Señales y Sistemas

Práctica 1

INTRODUCCIÓN A MATLAB

ÍNDICE

1. ¿Qué es Matlab?

MATLAB (del término anglosajón MATrix LABoratory) es un sistema interactivo, originalmente escrito en FORTRAN por Cleve Moler de la Universidad de Nuevo México, USA, que sirve para la resolución de cálculos numéricos en aplicaciones científicas y de ingeniería. Actualmente MATLAB se encuentra en su segunda generación, la cual es una evolución, escrita en lenguaje C, de la versión FORTRAN original. Esta segunda generación de MATLAB incluye ya capacidades gráficas, macros programables, aritmética IEEE, un intérprete rápido y muchos más programas de aplicación.

Los responsables de esta versión C son Steve Bangert, encargado del intérprete de comandos, Steve KIeiman, que implementó la parte gráfica, y John Little y Cleve Moler, que escribieron las rutinas analíticas, la guía de usuario y la mayor parte de los ficheros M. Todos ellos, a su vez, fueron los fundadores de The MathWorks, empresa propietaria de MATLAB.

Uno de los puntos fuertes de MATLAB consiste en que las unidades básicas de cálculo son matrices que no necesitan ser previamente dimensionadas. Este hecho permite resolver problemas numéricos más fácilmente que utilizando directamente lenguajes de programación convencionales, como son FORTRAN, BASIC, PASCAL o C. Además MATLAB expresa la solución de los problemas casi exactamente igual a como se escribirían matemáticamente.

MATLAB se utiliza especialmente en cálculos numéricos, algoritmos y problemas con formulación matricial. Su capacidad va desde efectuar una sencilla suma hasta realizar la inversión de matrices o su descomposición en autovectores.

Posiblemente, la característica más importante de MATLAB es la facilidad con la que se puede ampliar, permitiendo a cualquiera crear aplicaciones y funciones integrables en MATLAB y utilizables por el resto de personas.

MATLAB interpreta los comandos que le son introducidos por teclado, procesándolos inmediatamente y mostrando el resultado. También permite ejecutar secuencias de comandos almacenados en un fichero con extensión *.m, llamados ficheros M-files. Los ficheros M son ficheros ASCII que pueden crearse o modificarse con cualquier editor de texto convencional. Los ficheros M pueden referenciar a otros ficheros M, o incluso referenciarse a sí mismos recursivamente.

Existen dos tipos de ficheros M: script-files y funciones.

Los **script-files** son ficheros externos de MATLAB que contienen una lista de instrucciones y cuyo objetivo es automatizar largas secuencias de comandos.

Los **ficheros de función** permiten definir nuevas funciones para MATLAB. Deben incluir en la primera línea la palabra reservada *`function`*. Por ejemplo:

```
function [mean,stdev] = stat(x,y,z)
```

donde *`stat`* es el nombre la función, *`x`*, *`y`* y *`z`* son las variables de entrada y *`mean`* y *`stdev`* son las variables que devuelve la función. Al crear un fichero de función se debe salvar con el nombre de la función (en nuestro ejemplo, `stat.m`).

Los ficheros de función distinguen tanto variables locales como externas que pueden ser pasadas a la función.

Otro tipo de ficheros de MATLAB son los tipo MAT, que se generan con el comando `save` de MATLAB. Este comando permite grabar las variables que actualmente se encuentran en memoria, en un fichero de disco de tipo binario. Al fichero que resulta se le llama fichero de tipo MA T porque la extensión del fichero es .MAT. El comando `load` es el inverso de `save`, es decir: `load` lee el fichero MAT del disco y lo carga en la memoria de MATLAB.

MATLAB ofrece también una serie de funciones diseñadas especialmente para aplicaciones específicas. Estas se agrupan en entidades conocidas como ***toolboxes***. Las *toolboxes* añaden aún más potencia a MATLAB cuando piensa utilizarse éste en aplicaciones tales como procesado digital de la señal. El programa MATLAB incluye unas *toolboxes* de señales, que contienen funciones destinadas entre otros propósitos, al análisis espectral en módulo y fase de señales, la implementación de filtros tanto analógicos como digitales, el análisis de modulaciones digitales, la evaluación de efectos introducidos por un canal de comunicación, etc.

1.1. Primeros pasos con MATLAB

Las variables empleadas en MATLAB deben empezar con una letra. MATLAB hace distinción entre letras mayúsculas y minúsculas.

Todas las variables se almacenan en forma de matriz de elementos complejos, aunque pueden introducirse datos reales o enteros. Si la parte imaginaria de todos los elementos de la matriz es cero, entonces MATLAB sólo muestra en pantalla la parte real.

