Sesión de prácticas nº 1: Eficiencia

Sobre gnuplot:

gnuplot es un programa interactivo que nos permite mostrar gráficamente funciones y ficheros de datos. Esta utilidad nos permitirá visualizar cómo los ficheros de datos se ajustan a las curvas teóricas. Además, también nos servirá para estimar las constantes correspondientes.

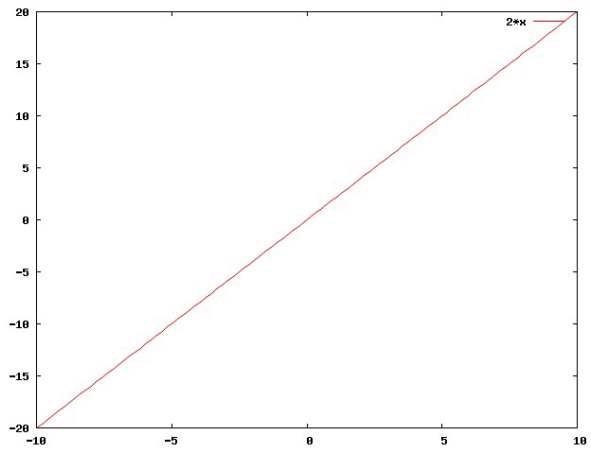
Para ejecutarlo tendremos que llamarlo desde la línea de órdenes mediante "gnuplot". Entraremos en la consola del programa gnuplot (en la pantalla aparecerán (entre otras) las líneas

```
Terminal type set to 'x11'
gnuplot>
```

Éste es el prompt característico de gnuplot, que nos indica que está esperando a que le demos alguna orden.

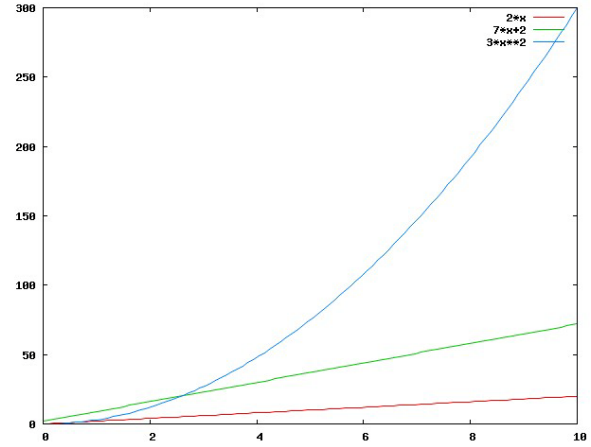
Por ejemplo, para dibujar la gráfica de la función y = 2x debemos escribir la orden

```
gnuplot> plot 2*x
```



También podemos dibujar varias funciones y= 2x, y= 7x+2 e $y=3y^2$ utilizando

```
gnuplot> plot [0:10] 2*x, 7*x+2, 3*x**2
```

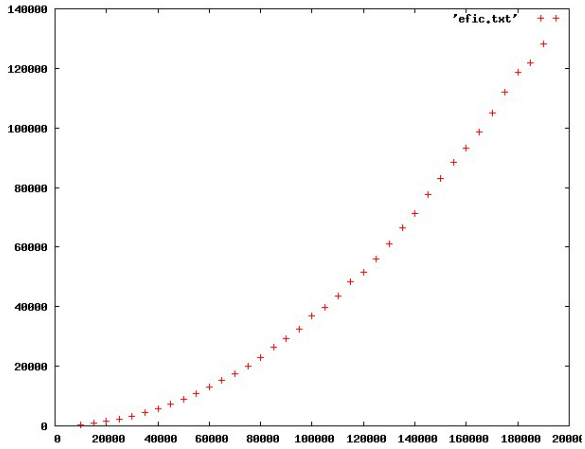


El primer ejercicio que vamos a plantear pretende que nos familiaricemos con gnuplot. Para ello, analizaremos un ejemplo muy sencillo, disponible en el programa efic.cpp. En este programa

implementamos el algoritmo de búsqueda lineal con centinela y forzamos la búsqueda de un elemento inexistente. Esto supondrá un tiempo de ejecución de O(n). Además, realizamos n llamadas a esta función para conseguir un tiempo de ejecución que sea $O(n^2)$, siendo n el tamaño del problema.

