Señales y Sistemas

Práctica 2:

REPRESENTACIÓN GRÁFICA

CON MATLAB

ÍNDICE

1. GRÁFICOS BIDIMENSIONALES

La representación gráfica con MATLAB está orientada fundamentalmente a la representación de vectores y matrices (lo cuál es bastante lógico teniendo si nos fijamos en el modo de trabajo de MATLAB). La función principal para la representación de gráficos 2-D es la función **`plot`**, como estudiaremos a continuación. MATLAB utiliza un tipo especial de ventanas para realizar las operaciones gráficas. Ciertos comandos abren una ventana nueva y otros dibujan sobre la ventana activa, bien sustituyendo lo que hubiera en ella, bien añadiendo nuevos elementos gráficos a un dibujo anterior. Todo ello se verá con más detalle en las siguientes secciones.

Esta segunda práctica está estructurada de la siguiente forma: en *primer lugar* se hace un estudio sobre la representación bidimensional en MATLAB, que funciones se utilizan, que modificaciones se pueden hacer, etc.; en *segundo lugar* se presenta brevemente la representación en 3-D (es sólo a nivel informativo para que el alumno conozca las posibilidades de MATLAB) y finalmente se proponen unos ***ejercicios que el alumno debe resolver y mostrar para su corrección al profesor de prácticas***.

1.1. Funciones gráficas 2-D elementales

MATLAB dispone de cuatro 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. Las funciones son las siguientes:

`plot`	Crea un gráfico a partir de vectores y/o columnas de matrices, con escalas lineales en ambos ejes.
`loglog`	Igual con escala logarítmica en ambos ejes.
`semilogx`	Igual con escala lineal en el eje de ordenadas y logarítmica en el eje de abscisas.

`semilogy`	Igual con escala lineal en el eje de abscisas y logarítmica en el eje de ordenadas.

En lo sucesivo se hará referencia casi exclusiva a la primera de estas funciones **`plot`**. Las demás se pueden utilizar de un modo similar. 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 título al texto
`xlabel('eje x')`	Añade una etiqueta el eje de abscisas. Con **xlabel off** desaparece.
`ylabel('eje y')`	Añade una etiqueta el 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.
`gtext('texto')`	Introduce **texto** con ayuda del ratón, 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; para más detalle consultar el **help**.
`grid`	Activa la inclusión de una cuadrícula en el dibujo. Con **grid off** desaparece la cuadrícula.

Borrar texto (u otros elementos gráficos) es un poco más complicado; de hecho hay que preverlo de antemano. Para poder hacerlo hay que recuperar previamente el *valor de retorno* del comando con el cual se ha creado. Después hay que llamar a la función **`delete`** con ese valor como argumento. Veamos un ejemplo:

```
>>v = text(1, .0 ,'seno')
v = 76.0001
```

```
>>delete(v)
```

Los dos grupos de funciones anteriores no actúan de la misma forma. Así la función **plot** dibuja una nueva figura en la ventana activa (en todo momento MATLAB tiene una ventana activa de entre todas las ventanas gráficas abiertas), o abre una nueva figura si no hay ninguna abierta, sustituyendo cualquier cosa que hubiera dibujada anteriormente en esa ventana. Para verlo, se comenzará creando un par de vectores **x** e **y** con los que trabajar:

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

Ahora se deben ejecutar los comandos siguientes (se comienza cerrando la ventana activa, para que al crear la nueva ventana aparezca en primer plano):

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

Se puede observar la diferencia con la secuencia que sigue:

```
>>close
>>plot(x,y)
>>grid
%Se añade la cuadrícula sin borrar la función seno
```

En el primer caso, MATLAB ha creado la cuadrícula en una ventana nueva y luego la ha borrado al ejecutar la función **plot**. En el segundo caso, primero ha dibujado la función y luego ha añadido la cuadrícula. Esto es así porque hay funciones como **plot** que por defecto crean una nueva, 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.

Más adelante se verá que con la función **`hold`** pueden añadirse gráficos a una figura ya existente respetando su contenido.