Para introducir números complejos, primero de todo es necesario asignar la raíz cuadrada de -1 a una variable (MATLAB realiza esta asignación por defecto con las variables `i` y `j`). A partir de ese momento, podrá emplearse dichas variables como identificador de la parte imaginaria de un número. Es conveniente recordar esta asignación, pues las variables `i` y `j` se suelen emplear generalmente en los bucles y en ese caso dejan de ser representativas del número imaginario.

En los apartados siguientes se aconseja ir repitiendo desde MATLAB todos los ejemplos que van a ir apareciendo. Esto ayudará a coger confianza e ir descubriendo todas las posibilidades básicas que nos ofrece este paquete de simulación.

El cursor de MATLAB es el símbolo “>>”. Al aparecer en la pantalla, MATLAB espera que se escriba una instrucción para ser ejecutada. Si se ejecuta una instrucción de tipo gráfico (como `plot`) el PC pasa a modo gráfico, sacando una nueva ventana gráfica, conservando en la pantalla de comandos el contenido existente. Pulsando cualquier tecla se abandona el modo gráfico sin perder el contenido. Con el comando `shg` (show graphics) puede volverse al modo gráfico.

Las instrucciones en la ventana de comandos se pueden encadenar en una misma línea, separadas por comas “`,`” o por punto y coma”`;`”. En el primer caso aparecen en la pantalla los resultados de cada instrucción, mientras que en el segundo caso no. Si una instrucción es demasiado larga y ocupa más de una línea, se finaliza la línea con dos puntos “**`..`**” y se continua en la línea siguiente.

El tipo de datos básico que utiliza MATLAB es la matriz (los vectores o las variables escalares son casos particulares de matrices). Para definir una variable en modo interactivo simplemente se ha de asignar un valor a un nombre cualquiera. Este nombre ha de comenzar por una letra y no debe tener más de 18 caracteres. Así por ejemplo haciendo:

```
>>a=5
```

MATLAB ha asignado el valor 5 a la variable a. La variable `ans` (abreviatura de *answer*) está reservada por el sistema para guardar el resultado de una operación sobre la que no se ha especificado ningún nombre, así la instrucción

```
>>3^5+17
```

proporciona una variable `ans=260` (3 elevado a la quinta + 17). Esta variable puede usarse como cualquier otra, así se puede hacer

```
>>A=2*ans
```

y nos dará A=520.

El sistema diferencia entre minúsculas y mayúsculas. Ejecutando,

```
>>aA=a*A
```

se obtiene como respuesta aA=2600.

Observamos que si se pone un "**;**" al final de la instrucción el resultado no aparece en pantalla.

```
>>aA=a*A;
```

Las operaciones básicas entre variables son suma, resta, producto, división, división inversa y potencia: +, -, *, /, \, ^. A continuación, algunos ejemplos:

```
>>1/4  %División normal
ans=0.25
>>1\4  %División inversa
ans=4
>>c=ans^2  %Potencia
c=16
```

1.1.1. Matrices

Existen cuatro modos de definir una matriz en MATLAB.

- Introduciendo explícitamente la lista de elementos de la matriz.
- Generándola mediante comandos y declaraciones disponibles en MATLAB.
- Creándola con ficheros m.
- Cargándola de ficheros MAT.

Un ejemplo típico del primer modo lo tenemos en la siguiente declaración:

```
>>A=[1 2 3;4 5 6;7 8 9]
```

que representa una matriz 3 por 3, llamada A, con los números naturales del 1 al 9.

Para ver el tamaño de la matriz:

```
>>[m,n]=size(A)
```

Los corchetes, [y], se emplean como delimitadores de los elementos de la matriz. El punto y coma se emplea aquí para marcar el fin de cada fila, aunque también puede hacerse esto último con el retorno de carro. Los elementos individuales de cada fila pueden separarse por espacios o por comas. Así que `[1 –2 3]` es lo mismo que `[1,-2,3]` pero no es igual que `[1 -2,3]`.

Los elementos de una matriz pueden ser expresiones matemáticas. Por ejemplo, la sentencia,

```
>>B = [-1.3  sqrt(3)  (1+2+3)*4/5]
```

produce el vector: `[-1.3000  1.7321  4.8000]`.

Puede accederse a los elementos de una matriz de la misma forma que se hace en los lenguajes de programación convencionales, es decir, indicando los subíndices entre paréntesis, donde el primer subíndice es el número de fila y el segundo subíndice el número de columna. Por ejemplo: `A(2,3)` hace referencia al tercer elemento de la segunda fila de la matriz A. Tecléese `A(2,3)` en la línea de comandos del MATLAB seguido de un retorno de carro y compruébese que el resultado devuelto es 6.

También puede tenerse acceso a una fila o columna completa por medio del símbolo "`:`" como subíndice libre. Por ejemplo `A(2,:)` es la segunda fila de A, y `B(:,j)` es la j-ésima columna de B.

