Funciones en MATLAB

Estructura de una función:

function [parametros_salida] = nombre_funcion (parametros_entrada)

% comentarios de la descripción de la función

Bloque de instrucciones / cálculos (dentro de los cuales se asignan valores a los parámetros de salida)

Obs.

- Las variables que fueron declaradas dentro de la función, “viven” sólo en esa función (estas no existen fuera de ese ámbito). Se dice que dichas variables son *locales* a la función.
- Las variables declaradas en el programa como GLOBAL (esto es, las variables definidas en un *script* o en el ambiente de trabajo de Matlab), si son validas dentro de las funciones. Sin embargo, no es una buena práctica de programación usar variables globales en una función.

Funciones en MATLAB

- Cada función debe ser almacenada en un archivo de extensión m, al igual que los *scripts*. La diferencia en el caso de las funciones es que el nombre del archivo donde está almacenada la función debe ser igual al nombre de la función.
- Una función puede ser invocada directamente desde la ventana de comandos de Matlab, dentro de otra función o en un *script*.

Ejemplo:

DEFINICIÓN DE LA FUNCIÓN

```
function yy = func(x,y)
% evaluación de  yy = - y + x +1
yy = - y + x + 1
```

Escrita en un archivo de de nombre "func.m"

INVOCACIÓN DE LA FUNCIÓN

```
>> yy = func(1,2)        →    yy = 0
```

Desde el ambiente de trabajo de Matlab

Funciones en MATLAB

nargin y nargout

- Si se usan dentro de la función :

  nargin es la cantidad de parámetros de entrada en la invocación de la función.

  nargout es la cantidad de parámetros de salida en la invocación de la función.

- Si se usan fuera de la función :

  nargin es la cantidad de parámetros de entrada en la definición de la función.

  nargout es la cantidad de parámetros de salida en la definición de la función.

Funciones en MATLAB

nargin y nargout

Ejemplo:

DEFINICIÓN DE LA FUNCIÓN

```
function yy = func(x,y)
% evaluación de  yy = - y + x +1
yy = - y + x + 1
```

Escrita en un archivo de de nombre “func.m”

```
>> in = nargin(‘func’)    →    in = 2
>> out = nargout(‘func’) →    out = 1
```

Desde el ambiente de trabajo de Matlab

archivo : func.m

Funciones en MATLAB

nargin y nargout

Ejemplo:

```
1  function imprime_matriz(A,archivo,i)
2  % Funcion que imprime una matriz A en un archivo cuyo nombre viene dado
3  % por el parametro 'archivo'. Si el parametro archivo es omitido, se
4  % imprime la matriz en pantalla.
5  if nargin == 2 %caso 2 parametros
6      id_archivo = fopen(archivo,'w');
7      [n,m] = size(A);
8      for i=1:n
9          for j=1:m
10             fprintf(id_archivo,' %f %',A(i,j));
11         end
12         fprintf(id_archivo,'%c%c\n',char(13),char(10));
13     end
14     fclose('all');
15 elseif nargin == 1 %caso 1 parametros
16     [n,m] = size(A);
17     for i=1:n
18         for j=1:m
19             fprintf(' %f %',A(i,j));
20         end
21         fprintf('\n');
22     end
23 else %caso no parametros
24     disp('no se dio la matriz de entrada');
25     return;
26 end
```

Se refiere a la cantidad de parámetros de entrada que se coloquen al momento de invocar la función

archivo : imprime_matriz.m

Funciones en MATLAB

nargin y nargout

En el ejemplo de la lámina anterior, si se invoca la función imprime_matriz desde el ambiente de trabajo de Matlab de la siguiente manera:

```
>> A = [ 1 2 3; 4 5 6];
>> imprime_matriz(A,'result.txt');
```

entonces se imprime la matriz A en el archivo de nombre result.txt,

mientras que al invocarla como:

```
>> A = [ 1 2 3; 4 5 6];
>> imprime_matriz(A);
```

entonces la matriz se imprime en pantalla.

