Manual de Introducción a MATLAB

Autores: Manuel López Martínez y José Ángel Acosta Rodríguez

Índice general

Capítulo 1

Introducción a MatLab. Parte I

1.1. Introducción

En estas notas se pretende realizar una introducción muy básica a MATLAB, orientada fundamentalmente al estudio de sistemas de control. En líneas generales, MATLAB es un sistema interactivo basado en matrices para cálculos científicos y de ingeniería. Desde el punto de vista del control, MATLAB se puede considerar un entorno matemático de simulación que puede utilizarse para modelar y analizar sistemas. Sirve para estudiar sistemas continuos, discretos, lineales y no lineales.

MATLAB constituye un entorno abierto, para el cual numerosas paquetes específicos adicionales (toolboxes) han sido desarrollados. En el caso que nos ocupa se utilizará fundamentalmente la 'Control System Toolbox' . Estos paquetes específicos adicionales están constituidos por un conjunto de funciones que pueden ser llamadas desde el programa y mediante las cuales se pueden realizar multitud de análisis.

Las notas se centrarán fundamentalmente en aquellos aspectos y funciones que más interés tengan desde el punto de vista de control, instando al lector a que busque en el manual de usuario cualquier información adicional que desee. Para el desarrollo de las mismas se ha utilizado tanto la experiencia programando en MATLAB de los autores, como una serie de referencias básicas.

El núcleo fundamental de MATLAB se encuentra en los subdirectorios BIN y MATLAB. En BIN se encuentran los programas ejecutables. El subdirectorio MATLAB contiene los ficheros .m (aunque serán explicados posteriormente, comentamos brevemente que consisten en ficheros escritos a base de comandos de MATLAB y que realizan una función determinada), que contienen las funciones básicas para el funcionamiento de MATLAB. En este

sentido, es necesario comentar que MATLAB cuenta con dos tipos básicos de funciones:

Las llamadas ***built-in functions***: Son funciones que MATLAB tiene incorporadas internamente y por tanto no son accesibles al usuario.

Funciones ***m functions***: Son funciones cuyo código es accesible. Las que se encuentran en el subdirectorio MATLAB son las básicas para el funcionamiento del sistema.

Las toolboxes se suelen instalar en forma de subdirectorios en el disco duro, colgando del subdirectorio TOOLBOX(en la versión WINDOWS). En ellos se encuentran también funciones .m orientadas al control de sistemas. Además, se pueden incorporar otros toolboxes (SIGNAL PROCESSING, ROBUST CONTROL, etc), e incluso funciones propias del usuario.

En el caso de las versiones para WINDOWS, el arranque del programa se realiza 'pinchando' con el ratón en el icono correspondiente. Para obtener información adicional se aconseja mirar el manual de usuario.

1.2. Entorno

Una vez arrancado MATLAB, aparece el prompt o línea de comandos del sistema (≫). Este es el momento de comentar la existencia del comando más famoso de cualquier aplicación: **help**. Introduciendo este comando aparecerán todas las citadas ***built-in functions***, las contenidas en el subdirectorio MATLAB y todas aquellas contenidas en los subdirectorios incluidos en el PATH(ver cuadro1.1).

Para obtener información sobre cualquiera de las funciones se introduce **help nombre-función**. Ejemplo: help cos (cos es una función que calcula el coseno de un número). Una cuestión importante a tener en cuenta es que MATLAB distingue entre **mayúsculas** y **minúsculas**. En este sentido, los nombres de función se introducirán en minúsculas. El comando **demo** permite obtener una demostración de las 'capacidades' del sistema.

1.2.1. Funciones y símbolos

- Si se quiere guardar toda la sesión en un archivo (comandos introducidos y resultados), basta usar el comando **diary** nombre-archivo y se guardará la sesión en un archivo llamado diary. Cuando no se quiera seguir almacenando la información se introducirá **diary off** .