Pueden manipularse submatrices empleando un vector como subíndice. El vector puede ser una matriz definida previamente, o puede constituirse con los delimitadores `[` y `]` tal como se explicó más arriba. Por ejemplo, si hacemos `v = [1 3]`, y cogemos el vector B anterior, tendríamos que `B(v)` es igual a `[-1.3000 4.8000]`.

El símbolo "`:`" permite una gran flexibilidad al construir matrices y submatrices, tal y como muestran los ejemplos siguientes:

```
>>x = j:k   % es equivalente a [j,j+l,j+2,...,k] (vacío si j>k).
>>x = j:i:k % es equivalente a [j,j+i, j+2i,...,k].
>>A([1,3],:) = A([3,1],:) % intercambia las filas 1 y 3 de la
                          % matriz A.
>>A(2:3,1:2) % es una submatriz de A formada por las filas 2 hasta
             % la 3 y las columnas 1 hasta la 2.
>>A(:)  % es un vector columna con TODOS los elementos de A
```

También es interesante conocer la existencia de la palabra reservada `end`. Esta palabra representa el índice del último elemento de la fila/columna para la que se ha usado, de esta forma no es necesario conocer de antemano el tamaño de la fila/columna deseada. Por ejemplo:

```
>> A(2,3:end)
```

representa una submatriz de A que contiene la segunda fila y las columnas de la tercera hasta la última.

En MATLAB las matrices se redimensionan automáticamente. Por ejemplo si:

```
>>B = [-1.3000  1.7321 4.8000]
```

entonces

```
>>B(5) = abs(B(1))
```

produce

```
>>B = [-1.3000  1.7321 4.8000  0.000  1.3000]
```

Otra operación matricial importante es el cálculo de la traspuesta de una matriz. La forma de realizarla es empleando el apóstrofe. Existen dos versiones: `A.'`, calcula la matriz traspuesta de A, mientras que `A'` calcula la matriz traspuesta conjugada de A. Una aplicación interesante y directa de esta operación es el cálculo del producto escalar de dos vectores, a y b, escribiendo: `a*b.'`. Por ejemplo, si `a = [1 -1 1]` y `b =[0 2 4]`, entonces tendríamos que

```
>>a*b.'
```

vale `[2]`.

Finalmente, la sentencia A=`[]` borra la variable A asignándole una matriz de dimensión cero por cero.

1.1.2. Números complejos y matrices

El formato de los números complejos en MATLAB es como sigue: el número imaginario `i` ó `j` al igual que el número `pi` están predefinidos

```
>>i %Devuelve ans = i
>>j %Devuelve ans = j
```

Un número complejo se puede generar de la siguiente manera:

```
>>z1=j
```

MATLAB incorpora funciones para la representación gráfica (como veremos mas adelante). La función básica para representar una gráfica es `plot()`.

```
>>plot(z1)  % Representa la parte imaginaria en función de la
           % parte real.
```

Al dibujar se ha hecho una selección automática de los ejes de la representación, y se ha representado un punto en las coordenadas correspondientes.

En ocasiones puede suceder que al dibujar un punto, este no sea visible en la pantalla. La función `plot` permite indicar uno de los siguientes marcadores (`'.'`, `'o'`, `'+'`, `'x'`, `'*'`), y un color para la representación (`'r'`, `'g'`, `'b'`, `'w'`: rojo, verde, azul, blanco).

```
>> plot(z1,'o') % Representa el símbolo o en las
                % coordenadas del punto z1
>>z2=1+2+2*z1 % Definición de un segundo número complejo
>>plot(z2,'x') % Representa el símbolo x en las
                % coordenadas del punto z2
```

Para representar conjuntamente los dos gráficos en uno sólo podemos utilizar la sentencia `hold on`

```
>>hold on %Congela un gráfico, permite la superposición
>>plot(z1, 'x') %Superpone el gráfico de z1 al de
z2
```

Es preciso tener en cuenta que los ejes de la representación son los del primer gráfico dibujado, y si un punto queda fuera de los ejes de la representación, simplemente no aparecerá en la pantalla.

```
>>z3=1/z1
>>plot(z3,'+') %Representa z3 superpuesto a z2 y z1
```

Ahora bien, `z3` no ha aparecido en el gráfico por quedar fuera de los márgenes de la representación. Para anular la superposición de gráficos se opera con la orden

```
>>hold off
```

Para imponer unos ejes concretos a la representación se emplea la función `axis`.

```
>>axis([-1 4 –2 4]) %Fija unos ejes adecuados a la representación
>>plot(z1,'o') %Representa “o” en las coordenadas del punto z1
>>hold on %Congela un gráfico para permitir la superposición
>>plot(z2,'x') %Representa “x” en las coordenadas del puntoz2
>>plot(z3,'+') %Representa z3 superpuesto a ambos z2 y z1
>>hold off %Desactiva el sistema de superposición de gráficos
```

