Curso basico de MATLAB

9. Números complejos

En muchos cálculos matriciales los datos y/o los resultados no son reales sino complejos, con parte real y parte imaginaria. ***Matlab*** trabaja sin ninguna dificultad con números complejos.

```
>> a=sqrt(-4)
a =
0 + 2.0000i
>> 3 + 4j
ans =
      3.0000 + 4.0000i
```

La asignación de valores complejos a vectores y matrices desde teclado puede hacerse de las dos formas, que se muestran en el ejemplo siguiente:

```
>> A = [1+2i 2+3i; -1+i 2-3i]
A =
       1.0000 + 2.0000i  2.0000 + 3.0000i
      -1.0000 + 1.0000i  2.0000 - 3.0000i
O bien

>> A = [1 2; -1 2] + [2 3; 1 -3]*I
A =
 1.0000 + 2.0000i  2.0000 + 3.0000i
-1.0000 + 1.0000i  2.0000 - 3.0000i
```

Matlab dispone asimismo de la función ***complex***, que crea un número complejo a partir de dos argumentos que representan la parte real e imaginaria, como en el ejemplo siguiente:

```
>> complex(1,2)
ans =
      1.0000 + 2.0000i
```

Existe una función (***conj( )***) que permite hallar simplemente la matriz conjugada y el operador punto y apóstrofo (**.'**) que calcula simplemente la matriz traspuesta.

10. Cálculos con polinomios

Para ***Matlab*** un polinomio se puede definir mediante un vector de coeficientes. Por ejemplo:

$$x_4 - 8x_2 + 6x - 10 = 0$$

Se puede representar mediante el vector [1, 0, -8, 6, -10].

Luego, se pueden realizar diversas operaciones sobre él, como por ejemplo:

- Evaluarlo para un determinado valor de **x.**
- Calcular las raíces.
- Calcular producto de polinomios.
- Calcular cociente de polinomios.
- Calcular la derivada de un polinomio.
- Interpolar polinomios

10.1 Funciones para cálculos con polinomios

Estando definido el polinomio como:

```
>> pol=[1 0 -8 6 -10]
pol =
     1 0 -8 6 -10
```

- ***Función roots( )***

roots(pol) *calcula las raíces del polinomio* ***pol***

```
>> A = roots(pol)
A =
          -3.2800
           2.6748
    0.3026 + 1.0238i
    0.3026 - 1.0238i
```

- ***Función poly( )***

poly(A) *el polinomio característico de la matriz* ***A***

```
>> pol = poly(A)
pol =
     1.0000    0.0000   -8.0000    6.0000  -10.0000
```

- ***Función polyval( )***

polyval(pol,x) *evalúa el polinomio* ***pol*** *para el valor de* ***x****. Si* ***x*** *es un vector,* ***pol*** *se evalúa para cada elemento de* ***x***

```
>> x = [1 2 3];
>> polyval(pol,x)
ans =
    -11.0000  -14.0000   17.0000
```

- ***Función conv( )***

  conv(p1,p2) *realiza el producto de convolución de dos polinomios* ***p1*** *y* ***p2.***

```
>> pol1=[1 -2 4];
>> pol2=[1 0 3 -4];
>> pol3=conv(pol1,pol2)
pol3 =
        1 -2 7 -10 20 -16
```

- ***Función deconv( )***

  [c,r]=deconv(p,q) *realiza la división del polinomio* ***p*** *por el polinomio* ***q****. En* ***c*** *se devuelve el cociente y en* ***r*** *el resto de la división.*

```
>> pol1=[1 -2 4];
>> pol2=[1 0 3 -4];
[c,r]=deconv(pol2,pol1)
c =
        1    2
r =
        0    0    3   -12
```

- ***Función residue( )***

  residue(p1,p2) *descompone el cociente entre* ***p1*** *y* ***p2*** *en suma de fracciones simples.*

```
P1(s)      R(1)       R(2)            R(n)
----  =  -------- + -------- + ... + -------- + K(s)
P2(s)    s - P(1)   s - P(2)        s - P(n)

>> pol2=[1 0 3 -4];
>> pol1=[1 -2 4];
[R,P,K]= residue(pol2,pol1)
R =
        1.5000 + 2.5981i
        1.5000 - 2.5981i
P =
        1.0000 + 1.7321i
        1.0000 - 1.7321i
K =
           1    2
```

- ***Función polyder( )***

  polyder(pol) *calcula la derivada del polinomio* ***pol****.*

```
>> pol=[1 -2 4];
>> polyder(pol)
ans =
        2   -2
```

- ***Función polyfit( )***

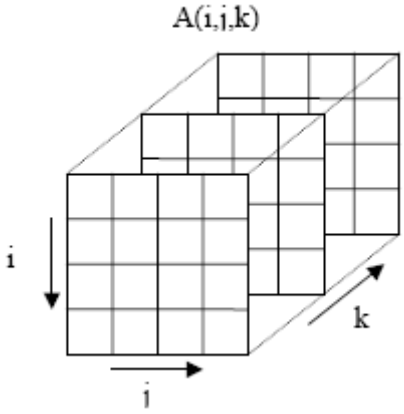
  pol = polyfit(x,y,n) *calcula los coeficientes de un polinomio* ***pol*** *de grado* ***n*** *que se ajusta a los datos* ***pol(x(i)) ~= y(i)****, en el sentido de mínimo error cuadrático medio.*

- ***Función interp1( )***

  interp1(xp,yp,x,'m') *calcula el valor interpolado para la abscisa* ***x*** *a partir de un conjunto de puntos dado por los vectores* ***xp*** *e* ***yp****. Con* ***'m'*** *se especifica también el método de interpolación. La cadena de caracteres* ***m*** *admite los valores 'nearest', 'linear', 'spline', 'pchip', 'cubic' y 'v5cubic'.*

11. Hipermatrices

Matlab permite trabajar con ***hipermatrices***, es decir con matrices de más de dos dimensiones. Una posible aplicación es almacenar con un único nombre distintas matrices del mismo tamaño (resulta una ***hipermatriz*** de 3 dimensiones).
Los elementos de una ***hipermatriz*** pueden ser números, caracteres, estructuras, y vectores o matrices de celdas.
El tercer subíndice representa la tercera dimensión: la ***"profundidad"*** de la ***hipermatriz.***



11.1 Ejemplo de definición de una hipermatriz

Por ejemplo, las siguientes sentencias generan, en dos pasos, una matriz de 2×3×2:

>> AA(:,:,1)=[1 2 3; 4 5 6]; *se define la matriz inicial*
>> AA(:,:,2)=[2 3 4; 5 6 7]; *se añade una segunda matriz*

Luego:

```
>> AA
 AA(:,:,1) =
              1 2 3
              4 5 6
 AA(:,:,2) =
              2 3 4
              5 6 7
```

11.2 Funciones que trabajan con hipermatrices

Algunas funciones de ***Matlab*** para generar matrices admiten más de dos subíndices y pueden ser utilizadas para generar hipermatrices. Entre las ya mencionadas están ***rand()***, ***randn()***, ***zeros(), ones(), size() y reshape().***
Todas las funciones de ***Matlab*** que operan sobre escalares (***sin()***, ***cos()***, etc.) se aplican sobre hipermatrices elemento a elemento (igual que sobre vectores y matrices).
Las funciones que operan sobre vectores (***sum()***, ***max()***, etc.) se aplican a matrices e hipermatrices según la primera dimensión, resultando un array de una dimensión inferior. Ejemplo:

```
>> sum(AA)
ans(:,:,1) =
             5     7     9
ans(:,:,2) =
             7     9    11
```

Las funciones matriciales propias del Álgebra Lineal (***det()***, ***inv()***, ***etc***.) no se pueden aplicar a hipermatrices. Para poderlas aplicar hay que extraer primero las matrices correspondientes (por ejemplo, con el operador dos puntos (:)).

