TEMA 2

Gráficos en MATLAB

1. Introducción **1**
2. Objetos gráficos **3**
2.1 Objetos LINE *4*
2.2 Objetos TEXT *9*
2.3 Objetos PATCH *10*
2.4 Objetos SURFACE *11*
2.5 Objetos LIGHT *14*
2.6 Objetos IMAGE *14*
3. Gráficos específicos **17**
3.1 Gráficos para presentaciones *17*
3.2 Probabilidad y estadística *19*
3.3 Respuesta frecuencial de sistemas lineales *25*
3.4 Respuesta temporal de sistemas lineales *25*
3.5 Otras funciones relacionadas con la teoría de sistemas *26*
4. Animaciones **27**

1. Introducción

El objetivo de este tema es presentar una panorámica de las capacidades gráficas del programa MATLAB.

Para ver las funciones relacionadas con los objetos gráficos de MATLAB, teclear:

```
>> help graph2d    %representaciones en dos dimensiones (2D)
>> help graph3d    %representaciones en tres dimensiones (3D)
>> help specgraph  %gráficos especiales

>> help graphics   %comandos de bajo nivel
```

El sistema gráfico de MATLAB permite:

- Presentar gráficamente datos mediante comandos de alto nivel. Esto incluye la visualización en 2 y 3 dimensiones, el procesado de imágenes/fotos, la elaboración de gráficos para presentaciones (diagramas de barras o de queso) y la inclusión de efectos (de animación, iluminación, movimientos de cámara).

- Crear y manipular objetos gráficos mediante comandos de bajo nivel. Esto se lleva a cabo mediante las utilidades GUI (*Graphics User Interface*) y permite el diseño de aplicaciones complejas, con ventanas, menús y controles. Por ejemplo, es posible programar ventanas como las que aparecen al invocar la herramienta gráfica (**`>>cftool`**) de la *Curve Fitting Toolbox*:

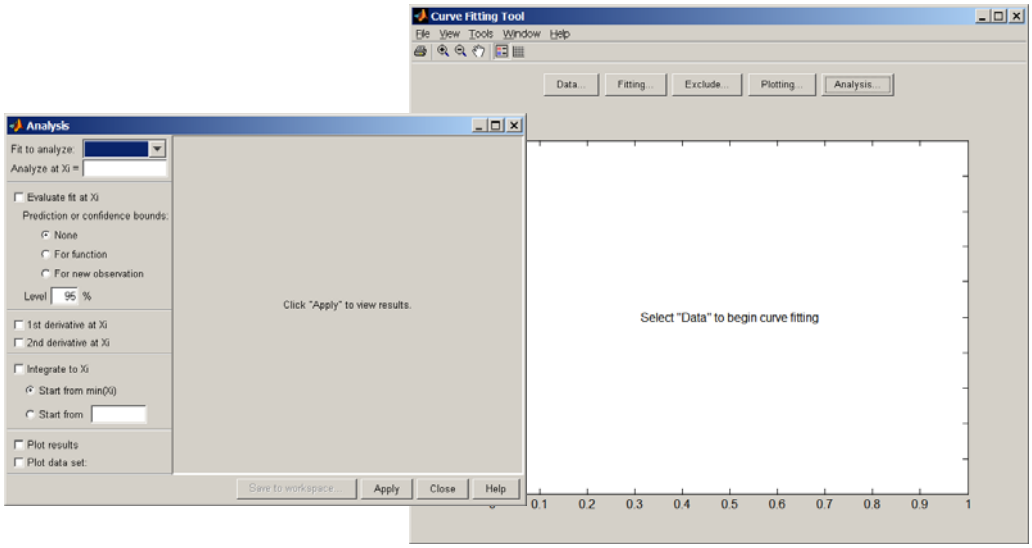


Fig. 1. Ejemplo de interficie gráfica de usuario

En el primer caso se trabaja a nivel de usuario mientras que en el segundo se trabaja a nivel de programador. Este segundo caso corresponde a un nivel más avanzado en el uso del MATLAB y se tratará en el Tema 4, aunque en este tema sí se verá una introducción a sus dos funciones centrales (**`get`** y **`set`**).

2. Objetos gráficos

Los objetos gráficos de MATLAB presentan la siguiente jerarquía básica:

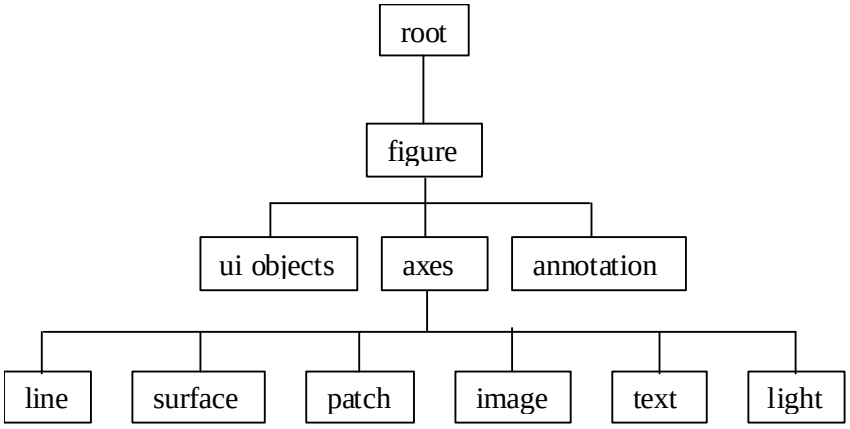


Fig. 2. Jerarquía de los objetos gráficos

(Nota: Hay más objetos y grupos de objetos pero aquí no se muestran a fin de dar más claridad a la exposición. Para más detalles consultar "graphics objects" en el help de MATLAB)

El objeto **`root`** es la ventana de comandos. Ningún objeto de los mostrados en el esquema anterior puede existir si no existe **`root`** (es otras palabras, si no está abierto el programa MATLAB no podemos tener ni **`figure`**'s, ni **`axes`**, etc.).

Cuando en **`root`** hacemos **`>>plot(t,y)`** se crean todos los objetos necesarios para la representación (**`figure`** → **`axes`** → **`line`**), si es que antes no existían.

```
>> x=0:10;
>> plot(x,x)
```

Handle: Todos los objetos gráficos tienen un *handle* (número que lo identifica) y un conjunto de propiedades (**`Color`**, **`Position`**, etc.). Algunas de las propiedades pueden modificarse pero otras no. El *handle* del objeto **`root`** es 0.

Para obtener el *handle* de un objeto **`figure`** se usa la función **`gcf`** (*get current figure*).

```
>> gcf
ans =
     1
```

Para obtener el *handle* de un objeto **`axes`** se usa la función **`gca`** (*get current axes*).

```
>> gca
ans =
  158.0017
```

Para ver todas la propiedades de un objeto se usa **`get`**, p. ej.: **`get(gca)`**

```
>> get(gca)
       ActivePositionProperty = outerposition
       ALim = [0 1]
        ⋮
       ZTickMode = auto

       BeingDeleted = off
        ⋮
       Visible = on
```

Para cambiar las propiedades de un objeto se usa **`set`**, por ejemplo:

```
>> set(gcf,'NumberTitle','off')
>> set(gcf,'Color','r')
```

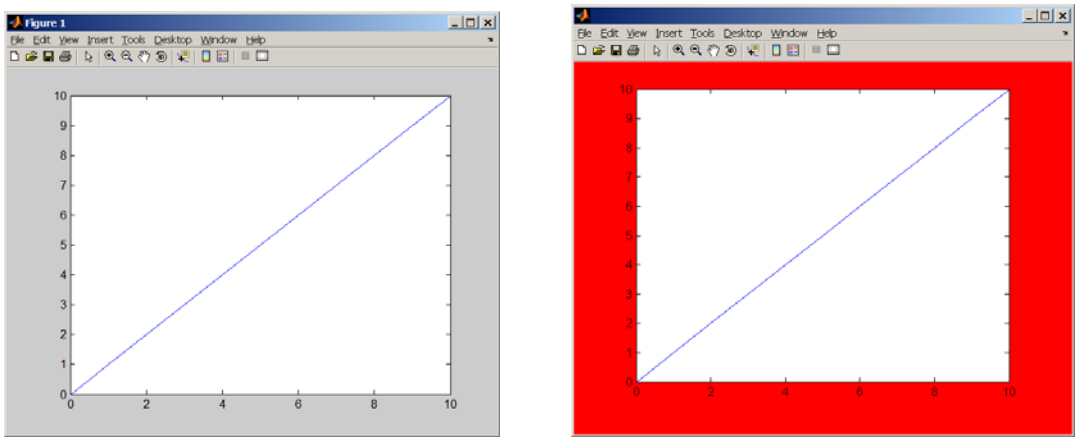


Fig. 3. Uso de los comandos de bajo nivel **`get`** y **`set`**.

Todas estas opciones también pueden consultarse y modificarse desde la barra de menús de las figuras. Por ejemplo, desde **`Edit → Figure Properties…`**.

2.1 Objetos LINE

Representación en 2D: Los pasos generales para la representación de curvas en dos dimensiones son los siguientes (notar, sin embargo, que muchas de las funciones de las *toolboxes* ya llevan incorporados automáticamente algunos de estos pasos con lo que no hace falta que el usuario los ejecute explícitamente):

Paso 1) Eje x: Crear un vector con los valores del eje de abscisas

Funciones: dos puntos (`:`), **`linspace`**, **`logspace`**

Por ejemplo:

>>x=0:.2:12; (nota: Valor inicial : Distancia entre valores : Valor final)

o bien

>>x=linspace(0,12,200); (nota: 200 valores espaciados uniformemente entre el primero, 0, y el último, 12)

Si necesitamos escalas logarítmicas (por ejemplo, en el caso de los diagramas de Bode), se utiliza **logspace**:

>>w=logspace(-1,3); (nota: 50 valores espaciados logarítmicamente entre 10^{-1}=0.1 y 10^{3}=1000)

Notar que tanto en **linspace** como en **logspace** el tercer argumento de entrada (número de puntos) es opcional. El valor por defecto es 100 y 50 respectivamente. Asimismo, en la función **:**, si no se especifica la distancia entre valores, el paso por defecto es 1 (p. ej.: **>>x=1:3** genera un vector fila cuyas componentes son **1 2 3**)

Paso 2) Eje y: Crear un vector con los valores correspondientes al eje de ordenadas.