1.1.1. Función *plot*.

Esta es la función clave de todos los gráficos 2-D en MATLAB. Ya se ha comentado que el elemento básico de los gráficos bidimensionales es el **vector**. Se utilizan también cadenas de 1, 2 ó 3 caracteres para dibujar *colores* y *tipos de línea*. La función **`plot()`**, en sus diversas variantes, no hace otra cosa que dibujar vectores. Un ejemplo muy sencillo de esta función, en el que se le pasa un único vector como argumento, es el siguiente:

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

¿Qué obtiene?

Por defecto, los distintos puntos del gráfico se unen con una línea continua y por defecto también es de color azul.

Cuando a la función **`plot`** se le pasa un único vector real como argumento, dicha función dibuja en ordenadas el calor de los **n** elementos del vector frente a los índices 1,2, ..., n del mismo en abscisas. Como ya hemos visto, si el vector es complejo el funcionamiento es diferente.

Una segunda forma de utilizar la función **`plot`** es con dos vectores como argumentos. En este caso los elementos del segundo vector se representan en ordenadas frente a los valores del primero, que se representan en abscisas. Veamos como se puede

dibujar un cuadrilátero de esta forma (obsérvese que para dibujar un polígono cerrado el último punto debe coincidir con el primero):

```
>> x = [1 6 5 2 1];  y = [1 0 4 3 1];
>>plot (x,y)
```

La función `plot` permite también dibujar múltiples curvas introduciendo varias parejas de vectores como argumentos. En este caso, cada uno de los segundos vectores se dibujan en ordenadas como función de los valores del primer vector de la pareja, que se representan en abscisas. Si el usuario no decide otra cosa, para las sucesivas líneas se utilizan colores que son permutaciones cíclicas el **azul**, **verde**, **rojo**, **cian**, **magenta, amarillo**, y **negro**. Observemos el siguiente ejemplo:

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

En el caso de **números complejos** es distinto. Si se pasan a **`plot`** varios vectores complejos como argumentos, MATLAB simplemente representa las partes reales y desprecia las partes imaginarias. Sin embargo, un único argumento complejo hace que se represente la parte real en abscisas, frente a la parte imaginaria en ordenadas. Veamos un ejemplo, donde para generar un vector complejo se utiliza el resultado del cálculo de valores propios de una matriz formada aleatoriamente:

```
>>plot (eig (rand (20,20)) , '+')
```

donde se ha hecho uso de elementos que se verán en la siguiente sección, respecto a dibujar con distintos "markers" (en este caso con signos +), en vez de con línea continua, que es la opción por defecto. En el comando anterior, el segundo argumento es un carácter que indica el tipo de "maker" elegido. El comando anterior es equivalente a:

```
>>z = eig(rand(20,20));
>>plot (real(z), imag(z), '+')
```

Como ya se ha indicado, si se incluye más de un vector complejo como argumento, se ignoran las partes imaginarias. Si se quiere dibujar varios vectores complejos, hay que separar explícitamente las partes reales e imaginarias de cada vector, como hemos visto en el ejemplo anterior.

El comando **`plot`** puede utilizarse también con matrices como argumentos. Para mas información sobre la función **`plot`** se aconseja hacer uso del comando **`help`**.

1.1.2. Estilos de línea y marcadores

En la sección anterior hemos visto como la tarea fundamental de la función **`plot`** era dibujar los valores de un vector en ordenadas, frente a los valores de otro vector en abscisas. En el caso general esto exige que se pasen como argumentos un par de vectores. En realidad, el conjunto básico de argumentos de esta función es una *tripleta* formada por dos vectores y una cadena de 1, 2 ó 3 caracteres que indica el color y el tipo de línea o de *marker*. En las tablas siguientes aparecen distintas posibilidades:

Símbolo	**Color**
y	Yellow
m	Magenta
c	Cyan
r	Red
g	Green
b	Blue

Símbolo	Color
w	White
k	Black

Símbolo	Estilo de línea
-	Líneas continuas
:	Líneas a puntos
-.	Líneas a barra-punto
--	Líneas a trazos