La función ***cat()*** permite concatenar matrices según las distintas “dimensiones”, como puede verse en el siguiente ejemplo:

```
>> A=zeros(2,3);
>> B=ones(2,3);

>> cat(1,A,B)
ans =
      0 0 0
      0 0 0
      1 1 1
      1 1 1

>> cat(2,A,B)
ans =
      0 0 0 1 1 1
      0 0 0 1 1 1

>> cat(3,A,B)
ans(:,:,1) =
             0 0 0
             0 0 0
ans(:,:,2) =
             1 1 1
             1 1 1
```

Las siguientes funciones de ***Matlab*** se pueden emplear también con hipermatrices:

ndims()	Devuelve el número de dimensiones
squeeze()	Elimina las dimensiones que son igual a uno
permute(A,v)	Permuta las dimensiones de **A** según los índices del vector **v**
ipermute(A,v)	Realiza la permutación inversa

12. Gráficos 2-D elementales

Matlab dispone de cinco funciones básicas para crear gráficos 2-D. Estas funciones se diferencian principalmente por el *tipo de escala* que utilizan en los ejes de abscisas y de ordenadas. Estas funciones son las siguientes:

plot()	Crea un gráfico a partir de vectores y/o columnas de matrices, con escalas lineales sobre ambos ejes
plotyy()	Crea un gráfico con dos escalas lineales diferentes para las ordenadas, una a la derecha y otra a la izquierda de la figura.
loglog()	Crea un gráfico con dos escalas logaritmicas diferentes para las ordenadas, una a la derecha y otra a la izquierda de la figura.
semilogx()	Crea un gráfico con una escala lineal en el eje de ordenadas y una logarítmica en el eje de abscisas.
semilogy()	Crea un gráfico con una escala logaritmica en el eje de ordenadas y una lineal en el eje de abscisas.

Existen además otras funciones orientadas a añadir títulos al gráfico, a cada uno de los ejes, a dibujar una cuadrícula auxiliar, a introducir texto, etc.
Estas funciones son las siguientes:

title('título')	Añade un título al dibujo.
xlabel('texto')	Añade una etiqueta al eje de abscisas. Con *xlabel off* desaparece.
ylabel('texto')	Añade una etiqueta al eje de ordenadas. Con *ylabel off* desaparece
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 'texto' es también un vector de cadenas de texto de la misma dimensión, cada elemento se escribe en las coordenadas correspondientes.
gtext('texto')	Introduce 'texto' con ayuda del mouse: el cursor cambia de forma y se espera un clic para introducir el texto en esa posición.
legend()	Define rótulos para las distintas líneas o ejes utilizados en la figura.
grid	Activa la inclusión de una cuadrícula en el dibujo. Con *grid off* desaparece la cuadrícula

Los dos grupos de funciones anteriores no actúan de la misma forma. Así, la función ***plot*** dibuja una nueva figura en la ventana activa o abre una nueva figura si no hay ninguna abierta, sustituyendo cualquier cosa que hubiera dibujada anteriormente en esa ventana:

```
>> x = [-10:0.2:10];
>> y = sin(x);
```

```
>> close          se cierra la ventana gráfica activa anterior
>> grid           se crea una ventana con una cuadrícula
>> plot(x,y)      se dibuja la función seno borrando la cuadrícula
```

Se puede observar la diferencia con la secuencia que sigue:

```
>> close          se cierra la ventana gráfica activa anterior
>> plot(x,y)      se crea una ventana y se dibuja la función seno
>> grid           se añade la cuadrícula sin borrar la función seno
```

En el primer caso ***Matlab*** crea la cuadrícula en una ventana nueva y luego la borra al ejecutar la función ***plot***. En el segundo caso, primero dibuja la función y luego añade la cuadrícula. Esto es así porque hay funciones como ***plot*** que por defecto crean una nueva ***figura***, y otras funciones como ***grid*** que se aplican a la ventana activa modificándola, y sólo crean una ventana nueva cuando no existe ninguna ya creada.

12.1 Tipos de argumentos para la función PLOT

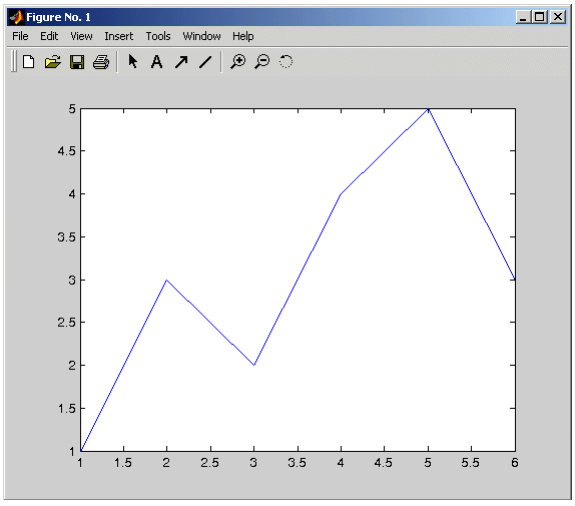
La función ***plot***, en sus diversas variantes, no hace otra cosa que dibujar vectores.
Por defecto:

- Los distintos puntos del gráfico se unen con una línea continua.
- El color que se utiliza para la primera línea es el azul.
- El color del fondo es blanco

12.1.1 Ejemplo en el que se le pasa un único vector como argumento

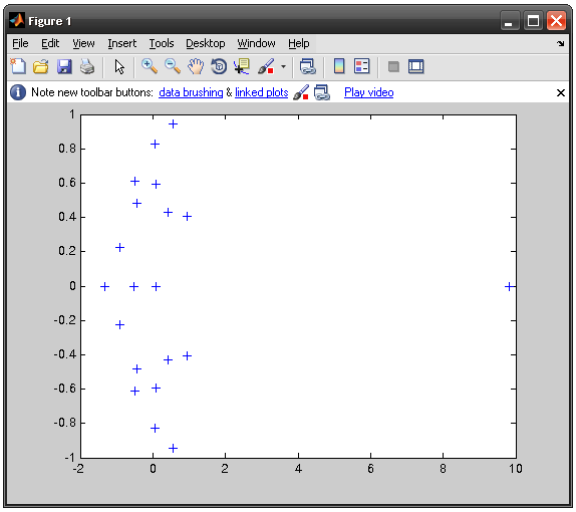
Cuando a la función ***plot*** se le pasa como único argumento, un vector de números reales, dibuja en ordenadas el valor de los n elementos del vector frente a los índices del mismo en abscisas.

```
>> x = [1 3 2 4 5 3];
>> plot(x)
```



Cuando a la función ***plot*** se le pasa como único argumento, un vector de números complejos como argumento, ***Matlab*** hace que se represente la parte real en abscisas, frente a la parte imaginaria en ordenadas.

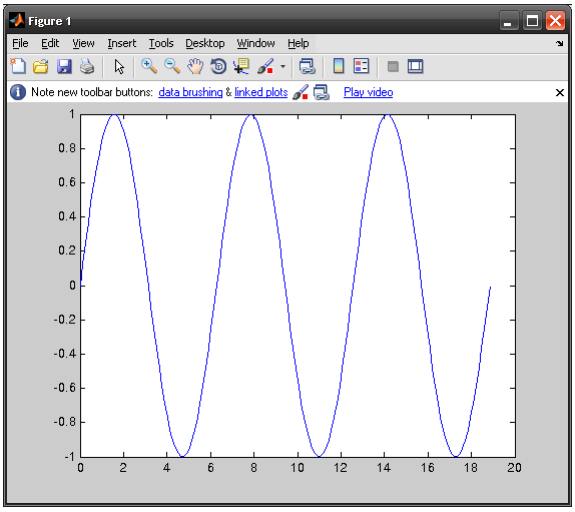
```
>> z = eig(rand(20,20));          se crea un vector de números complejos
>> plot(z,'+')                    se dibujan cruces en las intersecciones
```



12.1.2 Ejemplo en el que se le pasan dos vectores como argumento

En este caso, el segundo vector se dibuja en ordenadas como función de los valores del primer vector, que se representa en abscisas.

```
>> x = 0:pi/25:6*pi;
>> y = sin(x);
>> plot(x,y)
```