- El símbolo % sirve para poner comentarios en los programas (todo lo escrito desde ese símbolo hasta el final de la línea no se ejecutará).

c:\matlab	Establece los parámetros de la sesión MATLAB
matlab\general	Comandos de propósito general
matlab\ops	Operadores y caracteres especiales
matlab\lang	Construcción del lenguaje y debugging
matlab\elmat	Matrices elementales y manipulación de matrices
matlab\specmat	Matrices especiales
matlab\elfun	Funciones matemáticas elementales
matlab\specfun	Funciones matemáticas especiales
matlab\matfun	Funciones matriciales - álgebra lineal numérica
matlab\datafun	Análisis de datos y funciones de transformada Fourier
matlab\polyfun	Funciones polinomiales y de interpolación
matlab\funfun	Funciones de funciones - métodos numéricos no lineales
matlab\sparfun	Funciones para matrices dispersas
matlab\plotxy	Gráficos en dos dimensiones
matlab\plotxyz	Gráficos en tres dimensiones
matlab\graphics	Funciones gráficas de propósito general
matlab\color	Funciones para control de color, brillo y contraste
matlab\sounds	Funciones para procesamiento de sonido
matlab\strfun	Funciones de cadenas de caracteres
matlab\iofun	Funciones de Entrada-Salida de bajo nivel
matlab\demos	La Expo de MATLAB y otras demostraciones
simulink\simulink	Análisis de modelos en SIMULINK y funciones de construcción.
simulink\blocks	Librería de Bloques de SIMULINK
simulink\simdemos	Demostraciones y ejemplos de SIMULINK
toolbox\control	Control System Toolbox
toolbox\local	Librería de funciones locales

Cuadro 1.1: Listado del comando help

- Si lo que se desea es almacenar todas las variables de memoria (y sus valores actuales) en un fichero, se usa el comando **save** nombre-fichero. Esto crea un fichero con el nombre introducido y con extensión .MAT. Si no se pone nombre del fichero crea uno llamado MATLAB.MAT. En caso que se desee guardar en un fichero con formato ASCII, se introducirá en el comando un modificador **save -ascii** nombre fichero ascii. Si sólo se quieren guardar una serie de variables se introducirá **save** nombre-fichero nombre-variables separadas por espacios.
- Para recuperar los ficheros generados con el comando **save** se utilizará **load** nombre-fichero.
- El comando **what** muestra los ficheros **.m** que se encuentran en el disco duro en el subdirectorio desde el cual se haya invocado a MATLAB.
- **dir** muestra todos los ficheros contenidos en el subdirectorio actual.
- Con el comando **delete** se puede borrar cualquier archivo del disco duro.
- **chdir** permite cambiar de directorio.
- El comando **type** permite ver el contenido de cualquier archivo en formato ASCII.
- Para borrar alguna variable de memoria se utiliza **clear** nombre-variables separadas por espacios.
- Para parar la ejecución de un comando se usa **Ctrl c**.
- Para finalizar la ejecución de MatLab se escribe **quit** o **exit**.

1.3. Variables y operadores

Los operadores básicos que usa Matlab son:

- **Aritméticos**:
  - Suma: +
  - Resta: -
  - Multiplicación: *
  - División : /

- Potencia: $\wedge$

- **Lógicos y Relacionales**: Permiten la comparación de escalares (o de matrices elemento a elemento). Si el resultado de la comparación es verdadero, devuelven un 1, en caso contrario devuelven un 0. Los operadores elementales son:

  - $<$ menor que
  - $<=$ menor o igual
  - $==$ igual
  - $>$ mayor que
  - $>=$ mayor o igual
  - $\sim=$ no igual

  Es importante no dejar espacios entre los operadores formados por dos símbolos. Para datos complejos se compara ($==$ y $\sim=$) tanto la parte real como la imaginaria.

  Por otro lado, se pueden usar variables de tipo carácter, cadena de caracteres, booleanas, bytes, enteros y flotantes.

  Para asignar un valor a una variable se escribe el nombre de la variable, el símbolo $=$, y el valor de la misma, o bien el nombre de otra variable previamente inicializada.

  Ejemplo:

```
>> a=100;
>> b=2;
>> c=a
c =
   100
```

  Si al final de la introducción del comando no se pone punto y coma (**;**), aparece el resultado explícitamente en pantalla. En caso contrario se ejecuta pero no muestra el resultado, almacenándolo en la variable a la que se asigna o si no se asigna se guarda en una variable de entorno llamada **ans**.

