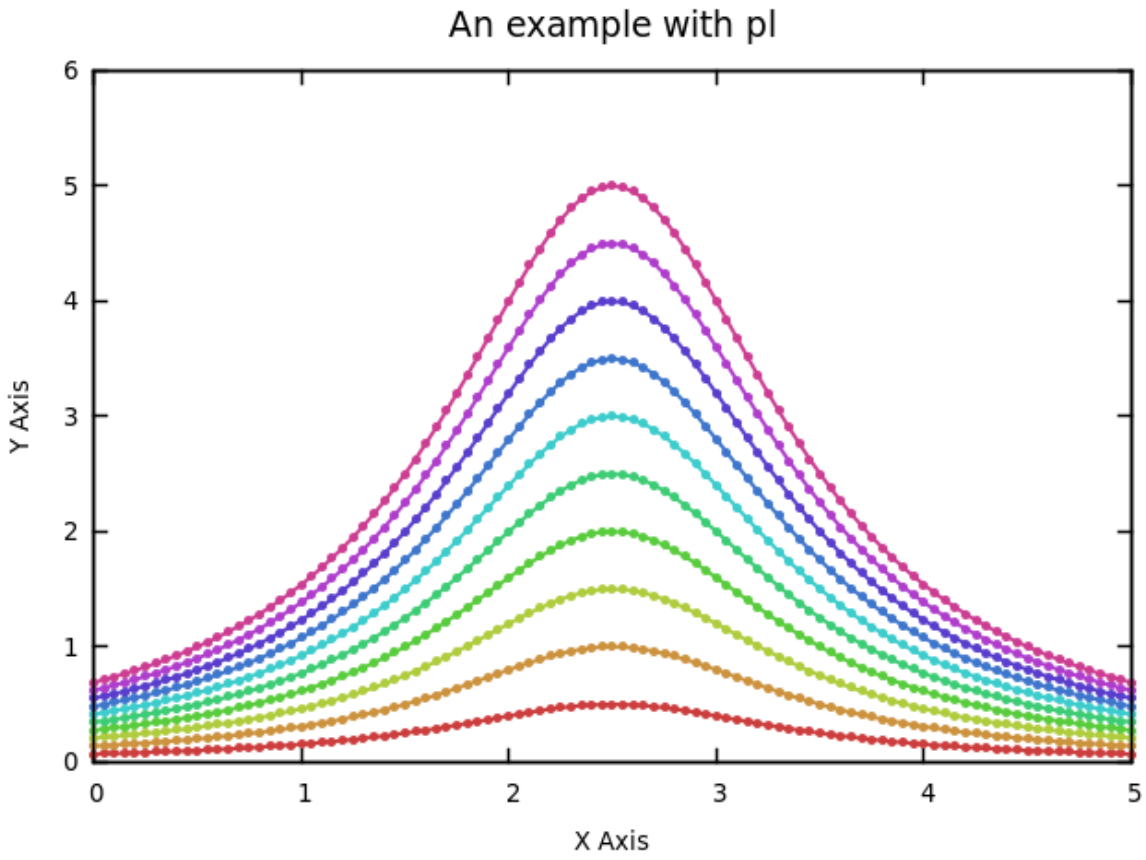
`pl`, un comando sencillo para la creación de gráficas

Práctica de *Lex*, Modelos de Computación



Andrés Herrera Poyatos

16 de diciembre de 2015

Índice

Códigos

1. Introducción

Una tarea habitual de los informáticos es crear gráficas para comparar resultados de diferentes algoritmos. Los algoritmos surgen de forma natural en cualquier área de la informática. Por tanto, rara es la asignatura de la carrera en la que no se pida implementar varios de ellos y realizar un estudio comparativo. En este ámbito, las gráficas son esenciales para sintetizar la información obtenida durante las ejecuciones.

Consecuentemente, hay que recurrir a programas para realizar estas gráficas. Aquellos similares a Excel son muy utilizados. En estos el proceso de creación de las gráficas es intuitivo y visual pero bastante lento. Cada vez que se obtienen nuevos datos hay que repetir el mismo procedimiento manual para generar la gráfica asociada.