Una forma alternativa para representar un conjunto de números complejos es construir un vector complejo cuyo valor de las coordenadas sea el de los valores complejos a representar. Una manera de hacerlo es asignar cada valor a una componente del vector:

```
>>z(1)=z1; z(2)=z2; z(3)=z3; %Creación del vector z de orden 1x3
```

Otra forma diferente de definir el vector es:

```
>>z=[z1 z2 z3] %Creación del vector z de orden 1x3
>>plot(z) %Representa las coordenadas de z unidas por líneas
>>plot(z,'o') %Representa “o” para cada coordenada del vector
```

MATLAB permite además obtener la parte real, parte imaginaria, módulo y fase de un número complejo mediante las funciones predefinidas **real**, **imag**, **abs**, **angle**. También incorpora la función que calcula el complejo conjugado **conj**.

Es posible generar un vector de números complejos mediante:

```
>>V_real=1:10 %Parte real del vector
>>V_imaginaria=1:10 %Parte imaginaria del vector
>>complex=V_real+V_imaginaria*j
>>rcmp=real(complex) %Componente real del vector complex
>>icmp=imag(complex) %Componente imaginaria del vector complex
>>mcmp=abs(complex) %Módulo del vector complex
>>fcmp=angle(complex) %Fase del vector complex
>>ccmp=conj(complex) %Vector complex conjugado
```

Para operar (`+` , `-` , `*` , `/` , `^`) con todos los elementos de la matriz uno a uno se emplean estos símbolos precedido de un punto (`.+`, `.-`, `.*`, `./`, `.^`).

```
>>cmp1=mcmp.*exp(j*fcmp) %Recupera el complejo con el módulo
                         % y la fase
```

Por otra parte, como ya hemos avanzado, los dos puntos ‘**:**’ se utilizan para indicar un margen de valores. Así por ejemplo, si se quiere generar un vector cuyas componentes sean números enteros desde el 1 al 10 se hace simplemente:

```
>>lista=0:10
```

El primer valor será el primero de la lista, el segundo el último. Por defecto el incremento es la unidad. Si se quiere hacer una lista de los números pares entre 0 y 10, podemos asignar un incremento de dos unidades a la instrucción anterior, intercalando el incremento correspondiente entre la primera y la última componente del vector:

```
>>pares=0:2:10 % Incremento de 2
>>decrece=10:-1:0 %Incremento de -1
```

1.1.3. Operadores aritméticos y lógicos

Los símbolos aritméticos: +, -, *, ^, se emplean respectivamente como operadores para sumar, restar, multiplicar y elevar una cantidad a una potencia. Las matrices que se sumen o resten deben tener la misma dimensión. Mientras que para la multiplicación de, digamos, A*C, el número de filas de A debe coincidir con el número de columnas de C. Para la potenciación, la matriz debe ser cuadrada y el exponente debe ser un número positivo real o entero.

Las barras de división a izquierda y derecha, A\C y C/A, corresponden formalmente a las multiplicaciones izquierda y derecha de C por la inversa de A, esto es $A^{-1}*C$ y $C*A^{-1}$. O dicho de otro modo, el operador división se define como la matriz solución, X, de la ecuación A*X=C, para la división izquierda A\C, y de X*A=C, para la división derecha C/A

Como ya hemos visto, también es posible efectuar la multiplicación o la división elemento a elemento de dos matrices. Para ello sólo hay que utilizar los símbolos `.*` (multiplicación), `.\` (división izquierda) y `./` (división derecha).

Introduzca en MATLAB por teclado dos matrices A y C y pruebe a efectuar con ellas todas las operaciones aritméticas que se han presentado.

Cuatro son los operadores lógicos que tiene MATLAB, el operador AND: &, el operador OR: |, el operador NOT: ~, y el operador OR EXCLUSIVO: `xor(a,b)`. Su utilidad radica en que permiten aunar o modificar operadores relacionales conforme a las reglas del álgebra de Boole. Por ejemplo, `5>4 & 6>4` será cierto (TRUE), mientras que `1==2 | 2==3` será falso (FALSE). Al responder a una operación lógica, MATLAB asigna 1 a los resultados TRUE y 0 a los FALSE. En el siguiente apartado, dedicado a los bucles, se entenderá mejor su uso en MATLAB.

1.1.4. Bucles

MATLAB permite ejecutar repetidamente una serie de instrucciones controlando el momento en que se desea terminar esta operación. Es lo que se conoce por el nombre de bucles. MATLAB tiene dos tipos de bucles: la sentencia **FOR** y la sentencia **WHILE**. La forma general de una sentencia **for** es:

```
for variable=expr
  declaracion;
  ........
  declaracion;
end
```

En la descripción anterior *expr* hace referencia a una expresión aritmética, usualmente un vector del tipo j:k, mientras que *variable* es una variable convencional de MATLAB cuyo contenido puede usarse dentro del bucle. En la sentencia **for** expuesta, *variable* va tomando a cada paso el contenido de *expr*, ejecutándose todas las declaraciones que figuran hasta la palabra **end**. El bucle se lleva a efecto tantas veces como diversos valores pueda tomar *expr*.

