DECSAI
Departamento de Ciencias de la Computación e I.A.
Universidad de Granada

Inteligencia Artificial en Investigación Operativa

Curso académico 2012/2013

Práctica 0

Tutorial de MATLAB

© *Fernando Berzal*

ENTREGA DE LA PRÁCTICA
(a través del acceso identificado de DECSAI)
`https://decsai.ugr.es/`

```
resultados.m
figura.m
macro.m
hipoteca.m
```

Contenido

MATLAB

MATLAB (http://www.mathworks.com/products/matlab/) es un paquete de software matemático muy utilizado en distintos ámbitos profesionales y científicos. MATLAB, cuyo nombre proviene de *MATrix LABoratory* ("laboratorio de matrices"), ofrece un entorno de desarrollo integrado (IDE) e incluye un lenguaje de programación propio (el lenguaje M). Web:

GNU Octave (http://www.gnu.org/software/octave/) es un programa "*open source*" que puede utilizarse como sustituto de MATLAB y también incluye un intérprete de un lenguaje "similar" al de MATLAB (casi idéntico en muchos sentidos).

Consideraciones previas: Algunos comandos útiles de MATLAB

- Consultar el directorio de trabajo actual [*print working directory*]:

```
pwd
```

- Cambiar de directorio [*change directory*]:

```
cd c:/matlab/proyecto
```

- Cambiar el "*path*" de MATLAB:

```
path(path,'c:/matlab/proyecto')
```

- Limpiar la pantalla de la línea de comandos [*clear command window*]:

```
clc
```

- Ayuda (opción "*Product Help*" del menú "*File*", accesible con la tecla F1).

```
help ...
```

- Salir de MATLAB (opción "*Exit MATLAB*" del menú "*File*" o combinación de teclas Control+Q).

```
exit
```

o también

```
quit
```

Operaciones básicas en MATLAB

Expresiones

Usando MATLAB como si fuese una calculadora, tecleamos

```
>> 1+1
```

y obtenemos el resultado

```
ans =
  2
```

El resultasdo de la operación queda almacenado, por defecto, en la variable `ans`.

Operaciones aritméticas

Operación aritmética	Operador
Suma	+
Resta	-
Multiplicación	*
División	/
Exponenciación	^

Compruebe el funcionamiento de los distintos operadores aritméticos tecleando las siguientes expresiones en la línea de comandos de MATLAB:

```
495+594     % D. Acheson, 1089 and All That: A Journey into Mathematics,
732-237     % Cualquier otro número de 3 cifras también vale ;-)
3333*3334
355/113
2^10
```

EJERCICIO 1: Indique el valor del número de Avogadro en el fichero `resultados.m`.

Los números reales se pueden expresar en notación científica, de forma que `1.234e56` equivale a `1.234*10^56`.

Las expresiones aritméticas se pueden combinar como deseemos si utilizamos paréntesis:

```
(3+4)^3
(4+1+9+3)^3
(1+9+6+8+3)^3
```

En caso de no utilizarlos, se evalúan utilizando el orden de precedencia habitual: paréntesis, exponenciación, multiplicación/división, suma/resta (de mayor a menor precedencia).

```
2^1+6^2+4^3+6^4+7^5+9^6+8^7
```

EJERCICIO 2: Evalúe la siguiente expresión en MATLAB e incluya el resultado en el fichero `resultados.m`:

$$\sqrt[8]{5}\,\sqrt[3]{35}$$

MATLAB incluye multitud de funcionaes predefinidas, que podemos utilizar para realizar cálculos de tipo científico: funciones trigonométricas (`sin, cos, tan, asin, acos, atan`), logaritmos (`log, log10, exp`), funciones de tipo estadístico (`min, max, mean, median, mode, var, std, cov, corrcoef`)...

