Tutorial de Octave

Sebastián Horacio Carbonetto

Revisión: Septiembre 2010

Universidad de Buenos Aires
Facultad de Ingeniería
66.25 Dispositivos Semiconductores

1. Introducción

GNU Octave es un lenguaje de programación de alto nivel especializado en cálculos numéricos. Es libre, gratuito y puede descargarse de

http://www.gnu.org/software/octave/download.html

Es un programa multiplataforma ya que corre bajo Windows, Linux y MacOS entre otros. Se maneja por línea de comando, aunque existen numerosas GUI's[1], como *qtOctave* (Ubuntu).

Este tutorial tiene como objetivo familiarizar al estudiante con la sintaxis del lenguaje, así como también con las funciones más básicas y las cuales serán de utilidad para la realización de los trabajos prácticos de la materia. Con este fin, la explicación se basará en ejemplos triviales para que la atención del estudiante se base en la comprensión de la herramienta y no en el entendimiento del problema planteado.

Para profundizar el estudio de esta herramienta, existen numerosos tutoriales en la web[2], aunque una de las maneras más útiles de aprender es mediante el comando `help`, como veremos más adelante.

También se recomienda leer el apéndice dónde se indican qué paquetes deben ser instalados, además del mismo programa, para que todas las funciones necesarias para la realización de los trabajos prácticos estén disponibles.

2. Primer vistazo

Al iniciar el programa, ya sea en un entorno gráfico o por línea de comando, nos encontraremos con algún mensaje de bienvenida y la línea de comando en blanco para comenzar a trabajar.

[1] Graphical User Interfase

[2] Por ejemplo, http://en.wikibooks.org/wiki/Octave_Programming_Tutorial o bien la documentación oficial http://www.gnu.org/software/octave/docs.html

```
GNU Octave, version 3.2.3
Copyright (C) 2009 John W. Eaton and others.
This is free software; see the source code for copying conditions.
There is ABSOLUTELY NO WARRANTY; not even for MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE. For details, type 'warranty'.

Octave was configured for ï486-pc-linux-gnu".

Additional information about Octave is available at
http://www.octave.org.

Please contribute if you find this software useful.
For more information, visit http://www.octave.org/help-wanted.html

Report bugs to <bug@octave.org>(but first, please read
http://www.octave.org/bugs.html to learn how to write a helpful report).



>> _
```

Existen dos maneras de trabajar con Octave: de forma directa, ingresando comandos por la línea de comandos, o bien generando un *script*. Un script es un archivo de texto plano que contiene una serie de instrucciones que octave puede interpretar y ejecutar, de extensión `.m`. Lo más usual es trabajar con scripts, pero para comenzar este tutorial, los primero ejemplos se trabajarán directamente por línea de comando.

Lo más sencillo que podemos hacer en Octave son cuentas, por ejemplo

```
>> 1+1
ans = 2
>> 65998-4855
ans = 61143
>>
```

Todo resultado puede ser guardado en una variable. Para asignar un valor a una variable, debemos utilizar el operador '`=`'.

```
>> a=1
a = 1
>> b=599
b = 599
>> c=a+b
c = 600
>> d=2*3
d = 6
>>
```

Como se ve en el ejemplo anterior, a una variable no sólo se le puede asignar un valor aislado, sino también el resultado de una operación.

En todos los ejemplos mostrados se ve que Octave siempre muestra el resultado de la operación realizada, lo que puede llegar a ser molesto, especialmente en un script que tiene más de mil operaciones y no queremos verlas todas en pantalla. Para decirle a Octave que no queremos visualizar el resultado en la pantalla es necesario agregar un '`;`' al final de la línea ejecutada.

```
>> a=1;
>> b=599;
>> c=a+b;
>> d=2*3;
>> c*d
ans = 3600
>>
```