El comando **`for`** permite entre otras cosas, generar matrices de modo distinto al visto en el apartado anterior. Concretamente, el siguiente ejemplo muestra cómo generar una matriz de Hilbert de 5 por 5:

```
n=5; % declara el tamaño de la matriz
for i = 1:n  %para las filas 1 a la n
   for j= 1:n  % y para las columnas 1 a la n
        A(i,j)=1/(i+j-1); % crea el elemento (i,j) de la
   matriz
   end
end
```

La forma general de una sentencia **`while`** es:

```
while expr1 relop expr2
  declaracion;
  ............
  declaracion;
end
```

En esta sintaxis, se ha llamado *expr1* y *expr2* a sendas expresiones aritméticas y *relop* a uno de los siguientes operadores relacionales: == (igual que), < (menor que), > (mayor que), <= (menor o igual que), >= (mayor o igual que) o ~= (distinto a). En un bucle **`while`** todas las declaraciones contenidas hasta la palabra *end*, se ejecutan repetidamente mientras que *expr1* y *expr2* estén relacionadas de forma verdadera. En el siguiente ejemplo, MATLAB pide que se introduzca un número hasta que éste esté comprendido entre 1 y 5 ambos inclusive:

```
A=0
while A<1 | A>5
  A=input ('Para salir teclee un numero del 1 al 5: ');
end
```

1.1.5. Caminos condicionados

Una vez que un programa MATLAB comienza su ejecución, es posible hacer que ejecute cosas distintas conforme a alguna condición establecida. Es lo que se conoce como sentencia **`IF-ELSE`**. La forma general de una sentencia **`if-else`** es la siguiente:

```
if expr1 relop expr2
     declaracion A;
     ..............
     declaracion A;
else
     declaracion B;
     ..............
     declaracion B;
end
```

En la sintaxis expuesta, *expr1*, *expr2* y *relop*, tienen el mismo significado que tenían cuando se explicó la sentencia **`while`**. Si *expr1* está relacionada de forma cierta con *expr2*, entonces se ejecutan las declaraciones etiquetadas con A; y, en caso contrario, se ejecutan las declaraciones etiquetadas con B.

En la sentencia **`if-else`** la palabra **`else`** puede ser omitida. Una variante de la sentencia `if-else` es la `if-elseif-else` que permite anidar bucles condicionales dentro de otros bucles condicionales tantas veces como se desee. El siguiente, es un ejemplo de uso de la sentencia `if-elseif-else`:

```
if I==J
  A(I,J) =2;
elseif abs(I-J) == 1;
  A(I,J)=-l;
```

```
else
  A(I,J)=0;
end
```

1.1.6. Formatos de presentación de resultados de MATLAB

Para presentar los resultados numéricos por pantalla, MATLAB posee diversos formatos. Los más relevantes para nosotros son:

short	Punto fijo con 5 dígitos.
long	Punto fijo con 15 dígitos.
short e	Punto flotante con 5 dígitos.
long e	Punto flotante con 15 dígitos.
hex	Hexadecimal.
+	Imprime el signo de los resultados. Ignora la parte imaginaria.
rat	Formato fraccional de presentación de los resultados.

Todos los cálculos en MATLAB se realizan con precisión de tipo doble aunque luego la presentación de los resultados no lo muestre. El comando **format** permite cambiar el formato de presentación de un tipo a otro de los mostrados arriba; para ello, hay que ejecutar **format** seguido de una de las palabras clave que acaban de verse.

1.1.7. Variables reservadas de MATLAB

Aparte de las variables que el usuario de MATLAB pueda declarar para su uso personal, existen una serie de ellas cuya definición y manejo corren por cuenta de MATLAB. Ello no significa que no puedan modificarse, sino que si se toma la determinación de alterar su valor, perderán el propósito para el que fueron diseñadas. Algunas de estas variables son.

eps	Es una variable de tolerancia usada en varios cálculos, por ejemplo, rango y singularidad. Su valor inicial es el número real positivo más pequeño disponible en el ordenador, pero puede ser fijado a otro valor cualquiera.
realmax	Es el número más grande que puede computarse en punto flotante. Cualquier otro mayor produce desbordamiento.
realmin	Es el número más pequeño que puede computarse en punto flotante.
flops	Es un contador que lleva la cuenta del número de operaciones en punto flotante realizadas desde el comienzo de la sesión de trabajo.
NaN	*Not a Number*. Es la representación IEEE que el MATLAB da a resultados de operaciones indeterminadas como 0/0.
inf	Es una variable permanente en formato IEEE que representa una infinidad positiva, como por ejemplo 1/0.
pi	Número π.
end	Representa el último índice de una expresión indexada. Ejemplo: `x=v(3:end);` selecciona todos los elementos de `v` que van desde el tercero hasta el último.