En el fichero efic.txt incluimos los tiempos de ejecución del algoritmo para diferentes tamaños del problema. Para visualizar la curva experimental de los datos obtenidos tendremos que usar la instrucción plot:

```
gnuplot> plot "efic.txt"
```



Podemos ver "a ojo" que efectivamente, la curva que forman los tiempos obtenidos parece un $O(n^2)$, pero esto no es suficiente. Vamos a dibujar tanto la curva teórica como la empírica para compararlas.

Lo que nos proponemos hacer es buscar los coeficientes del polinomio $y=ax^2+bx+c$ que mejor ajustan a los datos, para lo que vamos a hacer uso de la orden fit de gnuplot. Esta función realiza un ajuste por mínimos cuadrados de la función a los datos, proporcionándonos los valores de a, b y c.

```
gnuplot> f(x) = a*x**2 + b*x + c

gnuplot> fit f(x) 'efic.txt' via a,b,c
```

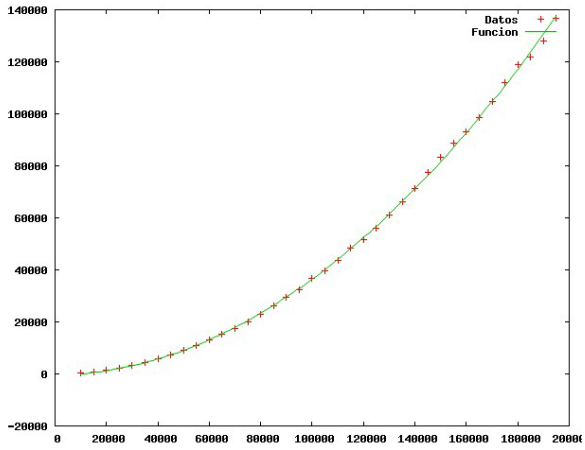
La orden fit implementa un algoritmo iterativo cuyos resultados aparecen en la consola de gnuplot. Podemos ver que los valores estimados de los parámetros son:

```
Final set of parameters            Asymptotic Standard Error
=======================            ==========================

a               = 3.55259e-006     +/- 4.877e-008   (1.373%)
b               = 0.0143477        +/- 0.01028      (71.64%)
c               = -505.107         +/- 459.2        (90.92%)
```

Ahora ya podemos dibujar las dos curvas, teórica y empírica:

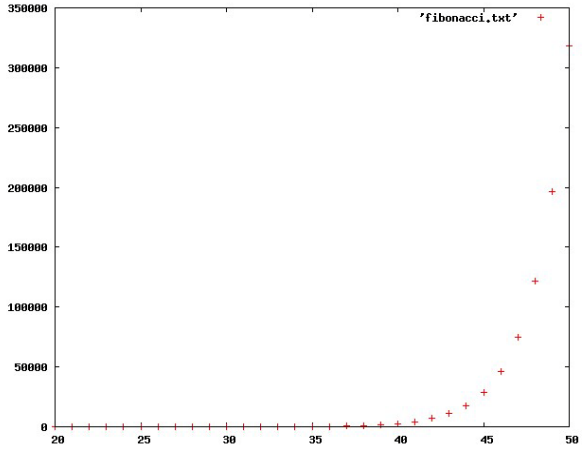
```
gnuplot> plot 'efic.txt' with p title 'Datos', f(x) with l title 'Funcion'
```



También proporcionamos otros algoritmos, como el de Fibonacci (versión recursiva), o el de las Torres de Hanoi, así como los ficheros con los datos empíricos obtenidos. Recordemos que Fibonacci pertenece al orden de $O\left(\left(\frac{1+\sqrt{5}}{2}\right)^n\right)$ y Hanoi pertenece al orden de $O(2^n)$.

Se han creado dos ficheros de datos que incluyen los tiempos de ejecución de ambos algoritmos: fibonacci.txt y hanoi.txt. Nos centraremos en fib.txt. Para visualizar este fichero tendremos que introducir la orden