3. Todos los elementos son matrices

Para Octave todos los elementos son matrices. En particular, los escalares son matrices de dimensión 1×1, los vectores fila son matrices de dimensión $1 \times n$, y los vectores columna son matrices de dimensión $n \times 1$. Las matrices se declaran entre corchetes (`[]`), las columnas se separan con espacios, y para iniciar una nueva fila se utiliza el '`;`'.

```
>> matrix=[1 2;3 4]
matrix =

    1    2
    3    4

>> rVector=[1 2 3 4]
rVector =

    1    2    3    4

>> cVector=[1;2;3;4]
cVector =

    1
    2
    3
    4

>>
```

Una vez que tenemos nuestra matriz almacenada en una variable, podemos acceder independientemente a cualquier elemento en forma individual. Basta con indicar entre paréntesis a cuál de todos los elementos queremos acceder.

```
>> matrix=[0.9 0.3 0.2;0.3 0.9 0.4; 0.4 0.2 0.9];
>> matrix(1,3)
ans = 0.20000
>> mini_matrix=matrix(2:3,2:3)
mini_matrix=

    0.90000    0.40000
    0.20000    0.90000

>>
```

En el primer ejemplo se accede al elemento de la fila 1 y columna 3. Es importante notar que la numeración de los vectores y matrices comienza en 1 y no en 0 como en otros lenguajes de programación (por ejemplo lenguaje C). En el segundo ejemplo, en lugar de acceder a un único elemento, se accede a una "submatriz" que corresponde a la inferior izquierda. El operador '`:`' puede

utilizarse para indicar que se desea tener acceso a todos los elementos de una fila o columna, y de esta manera tomar filas y/o columnas individuales dentro de una matriz

```
>> matrix(1,:)
ans =

    0.90000    0.30000    0.20000

>> matrix(:,1)
ans =

    0.90000
    0.30000
    0.40000

>>
```

Otro caso especial de indexado de elementos en un vector sucede cuando queremos acceder desde el elemento n hasta el último, pero desconocemos la longitud del mismo. En este caso, se puede utilizar la palabra reservada `end` que indica "hasta el final".

```
>> n=96;
>> x(n:end)
ans =

    96    97    98    99    100

>> n=5;
>> x(end-n+1:end)
ans =

    96    97    98    99    100

>>
```

El resultado en los últimos dos ejemplos es equivalente, sin embargo son dos operaciones esencialmente diferentes. En el primer caso se accede a todos los elementos a partir del n–ésimo elemento, mientras que en el segundo ejemplo se accede a los últimos n elementos.

¿Qué sucede si necesitamos un vector con valores de 1 a 100? ¿Tenemos que ingresar los 100 valores manualmente? Por suerte, no. Con el operador '`:`' pueden construirse vectores indicando el primer valor, el último valor, y el "paso" entre elementos. Si no se indica un valor para el paso, Octave entiende por defecto que éste es unitario.

$$x = valor\_inicial : paso : valor\_final$$

```
>> [1:5]
ans =

    1    2    3    4    5

>> [5:-1:1]
ans =

    5    4    3    2    1

>> 11:3:20
ans =

    11    14    17    20

>> [0:.2:1]
ans =

    0.00000    0.20000    0.40000    0.60000    0.80000    1.00000

>>
```

Notar dos cosas del ejemplo anterior. Cuando se utiliza el operador ‘:’ no es necesario utilizar los corchetes, Octave entiende que se está construyendo un vector. Además, cuando se ingresan números decimales, no es necesario ingresar el cero a la izquierda.

Como todos los elementos son matrices, todas las operaciones que se realizan en Octave son operaciones matriciales, por lo tanto las dimensiones de los elementos involucrados en la operación deben ser acordes a la misma, en caso contrario obtendremos un mensaje de error. Por ejemplo, no podemos sumar un vector fila con un vector columna

```
>> x=rVector+cVector;
error: operator +: nonconformant arguments (op1 is 1x4, op2 is 4x1)
>>
```

pero sí podemos multiplicarlos, y dependiendo del orden de los factores el resultado será un escalar o una matriz.

```
>> rVector*cVector
ans = 30
>> cVector*rVector
ans =

    1    2    3    4
    2    4    6    8
    3    6    9   12
    4    8   12   16

>>
```