EJERCICIO 3: Evalúe la siguiente expresión en MATLAB para estimar el valor de la constante π e incluya el resultado en el fichero `resultados.m`:

$$\pi \approx \frac{\ln(640320^3 + 744)}{\sqrt{163}}$$

Operaciones lógicas

Operación lógica	Operador
Y (and)	`a&b`
O (or)	`a|b`
NO (not)	`~a`
XOR	`xor(a,b)`

```
1 & 0
1 | 0
~ 1
xor(1,0)
```

Operaciones de comparación

Operación de comparación	Operador
Igualdad	`a == b`
Desigualdad	`a ~= b`
Menor que	`a < b`
Mayor que	`a > b`
Menor o igual	`a <= b`
Mayor o igual	`a >= b`

```
1 == 2      % false
1 ~= 2      % true (¡ojo! no se usa "!=" como en C)
```

Uso de variables

- Los identificadores de variables en MATLAB comienzan siempre con una letra, mayúscula o minúscula.
- Los identificadores pueden incluir letras, dígitos o símbolos de subrayado (_), pero no espacios ni otros símbolos.
- Se distinguen mayúsculas de minúsculas: la variable `total` es distinta a `Total` y ambas son diferentes a `ToTaL`.

Existen algunas variables predefinidas, como las utilizadas para algunas constantes matemáticas (`pi` para la constante π o `j` para la representación de la parte imaginaria de los números complejos, según la convención utilizada por físicos e ingenieros) y aquéllas que se emplean para representar situaciones especiales que pueden producirse al realizar operaciones aritméticas en coma flotante (`Inf` [infinity] y `NaN` [not a number], el resultado de una operación como 0/0).

Asignación de un valor a una variable:

```
>> e = 2.71828182845904

e =

  2.7183
```

En ocasiones, no querremos que MATLAB nos muestre continuamente los resultados de todas las operaciones que vayamos efectuando, para lo que añadiremos un punto y coma al final de la sentencia de asignación:

```
>> e = 2.71828182845904;
>>
```

Acceso al valor de una variable ya creada:

```
>> avogadro

avogadro =

  6.0221e+023
```

Borrar el valor de una variable:

```
>> clear avogadro
```

Borrar el valor de todas las variables definidas:

```
>> clear all
```

O, simplemente,

```
>> clear
```

Obviamente, no siempre trabajaremos con variables de tipo real o entero, p.ej.

```
cadena = 'hi';
booleano = 3>=1;
```

Acceso a las variables definidas en una sesión de trabajo

Podemos consultar las variables que tenemos definidas en nuestro entorno de trabajo utilizando el comando `whos` (o la ventana "Workspace"), que también nos muestra información sobre el tipo de cada variable.

```
>> whos

  Name          Size              Bytes  Class     Attributes
  avogadro      1x1                   8  double
```

EJERCICIO: Compruebe el funcionamiento de `whos` para variables de los distintos tipos vistos hasta el momento.

Para visualizar el valor de una variable, no tenemos más que teclear la cadena correspondiente a su identificador (o no terminar con punto y coma la sentencia de asignación en la que le demos un valor)

```
x = pi
```

También podemos utilizar cadenas de formato similares a las utilizadas en C:

```
disp(sprintf('con 2 decimales: %0.2f', x))
disp(sprintf('con 6 decimales: %0.6f', x))
```

La función `disp` le resultará muy útil cuando desee depurar sus implementaciones en MATLAB para visualizar mensajes informativos en pantalla:

```
disp('Llegó hasta aquí')
disp(['El valor de x es ', num2str(x)])
```

Si queremos, también podemos cambiar el formato estándar en el que se nos muestran los resultados de tipo numérico. Si escribimos

```
format long
pi
```

veremos el valor de π con 14 decimales, mientras que si elegimos

```
format short
pi
```

volveremos a su presentación estándar con 5 dígitos (4 decimales). Utilice la ayuda de MATLAB para consultar otras opciones disponibles tecleando `help format`.

Uso de matrices en MATLAB

En MATLAB, las matrices se definen especificando su contenido entre corchetes. Cada fila de la matriz se define mediante secuencias de valores separados por espacios o comas, con puntos y coma para separar las distintas filas de la matriz:

```
>> r = [1 2 3]

r =
     1     2     3

>> c = [1; 2; 3]

c =
     1
     2
     3

>> m = [1 2 3; 4 5 6]

m =
     1     2     3
     4     5     6

>> whos

  Name          Size                Bytes  Class     Attributes
  r             1x3                    24  double
  c             3x1                    24  double
  m             2x3                    48  double
```

NOTA: En MATLAB, un valor numérico simple se interpreta como una matriz de tamaño 1x1.

La función `size` nos devuelve el tamaño de cada una de las dimensiones de una matriz:

```
>> size(m)

ans =
     2     3
```

`size(m,1)` nos da el número de filas, mientras que `size(m,2)` nos indica el número de columnas. Por su parte, la función `length` nos devuelve la longitud de un vector y equivale a `max(size(m))` cuando trabajamos con matrices.