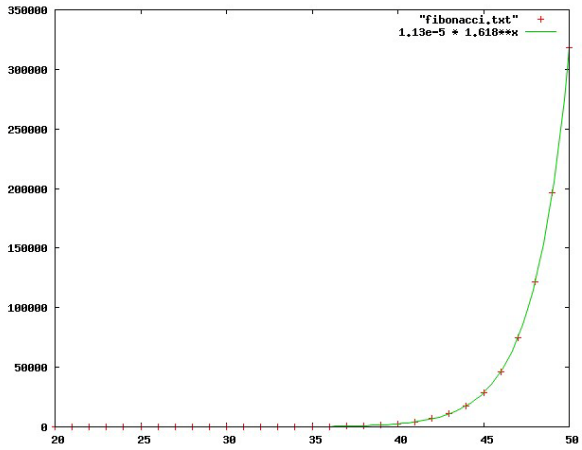
```
gnuplot> plot "fibonacci.txt"
```



El siguiente problema es calcular cuál es la constante oculta para el algoritmo de Fibonacci. Veamos cómo se puede realizar. Sabemos que Fibonacci pertenece al orden de $O\left(\left(\frac{1+\sqrt{5}}{2}\right)^x\right)$, o lo que es lo mismo $O(0{,}168^x)$ (aquí x representa el tamaño del problema). Para calcular la constante nos centramos en la ultima línea del fichero fib.txt y tenemos que t(50) = 317991 y considerando la expresión del orden tenemos que $t(50) <= c\ 1.618^{50}$. Despejamos c, obteniendo que $c = 1.13 \times 10^{-5}$.

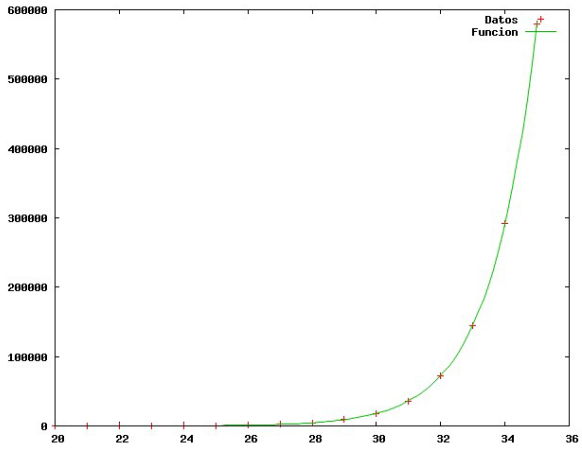
Ahora podemos pintar tanto la curva teórica como la empírica de Fibonacci.

```
gnuplot> plot "fibonacci.txt", 1.13e-5 * 1.618**x
```



De la misma forma, para Hanoi tenemos que t(35) = 580164, y considerando la expresión del orden tenemos que t(35) = c 2^{35}. Despejando, obtendremos que la constante oculta será c = 1.69×10^{-5}.

```
gnuplot> plot "hanoi.txt" with p title 'Datos', 1.69e-5*2**x with l title 'Funcion'
```



Para guardar las gráficas generadas en un fichero debemos indicar el tipo de formato de salida con el que lo queremos guardar (postscript, jpeg, png, etc.) y el nombre del fichero donde se almacenará. Para ello, haremos uso de las órdenes set terminal y set output. Por ejemplo, si queremos guardar la gráfica en el fichero 'grafica.jpg' (en formato .jpg):

```
gnuplot> set terminal jpeg
gnuplot> set output 'grafica.jpg'
gnuplot> "hanoi.txt" with p title 'Datos', 1.69e-5*2**x with l title 'Funcion'
```

Para volver a tener la salida en pantalla podemos hacer

```
gnuplot> set terminal x11
gnuplot> set output
```

Podemos averiguar el conjunto de formatos de salida mediante

```
gnuplot> set terminal
```

Y obtener ayuda sobre uno específico mediante

```
gnuplot> help set terminal jpeg
```

PLOT COMMAND
============

 `plot` is the primary command for drawing plots with `gnuplot`. It creates plots of functions and data in many, many ways. `plot` is used to draw 2-d functions and data; `splot` draws 2-d projections of 3-d surfaces and data. `plot` and `splot` contain many common features; see `splot` for differences. Note specifically that `splot`'s `binary` and `matrix` options do not exist for `plot`, and `plot`'s `axes` option does not exist for `splot`.

 Syntax:
```
       plot {<ranges>}
            {<function> | {"<datafile>" {datafile-modifiers}}}
            {axes <axes>} {<title-spec>} {with <style>}
            {, {definitions,} <function> ...}
```

where either a <function> or the name of a data file enclosed in quotes is supplied. A function is a mathematical expression or a pair of mathematical expressions in parametric mode. The expressions may be defined completely or in part earlier in the stream of `gnuplot` commands (see `user-defined`).

It is also possible to define functions and parameters on the `plot` command itself. This is done merely by isolating them from other items with commas.