¿Qué sucede si tengo datos almacenados en dos matrices distintas donde cada elemento de la matriz representa información independiente del resto de los elementos, pero se relaciona con el mismo elemento de la otra matriz? En este caso es posible que queramos realizar operaciones “elemento a elemento” en lugar de operaciones matriciales. En estos casos, será necesario anteponer un ‘.’ al operador. El ‘.’ justamente lo que hace es indicar al operador que la operación no es matricial sino elemento a elemento.

```
>> A=[9 8;7 6];
>> B=[1 0;0 1];
>> C=[1 2;3 4];
>> A*B
ans =

    9    8
    7    6

>> A.*B
ans =

    9    0
    0    6

>> C^2
ans =

    7   10
   15   22

>> C.^2
ans =

    1    4
    9   16

>>
```

Una operación muy frecuente que se realiza sobre matrices es la transposición. En forma estricta, el operador de transposición es '.', mientras que '' (sin el punto) es el operador hermítico. Cuando se trabaja con número reales ambos son equivalentes.

```
>> a=[(1+i) 1]
a =

    1 + 1i    1 + 0i

>> D'
ans =

    1 - 1i
    1 - 0i

>> D.'
ans =

    1 + 1i
    1 + 0i

>>
```

4. Funciones

Un comando es una orden que el usuario le indica al programa. En las secciones anteriores ya le estuvimos indicando comandos al programa, ya que los operadores matemáticos y el indexado de variables son ejemplos de comandos.

Existen otros comandos que pueden llegar a ser de utilidad, como por ejemplo el comando `exit`, cerrar el programa.

Un comando que es de vital utilidad es `help`, el cual brinda información de cualquier comando o función.

```
>> help round
'round' is a built-in function

 -- Mapping Function: round (X)
    Return the integer nearest to X. If X is complex, return 'round
    (real (X)) + round (imag (X)) * I'.
        round ([-2.7, 2.7])
            =>  -3    3

    See also: ceil, floor, fix

>>
```

No hay diferencia entre funciones y comandos, en Octave ambos términos son sinónimos, sin embargo suele hacerse una diferenciación entre según la utilidad. Se suele hablar de funciones a todo comando que obligatoriamente toma argumentos. El ejemplo más sencillo son las funciones matemáticas como por ejemplo `sqrt()` (raíz cuadrada), `log()` (logaritmo natural), `log10()` (logaritmo en base 10), o las funciones trigonométricas (`sin()`, `cos()`, `tan()`, etc). Una extensa lista de funciones matemáticas puede encontrarse en

http://www.math.utah.edu/docs/info/octave_9.html

Además de existir funciones matemáticas definidas, también existen constantes matemáticas predefinidas. La lista completa de constantes matemáticas disponibles en Octave se encuentra en

http://www.math.utah.edu/docs/info/octave_8.html

donde las más importantes son el número imaginario `i` (o también `I`, `j` o `J`), el número de Euler `e` y `pi`.

Si se quiere calcular alguna de las siguientes expresiones matemáticas en Octave

$$\frac{\ln 100}{\ln 10}$$

$$\sqrt{3^2 + 4^2}$$

debemos ingresar los comandos

```
>> log(100)/log(10)
ans = 2
>> sqrt(3^2 + 4^2)
ans = 5
>>
```

La mayoría de las funciones operan elemento a elemento cuando el argumento es una matriz, sin embargo no siempre es así. Por ejemplo

```
>> x
x =

    1    10    100    1000    10000

>> log10(x)
ans =

    0    1    2    3    4

>> y
y =

    1    4    9    16    25

>> sqrt(y)
ans =

    1    2    3    4    5

>>
```

Se recomienda consultar la ayuda (comando `help`) ante cualquier duda sobre la compatibilidad matricial de las distintas funciones.