Las dimensiones de los vectores **x** e **y** deben ser iguales. De hecho, y en general, **y** se calcula a partir de **x**, con lo cual la compatibilidad de dimensiones está garantizada.

Por ejemplo,

```
>>y1=bessel(1,x);
>>y2=bessel(2,x);
>>y3=bessel(3,x);
```

Por otro lado, si quisiéramos representar un valor constante a lo largo del eje x, lo que podemos hacer es **>>plot(x,2*ones(size(x)))** ya que la función **ones** crea un vector con las mismas dimensiones que **x** pero cuyas componentes son todo "unos". Otra opción sería **>>plot(x,x*0+2)**.

Paso 3) Representación: Ejecutar una instrucción gráfica

Funciones: **plot**, **semilogx** (eje x en logarítmico y eje y en lineal), **semilogy**, **loglog**, **polar**, **plotyy** (para tener dos ejes de ordenadas), **stem**, **stairs**.

Por ejemplo:

```
>>plot(x,y1,x,y2,x,y3)
```

En el caso de números complejos (por ejemplo **>> n=3+j*5;**), hacer **>>plot(n,'x')** es equivalente a hacer **>>plot(real(n),imag(n),'x')**.

Paso 4) Ajuste (ampliación/reducción): Si es necesario, se pueden cambiar los valores inicial y final de los ejes de la representación.

Funciones: **`axis`**, **`zoom`** (**`zoom on`** y **`zoom off`**).

```
>>axis([xmin xmax ymin ymax])
```

También es posible:

```
>>axis square
>>axis normal
```

La función axis tiene muchas opciones (auto, manual, normal, square, ij, xy, tight, on, off)

Paso 5) Rejilla: Para poner una rejilla la instrucción es **`grid`** (también **`grid on`** y **`grid off`**). Se pueden poner o quitar las líneas superior y derecha (**`box on, off`**) e incluso los ejes (**`axis on, off`**)

```
>> th=linspace(0,2*pi,101);
>> x=sin(th);
>> y=sin(2*th+pi/4);
>> plot(x,y,'k-')
>> grid on
>> box off
>> axis off
```

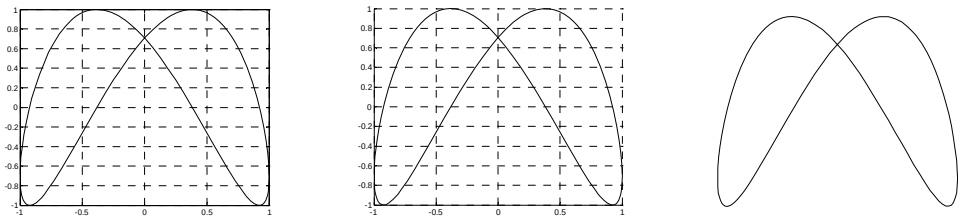


Fig. 4. Ejes y rejilla

Paso 6) Tipo de trazo: Se pueden usar trazos continuos/discontinuos, diversos colores y diversos símbolos. Se recomienda hacer **`>>help plot`** para ver las opciones disponibles. Las diferentes opciones se pueden combinar entre sí (el orden no importa) y se invocan entre apóstrofes (**`'*r'`**).

Colores: **`'r'`**, **`'b'`**, **`'g'`**, **`'y'`**, **`'k'`**, **`'m'`**, **`'w'`**, **`'c'`**,…
Tipo de trazo: **`'-'`**, **`'-.'`**, **`'--'`**, **`':'`**, **`'.'`**, **`'o'`**, **`'*'`**, **`'x'`**, **`'+'`**,…
Símbolos: **`'s'`**, **`'h'`**, **`'p'`**, **`'d'`**,…

Por ejemplo,

```
>>plot(t,y1,t,y2, 'r--',t,y2, '.-g')
```

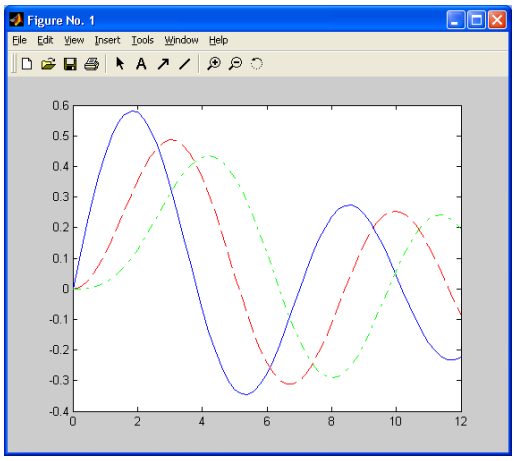


Fig. 5. Tipos de trazo

Paso 7) Superposición: Para superponer gráficas, también es posible utilizar la función **`hold`** (también **`hold on`**, **`hold off`**)

```
>>plot(x,y1),hold
Current plot held
>>plot(x,y2)
>>plot(x,y3)
>>hold
Current plot released
```

O bien

```
>>plot(t,y1),hold on
>>plot(t,y2)
>>plot(t,y3),hold off
```

Paso 8) División de la ventana gráfica Para subdividir la pantalla se utiliza **`subplot(a,b,c)`** donde **`a`** corresponde al número de filas, **`b`** al número de columnas y **`c`** a la subgráfica actual (de **`1`** a **`a`**×**`b`**). Por ejemplo,

Dos gráficas:

```
>> subplot(212),plot(x,y2)
>> subplot(211),plot(n,'og')
```

Cuatro gráficas:

```
>> x=linspace(0,12);y1=bessel(1,x);y2=bessel(2,x);y3=bessel(3,x);
>> subplot(221),plot(x,y1)
>> subplot(222),plot(x,y2)
>> subplot(223),plot(x,y3)
>> subplot(224),plot(x,x)
```

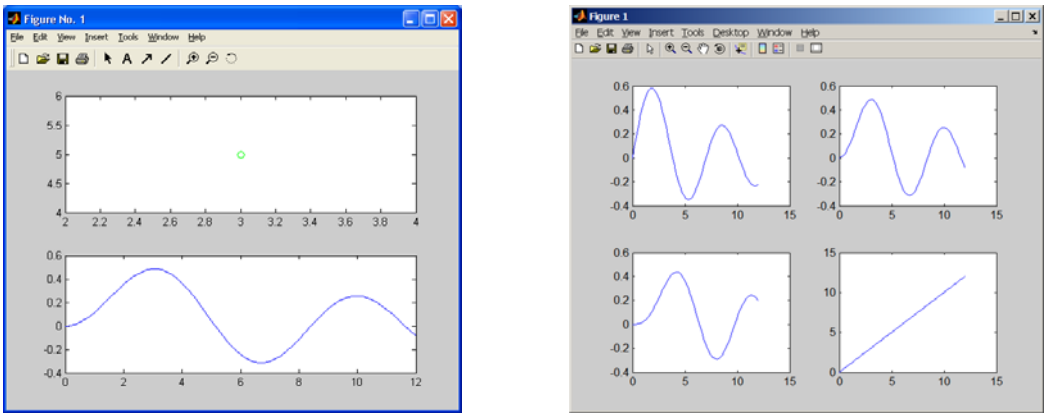


Fig. 6. Subgráficas con **`subplot`**

Paso 9) Captura de coordenadas: Para capturar los valores de las coordenadas *x, y* de una representación se utiliza la función **`ginput`** (*graphics input*). Para usarla basta con hacer:

```
>>ginput
```

con lo que aparece un cursor sobre la gráfica actual. Con el ratón se van capturando puntos y cuando se tienen los suficientes se pulsa <*Enter*> para volver a la ventana de comandos y ver las coordenadas capturadas.

Paso 10) Salvar y recuperar figuras: Para guardar un objeto **`figure`** seleccionar, en la barra de menús las opciones **`File → Save`**. La figura se guardará en un fichero de extensión **`*.fig`**, por ejemplo, **`figu.fig`**. Para recuperar la figura, basta con teclear

```
>> openfig('figu')
```

Representación en 3D: Los pasos para la representación de curvas en tres dimensiones son los mismos que en el caso de curvas en 2D. La única diferencia es que hay que usar instrucciones de representación en 3D. Éstas, en general, son las mismas que para 2D pero llevan el número 3 al final del nombre (**`plot3`**, **`comet3`**, **`stem3`**).