En el primer caso nargin tomó el valor 2, mientras que en el segundo caso tomó el valor 1.

Obs. Nótese que una función no tiene por qué tener parámetros de salida (igual observación vale para los parámetros de entrada).

Funciones en MATLAB

nargin y nargout

Ejemplo:

```
1  function [raiz,numIt,traza] = biseccion(f,a,b,tol)
2  debo_crear_traza = (nargout == 3); dar_numIt = (nargout == 2);
3  if debo_crear_traza, traza = []; numIt = 0; end
4  if dar_numIt, numIt = 0; end
5  if (nargin ~= 3) & (nargin ~= 4)
6      fprintf('\n Esta función espera 3 o 4 parámetros. \n\n');
7      raiz = 0; return
8  end
9  if nargin == 3, tol = eps; end
10 if a > b, temp = a; a = b; b = temp; end
11 fa = f(a); fb = f(b);
12 if sign(fa) == sign(fb)
13     fprintf('\n La funcion no cambia de signo en [%f,%f] \n',a,b);
14     raiz = 0; return
15 end
16 c = (a+b)/2; fc = f(c);
17 if debo_crear_traza, traza = [traza c]; numIt = 1; end
18 if dar_numIt, numIt = 1; end
19 while b-a > tol & abs(fc) > tol
20     if sign(fc) == sign(fa)
21         a = c; fa = fc;
22     else
23         b = c; fb = fc;
24     end
25     c = (a+b)/2; fc = f(c);
26     if debo_crear_traza, traza = [traza c]; numIt = numIt + 1; end
27     if dar_numIt, numIt = numIt + 1; end
28 end
29 raiz = c;
```

Se refiere a la cantidad de parámetros de salida que se coloquen al momento de invocar la función

archivo : biseccion_nargout.m

Funciones en MATLAB

nargin y nargout

En el ejemplo de la lámina anterior, si se invoca la función bisección desde el ambiente de trabajo de Matlab de alguna de las siguiente maneras, usando f = inline('cos(x)-x')

```
>> raiz = biseccion(f,0,pi/2,0.01)
     raiz =
           0.7363
>> [raiz,numIt] = biseccion(f,0,pi/2,0.01)
      raiz =
           0.7363
      numIt =
           5
>> [raiz,numIt,traza] = biseccion(f,0,pi/2,0.01)
      raiz =
           0.7363
      numIt =
           5
      traza =
           0.7854    0.3927    0.5890    0.6872    0.7363
```

Funciones en MATLAB

Funciones recursivas

Son aquellas que recurren a si mismas en su declaración (bloque de instrucciones que la definen), en otras palabras se invocan a sí mismas.

Ejemplo 1: calculo de la potencia n de x, es decir x^n

Función no recursiva (calpot)

```
1  % funcion que calcula la potencia n de x
2
3  function x = calpot(x,n)
4  y = 1;
5  if n > 0
6      for i=1:n
7          y = y*x;
8      end
9  end
10 x = y;
```

Función recursiva (calpot_r)

```
1  % funcion que calcula la potencia n de x
2  % como funcion recursiva
3
4  function x = calpot_r(x,n)
5  if n == 0
6      y = 1;
7  elseif n == 1
8      y = x;
9  else
10     y = x * calpot_r(x,n-1);
11 end
12 x = y;
```

Basado en $x^n = x * x^{n-1}$

Obs. Las 2 funciones producen el mismo resultado, pero las funciones recursivas en general son más costosas desde el punto de vista de ejecución y memoria utilizada.