Además de la gran cantidad de funciones que existen predefinidas en octave, o que pueden agregarse instalando paquetes especiales, uno puede definir sus propias funciones como en cualquier lenguaje de programación. Como esta funcionalidad excede a los contenidos de la materia, no será explicado en este tutorial.

5. Cadenas de caracteres

Hasta el momento tratamos a las variables como numéricas, sin embargo estas pueden tener carácter lógico (verdadero–falso), pueden ser registros, o incluso cadenas de caracteres (*strings*). En esta sección trataremos a las variables como strings, lo que puede ser de utilidad a la hora de realizar y rotular gráficos. Se describen las nociones básicas del manejo de strings así como las funciones que serán de utilidad para el desarrollo de los trabajos prácticos.

Las strings se definen entre ‘’ y se asignan a las variables de la misma manera que los números.

```
>> string1='¡Hola mundo!'
string1 = ¡Hola mundo!
>> string2='Qué original, ¿no?'
string2=Qué original, ¿no?
>>
```

Las cadenas de caracteres son vectores de caracteres, por lo que se pueden indexar

```
>> string1(2:5)
ans = Hola
>> string1(end)
ans = !
>>
```

Así como existen constantes numéricas también existen constantes de caracteres. Algunos de ellos son TAB `\t`, la barra invertida `\\`, el caracter de nueva línea `\n`, el apóstrofe `\'` y el doble apóstrofe `\"`. El siguiente ejemplo muestra cómo se utilizan algunos de ellos

```
>> conversacion='Don José:\t\t\"Hola Don Pepito\"\nDon Pepito:\t\"Hola
Don José\"\n'
conversacion = Don José:\t\t\"Hola Don Pepito\"\nDon Pepito:\t\"Hola Don
José\"\n
>> printf(conversacion)
Don José:       "Hola Don Pepito"
Don Pepito:     "Hola Don José"
>>
```

En el ejemplo anterior se utilizó la función `printf()`. Esta función es idéntica a aquella en el Lenguaje C, e imprime por "standard output".

Es muy común necesitar crear un string a partir de variables numéricas. Para ello existe la función `sprintf()` que es similar a `printf()`, pero devuelve una cadena de caracteres en lugar de imprimir su contenido por "standard output". El primer argumento de la función debe ser un string que controla el formato de salida del string. El caracter de control de formato es el '`%`'. Cualquier caracter de este string que no sea un caracter de control de formato, será tratado como un caracter corriente. Para más información sobre la sintaxis del control de formato, pueden referirse a

http://www.gnu.org/software/octave/doc/interpreter/Output-Conversion-Syntax.html
http://www.gnu.org/software/octave/doc/interpreter/Table-of-Output-Conversions.html

Algunos ejemplos

```
>> sprintf('La función rand() genera un número aleatorio como por
ejemplo%f.',rand())
ans = La función rand() genera un número aleatorio como por ejemplo
0.322414.
>> r=10*rand();
>> n=round(r);
>> sprintf('%i es el entero más cercano a%.2f.',n,r)
ans = 5 es el entero más cercano a 4.97.
>> sprintf('\n%03o\n%03o\n%03o',1,8,64)
ans =
001
010
100
>>
```