  De igual modo podemos realizar operaciones entre variables, del ejemplo anterior vamos a multiplicar a y b.

  Ejemplo:

```
>> d=a*b
d =
    200
```

MatLab tiene predefinidas una serie de **variables y constantes especiales**

- ans : respuesta cuando no se asigna expresión.
- eps : precisión mínima de la máquina.
- pi : π
- i, j : $\sqrt{-1}$
- inf: ∞
- NaN: Not a number.
- clock: Reloj.
- date : Fecha.
- flops: Número de operaciones en coma flotante.

Las variables a las que se asignan resultados, así como las variables de entorno, se almacenan en el 'espacio de trabajo'(workspace).

El comando **who** muestra las variables existentes en el entorno generadas por el usuario (pero no las variables especiales). El formato de salida puede modificarse usando **format** (short, long etc).

1.4. Vectores y Polinomios

Los vectores se introducen entre corchetes, y sus elementos están separados por espacios o comas.

Ejemplo:

```
>>v=[77 69 11 88]
v =
    77     69     11     88
```

Los elementos de los vectores se referencian usando índices entre paréntesis. Los índices en MatLab empiezan en 1.

Ejemplo: Para el elemento 2 del vector v

```
>>v(2)
ans =
    69
```

Se pueden referenciar varios elementos a la vez usando el operador **:**.
Ejemplo:

```
>>v(2:3)
ans =
    69 11
```

Los **polinomios** se representan por **vectores**, conteniendo los coeficientes del polinomio en orden descendente. Por ejemplo, el polinomio s^3+2s^2+3s+4 se representa:

```
p=[ 1 2 3 4] ;
```

Mediante la función **roots** se pueden encontrar las raíces de esa ecuación.

```
roots(p)
```

Del mismo modo, se puede calcular un polinomio a partir de sus raíces usando la función **poly**.

```
p2=poly([-1 -2]);
```

Si el argumento de entrada a **poly** es una matriz, devuelve el polinomio característico de la matriz ($det|\lambda I - A|$) como un vector fila.

Un polinomio puede ser evaluado en un punto determinado usando **polyval**.

```
ps=polyval(p,s)
```

donde **p** es el polinomio y **s** es el punto donde va a ser evaluado. Por ejemplo:

```
p2=[ 1 3 2] ; a=[ 1 2; 3 4] ; polyval(p2,a)
```

si se introduce en vez de un valor un vector o una matriz, la evaluación se hace elemento a elemento.

Los polinomios se pueden multiplicar y dividir usando las funciones **conv** y **deconv** respectivamente.

Ejemplo:

```
>> A=[1 -1]; % x-1
>> B=[1  1]; % x+1
>> C= conv(A,B)  %  x^2-1
C =
     1     0    -1
>> polyval(C,1)
ans =
     0
```

1.5. Matrices

El elemento básico en MATLAB es una matriz compleja de doble precisión, de forma que abarca realmente todo tipo de datos (desde números reales hasta complejos) y de estructuras de datos (escalares, vectores y matrices). Así por ejemplo, se pueden introducir:

A=[1 0 2; 2 2 0; 0 0 1]

A partir de esta representación se pueden comentar varias cosas:

- Para separar filas se usa **;** o bien al introducirlas se pulsa **return**.
- Para transponer matrices se usa el apóstrofe **'**.
- Los elementos de vectores y matrices pueden ser reales, complejos e incluso expresiones.
- Si se está introduciendo un comando o conjunto de ellos cuya sintaxis es muy larga, se puede continuar en la siguiente línea introduciendo al final de la actual tres puntos seguidos (**. . .**).
- Otras formas de introducir matrices:
  - Lista explícita de elementos.
  - Generándola mediante funciones y declaraciones.
  - Creándola en un archivo **.m** (matrices .m).
  - Cargándola de un archivo de datos externo (ficheros de datos ASCII y ficheros con formato **.mat**).

El comando **size** devuelve el número de filas y columnas de una matriz y **length** la mayor dimensión.