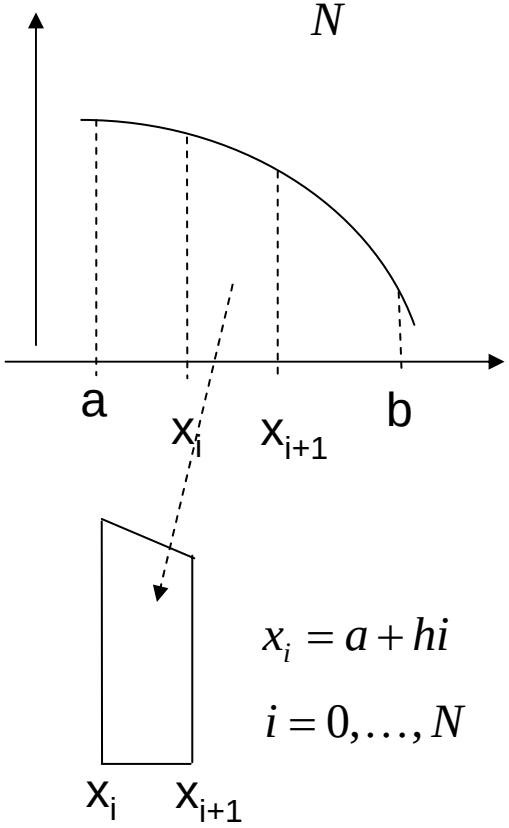
tic; x = calpot(2,500); toc
tic; x = calpot_r(2,500); toc

Funciones en MATLAB

Ejemplo: Cálculo de la integral de un polinomio en un intervalo [a,b] dado.

N: número de subintervalos en [a,b]