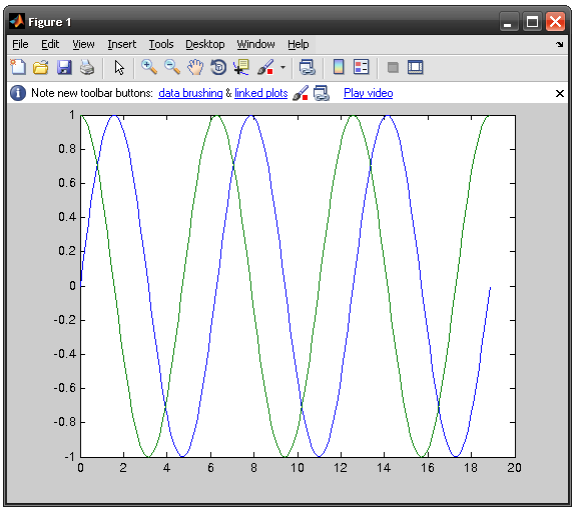
Cuando los argumentos son vectores de números complejos, simplemente se representan las partes reales y se desprecian las partes imaginarias. Si se quiere dibujar varios vectores complejos, hay que separar explícitamente las partes reales e imaginarias de cada vector.

```
>> plot(real(z),imag(z),'+')
```

12.1.3 Ejemplo en el que se le pasan más de dos vectores como argumento

La función ***plot*** permite también dibujar múltiples curvas introduciendo varias parejas de vectores como argumentos.
Si el usuario no decide otra cosa, para las sucesivas líneas se utilizan colores que son permutaciones cíclicas del ***azul***, ***verde***, ***rojo***, ***cyan***, ***magenta***, ***amarillo*** y ***negro***.

```
>> x = 0:pi/25:6*pi;
>> y = sin(x);
>> z = cos(x);
>> plot(x,y,x,z)
```



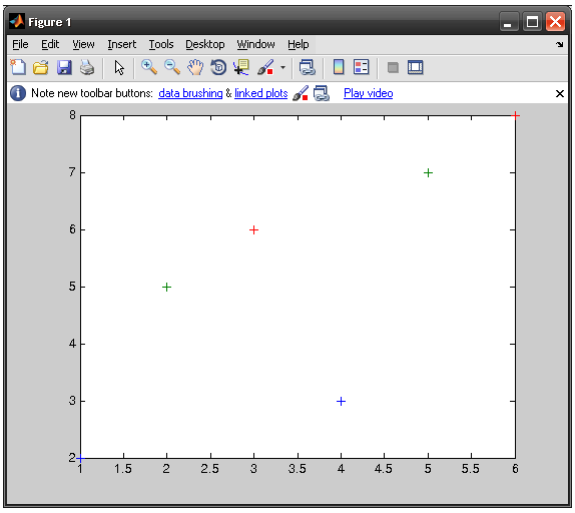
12.1.4 Ejemplo en el que se le pasan matrices como argumento

El comando ***plot*** puede utilizarse también con matrices como argumentos.

plot(A)	Dibuja una línea por cada columna de A en ordenadas, frente al índice de los elementos en abscisas.
plot(x,A)	Dibuja las columnas (o filas) de **A** en ordenadas frente al vector **x** en abscisas. Las dimensiones de **A** y **x** deben ser coherentes: si la matriz **A** es cuadrada se dibujan las columnas, pero si no lo es y la dimensión de las filas coincide con la de **x**, se dibujan las filas
plot(A,x)	Análogo al anterior, pero dibujando las columnas (o filas) de **A** en abscisas, frente al valor de **x** en ordenadas
plot(A,B)	Dibuja las columnas de **B** en ordenadas frente a las columnas de **A** en abscisas, dos a dos. Las dimensiones deben coincidir

plot(A,B,C,D) Análogo al anterior para cada par de matrices. Las dimensiones de cada par deben coincidir, aunque pueden ser diferentes de las dimensiones de los demás pares

```
>> A = [1 2 3; 4 5 6];
>> B = [2 5 6;3 7 8];
>> plot(A,B,'+')
```



12.2 Estilos de líneas y marcadores para la función PLOT

En la tabla siguiente se pueden observar las distintas posibilidades de estilos de líneas y marcadores.

Símbolo	Color	Símbolo	Marcadores (markers)
y	yellow	.	puntos
m	magenta	o	círculos
c	cyan	x	marcas en x
r	red	+	marcas en +
g	green	*	marcas en *
b	blue	s	marcas cuadradas (square)
w	white	d	marcas en diamante (diamond)
k	black	^	triángulo apuntando arriba
		v	triángulo apuntando abajo
Símbolo	**Estilo de línea**	>	triángulo apuntando a la dcha
-	líneas continuas	<	triángulo apuntando a la izda
:	líneas a puntos	p	estrella de 5 puntas
-.	líneas a barra-punto	h	estrella se seis puntas
--	líneas a trazos		

Cuando hay que dibujar varias líneas, por defecto se van utilizando sucesivamente los colores de la tabla comenzando por el azul, hacia arriba, y cuando se terminan se vuelve a empezar otra vez por el azul. Si el fondo es blanco, este color no se utiliza para las líneas.
También es posible añadir en la función ***plot*** algunos especificadotes de línea que controlan el espesor de la línea, el tamaño de los marcadores, etc.

12.3 Superposición de un grafico con uno ya existente

Existe la posibilidad de superponer un grafico nuevo a un gráfico ya existente, sin destruirlo o sin abrir una nueva ventana.

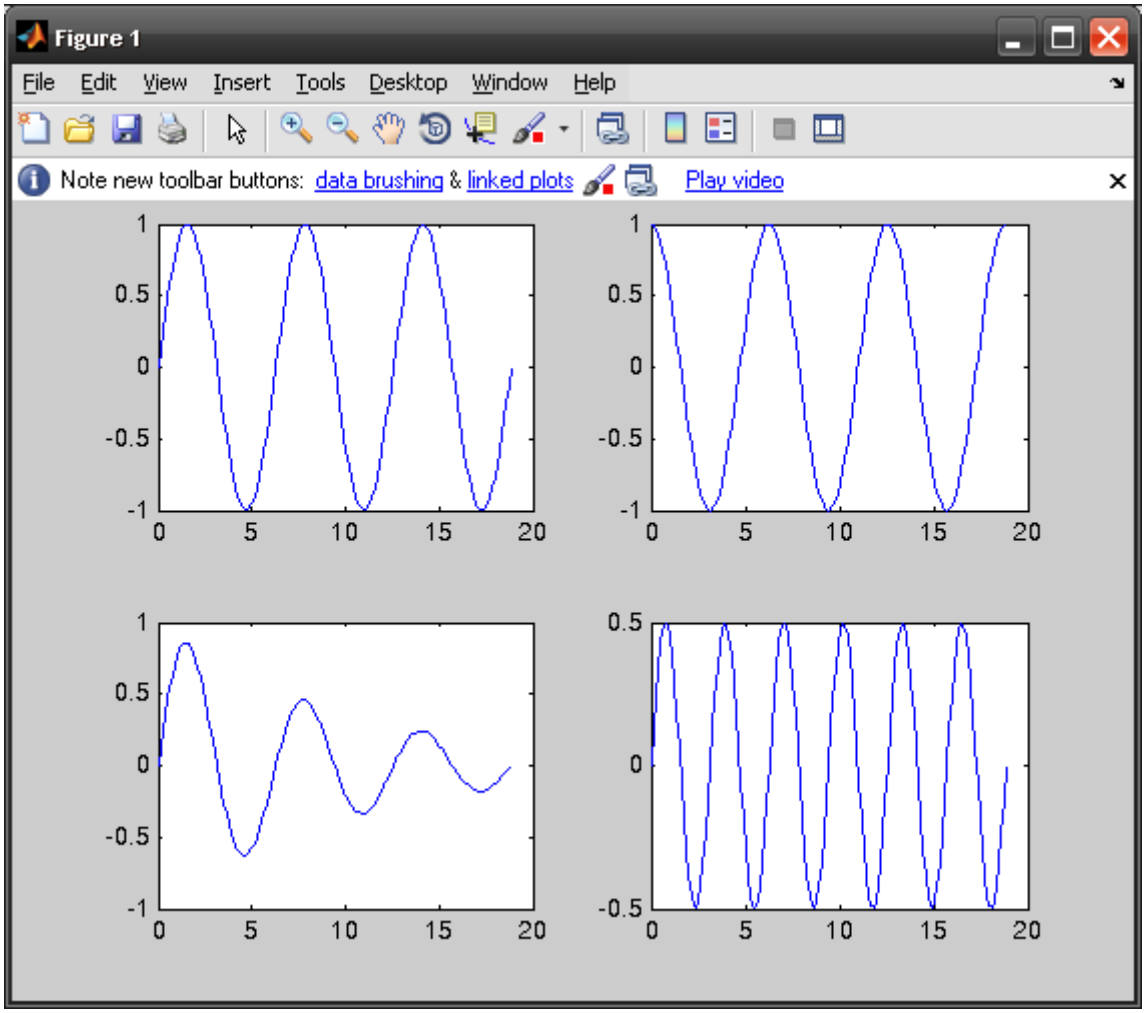
hold on	Hace que los gráficos sucesivos respeten los que ya se han dibujado en la figura.
hold off	Deshace el efecto de ***hold on***.