1.1.8. Medidas de tiempos y de esfuerzo de cálculo

MATLAB dispone de funciones que permiten calcular el tiempo empleado en las operaciones matemáticas realizadas. Algunas de estas funcione son las siguientes:

cputime	Devuelve el tiempo de CPU (con precisión de centésimas de segundo) desde que el programa arrancó. Llamando antes y después de realizar una operación y restando los valores devueltos, se puede saber el tiempo de CPU empleado en esa operación. Este tiempo sigue corriendo aunque MATLAB esté inactivo.
etime(t1,t2)	Tiempo transcurrido entre los vectores **t1** y **t2** obtenidos como respuesta al comando **clock**.
tic *ops* toc	Imprime el tiempo en segundos requerido por ***ops***. El comando **tic** pone el reloj a cero y **toc** obtiene el tiempo transcurrido.

Otras funciones permiten calcular el número de operaciones de coma flotante realizadas, como las siguientes:

flops(0)	Inicializa a cero el contador de número de operaciones aritméticas de punto flotante (flops).
flops	Devuelve el número de *flops* realizados hasta el momento.

Por ejemplo, el siguiente código mide de varias formas el tiempo necesario para resolver un sistema de 500 ecuaciones con 500 incógnitas. Téngase en cuenta que los tiempos pequeños (del orden de las décimas o centésimas de segundo), no se pueden medir con gran precisión.

```
>>A=rand(500); b=rand(100,1); x=zeros(500,1);
>>tiempo=clock; x=A\b; tiempo=etime(clock, tiempo)
>>time=cputime; x=A\b; time =cputime-time
>>tic; x=A\b; toc
```

donde se han colocado varias sentencias en la misma línea para que se ejecuten todas sin tiempos muertos al pulsar *intro*. Esto es especialmente importante en la línea de comandos en la que se quiere medir los tiempos. Todas las sentencias de cálculos matriciales van seguidas de punto y coma con objeto de evitar la impresión de los resultados.

1.1.9. Otros comandos de MATLAB

help	Da una breve descripción de diversas características de MATLAB. Si help se antepone a un nombre de función, devuelve una breve ayuda sobre ésta.
demo	Muestra un menú con las demostraciones disponibles.
clear	Borra todas las variables excepto las reservadas de MATLAB. Clear x borra únicamente la variable x.
who	Lista las variables actualmente utilizadas.
whos	Muestra las variables actualmente utilizadas así como su tamaño.
what	Lista los comandos y funciones disponibles actualmente en el directorio de trabajo del disco.

dir	Da una relación de los ficheros que hay en el directorio actual.
echo	Reproduce por consola el texto íntegro de la línea de comandos que se ejecuta en cada instante.
chdir	Cambia el directorio actual por el que se indique a continuación de `chdir`.
type	Muestra el contenido de cualquier fichero ASCII. Si no se indica extensión, MATLAB toma `.m` por defecto.
quit o exit	Finaliza la sesión de trabajo con MATLAB.
why	Siempre tiene respuesta para cualquier problema.
;	Al final de una declaración inhibe la salida por pantalla del resultado.
...	Indica que la declaración continúa en la siguiente línea física.
%	Indica que el resto de la línea es un comentario.
!	Invoca a un comando del sistema operativo.

Si se desea conocer con más profundidad el entorno de simulación MATLAB, puede acudirse a las referencias que se detallan al final de la práctica.

1.2. Modo de trabajo y ficheros editados por el usuario

El sistema MATLAB permite editar en un fichero la secuencia de instrucciones que se quiere que sean ejecutables (scrip-file). Para ello ha de crear y editar un fichero de extensión ".m". Por ejemplo, crear para cada uno de los ejemplos del apartado siguiente un fichero con un nombre determinado (ejemplo.m). Este fichero deberá situarse en un directorio accesible desde MATLAB. Para ejecutarlo bastará teclear `>>ejemplo`, y después pulsar la tecla de retorno de carro.

Se puede utilizar cualquier editor que permita crear un fichero de caracteres. En cambio, no es válido un procesador de textos que guarde informaciones de formato, tipos de letra, etc. ya que los correspondientes caracteres de control serán mal interpretados por MATLAB. En general usaremos el propio editor de texto de MATLAB.

En general, durante el desarrollo de las prácticas de esta asignatura (y del resto de asignaturas de la carrera que empleen Matlab), es conveniente que empleéis una carpeta alojada en el escritorio como directorio de trabajo. En el escritorio, vuestro usuario tiene permisos de lectura y escritura, sin embargo en el directorio por defecto de trabajo de Matlab, los permisos pueden estar restringidos y puede haber problemas a la hora de guardar el trabajo.