Por ejemplo, **`>> plot3(y1,y2,y3)`**

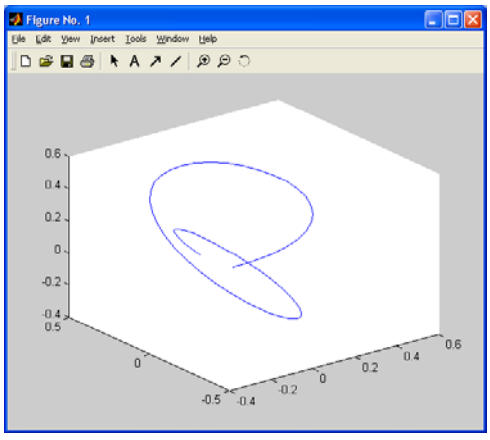


Fig. 7. Objeto LINE en 3D

Los ejes pueden modificarse con **axis**:

```
axis([xmin xmax ymin ymax zmin zmax])
```

2.2 Objetos TEXT

Suponer de nuevo la representación de las tres funciones de Bessel. Para etiquetar la representación se pueden utilizar las funciones **xlabel**, **ylabel**, **zlabel**, **title**, **text**, **gtext**, **legend**. Observar cómo se indica lo que son subíndices y superíndices.

```
>> xlabel('x')
>> ylabel('y_1 , y_2 , y_3')
>> title('Funciones de Bessel de primera especie')
>> legend('1^e^r orden','2^o orden','3^{er} orden',-1)
>> text(6,0.5,'hoolaaa')
```

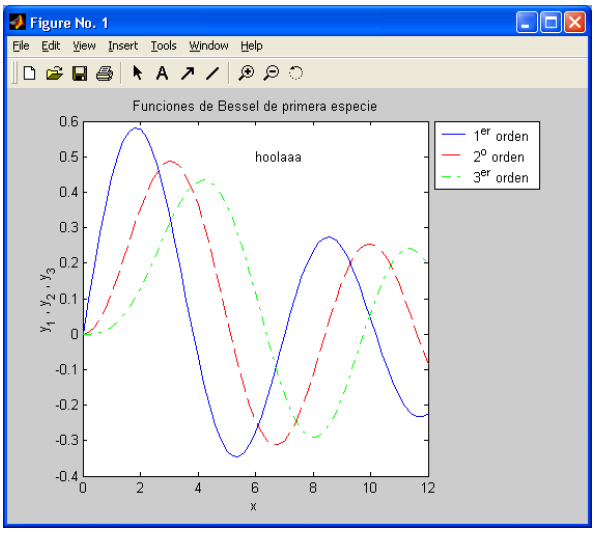


Fig. 8. Objetos TEXT

La función **gtext** (*graphics text*) hace lo mismo que **text** pero inserta el texto allá donde se clica con el ratón, en vez de indicarle las coordenadas. Si se quiere escribir varias líneas en un **gtext**, se hace así:

```
>> gtext({'y_1: 1^e^r orden','y_2: 2^o orden','y_3: 3^e^r orden'});
```

También es posible usar letras del alfabeto griego. Basta con poner el nombre de la letra detrás del símbolo \: **\alpha**, **\beta**,...

```
>> title('y_1(\phi)=(\phi-sin(\phi))/2');
```

Se recomienda hacer **>>help TeX**. Esta utilidad permite también escribir expresiones matemáticas (fracciones, raíces cuadradas,…).

Finalmente, también es posible insertar otros símbolos, como por ejemplo flechas:

```
>> text(0.3,0.4,'\downarrow','FontSize',10)
```

Otra función útil para usar dentro de un objeto **`text`** es **`num2str`** (*number to string*):

```
>> r=2.5;
>> text(0.4,0.3,['radio = ',num2str(r)])
```

2.3 Objetos PATCH

Son objetos compuestos por uno o más polígonos que pueden estar o no conectados. Es útil para presentaciones y animaciones (puesto que permite dibujar objetos complejos con múltiples caras).

Las tres funciones básicas son **`fill`**, **`fill3`** y **`patch`**. En ellas hay que especificar los vértices del polígono y el color de relleno.

El orden de especificación de los vértices es importante. Ver el siguiente ejemplo:

```
>> x=[3 3 7 7];
>> y=[5 6.5 6.5 5];
>> fill(x,y,'r')
>> axis([0 10 4 7.5])
```

```
>> x=[3 3 7 7];
>> y=[5 6.5 5 6.5];
>> fill(x,y,'r')
>> axis([0 10 4 7.5])
```

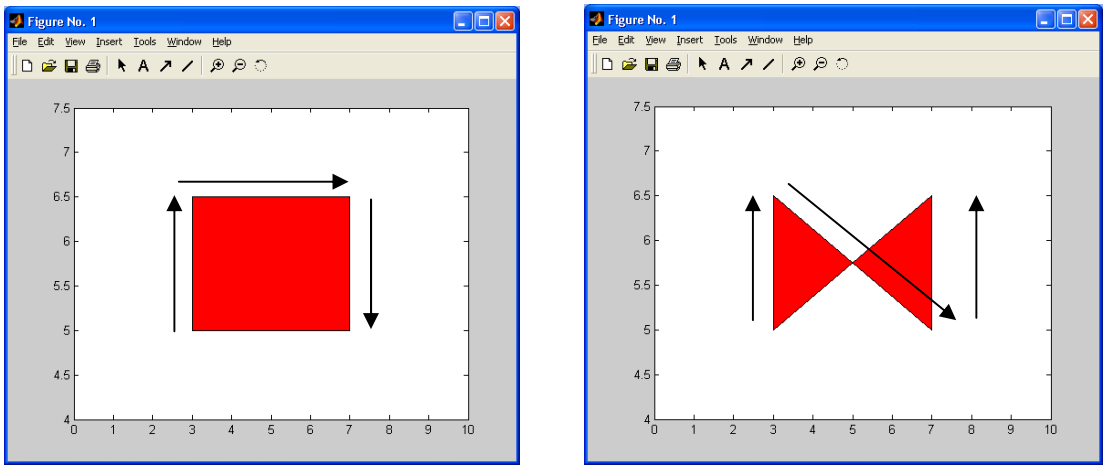


Fig. 9. Orden de los vértices en objetos PATCH

También es posible hacer **`fill(x1,y1,c1,x2,y2,c2,...)`**.

Con respecto al color, es posible usar colores predefinidos o crear nuevos colores a partir de ternas **`[r g b]`** donde las tres componentes de dicha terna varían entre 0 y 1.

Ejemplo:

Rojo oscuro = [0.5 0 0]
Cobre = [1 .62 .4]
Gris = [0.5 0.5 0.5]

Predefinidos:

Rojo = [1 0 0]	'r'	Verde = [0 1 0]	'g'

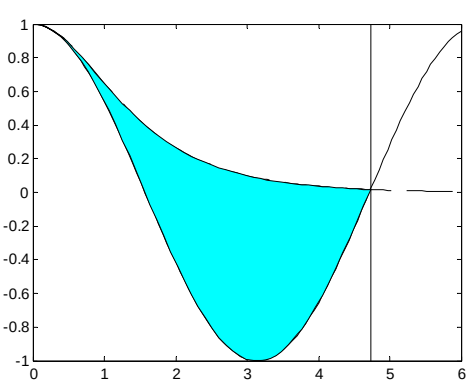
Azul = [0 0 1]	'b'	Cyan = [0 1 1]	'c'
Magenta = [1 0 1]	'm'	Amarillo = [1 1 0]	'y'
Negro = [0 0 0]	'k'	Blanco = [1 1 1]	'w'

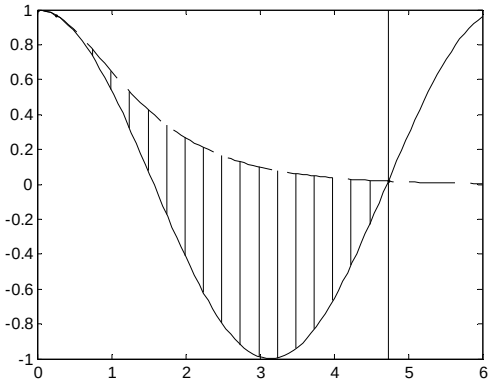
También existen mapas de colores (**`colormap`**) **`hsv`**, **`hot`**, **`cool`**, **`summer`**, **`gray`**, **`jet`**.

Ejemplo 1. Objetos PATCH

El siguiente ejemplo muestra cómo sombrear un área (Magrab,05) :

```
>> x=linspace(0,6,100);
>> plot(x,cos(x),'k-',x,1./cosh(x),'k--',[4.73 4.73],[-1 1],'k')
>> hold on
>> xn=linspace(0,4.73,50);
>> fill([xn,fliplr(xn)],[1./cosh(xn),fliplr(cos(xn))],'c')
```





```
>> x=linspace(0,6,100);
>> plot(x,cos(x),'k-',x,1./cosh(x),'k--',[4.73 4.73],[-1 1],'k')
>> hold on
>> xx=linspace(0,4.73,20);
>> plot([xx;xx],[cos(xx);1./cosh(xx)],'k-')
```

2.4 Objetos SURFACE

Para la representación de superficies en 3 dimensiones se procede como en los objetos LINE.

En primer lugar se definen los valores de los ejes x, y

```
>> x=-10:0.1:10;
>> y=x;
```

A continuación, se combinan los valores de **`x`** e **`y`** (vectores) a fin de tener una rejilla (matriz) sobre la que representar las coordenadas **`z`**.

```
>> [xx,yy]=meshgrid(x,y);
```

Las coordenadas **z** se calculan sobre las coordenadas cruzadas **xx**, **yy**:

```
>> z=xx.^3+yy.^3+2*xx.*yy;
```

Finalmente, se representa la superficie (con **mesh**, **surf**, **surface**, **waterfall**). Si se quiere cambiar la orientación se puede usar la función **view**.

```
>> mesh(x,y,z)
```

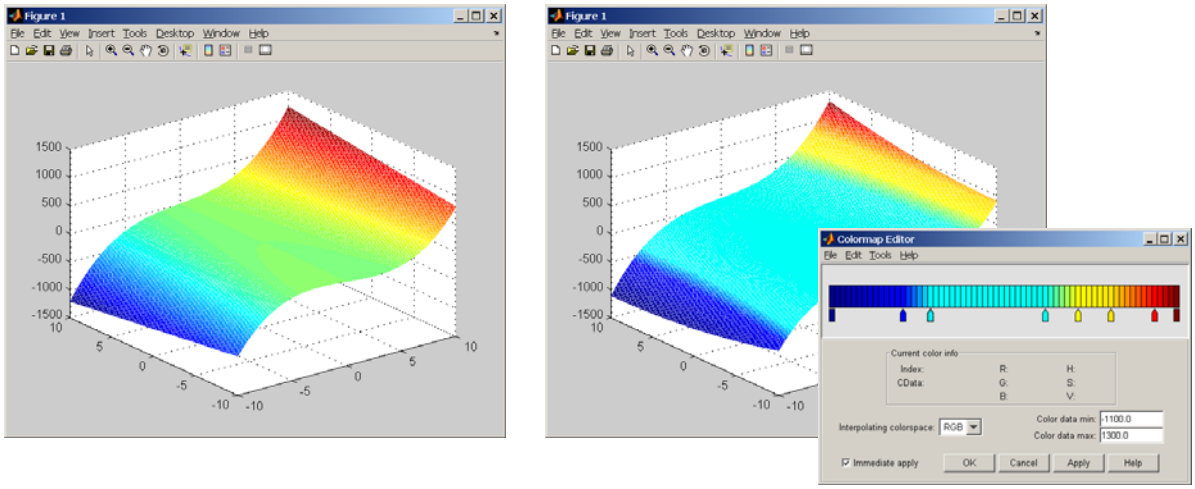


Fig. 10. Objetos SURFACE.