$$h = \frac{b-a}{N}$$



$$x_i = a + hi$$

$$i = 0, \ldots, N$$

$$area = \frac{f(x_i) + f(x_{i+1})}{2} h$$

$$area\ \ total = h \sum_{i=0}^{N-1} \frac{f(x_i) + f(x_{i+1})}{2} =$$

$$\frac{h}{2}(f(x_0) + 2f(x_1) + \cdots + 2f(x_{N-1}) + f(x_N)) =$$

$$h\left\{\left(\frac{f(a) + f(b)}{2}\right) + \sum_{i=1}^{N-1} f(x_i)\right\}$$

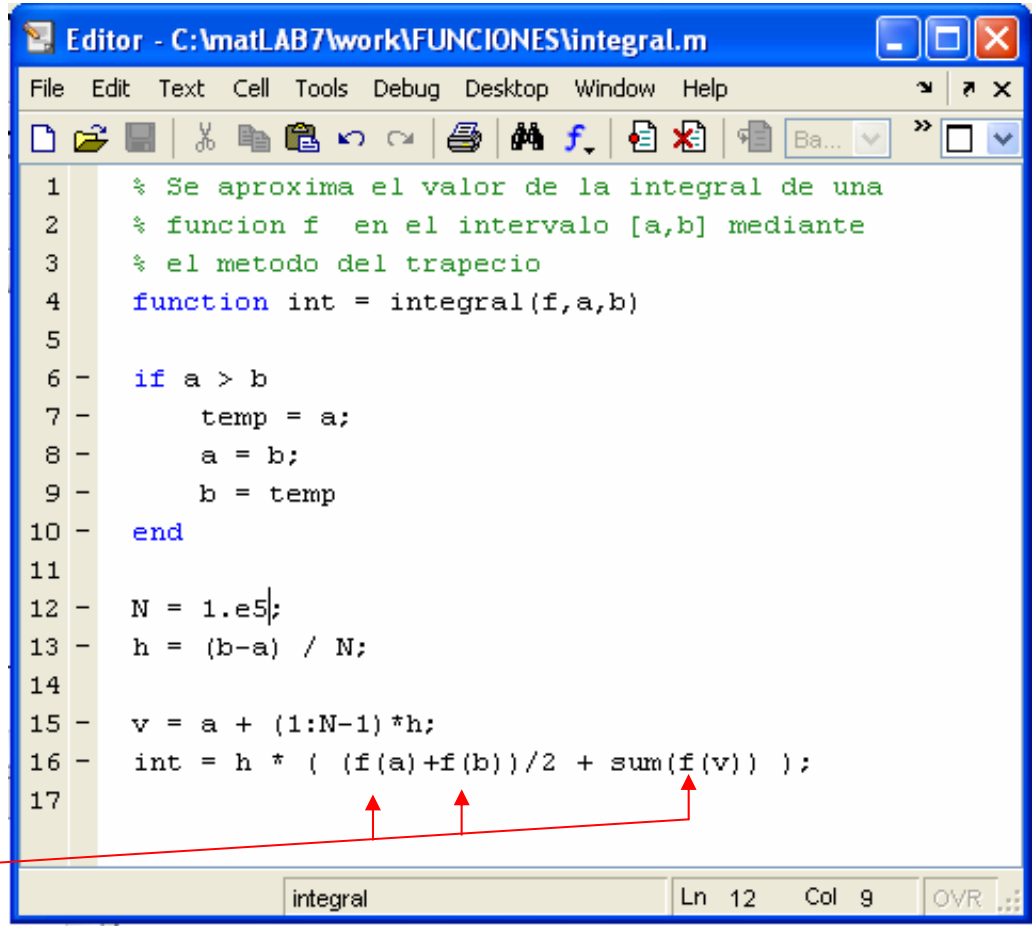
Funciones en MATLAB

Funciones como parámetros de otras funciones

En MATLAB, cualquiera de los parámetros de entrada de una función puede ser, a su vez, una función, la cual es invocada de manera usual en el bloque de instrucciones de la función que se está definiendo.

Ejemplo:

La función “integral” aproxima la integral de una función f dada, usando el método del trapecio. Así, la función f es el primer parámetro de la función “integral”. Nótese la invocación de la función f en el cálculo de int



archivo : integral.m

Funciones en MATLAB

Funciones como parámetros de otras funciones (cont.)

Si ahora se quiere invocar la función “integral” en el ambiente de trabajo de MATLAB, debe pasársele como primer parámetro otra función. Por ejemplo, supóngase que se quiere aproximar la integral entre 0 y 1 de la función x^2+1. Si ésta se define en un archivo “func1.m” de la siguiente manera:

```
function y = func1(x)
y = x.^2 + 1;
```

entonces podemos invocar a “integral” como:

```
>> int_aprox = integral(@func1,0,1)   →   int_aprox = 1.33333333334999
```

o equivalentemente:

```
>> g = @func1;
>> int_aprox = integral(g,0,1)   →   int_aprox = 1.33333333334999
```

Funciones en MATLAB

Funciones como parámetros de otras funciones (cont.)

En el caso de funciones cuya definición sea una fórmula, como en el ejemplo anterior, se puede utilizar lo que se conoce como “funciones anónimas”. Este método es muy práctico ya que no requiere que la función sea definida en un archivo .m

En el caso del ejemplo anterior, se puede invocar la función “integral” para calcular una aproximación de la integral entre 0 y 1 de la función x^2+1, de la siguiente manera:

```
>> int_aprox = integral(@(x) x.^2+1,0,1)   →   int_aprox = 1.33333333334999
```

o equivalentemente:

```
>> g = @(x) x.^2+1;
>> int_aprox = integral(g,0,1)   →   int_aprox = 1.33333333334999
```

Funciones en MATLAB

Funciones como parámetros de otras funciones (cont.)

En el caso de funciones predefinidas en Matlab (funciones intrínsecas), tales como *sin*, *cos*, *exp*, etc., se trabaja como en el caso de funciones definidas en archivos .m. Por ejemplo, una aproximación de la integral de la función seno en el intervalo [0,pi/2] se obtendría así:

```
>> int_aprox = integral(@sin,pi/2,0)   →   int_aprox = 0.99999999997944
```

o equivalentemente:

```
>> g = @sin;
>> int_aprox = integral(g, pi/2,0)   →   int_aprox = 0.99999999997944
```

Obs: Nótese que en esta invocación de la función “integral”, los extremos del intervalo de integración fueron pasados en orden inverso respecto a los ejemplos anteriores. Esto no representa un problema porque lo primero que se hace en la función “integral” es asegurar que en el parámetro *a* quede almacenado el extremo izquierdo del intervalo y en el parámetro *b* el extremo derecho.