Para acceder individualmente a elementos concretos de una matriz, podemos utilizar paréntesis e indicar la posición concreta que nos interese (comenzando a contar desde uno):

```
>> m(1,2)

ans =
     2

>> m(2,1)

ans =
     4

>> m(2,2) = m(1,2) + m(2,1)

m =
     1     2     3
     4     6     6
```

También podemos seleccionar fragmentos completos de la matriz utilizando vectores de índices:

```
>> m([1,2],3)

ans =
     3
     6

>> m(1,[1,3])

ans =
     1     3

>> m([1,2],[1,3])

ans =
     1     3
     4     6
```

Otra forma más cómoda de seleccionar secciones completas de una matriz consiste en la utilización de, utilizando la sintaxis `inicio:fin`

```
>> m(1,1:3)

ans =
     1     2     3
```

que, en este caso, equivale a

```
>> m(1,:)

ans =
     1     2     3
```

Esto es, no hace falta que especifiquemos los límites del rango cuando queramos quedarnos con filas o columnas completas de la matriz.

MATLAB también permite definir rangos más complejos utilizando la sintaxis `inicio:salto:fin`, lo que nos facilita la selección de las columnas pares de una matriz:

```
>> f = [1:10]

f =
     1     2     3     4     5     6     7     8     9    10

>> x = [f;f;f;f]

x =
     1     2     3     4     5     6     7     8     9    10
     1     2     3     4     5     6     7     8     9    10
     1     2     3     4     5     6     7     8     9    10
     1     2     3     4     5     6     7     8     9    10

>> y = x(:, 2:2:10)

y =
     2     4     6     8    10
     2     4     6     8    10
     2     4     6     8    10
     2     4     6     8    10
```

Este tipo de construcción nos será útil para definir los ejes de una figura, p.ej. `v = [1:0.1:2]` define un vector de 1 a 2, con saltos de 0.1.

MATLAB incluye también algunas funciones que nos facilitan crear matrices comunes:

```
>> zeros(1,3)

ans =
     0     0     0

>> ones(3,2)

ans =
     1     1
     1     1
     1     1

>> C=2*ones(2,3)

C =
     2     2     2
     2     2     2

>> eye(3,3)                    % Matriz identidad 3x3

ans =
     1     0     0
     0     1     0
     0     0     1

>> r = rand(1,3)               % usando una distribución uniforme

r =
    0.8147    0.9058    0.1270

>> r = randn(1,3)              % usando una distribución normal

r =
    0.8622    0.3188   -1.3077

>> w = 3 + sqrt(10)*(randn(1,5))  % con media 3 y varianza 10

w =
    3.4571    3.0856    2.9920   12.5101    6.3101
```

Como siempre, consulte la ayuda siempre que tenga alguna duda sobre las opciones de una función determinada, p.ej. `help rand` o `help randn`.

Operaciones con matrices

Operación	Operador	Ejemplo
Suma	+	A+B
Resta	-	A-B
Multiplicación	*	A*B
Multiplicación escalar	*	s*A
Multiplicación elemento a elemento	.*	A.*B
División elemento a elemento	./	A./B
Exponenciación elemento a elemento	.^	A.^B
Transposición (intercambio de filas por columnas)	'	A'

```
>> A=[1 2 3]

A =
     1     2     3

>> B=[1;2;3]

B =
     1
     2
     3

>> C=[1 2;3 4; 5 6]

C =
     1     2
     3     4
     5     6

>> C'

ans =

     1     3     5
     2     4     6
```

```
>> A*B

ans =
    14

>> A*C

ans =
    22    28


>> A*B'
??? Error using ==> mtimes
Inner matrix dimensions must agree.

>> A.*B
??? Error using ==> times
Matrix dimensions must agree.

>> A.*B'

ans =
     1     4     9

>> A.^2

ans =
     1     4     9

>> A==B'

ans =
     1     1     1

>> -A                  % Equivale a (-1) * A

ans =
    -1    -2    -3

>> 1./A

ans =
    1.0000    0.5000    0.3333
```

```
>> A + ones(1,length(A))

ans =
     2     3     4

>> A + 1              % Una forma más sencilla de hacer lo mismo

ans =
     2     3     4
```

Como no cabría esperar otra cosa, MATLAB incluye una amplia biblioteca de funciones para realizar otras operaciones con matrices: matriz inversa `inv(A)`, pseudoinversa `pinv(A)`, determinante `det(A)`, valores propios `eig(A)` [*eigenvalues*], descomposición en valores singulares `svd(A)` [*singular value decomposition*], factorización LU `lu(A)`...