También se pueden obtener curvas de nivel con **contour** o **meshc** (que combina la superficie con las curvas de nivel). Los diagramas de contorno se rotulan con **clabel**.

Se puede cambiar el color de la representación con **colormap** (por ejemplo **>>colormap gray**), **shading**, **hidden**, **brighten**. También se puede cambiar desde la barra de menús (*Edit* → *Colormap*…)

Ejemplo 2. Objetos SURFACE

Las demos de MATLAB tienen ejemplos como el siguiente:

```
z = peaks;
surf(z); hold on
shading interp;
[c ch] = contour3(z,20); set(ch, 'edgecolor', 'b')
[u v] = gradient(z);
h = streamslice(-u,-v);  % downhill
set(h, 'color', 'k')
for i=1:length(h);
  zi = interp2(z,get(h(i), 'xdata'), get(h(i),'ydata'));
  set(h(i),'zdata', zi);
end
view(30,50); axis tight
```

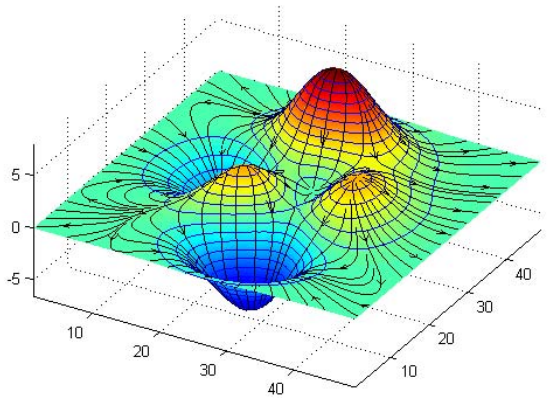


Fig. 11. Objetos SURFACE (función **`peaks`**).

Se pueden controlar los ejes con **`axis`**, **`zoom`**, **`grid`**, **`box`**, **`hold`**, **`axes`**, **`subplot`**. Se puede rotar la representación con **`rotate3d`**, **`viewmtx`** o **`view`** (o con ayuda de **`View`** → **`Camera Toolbar`**.) Se sugiere explorar las posibilidades del menú de la ventana de Figura.

```
>> z=peaks;surf(z)
```

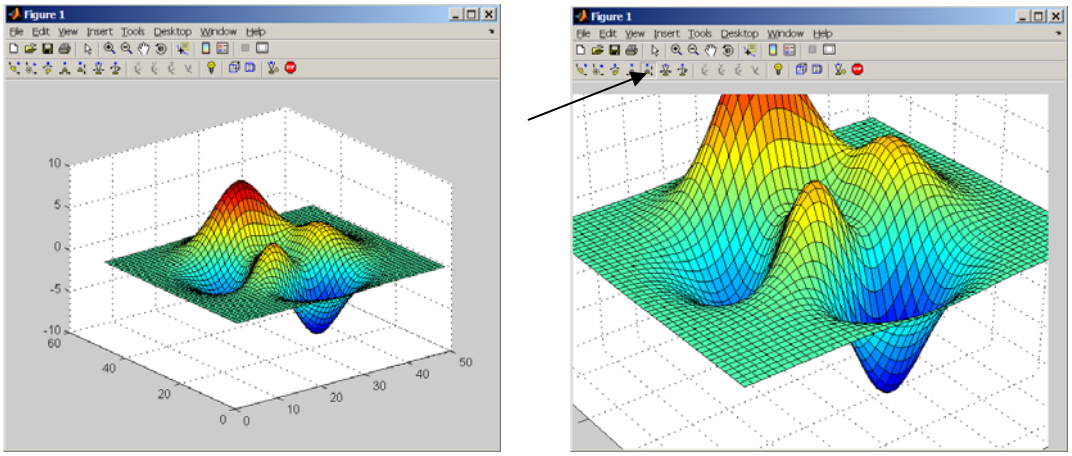


Fig. 12. Barra de herramientas en la ventana FIGURE.

Para ver más efectos hacer **`>>help graph2d`** y **`>>help graph3d`**.

Volúmenes predeterminados: Se pueden generar con ayuda de las funciones **`cylinder`**, **`sphere`**, **`ellipsoid`**.

```
>> cylinder([2 1 1 0.5],20);
>> sphere(50),axis('square')
```

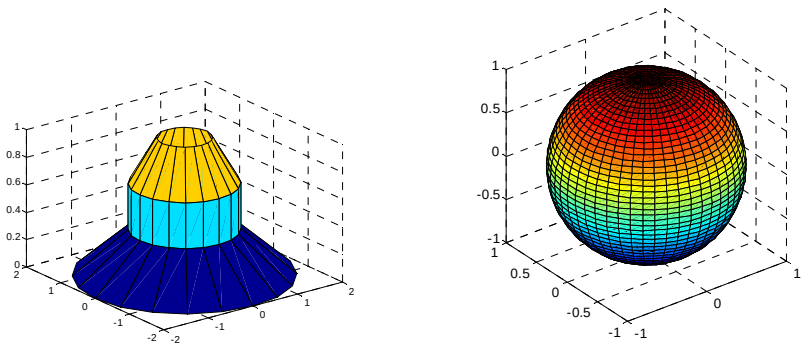


Fig. 13. Volúmenes predeterminados.

2.5 Objetos LIGHT

Los objetos LIGHT sirven para cambiar la apariencia de las representaciones en 3D. Las funciones más importantes son **`lighting (flat,none,phong,gouraud)`**, **`material (metal,dull,shiny)`**, **`surfl`**, **`specular`**, **`diffuse`**, **`surfnorm`**.

Ejemplo:

```
>> z=peaks;surf(z)
>> colormap('gray')
>> lighting phong
```

Desde la barra de menús de la figura también es posible seleccionar *Insert* → *Ligth*.

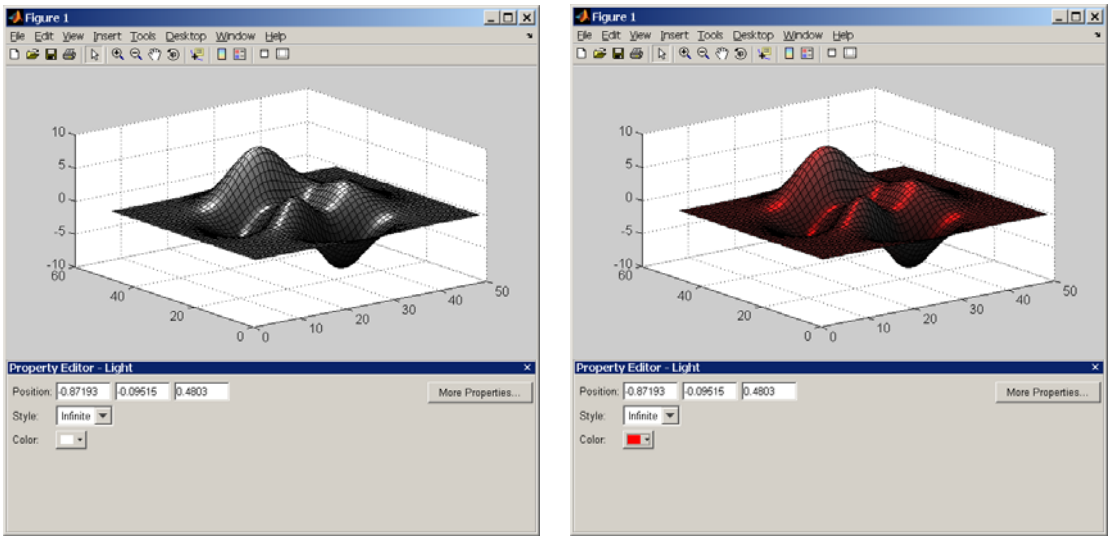


Fig. 14. Objeto LIGHT.

2.6 Objetos IMAGE

MATLAB escribe/lee diferentes formatos gráficos (TIFF, JPEG, BMP, PCX, XWD, HDF). Las principales funciones son **`imread`**, **`imwrite`** y **`imfinfo`**.

```
>>X=imread('earth1','gif');
>>X=imread('earth1.gif');

>>imfinfo('earth1.gif')
ans =
             Filename: 'earth1.gif'
          FileModDate: '17-May-2000 01:49:46'
             FileSize: 58178
               Format: 'GIF'
        FormatVersion: '87a'
                Width: 601
               Height: 353
             BitDepth: 7
            ColorType: 'indexed'
      FormatSignature: 'GIF87a'
      BackgroundColor: 0
          AspectRatio: 0
           ColorTable: [128x3 double]
           Interlaced: 'no'
```

El *display* de la imagen se hace con 2 posibles tipos de datos: **`double`** (doble precisión coma flotante, 64 bits) y **`uint8`** (entero sin signo, 8bit). Las funciones son **`image`** y **`imagesc`**. Es posible poner una barra con los colores presentes, **`colorbar`**.