Funciones en MATLAB

Funciones como parámetros de otras funciones (cont.)

Obs: En la última instrucción de la función “integral”, destaca la expresión sum(f(v)), lo cual obliga a que la función parámetro f sea aplicable a vectores. Es por eso que en la definición de la función “func1”

```
function y = func1(x)

y = x.^2 + 1;
```

era estrictamente necesario el uso del operador .^ en lugar de ^ (como sería en el caso escalar).

Funciones en MATLAB

Funciones como parámetros de otras funciones (cont.)

Un ejemplo en el cual no se puede usar una función anónima como parámetro de entrada, sino que es necesario definirla en un archivo .m, es el siguiente:

$$f(x) = \begin{cases} 1 & \text{si } -1 \le x < 0 \\ x^2 + 1 & \text{si } 0 \le x \le 1 \end{cases}$$

Se define la función “funcion_a_trozos” en el archivo “funcion_a_trozos.m” como

```
function y = funcion_a_trozos(x)
y = zeros(1,length(x));
for i=1:length(x)
    if x(i) < 0
        y(i) = 1;
    else
        y(i) = x(i)^2 + 1;
    end
end
```

Funciones en MATLAB

Funciones como parámetros de otras funciones (cont.)

Luego, una aproximación de la integral de esta función en el intervalo [-1,1] es:

```
>> int_aprox = integral(@funcion_a_trozos,-1,1)

int_aprox =

   2.33333333340002.
```

Funciones en MATLAB

Ejemplo: Cálculo de la integral de un polinomio en un intervalo [a,b] dado.

N: número de intervalos en [a,b]

$$h = \frac{b-a}{N}$$

$$area\ \ total = h\left\{\left(\frac{f(a)+f(b)}{2}\right)+\sum_{i=2}^{N} f(x_i)\right\}$$

```
1  p5 = input('ingrese los coeficientes del polinomio  ');
2  intervalo = input('ingrese el intervalo de integración  ');
3  f = const_polinomio(p5);
4  a = intervalo(1); b = intervalo(2);
5  int = integral(f,a,b);
6  valor = num2str(int,10);
7  xx = ['la integral de la función es' blanks(1) valor];
8  disp(xx);
```

archivos :
calculo_integral_polinomio.m
const_polinomio.m
integral.m

```
1  function int = integral(f,a,b)
2  % Se aproxima el valor de la integral de una
3  % funcion f en el intervalo [a,b] mediante
4  % el metodo del trapecio
5  %sintaxis: int = integral(f,a,b)
6  if a > b
7      temp = a;
8      a = b;
9      b = temp;
10 end
11 N = 1.e5;
12 h = (b-a)/N;
13 v = a + (1:N-1)*h;
14 int = h * ( (f(a)+f(b))/2 + sum(f(v)) );
15 end
```

```
1  function f = const_polinomio(p5)
2  % construccion de la formula f de un polinomio dado
3  % sus coeficientes, en orden decreciente de sus
4  % potencias, como el vector p5
5  %  sintaxis: f = const_polinomio(p5)
6  n = length(p5);
7  pp = [];
8  for i = 1:n
9      ss = '+';
10     if sign(p5(i)) == -1
11         ss = '-';
12     end
13     pp = [pp ss num2str(abs(p5(i)),8) '*x.^' num2str(n-i,1)];
14 end
15 disp(['polinomio:' blanks(1) pp]);
16 f = inline(pp);
17 % caso especial
18 % pp = poly2sym(p5);
19 % disp(pp);
```