Por ejemplo, podemos calcular la inversa de una matriz cuadrada:

```
>> A = magic(3)

A =
     8     1     6
     3     5     7
     4     9     2

>> inv(A)

ans =
    0.1472   -0.1444    0.0639
   -0.0611    0.0222    0.1056
   -0.0194    0.1889   -0.1028
```

O su pseudo-inversa, que en ocasiones resulta útil desde el punto de vista numérico (http://en.wikipedia.org/wiki/Moore%E2%80%93Penrose_pseudoinverse):

```
>> pinv(A)        % inv(A'*A)*A'

ans =
    0.1472   -0.1444    0.0639
   -0.0611    0.0222    0.1056
   -0.0194    0.1889   -0.1028
```

Muchas de las funciones predefinidas de MATLAB están diseñadas para operar elemento a elemento sobre vectores y matrices:

```
>> log(A)

ans =
         0    0.6931    1.0986

>> exp(A)

ans =
    2.7183    7.3891   20.0855

>> abs(log(A)-1)

ans =
    1.0000    0.3069    0.0986
```

Otras funciones útiles sobre matrices:

```
% Máximos con max  (o mínimos con min)

v = [1 15 2 0.5]
val = max(v)
[val,ind] = max(v)

max(rand(3),rand(3))

max(A,[],1)
min(A,[],2)

% Encontrar valores que cumplan una condición

v < 3
find(v < 3)

A = magic(3)
[r,c] = find(A>=7)
```

Otras funciones que pueden resultarle útiles al trabajar con vectores y matrices:

```
sum(v)

diff(v)

prod(v)

dot(v,w)

cross(v,w)

floor(v)

ceil(v)
```

Consulte en la ayuda de MATLAB para qué sirve cada una de las funciones anteriores.

EJERCICIO 4: ¿Cuál es el valor que devuelven las siguientes expresiones sobre la matriz A, que hemos definido como un cuadrado mágico de tamaño 9x9 utilizando la función `magic(9)`? Indique su respuesta en el fichero `resultados.m`.

```
sum(A,1)
sum(A,2)
sum(sum( A .* eye(9) ))
sum(sum( A .* flipud(eye(9)) ))
```

EJERCICIO 5: Dada una matriz cualquiera A de tamaño 3x3, indique en el fichero `resultados.m` cómo obtener los siguientes valores:

- La segunda fila de la matriz.
- La segunda columna de la matriz.
- La sección de la matriz formada por las filas 1 y 3.

Manejo de datos en MATLAB

Cuando trabajemos con MATLAB, lo primero que haremos será irnos al directorio donde estén los ficheros del proyecto con el que queramos trabajar.

```
% print working directory (muestra el directorio actual)

pwd

% change directory (cambia al directorio del proyecto)

cd 'C:\MATLAB\proyecto'

% list (muestra los ficheros del directorio actual)

ls
```

Habitualmente, trabajaremos con datos en forma de matriz, de forma que podremos hacer cosas como:

```
A(:,2) = [10; 11; 12]
```

que cambia la segunda columna de la matrix A, que debe ser de tamaño 3xn.

También podemos añadir columnas a una matriz ya existente poniendo algo como:

```
A = [A, [100; 101; 102]];
```

O bien:

```
A = [A [100; 101; 102]];
```

Incluso podemos combinar varias matrices si sus dimensiones concuerdan, p.ej.

```
B = [11 12; 13 14; 15 16]

C = [A B]
C = [A; B']
```

Cuando queramos convertir una matriz en un vector (de tipo columna), basta con poner lo siguiente:

```
A(:)
```

En ocasiones, los datos nos los darán en un fichero, que podremos leer fácilmente mediante el comando `load`:

```
load datos.mat
load otros.mat
```

Dichos ficheros los podemos generar fácilmente a partir de las matrices con las que estemos trabajando si utilizamos el comando `save`. Por ejemplo:

```
save datos A;
```

almacena la matriz `A` en el fichero `datos.mat`.