Una imagen consiste en una matriz de datos X (formada por píxeles) y una matriz con los colores que pueden tomar cada uno de los píxeles. Hay cuatro tipos de imágenes: indexada, de intensidad, binaria y *truecolor*.

Imagen indexada: Los elementos de la matriz de píxeles X son los índices de las filas de la matriz de color (ésta tiene 3 columnas R,G,B y tantas filas como colores presentes en la imagen).

```
>> load earth
>> image(X),colormap(map),colorbar('vert')
```

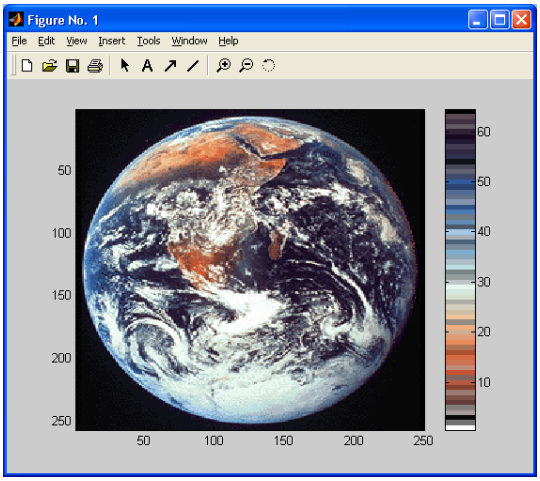


Fig. 15. Imagen indexada.

Imagen de intensidad: La matriz I representa intensidades (niveles de gris). Éstas van del negro al blanco (del 0 al 1, del 0 al 255 o del 0 al 65535)

```
>> Y=X/64;
>> imagesc(Y,[0 1]),colormap(gray),colorbar('vert')
```

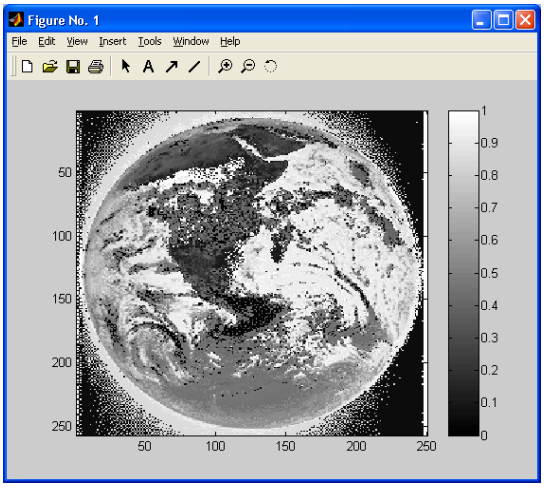


Fig. 16. Imagen de intensidad.

Imagen binaria: Los elementos de la matriz X son 1s y 0s.

Imagen truecolor: Es un 3D y no usa el colormap. X tiene dimensiones m×n×3. Cada píxel de la matriz X, X(m,n) viene definido por tres números: RGB(m,n,1) que corresponde al nivel de rojo, RGB(m,n,2) que corresponde al nivel de verde y RGB(m,n,3) que corresponde al nivel de azul.

```
>> rgb=imread('ngc6543a.jpg');
>> size(rgb)
ans =
   650   600     3

>> image(rgb)
```

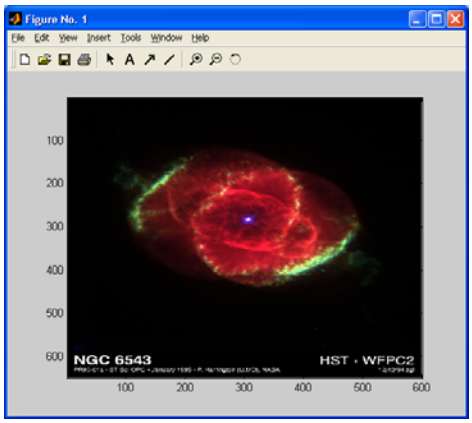


Fig. 17. Imagen *truecolor*.

El colormap por defecto es **`colormap('default')`** que corresponde al `hsv` (*Hue Saturation Value*). Teclear **`>>help graph3d`** para ver mapas de colores alternativos.

Hay toolboxes específicas que hacen un uso más intensivo de las imágenes. Ver por ejemplo, las demos de las *Image Processing Toolbox*, *Mapping Toolbox*, *Virtual Reality Toolbox*.

3. Gráficos específicos

Según sean las aplicaciones, existen tipos de representación específicos. Por ejemplo, en estadística es común representar la información por medio de histogramas, diagramas de dispersión, barras de error, etc. A medida que han ido apareciendo más *toolboxes*, MATLAB ha ido incorporando funciones para la representación de diferentes tipos de gráficos especiales.

3.1 Gráficos para presentaciones

Diagramas de queso: La función es **pie**. Si la suma es menor que 1, el queso queda incompleto,

```
>> x = [.19 .22 .41];
>> pie(x)
>> pie3(x)
```

Si se quiere extraer una "tajada",

```
>> x = [1 3 0.5 2.5 2];
>> pct=x/sum(x)
pct =
    0.1111    0.3333    0.0556    0.2778    0.2222

>> tajada = [0 1 0 0 0];
>> pie(x,tajada)
>> pie3(x,tajada),colormap summer
```

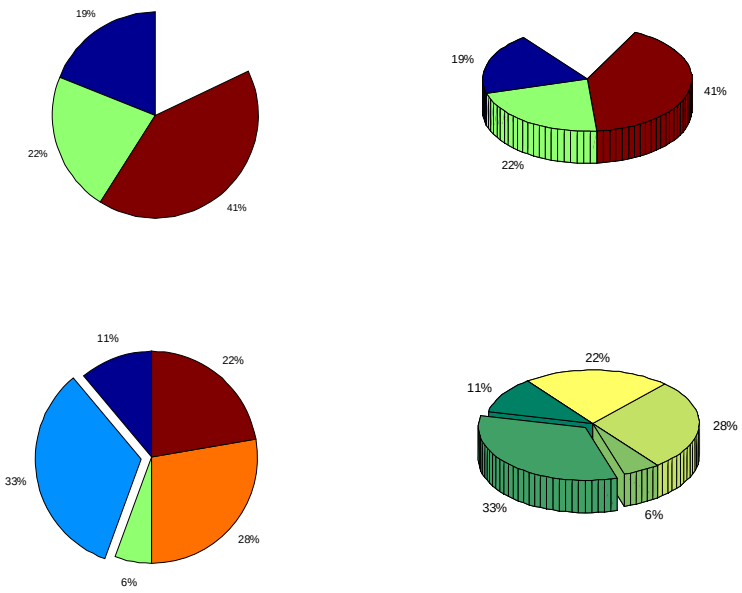


Fig. 18. Diagramas de queso

Histogramas: Funciones **hist**, **histfit**

```
yn = randn(10000,1);
hist(yn),
colormap autumn

Y = randn(10000,3);
hist(Y),
colormap summer

r = normrnd(10,1,100,1);
histfit(r)
```

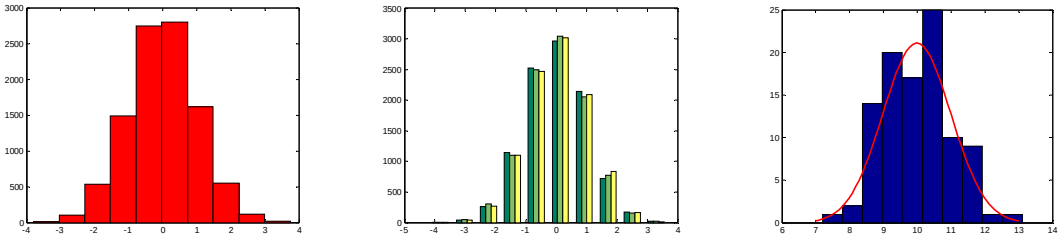


Fig. 19. Histogramas

Diagramas de barras: Funciones **bar**, **barh**

```
Y= round(rand(5,3)*10);
subplot(2,2,1),bar(Y,'group'),title('Grupo')
subplot(2,2,2),bar(Y,'stack'),title('Pila')
subplot(2,2,3),barh(Y,'stack'),title('Pila horizontal')
subplot(2,2,4),bar(Y,1.5),title('Grosor 1.5')
```