6. Gráficos

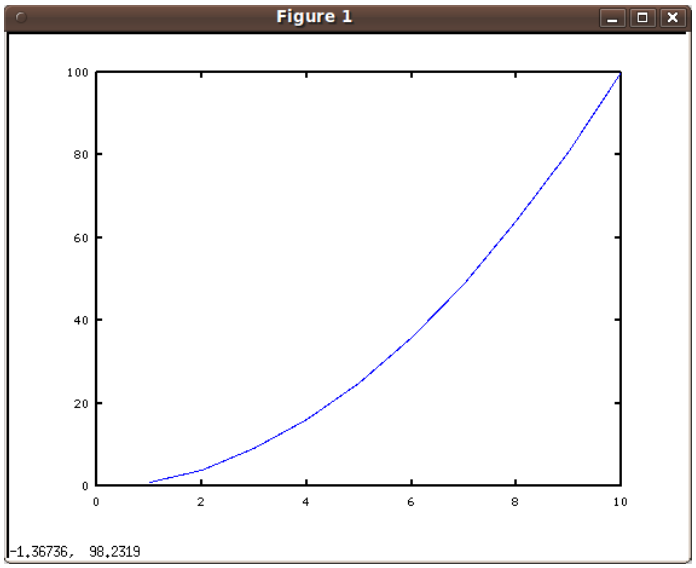
Octave es una poderosa herramienta para manipular datos gracias a su capacidad de cálculo numérico. Sin embargo, ver resultados numéricos no siempre es útil. En muchísimos casos resulta de mayor utilidad interpretar los resultados de forma gráfica. Para ello Octave cuenta con una gran variedad de funciones destinadas a realizar gráficos. Entre ellas encontramos `plot()`, `stem()`, `bar()`,

`polar()`, `semilogx()`, y muchas otras. A modo de ejemplo, se explicará la función `plot()` que será de mayor utilidad en la materia, pero se debe tener en cuenta que el uso de las otras funciones destinadas a realizar gráficos es similar.

La función `plot()` puede tomar múltiples argumentos, sin embargo en su forma más simple tiene un único argumento, los valores del eje y.

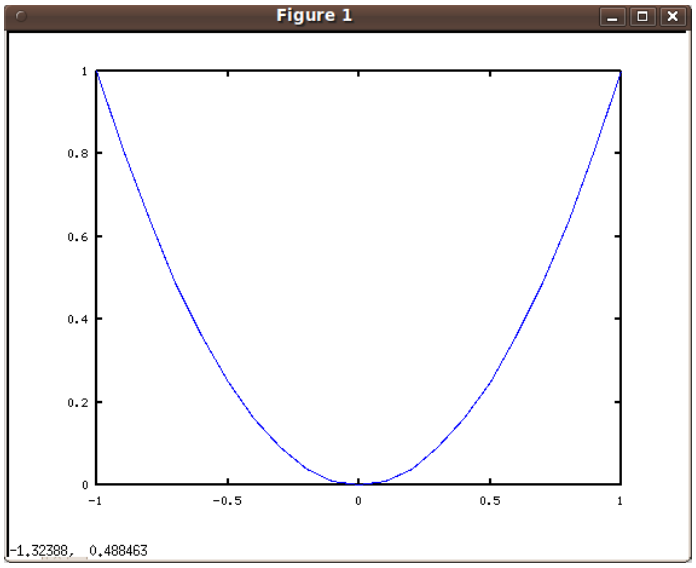
```
>> y=(1:10).^2;
>> plot(y);
>>
```

De esta manera se obtiene como resultado el siguiente gráfico.



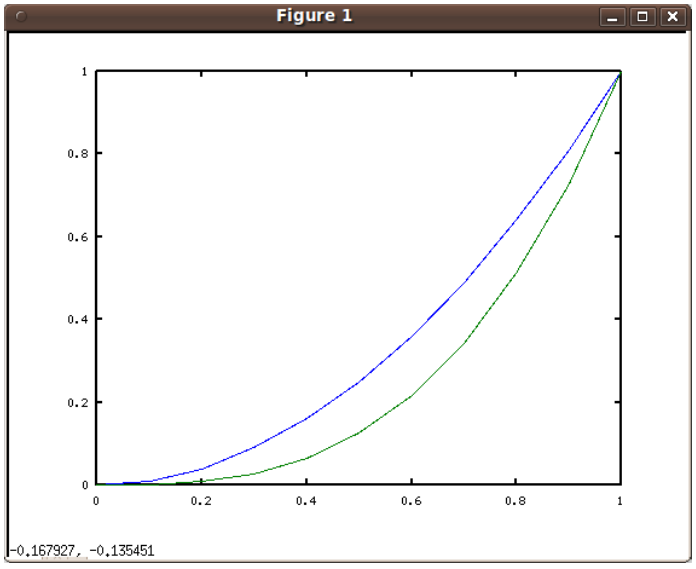
En este caso, el eje y toma los valores del vector `y` mientras que el eje x representa el x–ésimo elemento del vector. Esta asignación al eje x puede no ser representativa de los valores graficados, por ello la función `plot()` nos permite asignarle valores específicos a cada uno de los ejes, como se ve en el siguiente ejemplo.

```
>> x=(-1:0.1:1);
>> y=x.^2;
>> plot(x,y);
>>
```