Ejemplo:

```
>> A=[ 1 0 2; 2 2 0; 0 0 1]

A =

     1     0     2
     2     2     0
     0     0     1

>> size(A)

ans =

     3     3
```

Los elementos de una matriz se referencian de la forma **A(i,j)** donde **i** y **j** son los índices del elemento correspondiente. En este punto es importante comentar uno de los elementos más potentes de MATLAB, que es el símbolo **:** , que permite referenciar varios elementos de una matriz, así por ejemplo:

A(1, 2: 3) daría como resultado los elementos de las columnas 2 y 3 pertenecientes a la primera fila.

A(: , 2) daría como resultado todos los elementos pertenecientes a la segunda columna.

1.5.1. Operaciones con matrices

Las operaciones comunes con matrices son:

- Suma: +
- Resta: -
- Multiplicación: *
- División derecha: / (x=b/A es la solución de x*A=b).
- División izquierda: \ (x=A\b es la solución de A*x=b).
- Potencia: $\wedge$
- Traspuesta: '

Las mismas operaciones se pueden realizar elemento por elemento anteponiendo un punto . a cualquiera de los operandos anteriores (ejemplo: Para hacer el producto de los elementos (i,j) de las matrices A y B, se haría A.*B).

Además de las operaciones anteriores existen las trigonométricas estándar (sin, cos , tan, asin, acos , atan, atan2), funciones hiperbólicas (sinh, cosh, tanh, asinh, acosh, atanh) , funciones trascendentales (log, log10, exp, sqrt) y funciones normales de manipulación matricial:

- det : determinante.
- inv: inversa.
- eig: Obtención de autovalores.
- rank: rango de la matriz.
- norm: norma.
- trace: traza de la matriz.
- real : parte real.
- imag: parte imaginaria.
- abs : valor absoluto.
- conj: conjugada.

Ejemplo:

```
>> A

A =

     1     0     2
     2     2     0
     0     0     1

>> det(A)   % Determinante de la matriz A

ans =

     2
```

```
>>  trace(A) % Traza de la matriz A

ans =

     4

>> inv(A) % Inversa de la matriz A

ans =

    1.0000         0   -2.0000
   -1.0000    0.5000    2.0000
         0         0    1.0000

>> B=rand(3) % Matriz 3X3 de elementos aleatorios entre 0 y 1

B =

    0.4447    0.9218    0.4057
    0.6154    0.7382    0.9355
    0.7919    0.1763    0.9169

>> D=A*B

D =

    2.0286    1.2743    2.2395
    2.1203    3.3200    2.6824
    0.7919    0.1763    0.9169
```

1.6. Funciones Avanzadas

En esta sección simplemente comentaremos que existen una serie de funciones, muy útiles en problemas de integración numérica (**quad**, **quad8**), solución de ecuaciones diferenciales, importantes cuando se estudian los sistemas dinámicos (**ode23**, **ode45**), ecuaciones no lineales e interpolación (**fmin**, **fsolve** etc.), interpolación (**spline**), funciones orientadas al análisis de datos, **min**, **max**, **mean**, **median**, **std**,**sum**, **prod**, **cumsum**, **cumprod** etc.

1.7. Ficheros Scripts

MATLAB puede ejecutar programas que se encuentren almacenados en ficheros ASCII que se encuentren en alguno de los subdirectorios indicados en el PATH o bien en el subdirectorio de trabajo actual y tengan además extensión .m.

Los Scripts son ficheros .m en los que se ponen secuencialmente comandos de MATLAB que se ejecutan en ese orden al introducir el nombre del fichero .m (sin extensión). Operan globalmente con los datos que se encuentran en la memoria, es decir, las variables usadas son variables globales, un cambio en el valor de la variable en el Script actúa sobre la variable en memoria del mismo nombre.

A continuación se va a mostrar un ejemplo de Script. Se muestra el código del fichero .m y se presentan los resultados obtenidos en MatLab tras ejecutar el script. Para ello basta escribir en línea de comando el nombre del fichero excluyendo la extensión.

