Ruby/Gnuplot Manual

Gordon James Miller (gmiller@bittwiddlers.com)

March 18, 2003

Contents

1 Overview

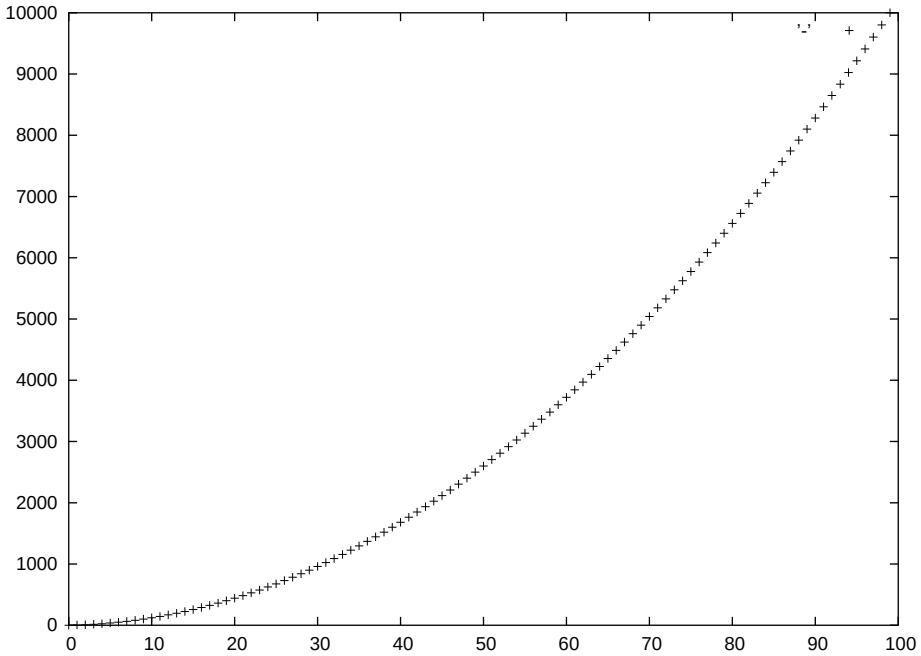
Ruby/Gnuplot (RGnuplot) is a wrapper that provides the ability to control a gnuplot process from within Ruby. Gnuplot is normally controlled by scripting directives in its own language. RGnuplot provides an object-oriented interface to allow a user to generate plots using Ruby objects.

This manual does not intend to replace Gnuplot's documentation by any means. Instead, the intent of the document is to allow users familiar with gnuplot to translate their knowledge into Ruby. Users that do not know Gnuplot will still benet from this, but I really encourage individuals to learn the tool.

2 QuickStart

2.1 One DataSet in 2D

```
require 'gplot/Gnuplot'

plot = Gnuplot::Plot.new
plot.terminal 'postscript eps'
plot.output   'demo1.eps'
```

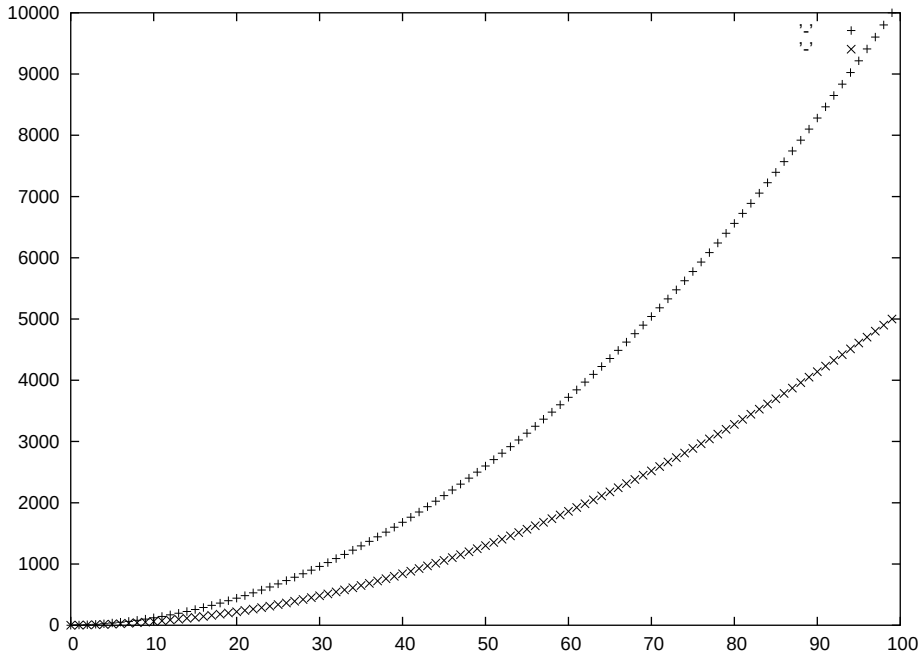



```
x = (1..100).to_a
y = x.collect { |i| i*i }
ds = y.gpds

plot.draw ds
```