Por ejemplo:
```
>> plot(x)
>> hold on
>> plot(x2,'--')
>> hold off
```

12.4 División de una ventana grafica, creación de subventanas

Una ventana gráfica se puede dividir en **m** particiones horizontales y **n** verticales, con objeto de representar múltiples gráficos en ella. Cada una de estas subventanas tiene sus propios ejes, aunque otras propiedades son comunes a toda la figura. La forma general de este comando es: ***subplot(m,n,i)*** donde **m** y **n** son el número de subdivisiones en filas y columnas, e **i** es la subdivisión que se convierte en activa. Las subdivisiones se numeran consecutivamente empezando por las de la primera fila, siguiendo por las de la segunda, etc.
Por ejemplo, la siguiente secuencia de comandos genera cuatro gráficos en la misma ventana:

```
>> x = 0:pi/25:6*pi;
>> y = sin(x);
>> z = cos(x);
>> w = exp(-x*.1).*y;
>> v=y.*z;
>> subplot(2,2,1), plot(x,y)
>> subplot(2,2,2), plot(x,z)
>> subplot(2,2,3), plot(x,w)
>> subplot(2,2,4), plot(x,v)
```

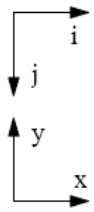


12.5 Control de los ejes

También en este punto ***Matlab*** tiene sus opciones por defecto, que en algunas ocasiones puede interesar cambiar. El comando básico es el comando ***axis***. Por defecto, ***Matlab*** ajusta la escala de cada uno de los ejes de modo que varíe entre el mínimo y el máximo valor de los vectores a representar. Este es el llamado modo ***"auto"***, o modo automático.
Para definir de modo explícito los valores máximo y mínimo según cada eje, se utiliza el comando: ***axis([xmin, xmax, ymin, ymax])*** mientras que : ***axis('auto')***
devuelve el escalado de los ejes al valor por defecto o automático.

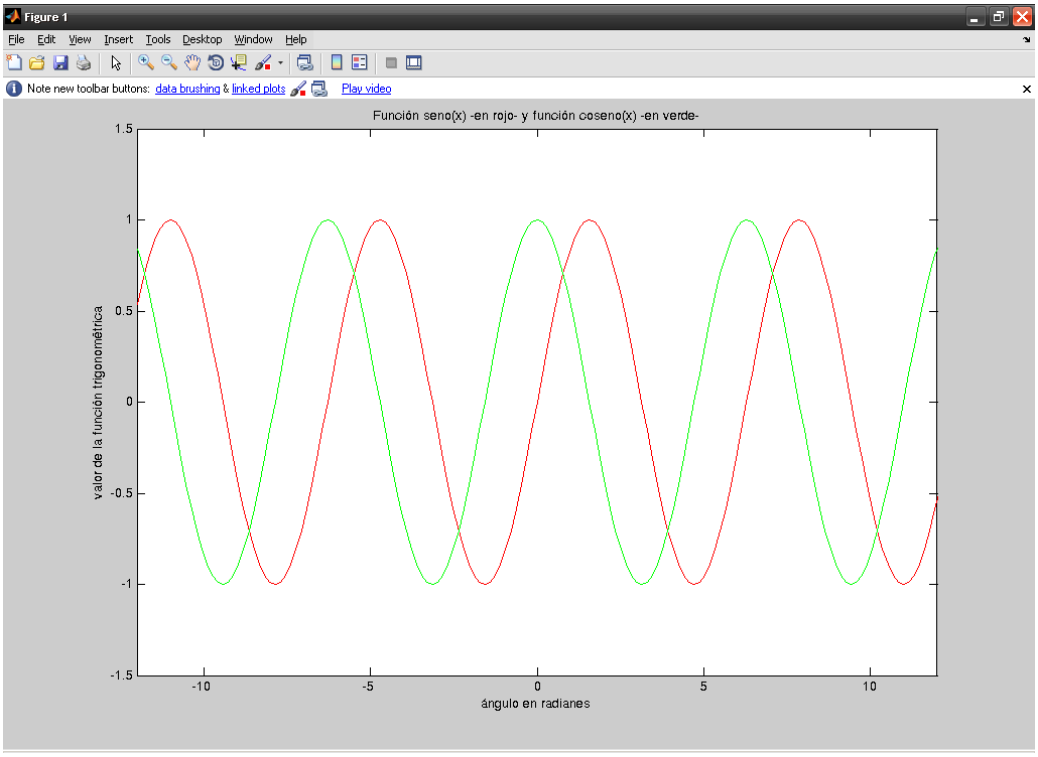
Otros posibles usos de este comando son los siguientes:

v = axis	Devuelve un vector **v** con los valores [xmin, xmax, ymin, ymax]
axis('ij')	Utiliza *ejes de pantalla*, con el origen en la esquina superior izquierda y el eje **j** en dirección vertical descendente
axis('xy')	Utiliza *ejes cartesianos* normales, con el origen en la esquina inferior izquierda y el eje **y** vertical ascendente



axis('auto x')	Utiliza el escalado automático sólo en dirección *x*
axis(axis)	Mantiene los ejes en sus actuales valores, de cara a posibles nuevos gráficos añadidos con ***hold on***
axis('tight')	Establece los mismos límites para los ejes que para los datos
axis('equal')	El escalado es igual en ambos ejes
axis('square')	La ventana será cuadrada
axis('image')	La ventana tendrá las proporciones de la imagen que se desea representar en ella (por ejemplo la de una imagen bitmap que se desee importar) y el escalado de los ejes será coherente con dicha imagen
axis('normal')	Elimina las restricciones introducidas por 'equal' y 'square'
axis('off')	Elimina las etiquetas, los números y los ejes
axis('on')	Restituye las etiquetas, los números y los ejes

```
>> x=[-4*pi:pi/20:4*pi];
>> plot(x,sin(x),'r',x,cos(x),'g')
>> title('Función seno(x) -en rojo- y función coseno(x) -en verde-')
>> xlabel('ángulo en radianes')
>> ylabel('valor de la función trigonométrica')
>> axis([-12,12,-1.5,1.5])
```



12.6 Creación de ventanas gráficas

Si se llama a la función ***figure*** sin argumentos, se crea una nueva ventana gráfica con el número consecutivo que le corresponda. El valor de retorno es dicho número.
Por otra parte, el comando ***figure(n)*** hace que la ventana **n** pase a ser la ventana o figura activa. Si dicha ventana no existe, se crea una nueva ventana con el número consecutivo que le corresponda (que se puede obtener como valor de retorno del comando).

- La función ***close*** cierra la figura activa, mientras que ***close(n)*** cierra la ventana o figura número **n**.
- La función ***clf*** elimina el contenido de la figura activa, es decir, la deja abierta pero vacía.
- La función ***gcf*** (*get current figure*) devuelve el número de la figura activa en ese momento.
- El comando ***figure(gcf)*** permite hacer visible la ventana de gráficos desde la ventana de comandos.