1.3. Algunos ejemplos sencillos con MATLAB

A parte de los ejemplos que se expondrán seguidamente, es muy recomendable ejecutar las demostraciones de que dispone MATLAB. Para ello, lo único que hay que hacer es escribir **demo** en la línea de comandos y pulsar la tecla de retorno de carro. Tras esto, aparecerá un menú con todas las demostraciones disponibles. Para moverse por ellas, basta con seguir las explicaciones que MATLAB va dando.

A continuación se dan tres ejemplos ilustrativos de programas escritos para MATLAB. Es aconsejable que el alumno intente razonar cómo funcionan y el efecto que produce cada uno de estos programas antes de pasar a ejecutarlos.

Ejemplo 1. El ejemplo siguiente es una función que demuestra cómo se pueden calcular

medias estadísticas. Si la variable de entrada es un vector, calcula la media y la desviación típica de todos sus elementos. Mientras que si la variable de entrada es una matriz, obtiene un vector fila con los valores medios y las desviaciones típicas de cada columna. Asimismo, este ejemplo sirve para ilustrar cual es el formato típico de un fichero de función tipo M.

Para ejecutar el programa, crear un vector `x`, por ejemplo `x = [4.5 6 7 3 10]`, y teclear en la línea de comandos de MATLAB: `[m,d] = stat (x)`. El resultado que debe obtenerse es `m = 6.1` y `d = 2.3749`.

```
function [mean,stdev] = stat(x) % Declaración de una función
% Esta función calcula la media y la desviación típica
% de todos los elementos de la entrada si
% la variable de entrada es un vector.
% Si la variable de entrada es una matriz,
% obtiene un vector fila con los valores medios
% y las desviaciones típicas de cada columna.

    [m,n]=size(x);
     if m==1
        m=n;
     end
     mean=sum(x)/m;
     stdev=sqrt(sum(x.^2)/m-mean.^2);
```

Ejemplo 2. El presente ejemplo muestra un problema de teoría de números. Consiste en lo siguiente: cojamos cualquier número entero positivo; si es par, se divide por dos; si es impar, se multiplica por 3 y se le suma 1 al resultado. Así se va repitiendo el proceso hasta que se consiga un número entero igual a uno. Lo atractivo del problema, aún no resuelto, es lo siguiente: ¿existe algún entero para el cual el proceso no termine nunca? El siguiente programa en MATLAB ilustra el empleo de los comandos **`while`** y **`if`**. También muestra cómo la función **`input`** permite introducir un número desde el teclado y como el comando **`break`** permite salir de un bucle:

```
%Problema "3n+1" clásico de teoría de números
[m,n]=size(x);
if m==1
   m=n;
end
mean=sum(x)/m;
stdev=sqrt(sum(x.^2)/m-mean.^2);
```

Este programa representa un buen ejemplo de un fichero de tipo script. Para ejecutarlo solo hay que teclear su nombre en la línea de comandos de MATLAB. El nombre que se ha dado al fichero del programa ha sido: *tres.m*.

Ejemplo 3. El siguiente ejemplo es otro fichero script que muestra cómo puede emplearse MATLAB para calcular la densidad espectral de potencia de una señal. Se trata de dos tonos de 50 y 120 Hz a los que se les ha añadido ruido blanco y gaussiano. Se ilustra el uso de la Transformada Rápida de Fourier, **`fft`**, y del cálculo de la conjugada, **`conj`**, con vistas a calcular el módulo al cuadrado de una señal.

```
% Calcula el espectro de dos tonos contaminados por ruido blanco
% gaussiano

fs=1e3;  %Frecuencia de muestreo
Ts=1/fs; %Periodo de muestreo
t=0:Ts:0.25; %Vector de instantes de muestreo
y=sin(2*pi*50*t)+2*sin(2*pi*120*t); %Señal muestreada, tonos de 50 Hz
                                    % y 120 Hz
yn=y+2*randn(size(t)); %Señal muestreada más ruido
plot(t,y),title('Señal en el dominio del tiempo');
figure;
plot(t,yn),title('Señal en dominio t contaminada con ruido');
figure;
plot(yn(1:50)),title('Representamos solo una parte');
Yn=fft(yn,256);   %Transformada de Fourier
Syy=Yn.*conj(Yn); %Densidad espectral de energía
f=1000/256*(0:127); %Creamos los ejes de frecuencia para la DEE
plot(f,Syy(1:128)),title('DEP'); %Solo para la mitad de frecuencias,
                                 % las otras son simétricas.
figure;
plot(f(1:50),Syy(1:50)),title('Zoom de la DEP');
echo off %Para que no salgan las instrucciones por pantalla
```

Para ejecutar este programa teclear *espectro* seguido del retorno de carro desde la línea de comando de MATLAB.