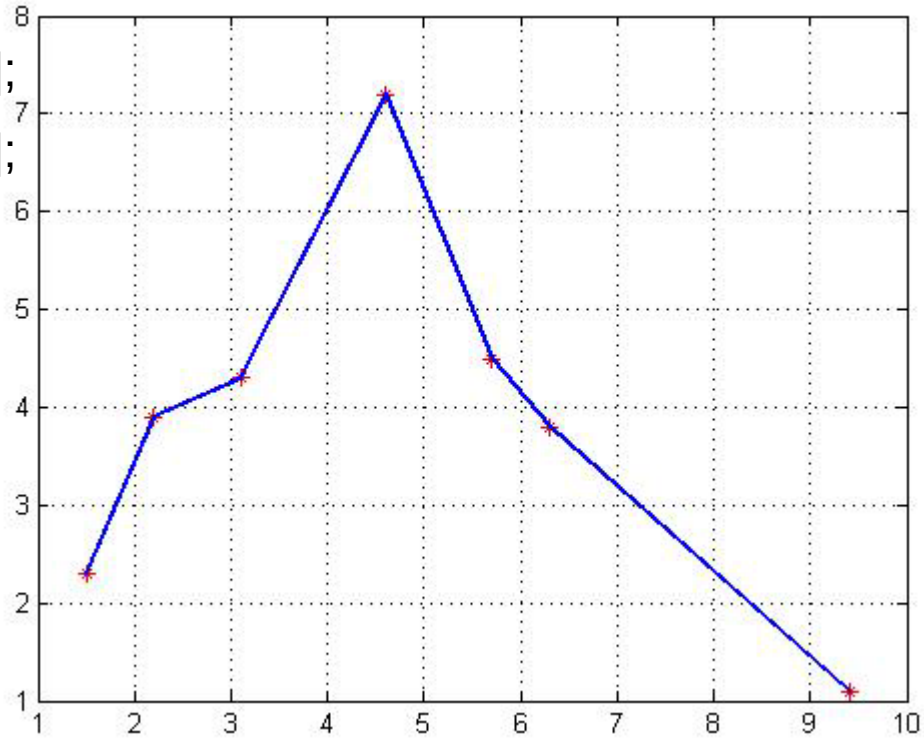
Graficas en 2 dimensiones

MATLAB tiene funciones básicas para crear gráficos 2D. Estas funciones se diferencian principalmente por el tipo de escala que utilizan en los ejes de las abscisas y las ordenadas.

Función “plot”

Crea un gráfico a partir de vectores y/o columnas de matrices, con escalas lineales sobre ambos ejes.

```
>> x = [ 1.5,  2.2,  3.1,  4.6,  5.7,  6.3,  9.4 ];
>> y = [ 2.3,  3.9,  4.3,  7.2,  4.5,  3.8,  1.1 ];
>> plot(x, y)
```



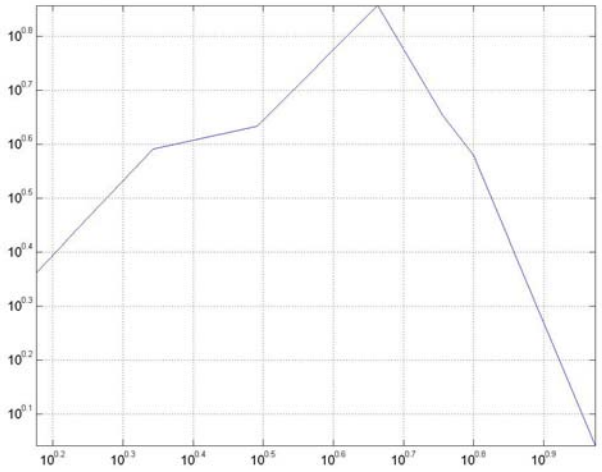
Graficas en 2 dimensiones

Funciones “loglog”, “semilogx” y “semilogy”