La función ***figure*** también admite que se fijen algunas de sus propiedades, como por ejemplo la posición y el tamaño con que aparecerá en la pantalla. Por ejemplo, el comando:
>> figure('position',[left,botton, width,height])

Abre una ventana cuya esquina inferior izquierda está en el punto **(left,botton)** respecto a la esquina inferior izquierda de la pantalla (en pixels), que tiene una anchura de **width** pixels y una altura de **height** pixels.
Otra característica muy importante de una ventana gráfica es la de representar animaciones utilizando la técnica del ***doble buffer***. De modo sencillo, esta técnica se puede explicar diciendo que es como si la computadora tuviera dos paneles de dibujo: mientras uno está a la vista, se está dibujando en el otro, y cuando el dibujo está terminado este segundo panel se hace visible. El resultado del ***doble buffer*** es que las animaciones y el movimiento se ven de modo perfecto, sin el parpadeo tan característico cuando no se utiliza esta técnica. Para dibujar con ***doble buffer*** en la ventana activa basta ejecutar los comandos siguientes:
>> set(gcf,'DoubleBuffer','on', 'Renderer','painters')

12.7 Funciones LINE, FINDOBJ Y FPLOT

- ***Función LINE***
  La función ***line*** permite dibujar una o más líneas que unen los puntos cuyas coordenadas se pasan como argumentos. Permite además especificar el color, grosor, tipo de trazo, marcador, etc. Es una función de más bajo nivel que la función ***plot***, pero ofrece una mayor flexibilidad.
  >> line([xini, xend], [yini, yend])

Si cada columna de la matriz **X** contiene la coordenada *x* inicial y final de un punto, y lo mismo las columnas de la matriz **Y** con las coordenadas *y*, la siguiente sentencia dibuja tantas líneas como columnas tengan las matrices **X** e **Y**:
>> line([X], [Y]);

- ***Función FINDOBJ***
  Si al dibujar una línea se recupera el valor de retorno de la función ***line*** y se almacena en una variable, más tarde es posible realizar un borrado selectivo de esa línea. Sin embargo, también es posible recuperar la referencia (*handle*) a un determinado elemento gráfico de una figura por medio de la función ***findobj*** a la que se pasan ciertas características del elemento gráfico que permiten su localización.
  Algunos posibles usos de esta función son los siguientes:

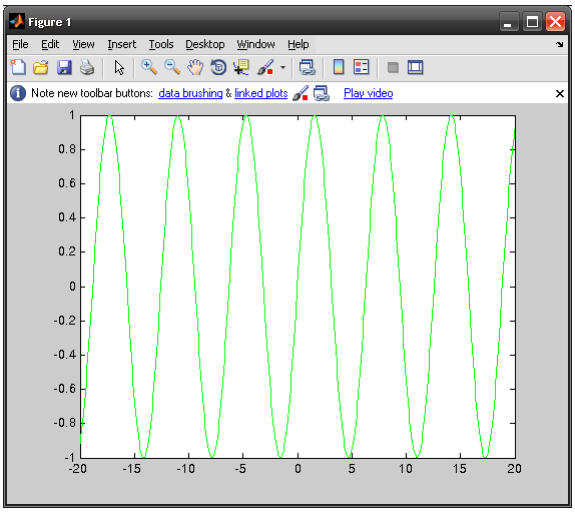
h = findobj	Recupera la referencia del objeto base de la jerarquía gráfica y de todos sus descendientes.
findobj('color','k')	Devuelve las referencias de todos los objetos de color negro.
findobj(gca,'Type','line','Color','b')	Encuentra los objetos de tipo línea y de color azul.

- ***Función FPLOT***
  La función ***plot*** vista anteriormente dibuja vectores. Si se quiere dibujar una función, antes de ser pasada a ***plot*** debe ser convertida en un vector de valores. La función ***fplot*** admite como argumento un *nombre de función* o un *nombre de fichero* ****.m*** en el cual esté definida una función de usuario. La función puede ser escalar (un único resultado por cada valor de **x**) o vectorial. La forma general de esta función es la siguiente:
  >> fplot('funcion', limites, 'cadena', tol)

  donde:

'funcion'	Representa el nombre de la función o del fichero ****.m*** entre apóstrofos
limites	Es un vector de 2 ó 4 elementos, cuyos valores son [**xmin,xmax**] o [**xmin,xmax,ymin,ymax**],
'cadena'	Tiene el mismo significado que en ***plot*** y permite controlar el color, los marcadores y el tipo de línea.
tol	Es la tolerancia de error relativo. El valor por defecto es 2×10^{-3}.

Ejemplo:
>> fplot('sin(x)', [-20 20], 'g')



12.8 Otras funciones graficas 2- D

Existen otras funciones gráficas bidimensionales orientadas a generar otro tipo de gráficos distintos de los que produce la función ***plot*** y sus análogas.
Algunas de estas funciones son las siguientes

bar()	Crea diagramas de barras verticales
barh()	Crea diagramas de barras horizontales
bar3()	Crea diagramas de barras verticales con aspecto 3-D
bar3h()	Crea diagramas de barras horizontales con aspecto 3-D
pie()	Crea gráficos con forma de “torta”
pie3()	Crea gráficos con forma de “torta” con aspecto 3-D
area()	Crea gráficos similares a ***plot***, pero rellenando en ordenadas de 0 a y
stairs()	Crea gráficos similares a ***bar()*** sin líneas internas
errorbar()	Representa sobre un gráfico, mediante barras, valores de errores
compass()	Dibuja los elementos de un vector complejo como un conjunto de vectores partiendo de un origen común
feather()	Dibuja los elementos de un vector complejo como un conjunto de vectores partiendo de orígenes uniformemente espaciados sobre el eje de abscisas
hist()	Dibuja histogramas de un vector
rose()	Dibuja histogramas de ángulos (en radianes)
quiver()	Dibuja campos vectoriales como conjunto de vectores

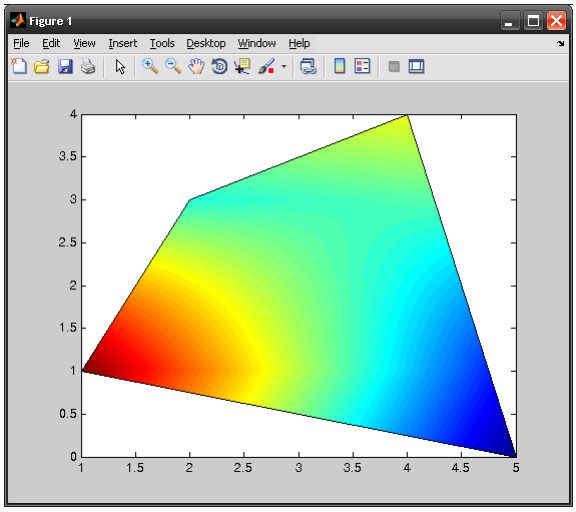
12.9 Función FILL PARA POLÍGONOS

Es una función especial para dibujar polígonos planos, rellenándolos de un determinado color. La forma general es la siguiente:
>> fill(x,y,c)

Dibuja un polígono definido por los vectores **x** e **y**, rellenándolo con el color especificado por **c**. Si es necesario, el polígono se cierra uniendo el último vértice con el primero.

- Si **c** es un carácter de color ('r','g','b','c','m','y','w','k'), o un vector de valores [r g b], el polígono se rellena de modo uniforme con el color especificado.
- Si **c** es un vector de la misma dimensión que **x** e **y**, sus elementos se trasforman de acuerdo con un mapa de colores determinado, y el llenado del polígono (no uniforme en este caso) se obtiene interpolando entre los colores de los vértices.

Ejemplo:
>> x = [1 5 4 2];
>> y = [1 0 4 3];
>> fill(x,y,[1 0.5 0.8 0.7])



Este comando puede utilizarse también con matrices:
>> fill(A,B,C)

Donde **A** y **B** son matrices del mismo tamaño. En este caso se dibuja un polígono por cada par de columnas de dichas matrices. **C** puede ser un vector fila de colores uniformes para cada polígono, o una matriz del mismo tamaño que las anteriores para obtener colores de relleno por interpolación.

12.10 Entrada de puntos con el mouse

Se realiza mediante la función ***ginput***, que permite introducir las coordenadas del punto sobre el que está el cursor, al clickear (o al pulsar una tecla).
Algunas formas de utilizar esta función son las siguientes:

[x,y] = ginput	Lee un número indefinido de puntos (cada vez que se clica o se pulsa una tecla cualquiera) hasta que se termina pulsando la tecla ***enter***
[x,y] = ginput(n)	Lee las coordenadas de **n** puntos

12.11 Creación de películas

Para crear pequeñas películas ***movies*** se pueden utilizar las funciones ***movie***, ***moviein*** y ***getframe***.
Una película se compone de varias imágenes, denominadas ***frames*****.**

- La función ***getframe*** devuelve un vector columna con la información necesaria para reproducir la imagen que se acaba de representar en la figura o ventana gráfica activa, por ejemplo con la función ***plot***. El tamaño de este vector columna depende del tamaño de la ventana, pero no de la complejidad del dibujo.
- La función ***moviein(n)*** reserva memoria para almacenar **n *frames.***
- Una vez creada la película se puede representar el número de veces que se desee con la función ***movie***.