El formato de los ficheros `.mat` es específico de MATLAB por lo que, cuando queramos acceder a dichos datos desde otros programas, resulta convenientes almacenarlos en formato ASCII:

```
save datos.txt matriz -ascii;
```

EJERCICIO: Compruebe el funcionamiento de los comandos `load` y `save` con sus propios datos, usando tanto el formato propio de MATLAB como el formato ASCII. Para ello, cree una matriz cualquiera, guárdela en un fichero con `save`, elimínela de la memoria de trabajo con `clear` y vuelva a cargarla en memoria utilizando `load`.

> Observe el identificador de la matriz que se obtiene como resultado al cargar los datos de un fichero ASCII y la diferencia con respecto al uso de ficheros .MAT (en los que, si lo desea, puede almacenar el valor de varias variables).

NOTA FINAL: MATLAB permite métodos más sofisticados para trabajar con ficheros, al estilo de C (`fopen`, `fread`, `fprintf` y `fscanf` funcionan) o Java, pero su uso no será necesario para la realización de estas prácticas.

Visualización de datos en MATLAB

Histogramas

Usando la función de generación de números pseudoaleatorios `randn`, podemos crear un conjunto de datos sintético que siga una distribución normal con media 180 y desviación 10:

```
alturas = 180 + sqrt(100)*(randn(1,10000));
```

A continuación, podemos visualizar el histograma correspondiente a nuestras 10000 muestras escribiendo:

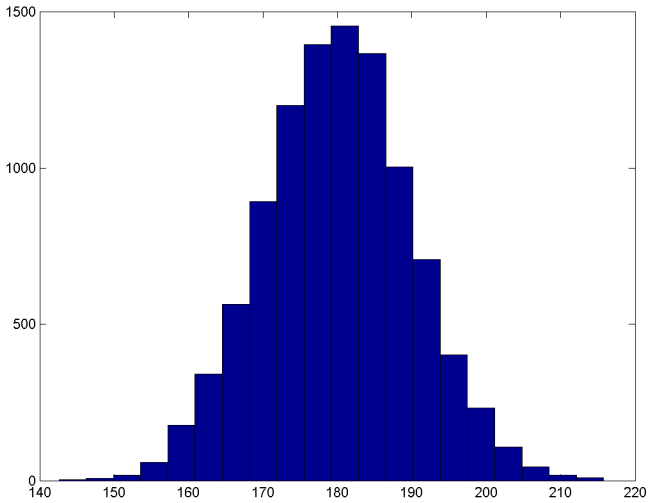
```
hist(alturas);
```

Para que se pueda apreciar mejor la distribución de nuestros datos, podemos modificar el número de barras del histograma [*bins*] usando un segundo parámetro opcional de la función `hist`:

```
hist(alturas,20);
```

Si queremos guardar la imagen del histograma, podemos utilizar el comando `print`, que admite distintos formatos de imagen, p.ej.:

```
print -dpng 'histograma.png'
```



Observe lo que sucede con el histograma si aumentamos su número de barras, p.ej. usando 50 ó 100.

Diagramas 2D

A continuación, generaremos diagramas utilizando la función `plot`:

```
t = [0:0.01:1];
y1 = sin(2*pi*4*t);
plot(t,y1);
```

Si en la misma figura queremos visualizar varias series de datos, tenemos que usar el comando `hold on` (`hold off` para desactivar esta opción):

```
hold on;
y2 = cos(2*pi*4*t);
plot(t,y2,'r');
```

Fíjese cómo hemos seleccionado el color de la nueva serie mediante el tercer parámetro del comando `plot` (consulte la ayuda con `help plot` para descubrir otras opciones disponibles).