There are four possible sets of axes available; the keyword <axes> is used to select the axes for which a particular line should be scaled. `x1y1` refers to the axes on the bottom and left; `x2y2` to those on the top and right; `x1y2` to those on the bottom and right; and `x2y1` to those on the top and left. Ranges specified on the `plot` command apply only to the first set of axes (bottom left).

 Examples:
```
       plot sin(x)
       plot f(x) = sin(x*a), a = .2, f(x), a = .4, f(x)
       plot [t=1:10] [-pi:pi*2] tan(t), \
            "data.1" using (tan($2)):($3/$4) smooth csplines \
                     axes x1y2 notitle with lines 5
```

 See also `show plot`.

Subtopics available for plot:

acsplines	bezier	csplines	datafile
errorbars	errorlines	every	example
frequency	index	parametric	ranges
sbezier	smooth	special-filenames	style
thru	title	unique	using
with			

WITH
=====

Functions and data may be displayed in one of a large number of styles. The `with` keyword provides the means of selection.

Syntax:

```
with <style> { {linestyle | ls <line_style>}
               | {{linetype  | lt <line_type>}
                  {linewidth | lw <line_width>}
                  {pointtype | pt <point_type>}
                  {pointsize | ps <point_size>}
                  {fill | fs <fillstyle>}
                  {palette}}
             }
```

where <style> is either `lines`, `points`, `linespoints`, `impulses`, `dots`, `steps`, `fsteps`, `histeps`, `errorbars`, `xerrorbars`, `yerrorbars`, `xyerrorbars`, `errorlines`, `xerrorlines`, `yerrorlines`, `xyerrorlines`, `boxes`, `filledcurves`, `boxerrorbars`, `boxxyerrorbars`, `financebars`, `candlesticks`, `vectors` or `pm3d`. Some of these styles require additional information. See `plotting styles` for details of each style. `fill` is relevant only to certain 2D plots (currently `boxes` `boxxyerrorbars` and `candlesticks`). Note that `filledcurves` and `pm3d` can take an additional option not listed above (the latter only when used in the `splot` command)---see their help or examples below for more details.

Default styles are chosen with the `set style function` and `set style data` commands.

By default, each function and data file will use a different line type and point type, up to the maximum number of available types. All terminal drivers support at least six different point types, and re-use them, in order, if more are required. The LaTeX driver supplies an additional six point types (all variants of a circle), and thus will only repeat after 12 curves are plotted with points. The PostScript drivers (`postscript`) supplies a total of 64.

If you wish to choose the line or point type for a single plot, <line_type> and <point_type> may be specified. These are positive integer constants (or expressions) that specify the line type and point type to be used for the plot. Use `test` to display the types available for your terminal.

You may also scale the line width and point size for a plot by using <line_width> and <point_size>, which are specified relative to the default values for each terminal. The pointsize may also be altered globally---see `set pointsize` for details. But note that both <point_size> as set here and as set by `set pointsize` multiply the default point size---their effects are not cumulative. That is, `set pointsize 2; plot x w p ps 3` will use points three times default size, not six.

If you have defined specific line type/width and point type/size combinations with `set style line`, one of these may be selected by setting <line_style> to the index of the desired style.

If gnuplot was built with `pm3d` support, the special keyword `palette` is allowed for smooth color change of lines, points and dots in `splots`. The color is chosen from a smooth palette which was set previously with the command `set palette`. The color value corresponds to the z-value of the point coordinates or to the color coordinate if specified by the 4th parameter in `using`. The 2d `plot` command ignores this option.

The keywords may be abbreviated as indicated.

Note that the `linewidth`, `pointsize` and `palette` options are not supported by all terminals.

Examples:

This plots sin(x) with impulses:
```
plot sin(x) with impulses
```

This plots x with points, x**2 with the default:
```
plot x w points, x**2
```

This plots tan(x) with the default function style, file "data.1" with lines:
```
plot [ ] [-2:5] tan(x), 'data.1' with l
```

This plots "leastsq.dat" with impulses:
```
plot 'leastsq.dat' w i
```

This plots the data file "population" with boxes:
```
plot 'population' with boxes
```

This plots "exper.dat" with errorbars and lines connecting the points (errorbars require three or four columns):
```
plot 'exper.dat' w lines, 'exper.dat' notitle w errorbars
```

Another way to plot "exper.dat" with errorlines (errorbars require three or four columns):
```
plot 'exper.dat' w errorlines
```