Es más productivo un lenguaje de programación mediante el cual se programe cómo debe ser la gráfica. De esta forma se puede ejecutar el código automáticamente dentro de un script que trate los datos. Además, el código redactado es reutilizable. Varias opciones son:

- Lenguajes interpretados de propósito general, como Python, que suelen disponer de librerías extensas para la creación de gráficos. En el caso de Python cabe destacar `matplotlib` [1].
- El lenguaje de programación R, que está enfocado al análisis de datos [2]. R dispone de librerías con mucha funcionalidad para generar gráficos, como `ggplot2` [3].
- Gnuplot, que es un lenguaje específico para la creación de gráficos [4]. Gnuplot da mucha libertad al usuario para personalizar sus gráficas pero, como consecuencia, el aprendizaje es dificultoso. Es más, el código necesario para crear una gráfica puede tener fácilmente 50 líneas.

En cualquier caso, hay que aprender un nuevo lenguaje de programación y trabajar con código para ello. Además, cada vez que se quiera añadir nuevos datos a la gráfica hay que modificar bastante el código correspondiente.

El objetivo de esta práctica es proporcionar un comando para la terminal que permita crear gráficos de forma sencilla, intuitiva y automática y que utilice *Lex* para tratar la entrada [5]. Para ello se genera un script con código de Gnuplot a partir de las opciones que ha indicado el usuario en la entrada.

2. ¿Por qué utilizar *Lex*?

Se pretende que el comando sea lo más intuitivo posible. Por tanto, los argumentos o parámetros deben ser independientes de la posición en la que se escriban, como sucede en los comandos de bash. *Lex* permite conseguir esto mediante la búsqueda de expresiones regulares en la entrada. Cada posible argumento se representa mediante una expresión regular a la que se le asocia la acción correspondiente.

Mediante este método el procesamiento de la entrada es lineal. Además, el código asociado se simplifica gracias a la sintaxis de *Lex*. *Lex* también permite comprobar que los datos dados como argumentos cumplan el léxico que se desea para ellos. Consecuentemente, evita muchas comprobaciones tediosas en el código. En la Sección 3.3 se profundiza en las expresiones regulares utilizadas.

Lex habitualmente se utiliza para buscar en ficheros palabras aceptadas por expresiones regulares. Sin embargo, también se puede asignar como entrada una cadena de c mediante el comando [1]:

[1] ¿Cómo utilizar `yy_scan_string` en *Lex*?: stackoverflow.com/questions/1907847/how-to-use-yy-scan-string-in-lex

```
void yy_scan_string(const char* str);
```

Por tanto, se puede utilizar *Lex* para tratar los argumentos de un programa uniendo previamente estos en una nueva cadena. Este proceso es el que se realiza en `pl`. En esta práctica se propone el uso de *Lex* como alternativa a un tratamiento de argumentos en C++ puro.

3. ¿Cómo funciona `pl`?

En esta sección se explica el funcionamiento del comando. En primer lugar se muestra qué puede hacer `pl` mediante un ejemplo. Posteriormente se profundiza en la sintaxis de `pl` explicando las expresiones regulares asociadas en *Lex*.

3.1. ¿Qué se puede hacer con `pl`?

Para usar `pl` solo hay que ejecutar el comando sobre un archivo con los datos. La Figura 1 muestra la imagen obtenida por el Código 1.