Por último, añadimos un título, etiquetamos los ejes y añadimos una leyenda para facilitar la interpretación de la figura:

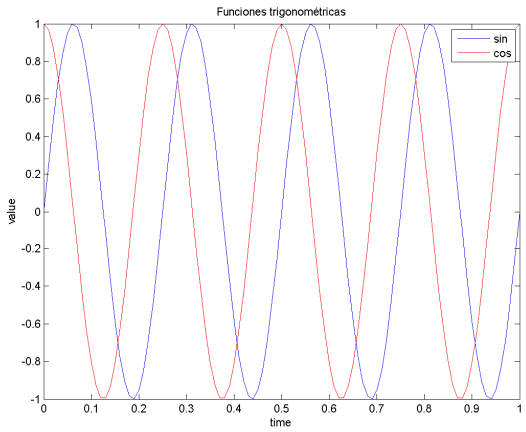
```
title('Funciones trigonométricas');
xlabel('tiempo');
ylabel('valor');
legend('sin','cos');
```

Cuando la figura quede a nuestro gusto, podemos exportarla en formato PNG, JPG, TIFF, EPS o WMF, para poder incluirla donde queramos:

```
print -dpng 'trig.png'
```

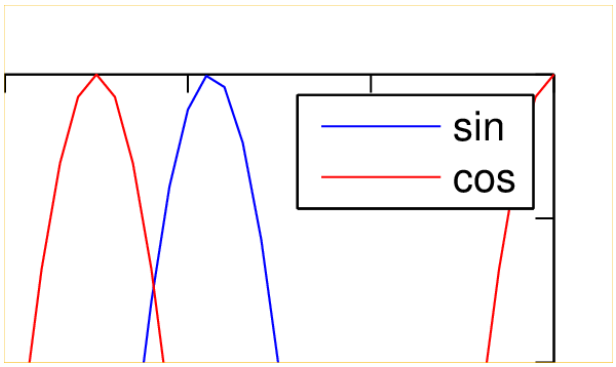
o bien

```
print -depsc2 'trig.eps'
```

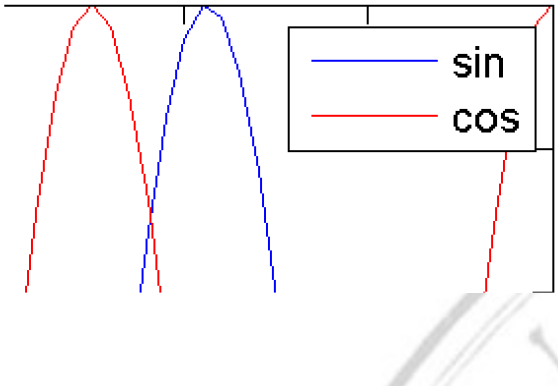


Se recomienda utilizar un formato vectorial como EPS en caso de que queramos incluir las imágenes en un documento, p.ej., elaborado con LaTeX.

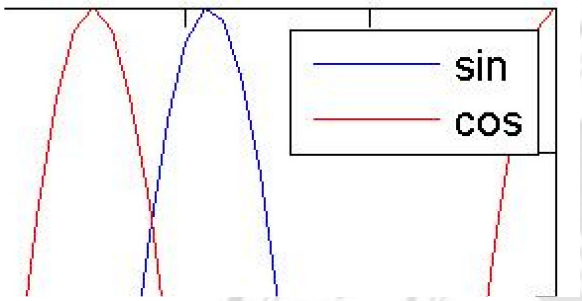
Si sólo queremos incluir la figura en una página web o mandarla por correo electrónico, el formato de compresión sin pérdidas PNG resulta adecuado. Por el contrario, el formato JPEG, que utiliza técnicas de compresión con pérdidas, puede distorsionar la imagen correspondiente a la figura creada desde MATLAB.



