
($1000)

Chapter 5: Using script files

Syntax

1. gnuplot -c scriptfile ARG1 ARG2 ...

Remarks

Basic usage can be displayed by typing `gnuplot -h`

```
$ gnuplot -h
Usage: gnuplot [OPTION] ... [FILE]
  -V, --version
  -h, --help
  -p  --persist
  -d  --default-settings
  -c  scriptfile ARG1 ARG2 ...
  -e  "command1; command2; ..."
gnuplot 5.0 patchlevel 3
```

Examples

Simple script file

Gnuplot is able to generate a graphic from a script file which allows for a sequence of commands necessary to draw a graphic to be executed in sequence instead of type in manually.

For the purpose of this example we'll create a simple script to draw a `sin(x)`.

Create a script file

Create a file `sinx.p` with the following contents:

```
# Set the output to a png file
set terminal png size 500,500
# The file we'll write to
set output 'sinx.png'
# The graphic title
set title 'Sin(x)'
#plot the graphic
plot sin(x)
```

In the example above you find the most common commands, however, there are several other commands to be explored such as `set xlabel`, `set ylabel`, etc.

You may customize the `set output` line with the path you would like the file to generate the file.

Run the script

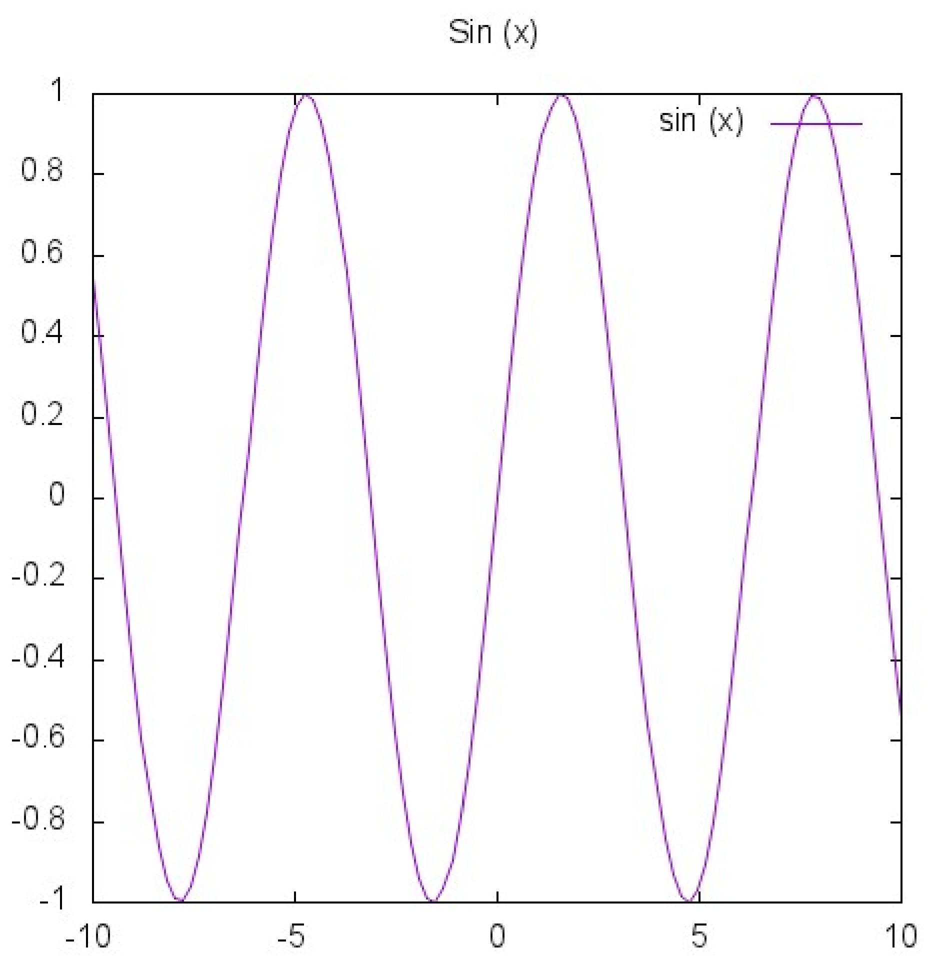
Open your terminal and type:

```
gnuplot path/to/sinx.p
```

In case your current folder contains the script you may enter the following instead:

```
gnuplot sinx.p
```

The script will run and generate the PNG file at the specified location. The resulting graphic should look like the following:



Read Using script files online: https://riptutorial.com/gnuplot/topic/4013/using-script-files

Credits

S. No	Chapters	Contributors
1	Getting started with Gnuplot	Community, Fawix, opisthofulax
2	2D Plotting Styles	Christoph
3	Basic plotting of data files	Christoph, Matthew, opisthofulax, Tom Solid
4	Fit data with gnuplot	opisthofulax
5	Using script files	Christoph, Fawix