Símbolo	Markers
.	Puntos
o	Círculos
x	Marcas en x
+	Marcas en +
*	Marcas en *
s	Cuadrados
d	Diamantes
^	Triangulo apuntado arriba
v	Triangulo apuntado abajo
p	Estrella de 5 puntas

Cuando hay que dibujar varias líneas, por defecto se van cogiendo sucesivamente los colores de la tabla comenzando por el azul, hacia arriba y cuando se terminan se vuelve a empezar otra vez por el mismo. Si el fondo es blanco, este color no se utiliza para las líneas.

1.1.3. Añadir líneas a un gráfico ya existente

Existe la posibilidad de añadir líneas a un gráfico ya existente, sin destruirlo o sin abrir una nueva ventana. Se utilizan para ello los comandos **`hold on`** y **`hold off`**. El primero de ellos hace que los gráficos sucesivos respeten los que ya se han dibujado en la figura (es posible que tengamos que modificar la escala de los ejes); el comando **`hold off`** deshace el efecto del anterior. El siguiente ejemplo muestra como se añaden las gráficas de **`x2`** y **`x3`** a la gráfica de **`x`** previamente creada (cada una con un tipo de línea diferente):

```
>>plot(x);
>>hold on
>>plot(x2, '–');
>>plot(x3, '-.');
>>hold off
```

1.1.4. Comando *subplot*

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 enumeran 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:

```
>>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)
```

Practique añadiendo títulos a cada `subplot`, así como rótulos a los ejes. Para volver a la opción por defecto basta con teclear el comando:

```
>>subplot(1,1,1)
```

1.1.5. Control de los ejes

MATLAB tiene unas opciones por defecto que 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(axis)`	Mantiene los ejes en sus actuales valores, de cara a posibles nuevas gráficas añadidas con **`hold on`**
`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('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
`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

1.2. Control de las ventanas gráficas: Función figure

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. La función **`close`** cierra la figura activa, mientras que **`close(n)`** cierra la ventana o figura número **`n`**. El comando **`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.

Ejecútense las siguientes instrucciones de MATLAB, observando los efectos de cada una de ellas en la ventana activa. El comando **`figure(gcf)`** permite hacer visible la ventana desde la ventana de comandos.

```
>>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'), figure(gcf)
>>ylabel('valor de la función trigonométrica'),...
 figure (gcf)
>>axis([-12, 12, -1.5, 1.5]), figure(gcf)
>>axis('equal'), figure(gcf)
>>axis('nomal'), figure(gcf)
>>axis('square'), figure(gcf)
>>axis('off'), figure(gcf)
>>axis('on'), figure(gcf)
>>axis('on'), grid, figure(gcf)
```

1.3. Otras funciones gráficas 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 (para mas información utilizar el comando `help`.):

`bar`	Crea diagramas de barras
`barh`	Diagrama de barras horizontales
`bar3`	Diagrama de barras con aspecto 3D
`bar3h`	Diagrama de barras horizontales con aspecto de 3D
`pie`	Gráficos con forma de tarta
`pie3`	Gráficos con forma de tarta y aspecto 3D
`stairs`	Análoga a bar sin líneas internas
`hist`	Histogramas de un vector
`rose`	Histogramas de ángulos (en radianes)

Ejecuta los siguientes comandos a modo de ejemplo:

```
>>x = [rand(1,100)*10];
>>plot(x)
>>bar(x)
>>stairs(x)
>>hist(x)
>>alfa = (rand(1,20)-0.5)*2*pi;
>>rose(alfa)
```

1.4. Entrada de puntos con el ratón

Se realiza mediante la función `ginput` que permite introducir las coordenadas del punto sobre el que está el cursor, al hacer un "clic" con el ratón. Algunas formas de utilizar esta función son las siguientes:

`[x,y]=ginput`	Lee un número indefinido de puntos cada vez que se pincha con el ratón sobre un punto de la figura y termina cuando se pulsa la techa intro.
`[x,y]=ginput(n)`	Lee las coordenadas de **n** puntos

Pruebe la instrucción sobre la figura que tenga activa.