This plots sin(x) and cos(x) with linespoints, using the same line type but different point types:
```
plot sin(x) with linesp lt 1 pt 3, cos(x) with linesp lt 1 pt 4
```

This plots file "data" with points of type 3 and twice usual size:
```
plot 'data' with points pointtype 3 pointsize 2
```

This plots two data sets with lines differing only by weight:
```
plot 'd1' t "good" w l lt 2 lw 3, 'd2' t "bad" w l lt 2 lw 1
```

This plots filled curve of x*x and a color stripe:
```
plot x*x with filledcurve closed, 40 with filledcurve y1=10
```

This plots x*x and a color box:
```
plot x*x, (x>=-5 && x<=5 ? 40 : 1/0) with filledcurve y1=10 lt 8
```

This plots a surface with color lines:
```
splot x*x-y*y with line palette
```

This plots two color surfaces at different altitudes:
```
splot x*x-y*y with pm3d, x*x+y*y with pm3d at t
```

See `set style` to change the default styles. See also styles demos.

FIT COMMAND
==========

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>'
    {datafile-modifiers}
    via '<parameter file>' | <var1>{,<var2>,...}
```

Ranges may be specified to temporarily limit the data which is to be fitted; any out-of-range data points are ignored. The syntax is

```
[{dummy_variable=}{<min>}{:<max>}],
```

analogous to `plot`; see `plot ranges`.

<function> is any valid `gnuplot` expression, although it is usual to use a previously user-defined function of the form f(x) or f(x,y).

<datafile> is treated as in the `plot` command. All the `plot datafile` modifiers (`using`, `every`,...) except `smooth` and the deprecated `thru` are applicable to `fit`. See `plot datafile`.

The default data formats for fitting functions with a single independent variable, y=f(x), are {x:}y or x:y:s; those formats can be changed with the datafile `using` qualifier. The third item (a column number or an expression), if present, is interpreted as the standard deviation of the corresponding y value and is used to compute a weight for the datum, 1/s**2. Otherwise, all data points are weighted equally, with a weight of one. Note that if you don't specify a `using` option at all, no y deviations are read from the datafile even if it does have a third column, so you'll always get unit weights.

To fit a function with two independent variables, z=f(x,y), the required format is `using` with four items, x:y:z:s. The complete format must be given---no default columns are assumed for a missing token. Weights for each data point are evaluated from 's' as above. If error estimates are not available, a constant value can be specified as a constant expression (see `plot datafile using`), e.g., `using 1:2:3:(1)`.

Multiple datasets may be simultaneously fit with functions of one independent variable by making y a 'pseudo-variable', e.g., the dataline number, and fitting as two independent variables. See `fit multi-branch`.

The `via` qualifier specifies which parameters are to be adjusted, either directly, or by referencing a parameter file.

Examples:

```
f(x) = a*x**2 + b*x + c
g(x,y) = a*x**2 + b*y**2 + c*x*y
FIT_LIMIT = 1e-6
fit f(x) 'measured.dat' via 'start.par'
fit f(x) 'measured.dat' using 3:($7-5) via 'start.par'
fit f(x) './data/trash.dat' using 1:2:3 via a, b, c
```

```
fit g(x,y) 'surface.dat' using 1:2:3:(1) via a, b, c
```

After each iteration step, detailed information about the current state of the fit is written to the display. The same information about the initial and final states is written to a log file, "fit.log". This file is always appended to, so as to not lose any previous fit history; it should be deleted or renamed as desired. By using the command `set fit logfile`, the name of the log file can be changed.

If gnuplot was built with this option, and you activated it using `set fit errorvariables`, the error for each fitted parameter will be stored in a variable named like the parameter, but with "_err" appended. Thus the errors can be used as input for further computations.

The fit may be interrupted by pressing Ctrl-C (any key but Ctrl-C under MSDOS and Atari Multitasking Systems). After the current iteration completes, you have the option to (1) stop the fit and accept the current parameter values, (2) continue the fit, (3) execute a `gnuplot` command as specified by the environment variable FIT_SCRIPT. The default for FIT_SCRIPT is `replot`, so if you had previously plotted both the data and the fitting function in one graph, you can display the current state of the fit.

Once `fit` has finished, the `update` command may be used to store final values in a file for subsequent use as a parameter file. See `update` for details.

Subtopics available for fit:

adjustable_parameters		beginners_guide	control
error	error_estimates	errors	guide
multi-branch	parameters	starting_values	tips