The routine creates a Plot object that will output an encapsulated postscript le to the le 'demo1.eps' then creates the dataset object with the data that is going to be plotted. In RGnuplot there are two types of plots, 2 dimensional and 3 dimensional, implemented by the Plot and SPlot classes. The names of these classes correspond to the gnuplot commands to do the same type of plot.

Plot and SPlot objects control the layout and formatting of the overall graphic being produced. Things like plot titles and x, y, and z axis information are controlled by these objects. When instantiated with no arguments the Plot object will spawn a gnuplot subprocess which is then written to to generate the plots. If the Plot object is created with a lename then the gnuplot commands will go to a le with that name instead of a gnuplot process. This will cause the plot to not be created at that moment but the generated script is capable of being loaded by the gnuplot interpreter and is useful for debugging the output of RGnuplot.

DataSet objects control the layout, formatting, and data of each set of data that is displayed in a Plot. There are numerous DataSet classes that apply to



dierent classes that are capable of providing data to Gnuplot: Array, Matrix, NArray, and String. RGnuplot adds the gpds method to the class that is capable of providing data. Each of these will be discussed later in this document.

DataSet objects are then added to the Plot object and the draw method on the Plot object is invoked. As a convenience, the draw method can also take a variable number of dataset objects.

In this plot there are no annotations added to either the plot or to its dataset so the legend and labelling is gnuplot defaults.

2.2 Multiple DataSets in 2D

Adding to the previous example, the following code displays multiple datasets but continues to not add any annotations.

```
   require 'gplot/Gnuplot'

plot = Gnuplot::Plot.new
plot.terminal 'postscript eps'
plot.output   'demo2.eps'

x = (1..100).to_a
y1 = x.collect { |i| i*i }
ds1 = y1.gpds
```

```
y2 = x.collect { |i| i*i / 2.0 }
ds2 = y2.gpds

plot.draw ds1, ds2
```

This example follows the same general outline as the rst gure. The Plot object is created, Plot characteristics are set, then DataSets are created and added to the plot.

2.3 Plotting functions

```
  require 'gplot/Gnuplot'

plot = Gnuplot::Plot.new
plot.terminal 'postscript eps'
plot.output   'demo3.eps'

x = (1..100).to_a
ds1 = "4 * x".gpds

y2 = x.collect { |i| i*i / 2.0 }
ds2 = "x**2".gpds

plot.draw ds1, ds2
```

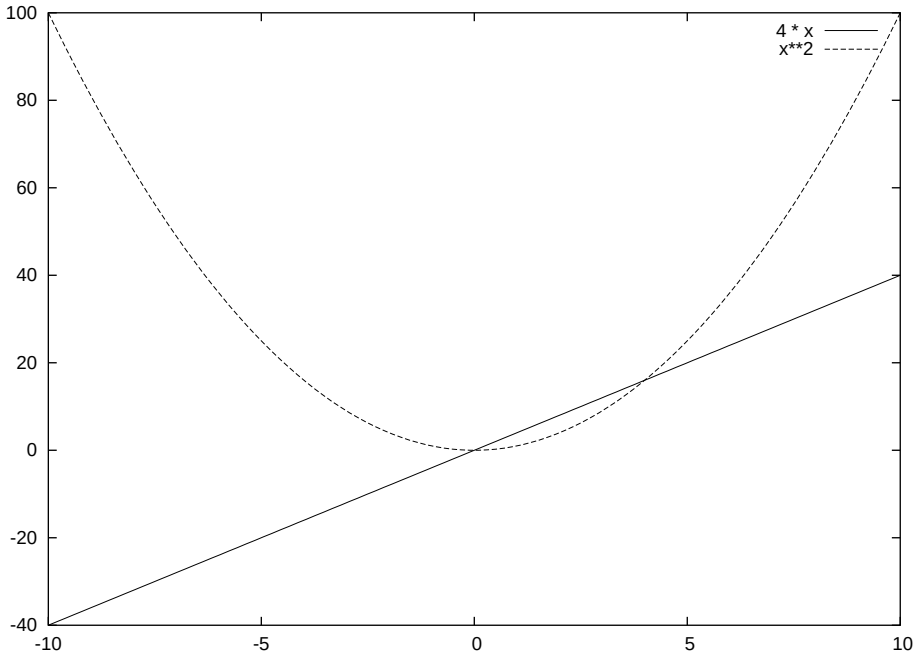
In this example, we rely on gnuplot's ability to specify a function as a string and plot it. In contrast to the previous examples, these datasets are created by invoking the gpds function on a string specifying a function of x. Gnuplot can intepret signicant functions including the the trig and hyperbolic functions as well as many others.