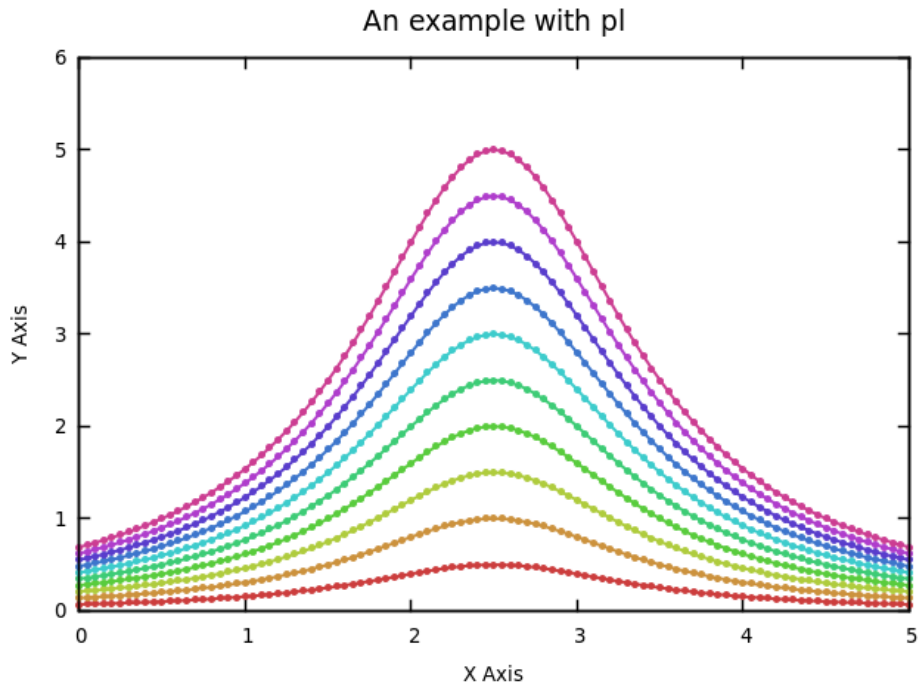


Figura 1: Una gráfica obtenida por `pl`.

```
pl -e # Genera el archivo example.csv
pl example.csv -t "An example with pl" -x "X Axis" -y "Y Axis"
```

Código 1: Ejemplo de uso de `pl`.

`pl` solo soporta gráficos en dos dimensiones en los cuales hay varios conjuntos de puntos distinguibles mediante colores. Estos son los gráficos más utilizados habitualmente en informática y el objetivo es que se puedan hacer lo más fácilmente posible.

Para realizar los gráficos `pl` genera un archivo `plot.sh` que contiene las sentencias de Gnuplot necesarias y que se ejecuta mediante bash.

3.2. Archivos aceptados por `pl`

`pl` puede utilizar múltiples archivos para una única gráfica. Los archivos aceptados por `pl` deben contener al menos dos columnas de números reales separadas por un delimitador. La primera columna se corresponde con las abscisas de los puntos a dibujar. Las siguientes columnas contienen una ordenada por cada abscisa, determinando un punto. Los puntos en una misma columna se dibujan con igual color. Por ejemplo, el archivo dado en el Código 2 representa 4 líneas horizontales. Se puede generar una gráfica a partir de este mediante el comando `pl lines1.csv`.

```
0,0,1,2,3
1,0,1,2,3
2,0,1,2,3
3,0,1,2,3
```

Código 2: Ejemplo de un archivo aceptado por `pl`.

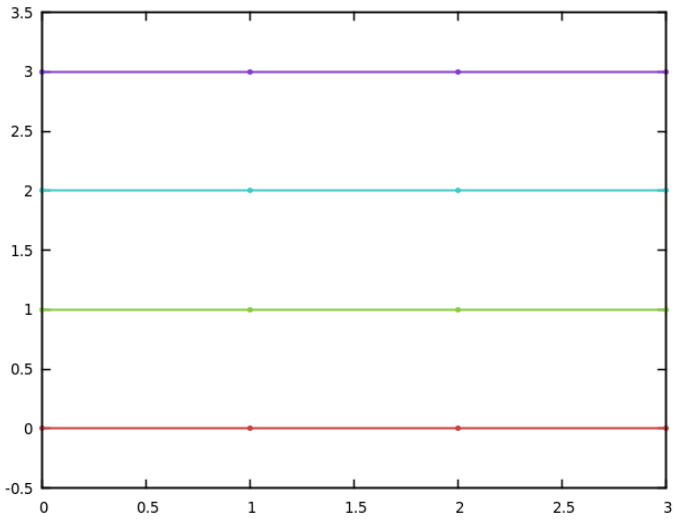


Figura 2: Gráfica asociada al fichero dado en el Código 2.