Ejemplo: Fichero .m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%   Ejemplo de Script:  prueba.m   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%   Operaciones con Matrices
%
%

A=[1 2 3; 4 5 6] B=[1 2; 3 4 ; 5 6]
C=A*B       % Producto de A por B
T=inv(C)^2  % Cuadrado de la inversa de C
Tt=T'       % Traspuesta de T
```

Ejemplo: Ejecución del Script

```
>> prueba

A =

     1     2     3
     4     5     6
```

```
B =

     1     2
     3     4
     5     6


C =

    22    28
    49    64


T =

    4.2191   -1.8580
   -3.2515    1.4321


Tt =

    4.2191   -3.2515
   -1.8580    1.4321
```

Capítulo 2

Introducción a MatLab.Parte II

En esta segunda práctica se van a tratar más herramientas de Matlab. Entre ellas se verán generación de gráficos y funciones en Matlab para los que será necesario estudiar el control de flujo de programas.

2.1. Gráficos

Para dibujar gráficos es preciso generar la tabla de valores correspondiente. Para ello MatLab dispone de dos funciones, linspace y logspace, que permiten generar vectores de puntos espaciados de forma lineal o logarítmica respectivamente.

- **x=linspace(a,b,n)** Genera un vector de **n** puntos desde **a** hasta **b**, cuyos componentes están espaciados linealmente.
- **x=logspace(a,b,n)** Genera un vector de **n** puntos desde **a** hasta **b**, cuyos componentes están espaciados logarítmicamente.

Para hacer gráficos en dos dimensiones (2D) se utiliza la función **plot** cuya sintaxis básica es:

- **plot(X,Y)** dibuja el vector Y frente al vector X. Se permite dibujar varios gráficos en una misma figura. Para ello la sintaxis es plot(X1,Y1,X2,Y2,...). Si se desea diferenciar las distintas gráficas, se pueden cambiar las propiedades de representación de las mismas, es decir, se puede especificar el color y tipo de línea. Esto se puede ver en la figura 2.1

Para poner título tanto a la figura como a los ejes coordenados existen una serie de funciones:

- **title('Título de la figura')**.
- **xlabel('Título del eje x')**.
- **ylabel('Título del eje y')**.
- **legend('gráfica1','gráfica2')**: Escribe una leyenda asociando un nombre a cada gráfica.
- **grid**: genera una rejilla sobre la gráfica para facilitar la interpretación de la misma.

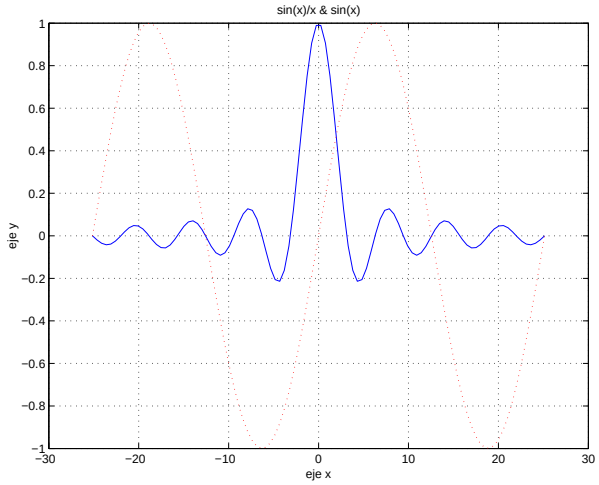

Figura 2.1: Ejemplo de función plot

Ejemplo: Script para generar una figura 2D

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%                                                                     %
%%         Funcion que genera un grafico de 2D                         %
%%                                                                     %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

x = linspace(-8*pi,8*pi,100);
y = sin(x)./x;      % ./ representa division elemento
                    % a elemento de dos vectores

figure; plot(x,y,'b',x,sin(x/4),'r:'), title('sin(x)/x & sin(x)'),
xlabel('eje x'),ylabel('eje y'), grid;
```

Por otro lado, Matlab permite realizar gráficas en tres dimensiones (3D). Las gráficas en 3D se definen mediante vectores o matrices de datos en función de que se dibuje una línea o una superficie.