2. Gráficos bidimensionales

La representación gráfica con MATLAB está orientada fundamentalmente a la representación de vectores y matrices (lo cual es bastante lógico 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,….; 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, ya que no es el objeto de esta práctica) y, finalmente, se proponen unos *ejercicios que el alumno debe resolver*.

2.1. Funciones gráficas 2-D elementales

MATLAB dispone de cuatro funciones básicas para crear gráficos 2-D. Estas funciones se diferencias 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 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.

Al trabajar con señales discretas, la información está representada por una serie de muestras individuales. El comando plot, por defecto une todas las muestras mediante una línea, dando lugar a la sensación de que la señal es continua y no discreta. Si se desea una representación en la que se vean todas las muestras por separado deberá emplearse el comando **stem** en lugar de plot.

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('titulo')`	Añade un título al gráfico
`xlabel('eje x ')`	Añade una etiqueta en el eje de abscisas. Con `xlabel off` desaparece.
`ylabel('eje y ')`	Añade una etiqueta en 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), sustituyendo cualquier cosa que hubiera dibujada anteriormente en esa ventana, o bien crea una nueva ventana si no hay ninguna abierta. 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.

2.1.1. 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 se 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 cómo 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= [10 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 dibuja 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)), '+')
```

Aquí 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 más información sobre la función **plot** se aconseja hacer uso del comando **help**.

2.1.2. Estilos de línea y marcadores

En la sección anterior hemos visto cómo 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
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.

2.1.3. 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 cómo 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
```

2.1.4. 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 subventana, así como rótulos a los ejes. Para volver a la opción por defecto basta con teclear el comando:

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

2.1.5. 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

2.2. 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.

Para practicas un poco con todo lo que se acaba de explicar, 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 coseno(x) –en verde-')
>>xlabel('Angulo 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('off'), figure(gcf)
>>axis('on'), figure(gcf)
>>axis('on'), grid, figure(gcf)
>>axis('square'), figure(gcf)
>>axis('off'), figure(gcf)
>>axis('on'), figure(gcf)
>>axis('on'), grid, figure(gcf)
```

2.3. 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 más 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)
```

2.4. 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.

2.5. Gráficos 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);
```

Ejecuta 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 de las representaciones tridimensionales en MATLAB está fuera de los objetivos de esta práctica, por tanto solo se han presentado nociones muy básicas. Para más información consultar el comando de ayuda **help**.

3. Ejercicios

En este apartado de la práctica se proponen tres ejercicios a realizar para que el estudiante comience a familiarizarse con MATLAB. Es aconsejable que el alumno guarde, de cara a estudiar para el examen, el código utilizado para resolverlos, las figuras que se obtengan como resultado y una breve explicación de lo realizado cuando sea necesario.

Ejercicio 1:

Genera una matriz cuadrada cualquiera y calcula:

```
>>C=A.^2
>>D=A^2
```

¿Son iguales? ¿Por qué?

Ejercicio 2:

Representa gráficamente la sinusoide $\mathrm{x(t)}$ y la parte real de la señal compleja $\mathrm{y(t)}$,

$$x(t) = 3 \cdot \cos\left(2\pi f_0 t + \frac{\pi}{4}\right)$$

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

con $f_0 = 1/T_0 = 100$ Hz y $f_1 = 300$ Hz.

Para ello, en primer lugar es necesario generar un vector de reales en el intervalo de tiempos adecuado. Esto 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.
```

Haz una representación aproximada de ambas señales eligiendo un intervalo de tiempo que se corresponda con dos periodos de la señal $x(t)$. Para obtener una buena representación debes considerar un número de puntos suficientemente grande. Debes calcular cuál ha de ser el incremento de tiempo que hay que utilizar en la representación en función del número de puntos.

Compara el resultado obtenido para $x(t)$ con la representación gráfica de la parte real de la señal $y(t)$, comentando la relación entre las frecuencias de las dos señales.

Ejercicio 3

Representa las señales siguientes con $f_2 = 0{,}01$ Hz en el intervalo comprendido entre $t = 0$ y $t = 500$ s, considerando un espaciado temporal entre muestras suficientemente cercano:

$$x(t) = e^{\frac{-t}{150}}$$

$$y(t) = \cos\left(2\pi f_2 t\right)$$

$$z(t) = 5 \cdot x(t) \cdot y(t)$$

4. Bibliografía

Para más información se aconseja al estudiante la consulta de los siguientes libros:

[1] The Math Works Inc., *Getting Started with MATLAB*, 2014:
http://www.mathworks.es/help/pdf_doc/matlab/getstart.pdf

[2] A.V. Oppenheim, A.S. Willsky, I.T. Young, *Señales y Sistemas (2ª edición)*, Prentice-Hall, 1998.