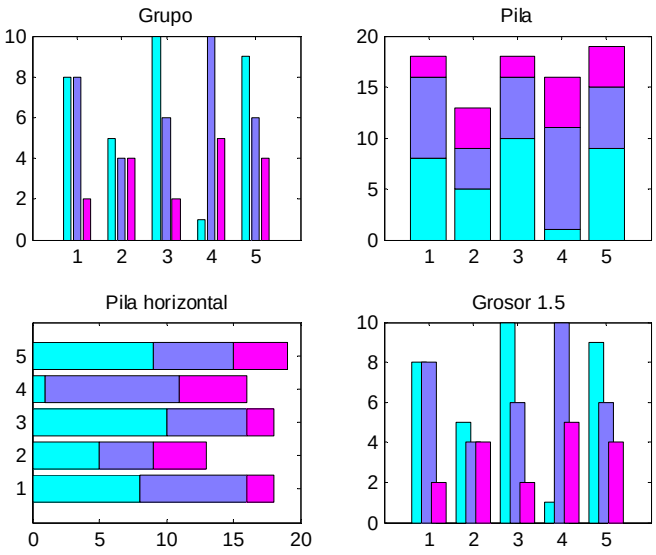


Fig. 20. Diagramas de barras

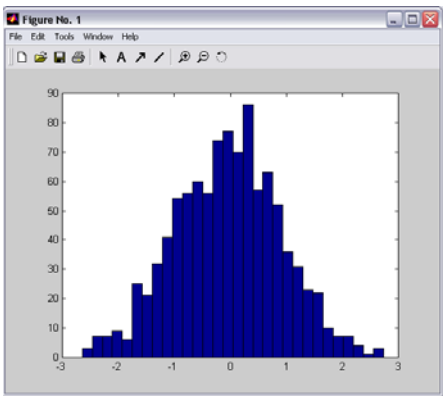
3.2 Probabilidad y estadística

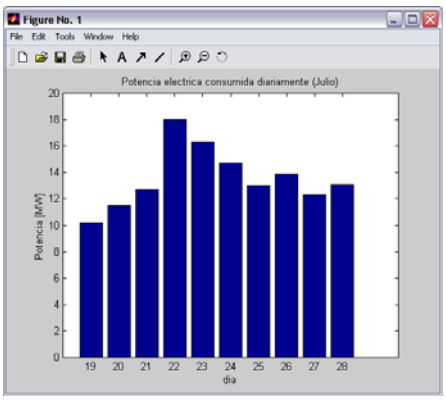
Ejemplo 3. Histogramas

En la primera figura el histograma se ha generado por medio de las siguientes instrucciones:

```
>>datos=randn(1000,1);  %generación de un vector de 1000
                        %muestras con distribución normal
>>hist(datos,30)  %histograma de 30 barras
```

Notar que la distribución de probabilidad de **datos** corresponde efectivamente a una Gaussiana (o distribución normal, de ahí la "n" de **randn**) de media cero y varianza unidad. ¿Qué distribución hubiera salido si se hubiera empleado la función **rand**?





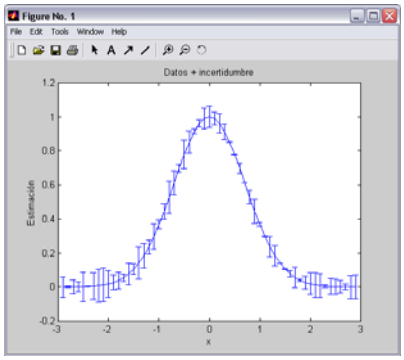
La segunda figura ilustra el consumo de energía de una población durante 10 días en MW. Los comandos para crear el gráfico de barras utilizados han sido:

```
» dias=19:28;
» potencia=[10.2 11.5 12.7 18 16.3 14.7 13.0 13.9 12.3 13.1];
» bar(dias,potencia);
```

Ejemplo 4. Representación de intervalos de error

Considerar por ejemplo un sistema cuya salida *y* es la exponencial decreciente del cuadrado de su entrada *u*. Las medidas de la salida para los distintos valores de *u* tienen una determinada incertidumbre (que aquí se ha generado de manera aleatoria). Esta incertidumbre puede representarse mediante la función **errorbar**:

```
» u=-2.9:0.1:2.9;
» e=0.1*rand(size(u));
» y=exp(-u.*u);
» errorbar(u,y,e)
```



Ejemplo 5. Diagramas de dispersión y de caja

Diagramas de dispersión:

```
load carsmall
figure,scatter(Weight,MPG),xlabel('Weight'),ylabel('MPG')
figure,gscatter(Weight,MPG,Model_Year,'bgr','xos')
```

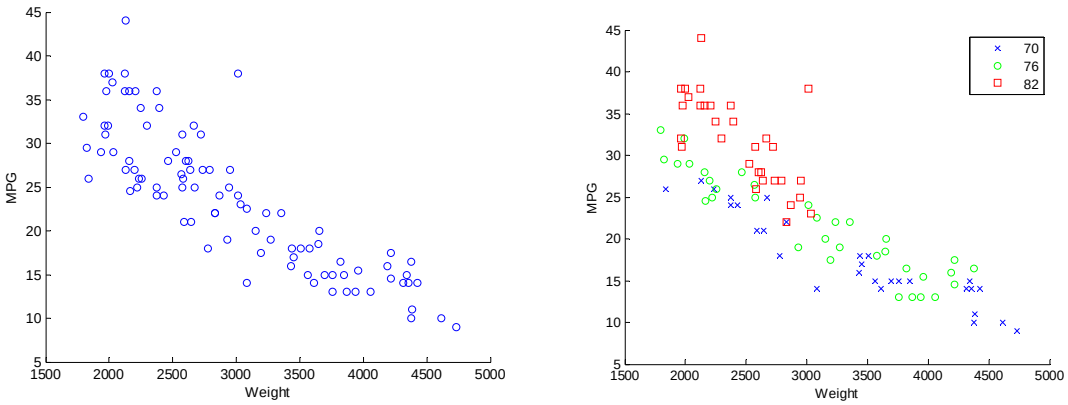


Fig. 21. Diagramas de dispersión

El fichero *carsmall.mat* contiene los siguientes datos sobre 100 coches: *Acceleration*, *Cylinders*, *Displacement*, *Horsepower*, *MPG* (consumo: miles-per-gallon), *Model*, *Model_Year*, *Origin*, *Weight.*

Diagramas de caja:

```
boxplot(MPG, Origin)
```

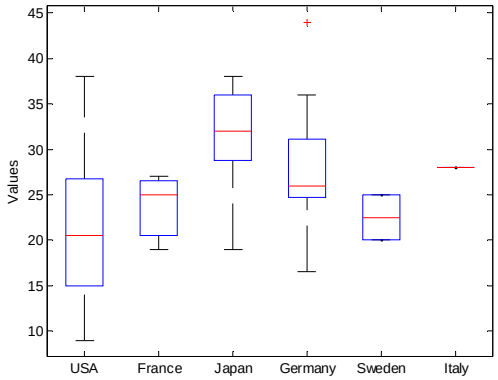


Fig. 22. Diagramas de caja

Hay un *outlier*. Se trata de un coche para el cual MPG>40. Para identificar cuál es, se puede hacer:

```
>> find(MPG>40)
ans =
    97
```

Parece que es un coche alemán:

```
>> Origin(97,:)
ans =
Germany
```

Podemos identificar el modelo y año, etc.

```
>> Model(97,:)
ans =
vw pickup

>> Model_Year(97,:)
ans =
    82
```

Ejemplo 6. Diagramas de distribución de probabilidad y test de hipótesis

Diagramas de distribución:

Diagrama de probabilidad normal: Sirve para determinar si un conjunto de datos está distribuido de forma Gaussiana. La línea continua conecta los percentiles 25 y 75.

```
x=normrnd(10,3,100,1);
figure,normplot(x)

x=exprnd(10,100,1);
figure,normplot(x)
```

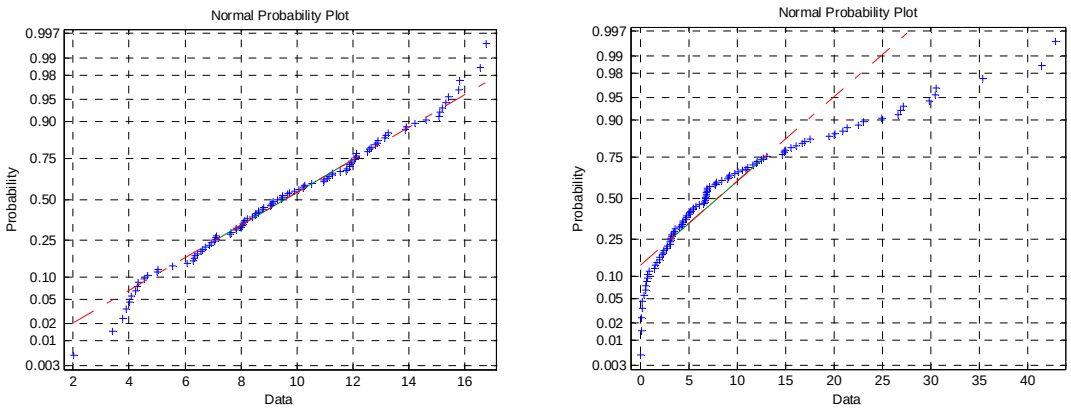


Fig. 23. Diagrama de probabilidad normal

Claramente, la segunda figura no corresponde a una distribución normal. Otra forma que ver que no es normal es mediante el test de Kolmogorov-Smirnov:

```
>> h=kstest(x)
```

```
h =
     1
```

La interpretación es la siguiente: Si el resultado es h=1 se puede rechazar la hipótesis nula. La hipótesis nula es que la muestra x tiene distribución normal estándar (media 0 y varianza 1). El resultado del test dice que se puede rechazar esta hipótesis. Por tanto, la muestra no está distribuida como N(0,1).

También nos podemos preguntar si la distribución es normal pero con otra media y desviación, N(m,σ). Por el diagrama PP ya sabemos que no pero vamos a comprobarlo:

```
>> [m,s]=normfit(x); %buscar media y desv que ajusten la muestra
>> [h,p]=kstest(x,[x normcdf(x,m,s)])  %y aplicar el test
h =
     1
p =
    0.0032
```

Puesto que de nuevo h=1, se puede rechazar la hipótesis nula de que la muestra está distribuida como N(m,σ). Nos quedamos pues con la hipótesis alternativa (que dice que la muestra NO está distribuida como N(m, σ)).

La función **`kstest`**, por defecto, rechaza la hipótesis nula (h=1) si el nivel de significancia es del 5%, esto es, si el valor de p es menor que 0.05, cosa que ocurre en nuestro caso. (Si el valor de p hubiera sido mayor que 0.05, h hubiera dado 0).