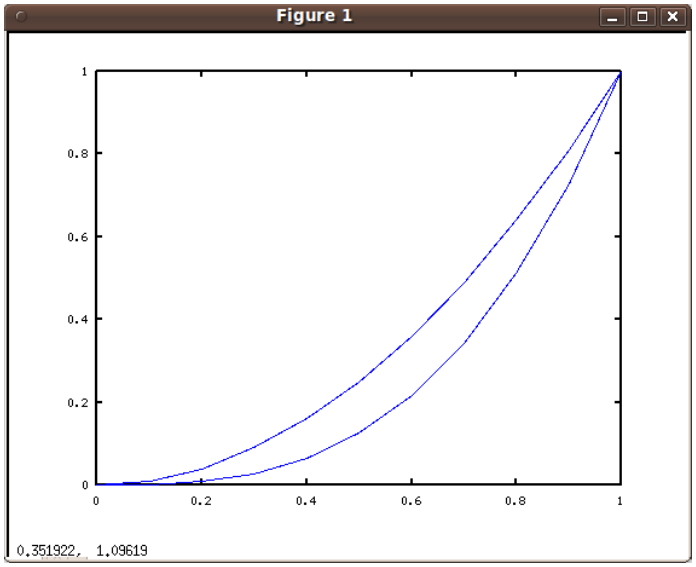


Suele ser necesario tener que graficar más de una curva en un mismo gráfico. Hay dos formas de hacer esto y cada una tiene sus ventajas y desventajas.

```
>> x1=(0:0.1:1);
>> y1=x1.^2;
>> x2=x1;
>> y2=x2.^3;
>> plot(x1,y1,x2,y2);
>>
```



```
>> x1=(0:0.1:1);
>> y1=x1.^2;
>> x2=x1;
>> y2=x2.^3;
>> hold on;
>> plot(x1,y1);
>> plot(x2,y2);
>>
```



En el primer ejemplo podemos observar que las curvas se diferencian por su color, cosa que no ocurre en el segundo caso. Además en el primer caso vemos que el gráfico se produce con una única invocación a la función `plot()`. Sin embargo, es necesario que todas las variables estén disponibles al momento de realizar el gráfico, lo cual puede no ocurrir en algunos casos.

En el segundo ejemplo para graficar simultáneamente distintas curvas es necesario indicarle al programa que una nueva invocación a `plot()` no debe "borrar" los gráficos realizados anteriormente. Esto se logra con el commando `hold on`. De forma similar, si es necesario volver a indicarle al programa que sólo debe graficar la última invocación de la función `plot()`, se debe usar el comando `hold off`. Por defecto, el commando `hold` se encuentra en el estado `off`.

El segundo inconveniente en la segunda implementación es que ambas curvas se encuentran graficadas con un mismo color. Como es de esperarse, uno puede elegir con qué color graficar las curvas. Para ello, es necesario dar un tercer argumento a la función `plot()`. Este tercer argumento es una cadena de caracteres que le da indicaciones a la función sobre la "estética" de la curva a graficar. La cadena de caracteres tiene típicamente tres caracteres que definen el color, el estilo de línea entre puntos y el tipo de marcador (ver cuadro 1).