La siguiente lista de comandos crea una película de 17 *frames*, que se almacenarán como las columnas de la matriz **M**. Luego se representara la 10 veces la pelicula a 15 imágenes por segundo.

Ejemplo:

```
>> M = moviein(17);
>> x=[-2*pi:0.1:2*pi]';
>> for j=1:17
>>       y=sin(x+j*pi/8);
>>       plot(x,y);
>>       M(:,j) = getframe;
>> end
>> movie(M,10,15)
```

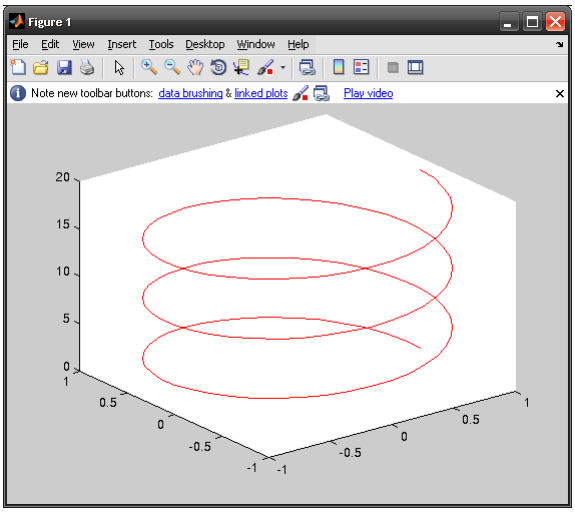
13. Gráficos 3-D

Quizás sea ésta una de las características de ***Matlab*** que más admiración despierta entre los usuarios no técnicos.

13.1 Dibujo de líneas: Función grafica PLOT3

La primera forma de gráfico 3-D es la función ***plot3***, que es el análogo tridimensional de la función ***plot***. Esta función dibuja puntos cuyas coordenadas están contenidas en tres vectores, uniéndolos mediante una línea continua (defecto), o bien mediante *markers*.

Ejemplo:
>> fi = [0:pi/20:6*pi];
>> plot3(cos(fi),sin(fi),fi,'r')

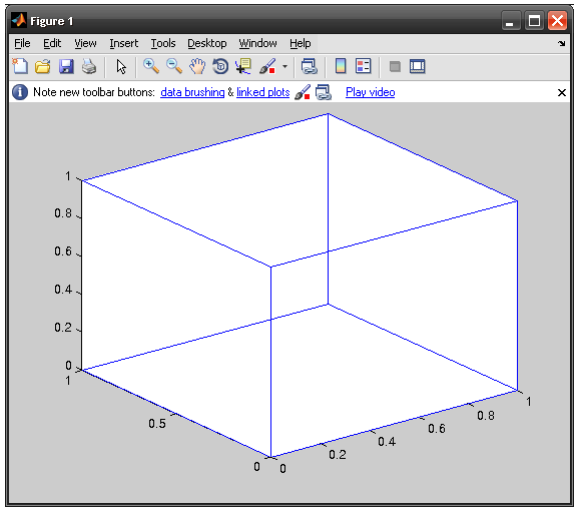


Su forma más sencilla es:
>> plot3(x,y,z)

Esto dibuja una línea que une los puntos (x(1), y(1), z(1)), (x(2), y(2), z(2)), etc. y la proyecta sobre un plano para poderla representar en la pantalla. Al igual que en el caso plano, se puede incluir una cadena de 1, 2 ó 3 caracteres para determinar el color, los markers, y el tipo de línea:
>> plot3(x,y,z,s)

También se pueden utilizar tres matrices **X**, **Y** y **Z** del mismo tamaño en cuyo caso se dibujan tantas líneas como columnas tienen estas 3 matrices, cada una de las cuales está definida por las 3 columnas homólogas de dichas matrices:
>> plot3(X,Y,Z)

Ejemplo:
>> A=[0 1 1 0 0 0 1 0 1 1 0 0 1 1 1 1 0 0
0 0 1 1 0 0 0 0 0 1 1 0 0 0 1 1 1 1
0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 1 1 0];
>> plot3(A(1,:)',A(2,:)',A(3,:)')



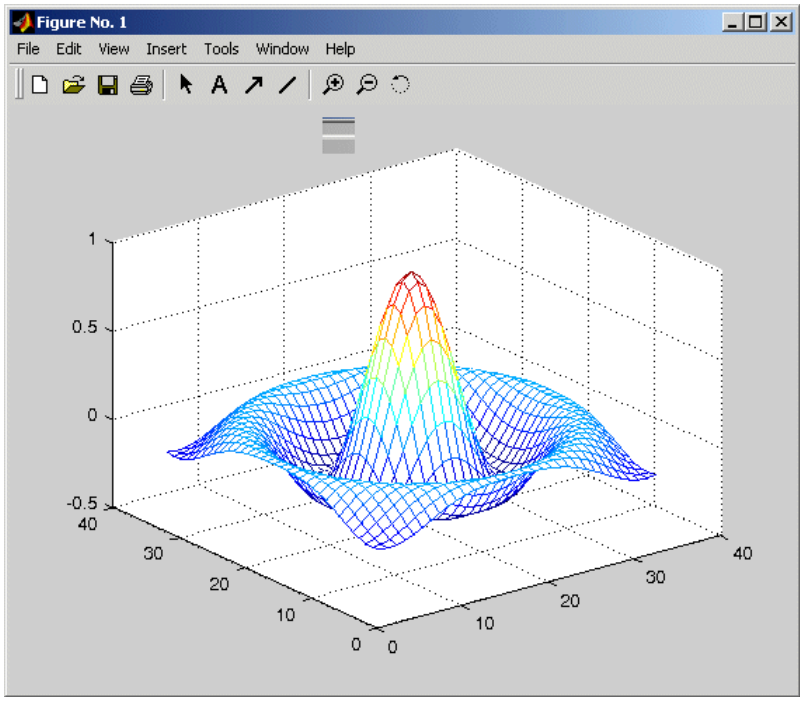
13.2 Dibujo de mallados: Funciones graficas MESHGRID, MESH Y SURF

Para dibujar una función de dos variables (z=f(x,y)) sobre un dominio rectangular, debemos definir:
Sean **x** e **y** dos vectores que contienen las coordenadas en una y otra dirección de la retícula sobre la que se va a dibujar la función.
Después hay que crear dos matrices **X** (cuyas filas son copias de **x**) e **Y** (cuyas columnas son copias de **y**). Estas matrices se crean con la función ***meshgrid***.
Estas matrices representan respectivamente las coordenadas *x* e *y* de todos los puntos de la retícula. La matriz de valores **Z** se calcula a partir de las matrices de coordenadas **X** e **Y**. Finalmente hay que dibujar esta matriz **Z** con la función ***mesh***, cuyos elementos son función elemento a elemento de los elementos de **X** e **Y**.

La función ***mesh*** dibuja en perspectiva una función en base a una *retícula de líneas de colores*, rodeando cuadriláteros del color de fondo, con eliminación de líneas ocultas.