2. GRAFICOS TRIDIMENSIONALES

MATLAB tiene posibilidades de realizar varios tipos de gráficos 3D. La primera forma de gráfico 3D es la función **`plot3`**, que es el análogo tridimensional de la función `plot`. Esta función dibuja puntos cuyas coordenadas están contenidas en 3 vectores, bien uniéndolos mediante una línea continua (por defecto), bien mediante *markers*.

Vamos a dibujar una línea espiral:

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

También podemos representar funciones de dos variables. Para ver un ejemplo es necesario definir una función en un fichero llamado **test3d.m** que será de la forma:

```
function z=test3d(x,y)
z = 3*(1-x) .^2 .*exp(-(x .^2) – (y+1) .^2) ...
     -10*(x/5 –x .^3 –y .^5) .*exp(-x .^2 – y .^2) ...
     -1/3*exp(-(x+1) .^2 – y .^2);
```

Ejecute la siguiente lista de comandos para ver los distintos tipos de representaciones obtenidas:

```
>>x =[-3:0.4:3]; y=x;
>>close
>>subplot(2,2,1)
>>figure(gcf),  fi=[0 : pi/20 : 6*pi];
>>plot3(cos(fi), sin(fi), fi, 'r')
>>[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)
```

El estudio en detalle de las representaciones tridimensionales en MATLAB está fuera de los objetivos de esta práctica, por tanto sólo se han presentado nociones muy básicas. Para más información consultar usando **`help`**.

3. EJERCICIOS

En este apartado de la práctica se proponen tres ejercicios a realizar para que el alumno comience a familiarizarse con MATLAB. Al finalizar la práctica, el alumno deberá mostrar a la profesora tanto el código utilizado para resolverlos, como las figuras que se obtengan como resultado.

Ejercicio 1

Halle y represente en el plano complejo los números siguientes:

$$x = 1 + j$$
$$z = \frac{1}{x}$$
$$w = \left(z^{-1}\right)^{*}$$
$$v = \frac{1}{z^{*} z} z$$
$$u = z^{4}$$
$$y = \sqrt[4]{z}$$

Compruebe los resultados obtenidos con MATLAB de forma numérica y gráfica.

Ejercicio 2

Represente gráficamente dos periodos de la sinusoide real en tiempo continuo *x(t)*,

$$x(t) = 3\cos(2\pi f_o t + \pi/4),$$

con $f_o = 1/T_0 = 100$ Hz y compara el resultado anterior con la representación gráfica de la parte real de la señal y(t),

$$y(t) = 2e^{(-200 + j2\pi f_1)t},$$

con f_1=300 Hz. Comente la relación entre las frecuencias de las dos señales. Haga una representación aproximada de las señales eligiendo un intervalo de tiempo adecuado para hacer la representación de dos periodos de la señal x(t).

Nota: En primer lugar es necesario generar un vector de reales en el intervalo de tiempos adecuado. Para ello se puede hacer de dos formas:

```
>>t=[Tmin:tinc:Tmax]
%Eje de tiempos de Tmin a Tmax con un incremento de tinc
>>t=linspace(Tmin,Tmax,N)
%N puntos equiespaciados entre Tmin y Tmax.
%Si no se especifica el valor de N, por defecto es 100.
```

Para obtener una buena representación debe considerar un número de puntos suficientemente grande. Debe calcular cuál ha de ser el incremento de tiempo que hay que utilizar en la representación en función del número de puntos.

Ejercicio 3

Represente las señales siguientes con f=0.01 Hz en el intervalo comprendido entre t=-100 y t=500 s., considerando un espaciado temporal entre muestras suficientemente cercano.

$$x(t) = e^{\frac{-t}{150}}$$
$$y(t) = \cos(2\pi ft)$$
$$z(t) = 3 \cdot x(t) \cdot y(t)$$