Por último, existen otros argumentos que pueden ser especificados para controlar otros aspectos visuales del gráfico como el espesor de la línea, el tamaño del marcador y los colores del marcador entre otros. Estos argumentos van de a pares, donde el primero indica la propiedad y el segundo el valor que se le asigna

Estilo de Línea		Tipo de Marcador		Color	
Caracter	Descripción	Caracter	Descripción	Caracter	Descripción
'-'	Línea sólida	'+'	Signo más	'r'	Rojo
'--'[3]	Línea a trazos	'o'	Círculo	'g'	Verde
'-.'[3]	Línea trazo-punto	'*'	Asterisco	'b'	Azul
':'[3]	Línea punteada	'.'	Punto	'c'	Cyan
'none'	Sin línea	'x'	Cruz	'm'	Magenta
		's'[3]	Cuadrado	'y'	Amarillo
		'd'[3]	Diamante	'k'	Negro
		'^'	Triángulo	'w'	Blanco
		'v'[3]	Triángulo		
		'>'[3]	Triángulo		
		'<'[3]	Triángulo		
		'p'[3]	Pentagrama		
		'h'[3]	Hexagrama		
		'none'	Sin marcador		

Cuadro 1: Descripción de caracteres para el formato de los gráficos

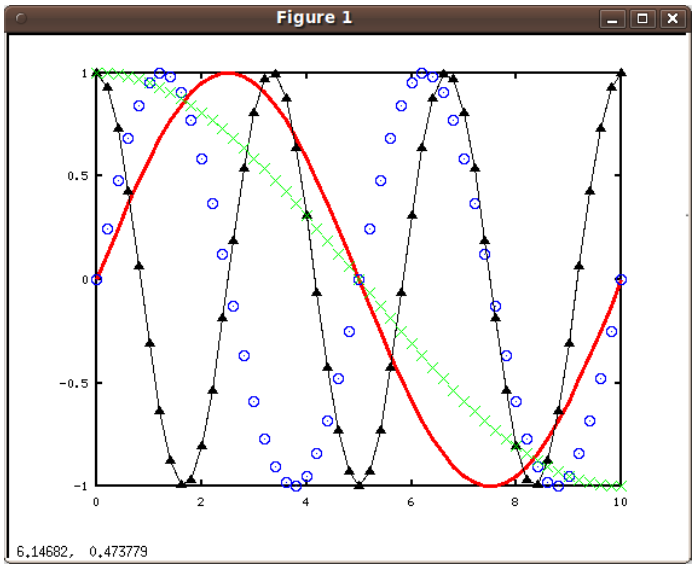
(por ejemplo: `plot(x,y,'propiedad1',valor1,'propiedad2',valor2,...)`). Para más detalles puede referirse a

http://www.mathworks.com/help/techdoc/ref/linespec.html

El siguiente es un ejemplo de cómo dar formato a distintas curvas en un mismo gráfico.

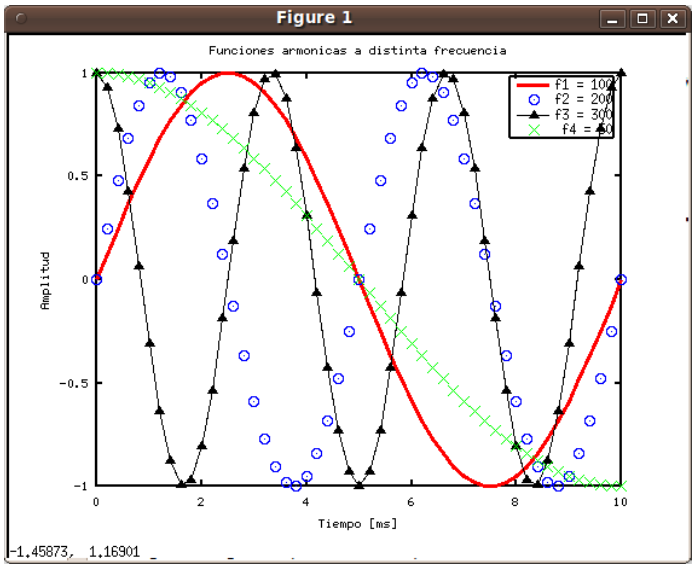
```
>> t=(0:0.2:10);
>> f1=100;f2=200;f3=300;f4=50;
>> x1=sin(2*pi*f1*t*1e-3);
>> x2=sin(2*pi*f2*t*1e-3);
>> x3=sin(2*pi*f3*t*1e-3);
>> x4=sin(2*pi*f4*t*1e-3);
>> hold on
>> plot(t,x1,'r','linewidth',3);
>> plot(t,x2,'bo','markersize',15);
>> plot(t,x3,'k^-','markersize',15);
>> plot(t,x4,'gx','markersize',15);
>>
```