Usaremos los siguientes comandos, además de los previamente comentados para gráficas 2D:

- **plot3(X,Y,Z)** Permite dibujar curvas en 3D. Dibuja el vector Z frente a los vectores X e Y. Se permite dibujar varios gráficos en una misma figura. Para ello la sintaxis es plot3(X1,Y1,Z1,X2,Y2,Z2...). Si se desea diferenciar las distintas gráficas, se pueden cambiar las propiedades de representación de las mismas, es decir, se puede especificar el color y tipo de línea. Esto se puede ver en la figura 2.2

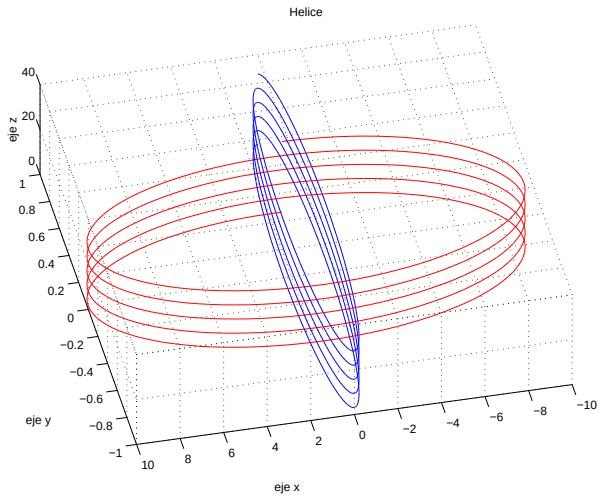

Figura 2.2: Ejemplo de función plot3

Ejemplo: Script para generar una curva 3D

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%                                                                     %
%%         Funcion que genera una curva de 3D                          %
%%                                                                     %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

t = 0:pi/50:10*pi; % : alternativa al comando linspace

figure; plot3(sin(t),cos(t),t,'b',10*sin(t),cos(t)/2,t,'r');
```

```
title('Helice'), xlabel('eje x'),
 ylabel('eje y'), zlabel('ejez'),grid;
```

- **[X,Y]=meshgrid(x,y)**: Genera una rejilla de puntos a partir de los vectores X e Y.
- **mesh(x,y,z), surf(x,y,z)** Para dibujar superficies en 3D. z es el valor que toma la función z=f(X,Y) en el punto de la rejilla X,Y.

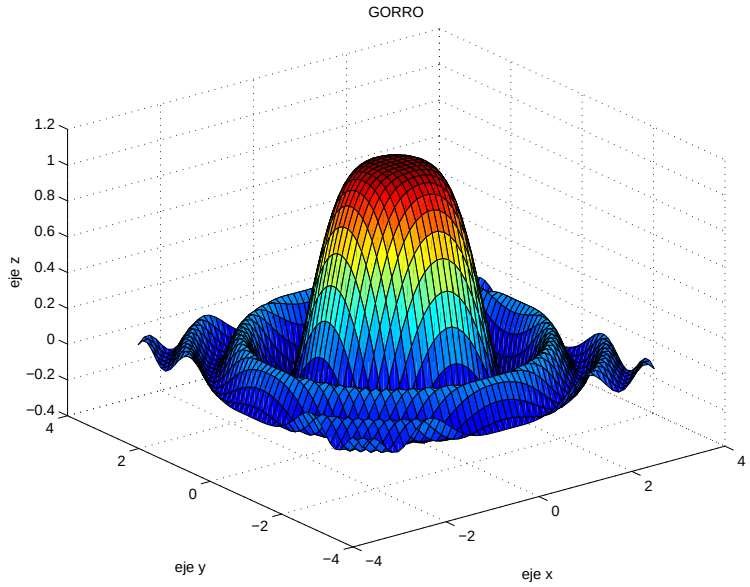

Figura 2.3: Ejemplo de función surf

Ejemplo: Script para generar una superficie 3D

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%                                                                      %
%%      Funcion que genera una superficie de 3D                         %
%%                                                                      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

x=linspace(-pi,pi,50);
y=linspace(-pi,pi,50);
[X,Y]=meshgrid(x,y);

z=sin(1.1*(X.^2+Y.^2))./(X.^2+Y.^2);
```

```
figure; surf(x,y,z);

title('GORRO'), xlabel('eje x'), ylabel('eje y'), zlabel('eje z'),
grid;
```

2.2. Programando en MatLab

MATLAB permite programar una serie de elementos controladores de flujo. La sintaxis es muy parecida a la de cualquier lenguaje de programación. Todos estos operadores se pueden usar en un fichero **.m**.