Ejemplo: *sen(r)/r* siendo $r=sqrt(x^2+ y^2)$

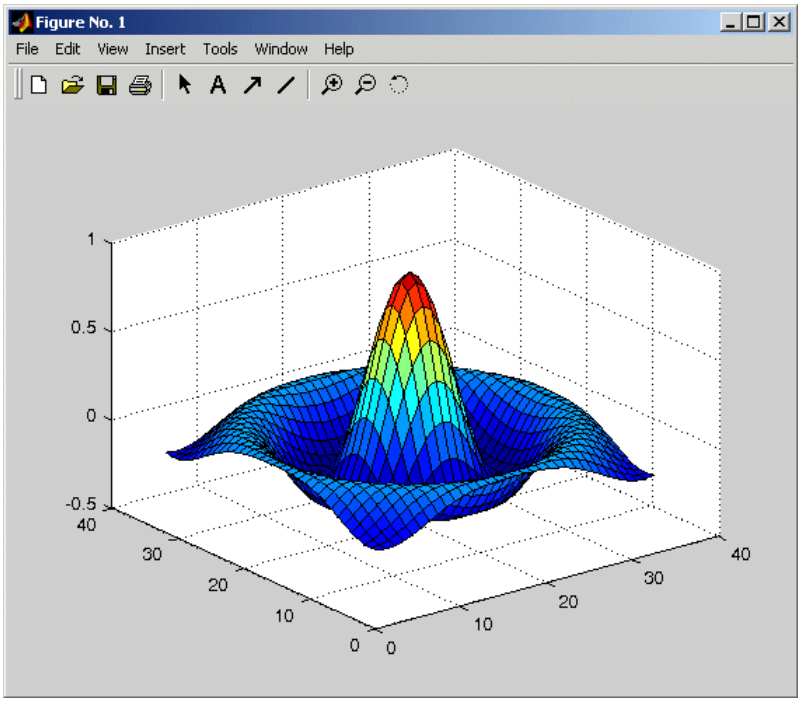
```
>> u = -8:0.5:8;
>> v = u;
>> [U,V] = meshgrid(u,v);
>> R = sqrt(U.^2+V.^2)+eps;
>> W = sin(R)./R;
>> mesh(W)
```



Con la función ***surf*** en vez de líneas aparece ahora una superficie *faceteada*, también con eliminación de líneas ocultas. El color de las facetas depende también del valor de la función.

Ejemplo: *sen(r)/r* siendo $r=sqrt(x^2+y^2)$

```
>> u = -8:0.5:8;
>> v = u;
>> [U,V] = meshgrid(u,v);
>> R = sqrt(U.^2+V.^2)+eps;
>> W = sin(R)./R;
>> surf(W)
```



La función ***surf*** tiene diversas posibilidades referentes a la forma en que son representadas las facetas o polígonos coloreados. Las tres posibilidades son las siguientes:

- ***shading flat*** determina sombreado con color constante para cada polígono. Este sombreado se llama plano o *flat*.
- ***shading interp*** establece que el sombreado se calculará por interpolación de colores entre los vértices de cada faceta. Se llama también sombreado de *Gouraud*
- ***shading faceted*** consiste en sombreado constante con líneas negras superpuestas. Esta es la opción por defecto

Por defecto, las funciones ***mesh*** y ***surf*** atribuyen color a los bordes y facetas en función de los valores de la función, es decir en función de los valores de la matriz **Z**. Ésta no es sin embargo la única posibilidad. En las siguientes funciones, las dos matrices argumento **Z** y **C** tienen el mismo tamaño:

>> mesh(Z,C)
>> surf(Z,C)

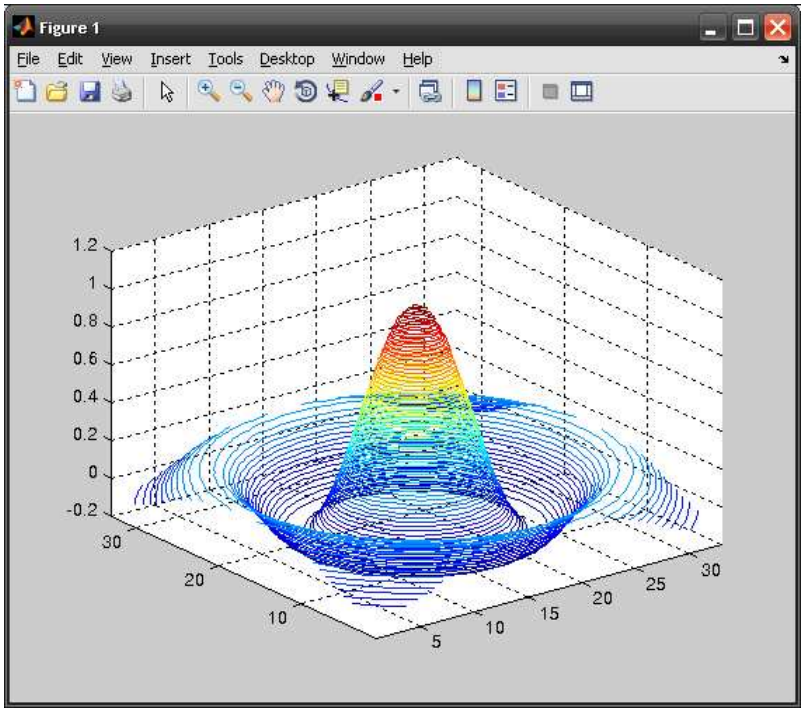
Como resultante, mientras se dibujan los valores de **Z**, los colores se obtienen de **C**.

13.3 Dibujo de líneas de contorno: Función CONTOUR3

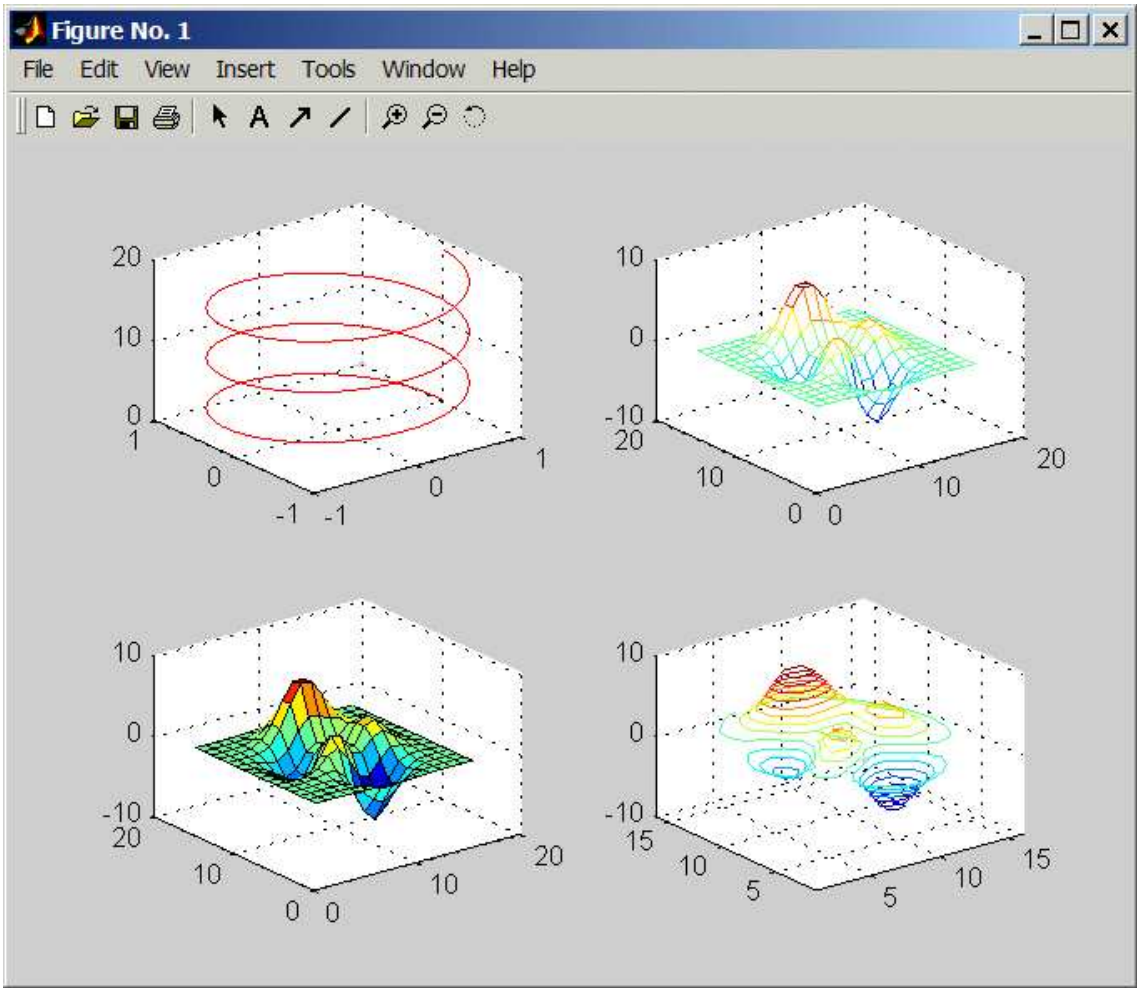
Una forma distinta de representar funciones tridimensionales es por medio de isolíneas o curvas de nivel. La expresión es ***contour3(Z, n***) siendo ***n*** el número de líneas de nivel a dibujar. Si no se pone se utiliza un número por defecto.