En el archivo del Código 2 se utiliza el estándar csv (una coma como separador). Sin embargo, se puede asignar un delimitador diferente mediante el argumento `-d <character>`. Este argumento responde a la siguiente expresión regular:

```
-d{space}{char}
```

donde space es [\t]* y char un carácter que puede estar delimitado o no por comillas.

Normalmente los datos a utilizar están almacenados en diferentes ficheros y los puntos de los distintos conjuntos de datos no tienen necesariamente las mismas abscisas. pl puede utilizar varios ficheros para una misma gráfica, con la única restricción de que tengan el mismo delimitador. Por ejemplo, el comando pl *.csv haría una gráfica con los datos de todos los archivos csv del directorio. Si estos fuesen los archivos dados por los Códigos 2 y 3, entonces el resultado es el mostrado por la Figura 3.

```
0,0.5,1.5,2.5,3.5
0.5,0.5,1.5,2.5,3.5
2.5,0.5,1.5,2.5,3.5
3,0.5,1.5,2.5,3.5
```

Código 3: Segundo ejemplo de un archivo aceptado por pl.

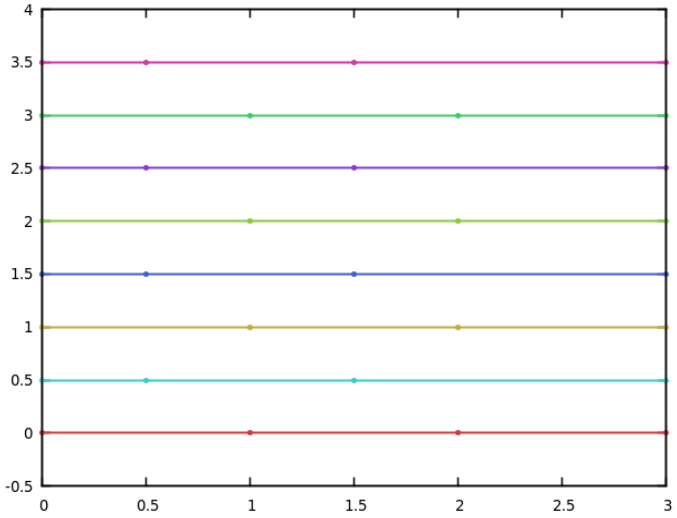


Figura 3: Gráfica asociada los ficheros dados en los Códigos 2 y 3.

pl detecta los nombres de archivos como palabras sin espacios que contienen al menos un punto, a partir del cual todo se considera como extensión del archivo. Si un archivo no contiene ningún punto se debe utilizar el argumento -f <file name>, pudiendo introducir un número arbitrario de archivos entre comillas.

3.3. Argumentos de pl

En esta sección se presentan los argumentos de pl que no se han mencionado previamente. Todos ellos siguen las expresiones regulares -{character}, -{character}{space}{word} o -{character}{space}{line}, donde una línea se considera como una sucesión de palabras y espacios delimitada por " o '.

- -o{space}{word} A la imagen creada se le asigna el nombre dado como argumento.

- `-t{space}({word}|{line})` Asigna un título a la imagen. El título puede ser una palabra o una línea.
- `-x{space}({word}|{line})` Asigna una etiqueta al eje de las abscisas. La etiqueta puede ser una palabra o una línea.
- `-y{space}({word}|{line})` Asigna una etiqueta al eje de las ordenadas. La etiqueta puede ser una palabra o una línea.
- `-l` La primera entrada de cada columna de los ficheros se toma como leyenda del conjunto de puntos correspondiente. Además, si no hay etiqueta para el eje de las abscisas, entonces la primera entrada de la columna de las abscisas se utiliza como etiqueta.

Por ejemplo, el Código 4 muestra un archivo con leyenda. El siguiente comando aplicado sobre este archivo genera la Figura 4.

```
pl legend.csv -o my_plot.png -l -t "Some lines with legend." -y "Y Axis"
```

```
X Axis,Line 1, Line 2, Line 3, Line 4
0,0,1,2,3
1,0,1,2,3
2,0,1,2,3
3,0,1,2,3
```

Código 4: Ejemplo de un archivo aceptado por `pl` con leyenda.