[3]Sólo MATLAB



Ahora las curvas tienen distinto formato, sin embargo todavía faltan elementos en el gráfico. Una persona que observe este gráfico puede preguntarse: ¿Qué se muestra en este gráfico? ¿Qué representan los ejes del gráfico? ¿Qué representa cada una de las curvas? Para responder estas preguntas es necesario agregar las etiquetas correspondientes que brinden la información necesaria. Para ello existen un grupo de funciones que se encargan de agregar estas etiquetas. Las más importantes son: `title()`, `xlabel()`, `ylabel` y `legend()`. En el siguiente ejemplo se agregan todas estas etiquetas al último gráfico.

```
>> title('Funciones armonicas a distinta frecuencia')
>> xlabel('Tiempo [ms]')
>> ylabel('Amplitud')
>> leyenda1=sprintf('f1 =%u',f1);
>> leyenda2=sprintf('f2 =%u',200);
>> leyenda3=sprintf('f3 =%u',300);
>> leyenda4=sprintf('f4 =%u',50);
>> legend(leyenda1,leyenda2,leyenda3,leyenda4)
>> legend('boxon')
>> legend('right')
>>
```



Existen otras funciones para manipular propiedades de los gráficos como `axes()`, `axis()`, `grid` y otras. Para conocer cómo funcionan, se recomienda consultar el comando `help`.

Como último detalle, mediante el comando `figure()` pueden generarse distintos gráficos, es decir, nuevas ventanas en donde graficar las distintas variables. Como argumento (opcional) esta función toma el número de identificación de cada ventana. Si ningún número es especificado, por defecto se toma el siguiente valor disponible. En el siguiente ejemplo

```
>> figure(2)
>> figure
>> figure
>>
```

primero se genera la "Figura 2", luego la "Figura 1" y finalmente la "Figura 3".

```
>> figure
>> figure
>> figure(1),hold on;
>> plot(t,x1,'b','linewidth',3),plot(t,x2,'r','linewidth',3);
>> figure(2),hold on;
>> plot(t,x3,'b','linewidth',3),plot(t,x4,'r','linewidth',3);
>>
```

En este último ejemplo, primero se generan dos figuras. Luego, se "activa" la figura 1 en donde se grafican en forma simultánea las variables `x1` y `x2` (las dos funciones *seno*). Finalmente se "activa" la figura 2 en donde se grafican en forma simultánea las variables `x3` y `x4` (las dos funciones *coseno*).

6.1. Exportar gráficos

Una vez que el gráfico está terminado, es necesario exportarlo para poder utilizarlo en un documento. La función `print()` se encarga de "imprimir" el gráfico a un archivo. Esta función tiene numerosos usos además de exportar

gráficos, los cuales no serán explicados en este tutorial. Para exportar a un archivo, es necesario primer especificar la ubicación y nombre del archivo. Si este argumento no es especificado, el programa toma por defecto que el gráfico debe ser impreso por la impresora. El segundo argumento especifica las opciones de impresión. Para indicar que la figura debe ser guardada como un archivo específico debe usarse la opción `'-dXXX'` donde `XXX` es el tipo de archivo. Los más comunes son: `ps`, `eps`, `png`, `jpg`, `gif`, `pdf`. Para más información consultar `help print`. A continuación se muestra a modo de ejemplo cómo se exporta el gráfico de las funciones armónicas en formato PNG.

```
>> print('figura_armonicas.png','-dpng')
>>
```