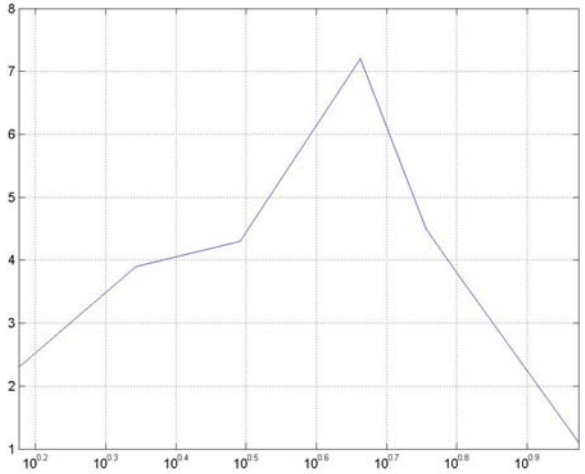
Crean un gráfico a partir de vectores y/o columnas de matrices, con escalas logarítmicas sobre ambos ejes, escala lineal en el eje de las ordenadas y logarítmica en el eje de las abscisas, y escala lineal en el eje de las abscisas y logarítmica en el eje de las ordenadas respectivamente.

```
>> x = [ 1.5,  2.2,  3.1,  4.6,  5.7,  6.3,  9.4 ];
>> y = [ 2.3,  3.9,  4.3,  7.2,  4.5,  3.8,  1.1 ];
```

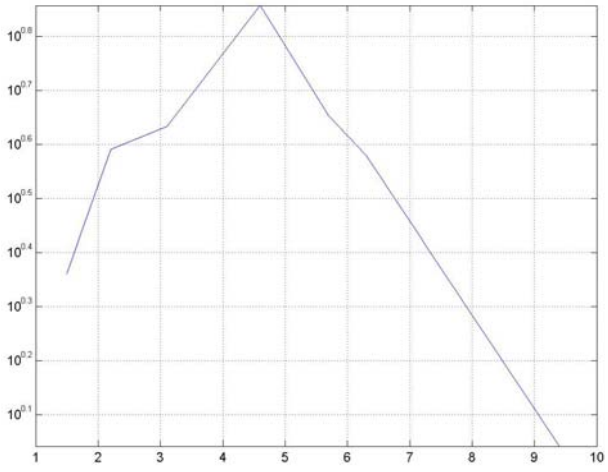
>> loglog(x,y)



>> semilogx(x,y)



>> semilogy(x,y)



Graficas en 2 dimensiones

Existen además otras funciones orientadas a añadir títulos al gráfico, a cada uno de los ejes, a dibujar una cuadrícula auxiliar, etc

title(‘título’)	Añade título al dibujo
xlabel(‘etiqueta’)	Añade una etiqueta al eje de las abscisas
ylabel(‘etiqueta’)	Añade una etiqueta al eje de las ordenadas
text(x,y´,’texto’)	Introduce ‘texto’ en el lugar especificado por las coordenadas x e y. Si x e y son vectores, el texto se repite por cada par de elementos. Si el texto es un arreglo de cadena de caracteres de la misma longitud que x e y, cada elemento se escribe en las coordenadas correspondientes.
grid on	Activa la inclusión de una cuadricula (mallado) en el dibujo
grid off	Desactiva la cuadricula (mallado) en el dibujo
hold on	Mantiene la ventana para añadir nuevos gráficos
hold off	Desactiva esta propiedad

Graficas en 2 dimensiones

Otras opciones para la función “plot” que definen el tipo de línea, color y símbolo para los puntos, son:

-	línea continua
:	línea punteada
-.	línea barra-punto
--	línea a trozos

y	amarillo
r	rojo
g	verde
b	azul
k	negro
w	blanco
m	magenta
c	cyan

.	punto
o	círculo
x	equis
+	mas
*	asterisco
s	cuadrado
d	diamante
v	triángulo abajo
^	triángulo arriba
<	triángulo izquierda
>	triángulo derecha
p	estrella

Graficas en 2 dimensiones

Ejemplo:

```
>> x = [ 1.5,  2.2,  3.1,  4.6,  5.7,  6.3,  9.4 ];
>> y = [ 2.3,  3.9,  4.3,  7.2,  4.5,  3.8,  1.1 ];
>> plot(x, y, 'b+:')
>> title('TITULO'); xlabel('etiqueta eje x'); ylabel('etiqueta eje y'); text(x,y,'P')
```