2.4 Annotating Data Sets

Until now we've not done anything except add data to plots. In order to make an acceptable plot we need to add annotations to the plot. Annotations are either added to the Plot or to a DataSet depending on what is being annotated. In gnuplot speak, anything that is passed as an argument to the 'plot' or 'splot' command is a DataSet attribute, everything else is a Plot attribute.

There are two dierent ways to add attributes to DataSets. The rst mechanism is to pass a hash to the gpds with the attribute name as strings. The second is to invoke the method with the same name as the attribute name on the DataSet.

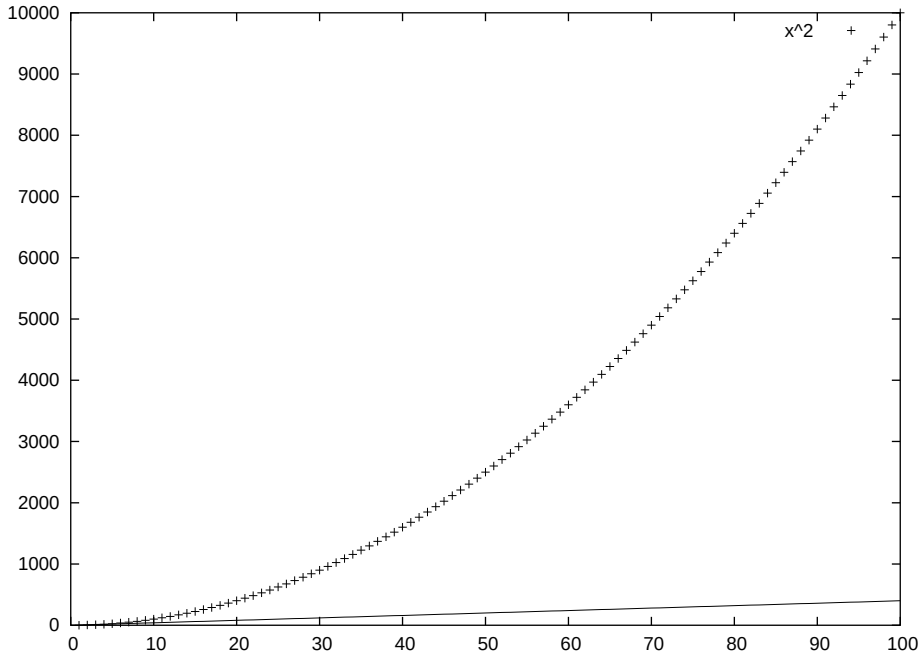
```
  require 'gplot/Gnuplot'
```




```
plot = Gnuplot::Plot.new
plot.terminal 'postscript eps'
plot.output   'demo4.eps'

x = (1..100).to_a
y = x.collect { |i| 4 * i }
ds1 = y.gpds
ds1.xgrid( x )
ds1.title("notitle")
ds1.with( "lines" )

y2 = x.collect { |i| i*i / 2.0 }
ds2 = "x**2".gpds( {"with" => "points", "title" => "x^2"} )

plot.draw ds1, ds2
```

In Figure 4 both methods are used. DataSet 1, ds1, is congured using methods while DataSet 2, ds2, is congured by passing the options in a hash to the initialize method. There is no advantage to using one over the other save a religious preference on the part of the coder.

The list below shows the most commonly used DataSet properties



with Selects the style that will be used to display the data. The argument to this should be one of either 'lines', 'points', 'linespoints', 'impulses', or any other acceptable string for 'with'. Type 'help plot with' within gnuplot to get help on this argument.

xgrid Set the values that should be used for the x axis. The argument to this should be an array of values for the x axis and should be the same length as the data. This is not the same as setting a range but simply associates an x value for every y value.

title Set the title for this particular dataset. Set to the string 'notitle' to turn o the label for this dataset.

2.5 Annotating Plots

2.6 Changing output format and writing to les