Ejemplo: *sen(r)/r* siendo $r=sqrt(x^2+ y^2)$

```
>> u = -8:0.5:8;
>> v = u;
>> [U,V] = meshgrid(u,v);
>> R = sqrt(U.^2+V.^2)+eps;
>> W = sin(R)./R;
>> contour3(W,60)
```



Resumiendo, tenemos el siguiente ejemplo:




```
>> x = [-3:0.4:3];
>> y = x;
>> subplot(2,2,1)
>> figure(gcf)
>> fi=[0:pi/20:6*pi];
>> plot3(cos(fi),sin(fi),fi,'r')
>> grid
>> [X,Y] = meshgrid(x,y);
>> Z = test3d(X,Y);
>> subplot(2,2,2)
>> figure(gcf)
>> mesh(Z)
>> subplot(2,2,3)
>> figure(gcf)
>> surf(Z)
>> subplot(2,2,4)
>> figure(gcf)
>> contour3(Z,16)
```

En donde la función ***test3d*** aplica la siguiente formula:

$$z = 3(1-x)^2 e^{-x^2-(y+1)^2} - 10\left(\frac{x}{5} - x^3 - y^5\right)e^{-x^2-y^2} - \frac{1}{3}e^{-(x+1)^2-y^2}$$

13.4 Mapas de colores

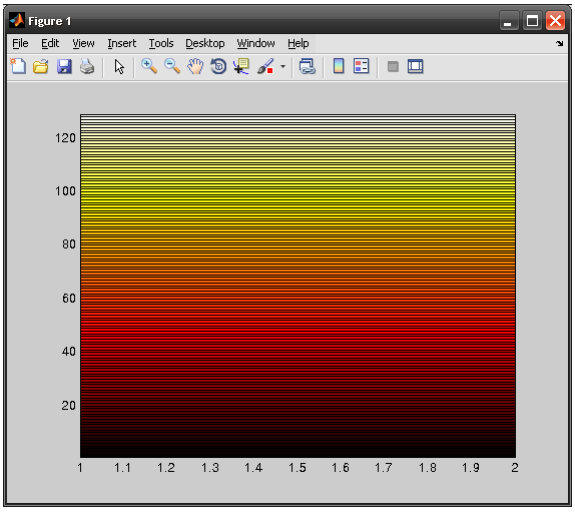
Un ***mapa de colores*** se define como una matriz de tres columnas, cada una de las cuales contiene un valor entre 0 y 1, que representa la intensidad de uno de los colores fundamentales: R (red o rojo), G (green o verde) y B (blue o azul).
La longitud por defecto de los mapas de colores de ***Matlab*** es 64, es decir, cada mapa de color contiene 64 colores. Esta longitud puede modificarse.
Algunos mapas de colores están predefinidos en ***Matlab***. Esto puede verse con la función ***colormap.*** Esta actúa sobre la figura activa, cambiando sus colores. Si no hay ninguna figura activa, sustituye al mapa de color anterior para las siguientes figuras que se vayan a dibujar.
El ***colormap*** por defecto es ***jet***. Para visualizar estos mapas de colores, cambiando al mismo tiempo su longitud, se pueden utilizar los comandos:

```
>> colormap(hot(128))
```

hot(128) genera una matriz de 128 elementos con el mapa de colores: negro – rojo – amarillo - blanco

```
>> pcolor([1:129;1:129]')
```

En donde la función ***pcolor*** permite visualizar por medio de colores la magnitud de los elementos de una matriz. La función ***pcolor*** es en cierta forma equivalente a la función ***surf*** con el punto de vista situado perpendicularmente al dibujo.



Cuando se desea dibujar una figura con un determinado mapa de colores se establece una correspondencia entre los valores de la función y los colores del mapa de colores. Esto hace que los valores pequeños se dibujen con los colores *bajos* del mapa, mientras que los valores grandes se dibujan con los colores *altos*.

Ejemplo:

```
>> A = rand(100,100);
>> colormap(hot);
>> pcolor(A);
>> pause(5)
>> pcolor(inv(A));
```

La función ***caxis*** permite ajustar manualmente la escala de colores. Su forma general es: **caxis([cmin, cmax])** donde ***cmin*** y ***cmax*** son los valores numéricos a los que se desea ajustar el mínimo y el máximo valor de la escala de colores.

13.5 Formas parametricas de las funciones MESH, SURF Y PCOLOR

Existen unas formas más generales de las funciones ***mesh***, ***surf*** y ***pcolor***. La principal ventaja de estas nuevas formas de las funciones ya conocidas es que es que admiten mayor variedad en la forma de representar la cuadrícula en el plano (x-y).
Por ejemplo, la función **mesh(X,Y,Z,C)** dibuja una superficie cuyos puntos tienen como coordenadas (X(i,j), Y(i,j), Z(i,j)) y como color C(i,j). Las cuatro matrices deben ser del mismo tamaño.

13.6 Otras funciones 3-D

surfc	Combinación de ***surf***, y ***contour*** en z=0
trisurf	Similar a ***surf***, dibuja una superficie 3-D a partir de los valores de una función en una malla de triángulos.
meshz	Similar a ***mesh*** con plano de referencia en el valor mínimo y una especie de “cortina” en los bordes del dominio de la función
trimesh	Similar a ***mesh***, dibuja una superficie 3-D a partir de los valores de una función en una malla de triángulos.
surfl	Controla la iluminación determinando la posición e intensidad de un foco de luz.
light	Crea un foco de luz en los ejes actuales capaz de actuar sobre superficies 3-D.
sphere	Dibuja una esfera 3-D de radio unidad. Por defecto se utiliza un faceteado de 20 (20 meridianos y 20 paralelos). Este número se puede cambiar. Es posible recoger las coordenadas como valor de retorno y multiplicarlas por un factor de escala.
cylinder	Dibuja una superficie cilíndrica de radio 1 y altura 1, con 20 facetas laterales. Este número se puede cambiar, como segundo argumento. El primer argumento puede ser un vector que indica como varía el radio en función de la altura del cilindro. También es posible recoger las coordenadas como valor de retorno y multiplicarlas por un factor de escala.

13.6 Elementos generales

Las funciones ***surf*** y ***mesh*** dibujan funciones tridimensionales en perspectiva. La localización del punto de vista o dirección de observación se puede hacer mediante la función ***view***, que tiene la siguiente forma:

view(azimut, elev)

En donde ***azimut*** es el ángulo de rotación de un plano horizontal, medido sobre el eje **z** a partir del eje **x** en sentido antihorario, y ***elev*** es el ángulo de elevación respecto al plano (x-y). Ambos ángulos se miden ***en grados***, y pueden tomar valores positivos y negativos (sus valores por defecto son -37.5 y 30).

También se puede definir la dirección del punto de vista mediante las tres coordenadas cartesianas de un vector (sólo se tiene en cuenta la dirección):

view([xd,yd,zd])

En los gráficos tridimensionales existen funciones para controlar los ejes, por ejemplo:
axis([xmin,xmax,ymin,ymax,zmin,zmax])
axis([xmin xmax ymin ymax zmin zmax cmin cmax])

También se pueden utilizar las funciones siguientes:
xlabel, ***ylabel***, ***zlabel***, ***xlim***, ***ylim***, ***zlim***, ***axis('auto')***, ***axis(axis)***, etc.

Las funciones ***mesh*** y ***surf*** disponen de un algoritmo de *eliminación de líneas ocultas* (los polígonos o facetas, no dejan ver las líneas que están detrás). El comando ***hidden*** activa y desactiva la eliminación de líneas ocultas.