Para otro tipo de distribuciones se puede usar la función *probplot*.

```
x=wblrnd(3,3,100,1);
probplot('weibull',x)
```

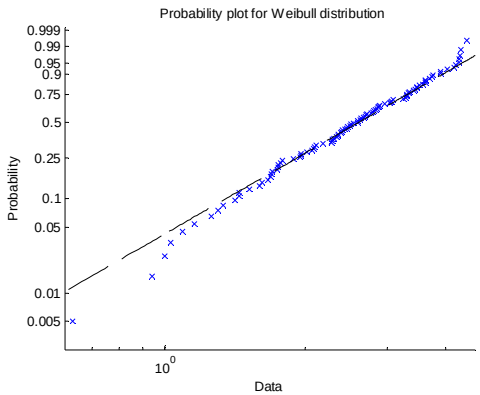


Fig. 24. Diagrama de probabilidad (otras distribuciones)

Diagrama Q-Q (quantile-quantile): Sirve para determinar si dos muestras provienen de la misma familia de distribución de probabilidad:

```
x=poissrnd(10,50,1);
y=poissrnd(5,100,1);
figure,qqplot(x,y)
```

```
x=normrnd(5,1,100,1);
y=wblrnd(2,0.5,100,1);
figure,qqplot(x,y)
```

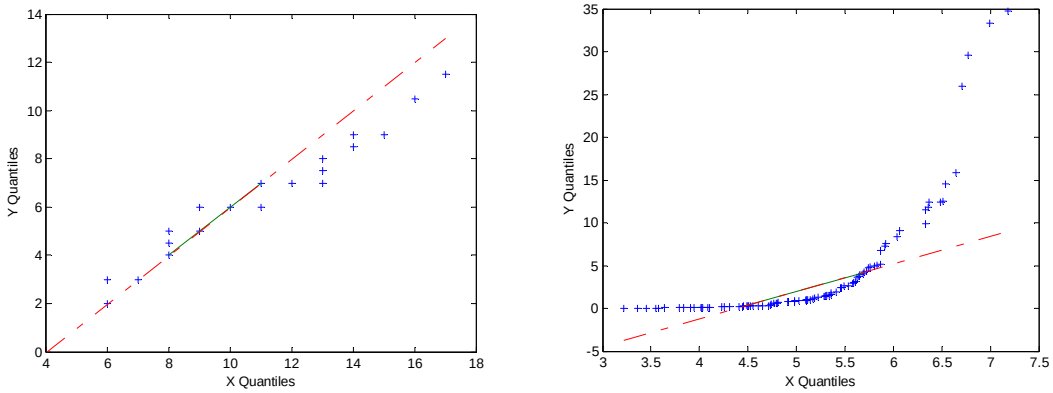
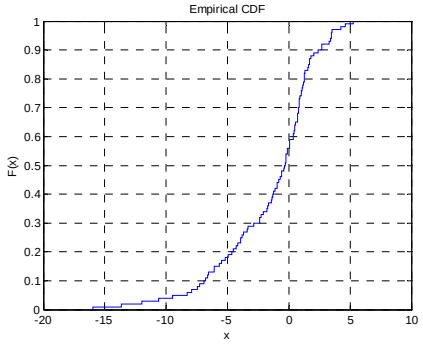


Fig. 25. Diagramas Q-Q

Diagrama de distribución acumulada: Es la función *cdfplot*.

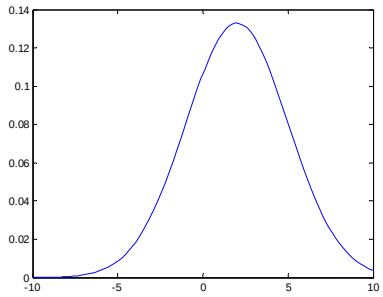
```
y=evrnd(0,3,100,1);
figure,cdfplot(y)
```

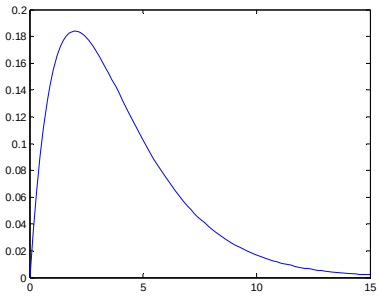


Ejemplo 7. Generación de funciones densidad de probabilidad

Generación pdfs: Las funciones que generan funciones densidad de probabilidad terminan por "…*pdf*" y empiezan por el nombre de la familia de probabilidad. Así *normpdf* genera la pdf correspondiente a la distribución normal y *chi2pdf* genera la pdf correspondiente a la distribución chi cuadrado.

```
x=linspace(-10,10);y=normpdf(x,2,3);figure,plot(x,y)
x=linspace(0,15);y=chi2pdf(x,4);figure,plot(x,y)
```

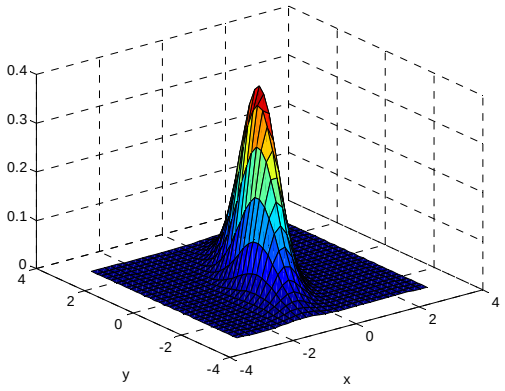


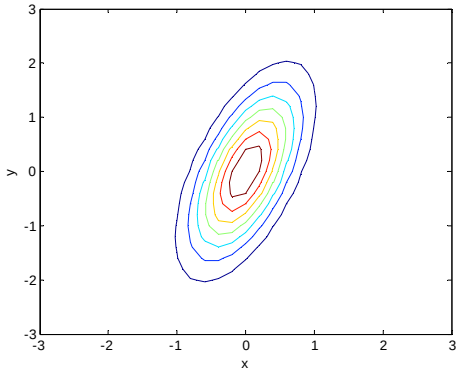


Otras funciones son: betapdf (Beta), binopdf (binomial), exppdf (exponencial), unifpdf (uniforme), etc…

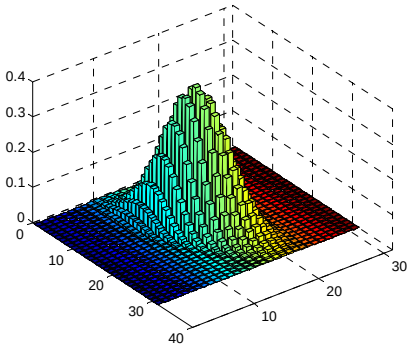
Distribución normal multivariable: La función es mvnpdf

```
media=[0 0];
matriz_cov=[.25 .3;.3 1];

x=-3:.2:3;
y=x;
[xx,yy]=meshgrid(x,y);

F=mvnpdf([xx(:) yy(:)],media,matriz_cov);
F=reshape(F,length(xx),length(yy));

figure,surf(x,y,F),xlabel('x'),ylabel('y')
figure,contour(x,y,F),xlabel('x'),ylabel('y')
```






```
bar3(F)
```



3.3 Respuesta frecuencial de sistemas lineales

Introducción del sistema: Se entran por separado el polinomio numerador y el polinomio denominador (notar el uso de los corchetes). Por ejemplo, para entrar $H(s)=\frac{2}{s^2+0.5s+1}$, se hace:

```
>> num=2;
>> den=[1 0.5 1];
```

Funciones: Las funciones son **`bode`** (para representar diagramas de Bode), **`nyquist`** (para representar diagramas polares), **`nichols`** (para representar diagramas fase-ganancia) y **`freqs`** (para obtener los valores de la representación en cartesianas). Se sugiere hacer `>>help nombre_funcion`.

Sintaxis: Hay varios niveles:

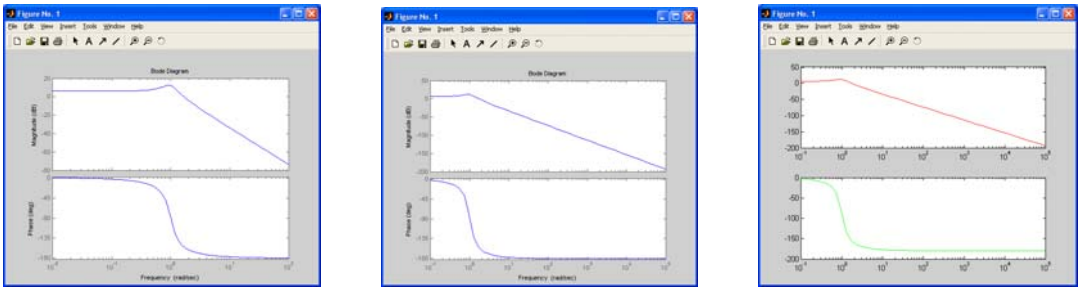
El más sencillo (ver Fig. a) es: `>> bode(num,den)`

Si se quiere especificar el eje frecuencial (ver Fig. b) hay que hacer

```
>> w=logspace(-1,5);  %frecuencias de 0.1 a 1e5
>> bode(num,den,w)
```

Si se quieren guardar en un vector las muestras del módulo y la fase de la respuesta frecuencial para, por ejemplo, representarla más tarde (ver Fig. c), se puede hacer

```
>> [mag,fase]=bode(num,den,w);
>> subplot(211),semilogx(w,20*log10(mag),'r')
>> subplot(212),semilogx(w,fase,'g')
```



(a) (b) (c)

Fig. 26. Respuesta frecuencial.

3.4 Respuesta temporal de sistemas lineales

El sistema $H(s)=\frac{2}{s^2+0.5s+1}$ se entra igual que en el apartado anterior.