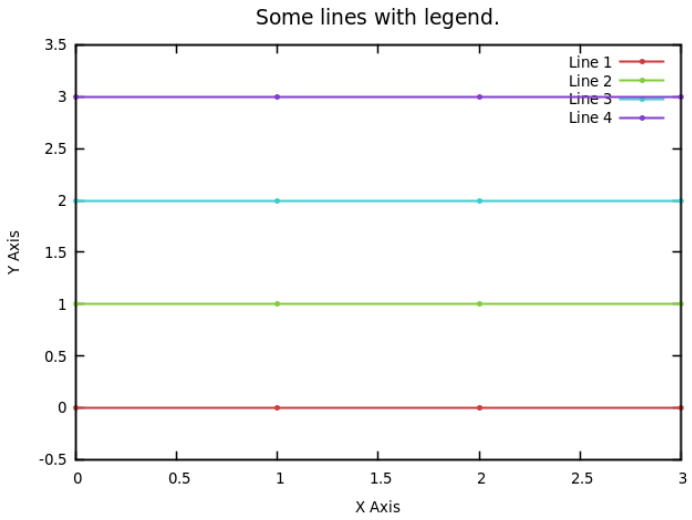


Figura 4: Gráfica con leyenda asociada al fichero dado en el Código 4.

Por defecto `pl` dibuja los conjuntos de puntos unidos por líneas. Sin embargo, también se pueden dibujar puntos o líneas solas.

- `-s{space}{style}` Cambia el estilo del gráfico. Hay tres estilos factibles: `lines`, `points` y `linespoints`. Cada uno de ellos tiene su correspondiente abreviación: `l`, `p` y `lp`.

- `-w{space}{number}` Cambia el ancho de línea por el número natural dado como parámetro. El ancho de línea es 2 por defecto.

Para ver la ayuda desde la terminal se puede utilizar el comando `pl -h`. Por otro lado, si se desea consultar el archivo `plot.sh` generado para crear la gráfica se puede utilizar el argumento `-k`, que evita que se elimine tras la ejecución.

3.4. Colores utilizados en las gráficas

Para seleccionar los colores de las gráficas `pl` utiliza el espacio HSV [6]. En este espacio los colores se representan por tres componentes:

- **Tonalidad**. Informalmente se entiende como la sensación visual según la cual el área coloreada se asemeja a uno de los tres colores básicos (rojo, verde y azul) o a una combinación de dos de ellos. Formalmente se corresponde con un ángulo (normalizado en [0,1)) que indica la posición del color en la circunferencia circunscrita al triángulo equilátero cuyos vértices son el rojo, el verde y el azul (en ese orden y comenzando el rojo en 0).
- **Brillo**. Se define como la cantidad de luz que refleja el color observado. Esto puede medirse como su distancia al color blanco normalizada.
- **Saturación** Se define como la intensidad del color divida por el brillo. También se normaliza en [0,1].

La transformación entre RGB y HSV es estándar y se puede encontrar con gran cantidad de detalles en la literatura. [2]

Por defecto, `pl` toma colores con igual saturación y brillo. La tonalidad se elige uniformemente en el intervalo [0,1). Exactamente, se toman los valores $\frac{i-1}{n}$ $\forall i = 1 \dots n$, donde n es el número de colores necesarios. De esta forma se garantiza que los colores sean distintos entre sí pero mantengan cierta concordancia.

`pl` también permite elegir colores con similar tonalidad. Para ello toma como argumento el color elegido y varía el brillo uniformemente sin llegar a colores muy oscuros. La saturación varía un poco para garantizar ciertas diferencias entre los colores. Se proporcionan dos argumentos para esto:

- `-c{space}{color_name}` : Los colores se eligen como variaciones del color dado como argumento. Se puede escoger entre rojo, amarillo, verde, azul, cian, y magenta.
- `-c{space}{color_rgb}` : Los colores se eligen como variaciones del color dado como argumento. Este consiste en una tupla R,G,B con la representación RGB del color escogido. Cada valor es un entero entre 0 y 255.