2.2.1. Bucles y estructuras condicionales

Veremos algunos de los comandos de control de flujo de programas en MATLAB: **for**, **while**, **if-else**.

for:

Sintaxis:

```
for variable = expresion
 hacer algo;
end
```

La expresión es un vector, una matriz o cualquier comando de MATLAB que produzca como salida un vector o una matriz. La ejecución se realiza una vez por cada elemento del vector o de una columna de la matriz.

Ejemplo, donde la variable i pasa por los valores 10, 9, ..., 1:

```
for i=10:-1:1
 kk(11-i)=i ;
end
```

Como se observa, los bucles (y las estructuras condicionales) se terminan con end. Es importante evitar en la medida de lo posible el uso de bucles en MATLAB, ya que consumen mucho tiempo, pudiéndose en muchos casos realizar las mismas operaciones de una forma más eficiente. Los siguientes ejemplos calculan logaritmos de números desde 1 a 10.000. Se hará de diferentes maneras para comparar. Se utilizan los comandos **clock** y **etime** para calcular el tiempo consumido en las operaciones.

- **clock**: Hora actual.
- **etime**: Devuelve el tiempo en segundos que ha transcurrido entre dos instantes de tiempo.

Método 1:

```
t1=clock;
for i=1: 10000,
    a(i)=log(i);
end;
e1=etime(clock,t1)
```

Método 2:

```
t1=clock; ind=[ 1: 10000];
a=zeros(1,10000);
a=log(ind);
e2=etime(clock,t1)
```

El tiempo de computación para el método 2 es del orden de 50 a 100 veces menor que para el método 1, dependiendo de la máquina.

Las causas de la disminución importante de tiempos es que en el primer método, MATLAB tiene que recalcular la dimensión del vector en cada pasada por el bucle (importancia de las inicializaciones), y además usa bucles for , que como se ha indicado, consumen mucho tiempo. Esto por supuesto no quiere decir que no deban usarse, pues habrá ocasiones en que no haya más remedio, pero siempre que haya una forma alternativa de hacerlo, ésta será preferible al uso de bucles.

while:

Permite bucles condicionales. Su sintaxis es:

```
while expresion,
   hacer algo,
end;
```

La expresión es de la forma X operador Y, donde X e Y son escalares o expresiones que devuelven escalares y los operadores suelen ser operadores relacionales. En el siguiente ejemplo se busca una matriz aleatoria con parte real de autovalores negativa:

```
rand(normal);
a=rand(2);
while max(real(eig(a)))>=0,
    a=rand(2);
end;
eig(a)
```

if, **else**, **elseif**:

La sintaxis es la siguiente:

```
if expresion1,
    hace algo,
    hace otras cosas,
elseif expresion2,
    hace algo,
    hace otras cosas,
else
    hace algo,
end
```

else y elseif son opcionales, no así end que es obligatorio para acabar la instrucción. Se puede usar **break** para salir de un bucle si se cumple la condición incluida en el if.

2.3. Funciones

Además de los script-files, hay otro tipo de ficheros .m: los **ficheros de funciones**.

A diferencia de los scripts anteriores, se le pueden pasar argumentos y pueden devolver resultados. Por tanto utilizan variables que se pasan por valor. La mayoría de los ficheros contenidos en las diferentes 'toolboxes' son funciones. La sintaxis de todas las funciones almacenadas en ficheros .m es la siguiente:

```
function[sal1,sal2,...] =nombre_fichero(ent1,ent2,...)
% Comentarios adicionales para el help
comandos de MATLAB
```

Una función puede tener múltiples entradas y salidas.
Ejemplo:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%                                                  %
%%          Funcion que calcula la media y          %
%%                                                  %
%%          la varianza de un vector de 3D          %
%%                                                  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%  [media,varianza] = funcion(vector)
%
 function [media,varianza] = funcion(x)
          n = length(x);
          media = med(x,n);
          varianza = sum((x-med(x,n)).^2)/n;

          %-------------------------
          function media = med(x,n)
          %subfuncion
          media = sum(x)/n;
```

Para calcular la media y la varianza del vector [6,4] se debe escribir lo siguiente:

```
>>[m,v]=funcion([6,4])

m =

     5


v =

     1
```