Funciones: Las funciones son **`impulse`** (para la respuesta a impulso), **`step`** (para la respuesta indicial, a escalón unitario) y **`lsim`** (*linear simulation*, para excitaciones arbitrarias tales como rampas, sinusoides, combinaciones de señales,...). Se sugiere hacer `>>help nombre_funcion.`

Sintaxis: Hay varios niveles (notar el uso del punto y coma):

El más sencillo (ver Fig. a) es:

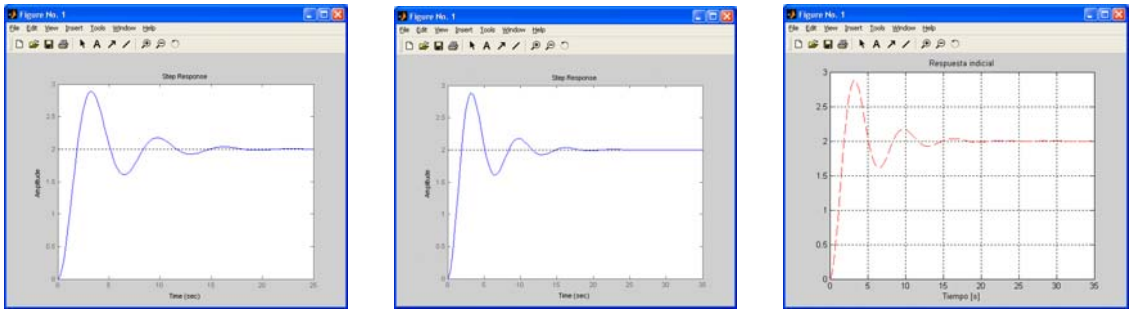
```
>> step(num,den)
```

Si se quiere especificar el eje temporal (ver Fig. b) hay que hacer

```
>> t=linspace(0,35);
>> step(num,den,t)
```

Si se quieren guardar en un vector las muestras de la respuesta para, por ejemplo, representarla más tarde (ver Fig. c), se puede hacer

```
>> y=step(num,den,t);
>> plot(t,y,'r--')
>> grid,title('Respuesta indicial'),xlabel('Tiempo [s]')
```

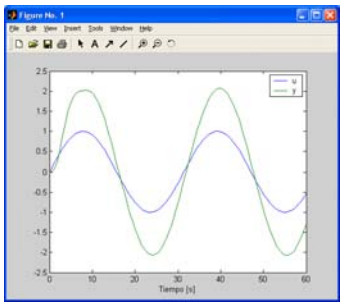


(a) (b) (c)

Fig. 27. Respuesta temporal.

Para simular excitaciones generales, primero hay que definir cuáles son éstas:

```
>> t=linspace(0,60,100);
>> u=sin(0.2*t);
>> y=lsim(num,den,u,t);
>> plot(t,u,t,y)
>> legend('u','y'),
>> xlabel('Tiempo [s]')
```

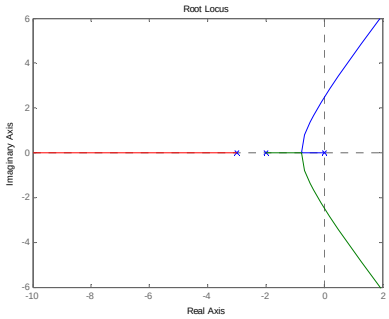


(d)

3.5 Otras funciones relacionadas con la teoría de sistemas

Para obtener el lugar geométrico de las raíces de Evans, la función es `rlocus`:

```
>> num=1;
>> den=conv([1 3 0],[1 2]);
>> rlocus(num,den)
```

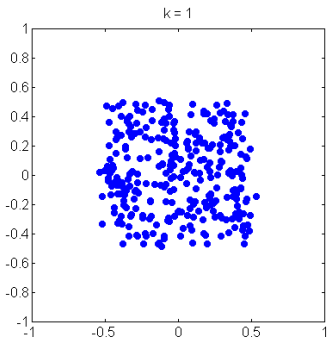


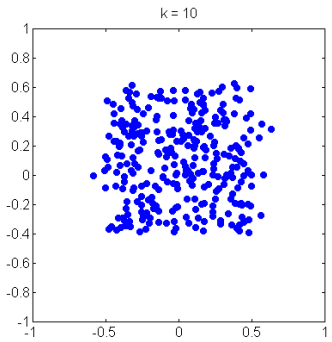
4. Animaciones

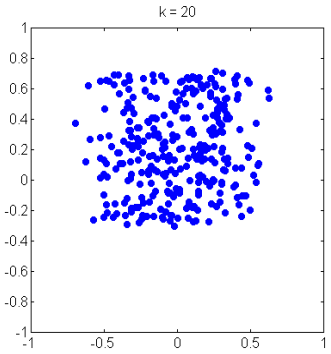
Animaciones: Las animaciones se consiguen con ayuda de las funciones **moviein** (de inicialización), **getframe** (captura de cada uno de los fotogramas o *frames*) y **movie** (animación propiamente dicha). (Nota: **moviein** no es necesaria en las versiones más recientes)

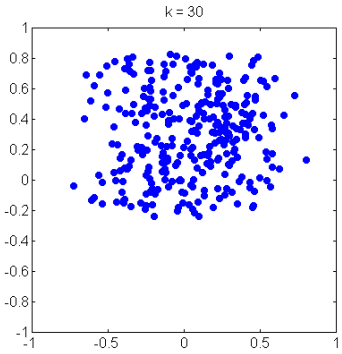
Ejemplo 8. Movimiento Browniano

```
n=300;s=0.02;
n_tr=50;
x=rand(n,1)-0.5;
y=rand(n,1)-0.5;
h=plot(x,y,'.')
set(h,'MarkerSize',18)
axis([-1 1 -1 1])
axis square
grid off

M=moviein(n_tr);

for k=1:n_tr
   x=x+s*randn(n,1);
   y=y+s*randn(n,1);
   set(h,'XData',x,'YData',y)
   M(:,k)=getframe;
end
movie(M,5)
```









Cuando se ejecuta la función **`movie`** primero hace un pase rápido de la animación y a continuación presenta la animación en sí. Tiene diversas opciones. Por ejemplo, **`movie(M,0)`** ejecuta el pase rápido pero no la animación; **`movie(M,-1)`** ejecuta el pase rápido y a continuación la animación hacia delante y después hacia detrás.

Por defecto, la velocidad de la película es la de la captura. Si se quiere modificar la velocidad hay que usar el tercer argumento. En **`movie(M,-2,50)`** la velocidad son 50 fotogramas por segundo. Ello quiere decir que una película con 50 fotogramas se verá en 1 segundo.

Vídeos: Para crear vídeos se puede utilizar, por ejemplo, la función **`avifile`**. Las tramas van añadiéndose al vídeo por medio de la función **`addframe`**. Finalmente, se cierra el vídeo con **`close`**.

El siguiente ejemplo muestra la creación de un fichero de vídeo (de nombre *soroll.avi*) basado en el movimiento browniano del apartado anterior.

Ejemplo 9. Creación de un fichero de vídeo

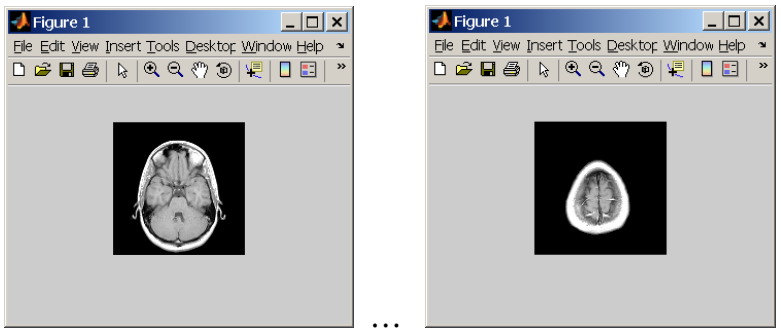
```
mov = avifile('soroll.avi')

n=300;
s=0.02;
x=rand(n,1)-0.5;
y=rand(n,1)-0.5;

h=plot(x,y,'.')
set(h,'MarkerSize',18)
axis([-1 1 -1 1]),axis square,grid off

n_frames=50;
for k=1:n_frames
   x=x+s*randn(n,1);
   y=y+s*randn(n,1);
   set(h,'XData',x,'YData',y)
   F=getframe(gca);
   mov=addframe(mov,F);
end
```

```
mov=close(mov);
```

El fichero de vídeo creado puede insertarse luego en otras aplicaciones, tales como el Power Point (Insertar → Películas y sonidos → Película de archivo). Ello resulta muy útil a la hora de presentar trabajos o proyectos fin de carrera.

Imágenes animadas: Para ver imagénes animadas (ficheros multi-trama, etc.) se puede usar la función **`immovie`**. Para ilustrarlo, ver el siguiente ejemplo de MATLAB:

```
>> load mri
>> mov=immovie(D,map);
>> movie(mov)
```



Para capturar los fotogramas de una imagen se puede hacer

```
[x,map]=imread('nombre_fichero.extensión',1:num:fotogramas);
```

o bien

```
[x,map]=imread('nombre_fichero.extensión','frames','all');
```

También es posible crear una imagen multiframe tomando los fotogramas por separado y juntándolos con ayuda de la función **`cat`**:

```
A = cat(4,A1,A2)
```

Aquí A1 y A2 serían las dos imágenes que forman la animación. El 4 está porque este tipo de variable es un array de 4 dimensiones m x n x 3 x 2, donde m x n son los pixels de la imagen, el 3 es corresponde a una imagen true-color (contiene una matriz de tres columnas correspondiendes a rgb, y cada pixel va referido a uno de estos 256 colores) y el 2 es el número de fotogramas.