El primer argumento simplemente transforma el color escogido a RGB para realizar los cómputos. Por ejemplo, la Figura 5 muestra una gráfica realizada con cian.

[2] Artículo extenso sobre los modelos de color HSV y HSL: https://en.wikipedia.org/wiki/HSL_and_HSV

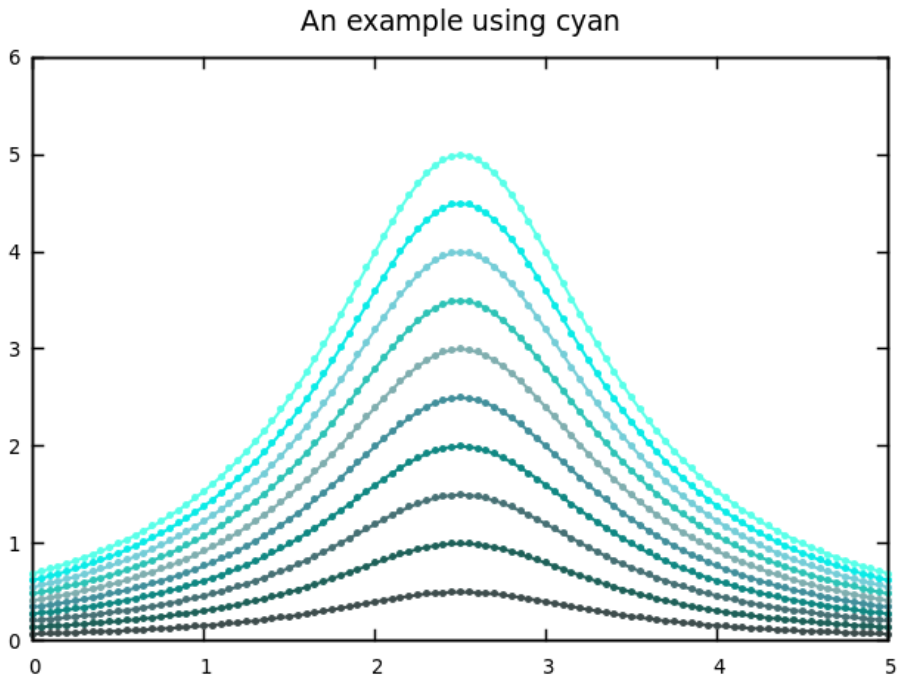


Figura 5: Gráfica donde se ha utilizado el argumento `-c cyan`.

4. Conclusión

El uso de *Lex* para tratar los argumentos de un programa facilita al programador esta sección del código. Las expresiones regulares proporcionan un mecanismo de detección de parámetros sencillo y eficaz, controlando además el léxico de los parámetros leídos.

Este hecho se prueba mediante la redacción de un programa para la creación de gráficas dos dimensionales, que se propone como una alternativa sencilla e intuitiva frente a las librerías utilizadas para este propósito. El programa presenta una completa selección de argumentos y automatiza todos los procesos asociados a la creación de gráficas. Además, los colores utilizados por este se eligen de forma que sean diferentes entre sí pero mantengan cierta concordancia.

Nota: El código se encuentra alojado en https://github.com/andreshp/pl

5. Bibliografía

1. Hunter, John D. "Matplotlib: A 2D graphics environment." Computing in science and engineering. Año 2007. Páginas 90-95.
2. Team R Core. R Language Definition." Año 2000.
3. Wickham, Hadley. "ggplot2: elegant graphics for data analysis."Springer Science and Business Media. Año 2009.

4. Janert, Philipp K. “Gnuplot in action.” Manning. Año 2010.

5. Lesk, Michael E., y Eric Schmidt. “Lex: A lexical analyzer generator.” Año 1975.

6. Tkalcic, Marko, y Jurij F. Tasic.“Colour spaces: perceptual, historical and applicational background.” Eurocon. Año 2003. Páginas 304-308.