Detalle de la imagen EPS



Detalle de la imagen PNG



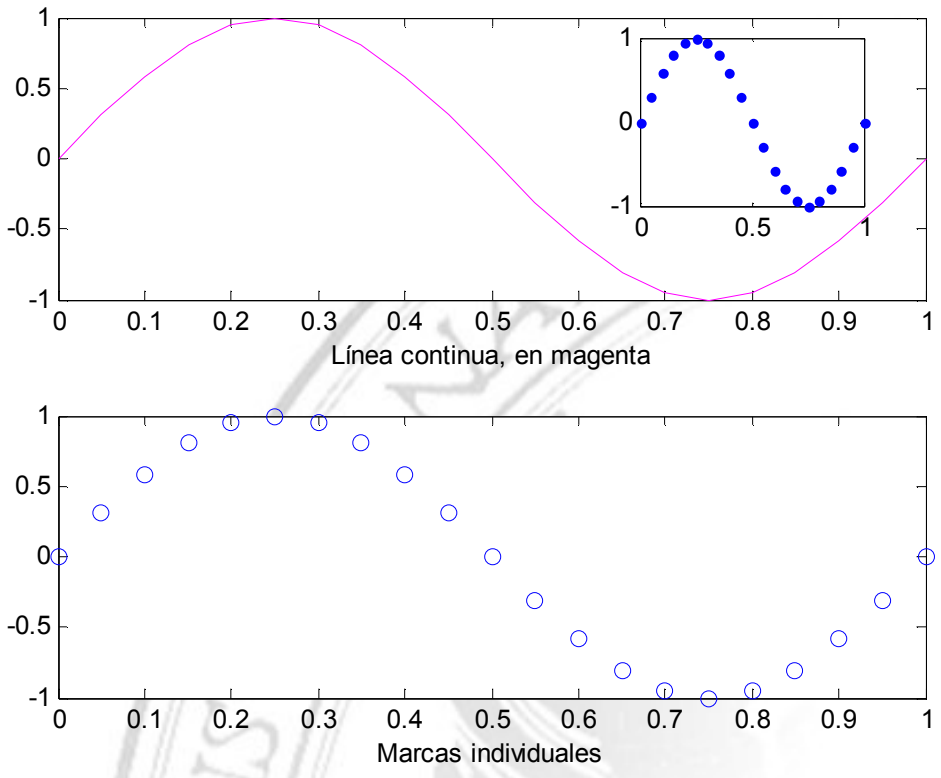
Detalle de la imagen JPEG

Otros comandos útiles a la hora de trabajar con imágenes son:

- `close` para cerrar las figuras que estén abiertas.
- `clf` para borrar la figura actual (sin cerrar su ventana).
- `figure` para crear una nueva figura, sin perder las anteriores.
- `figure(n)` para cambiar la figura actual cuando tengamos varias abiertas.
- `axis([xmin xmax ymin ymax])` para cambiar la escala de los ejes.

También pueden resultar útiles comandos como `axes` (que permite crear figuras en miniatura dentro de otras) o `subplot` (que permite apilar figuras).

EJERCICIO: Utilizando los distintos comandos descritos en esta sección, incluya en el fichero `figura.m` todos los comandos necesarios para crear una figura como la siguiente y guárdela en el fichero `figura.png`.

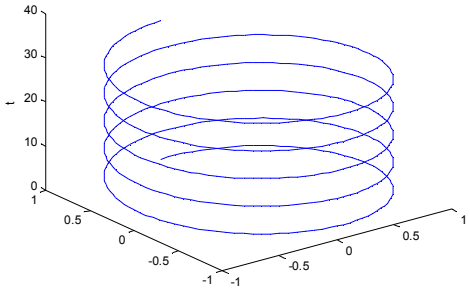


NOTA: Para crear este tipo de figuras apiladas, deberá utilizar `subplot(2,1,X)`.

Diagramas 3D

`plot3` es el equivalente a `plot` en 3 dimensiones:

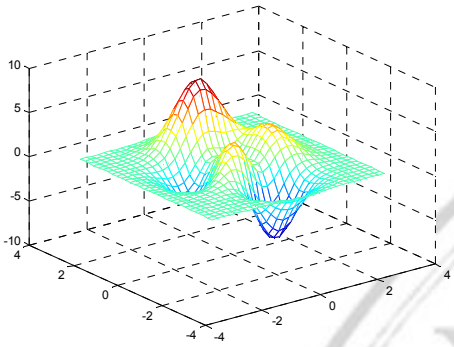
```
t = 0:pi/50:10*pi;
plot3(sin(t),cos(t),t);
zlabel('t');
```



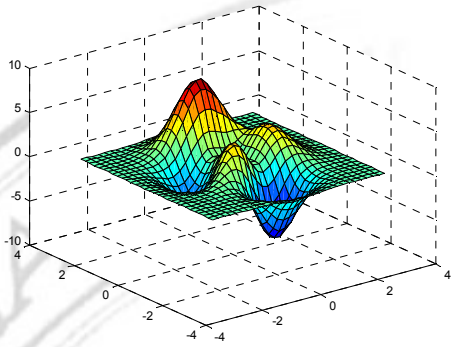
No obstante, este tipo de diagramas se suele utilizar más a menudo para visualizar superficies en vez de trayectorias, para lo que recurriremos a las funciones `mesh` (malla) o `surf` (superficie):

```
[x y z] = peaks(30);
```

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

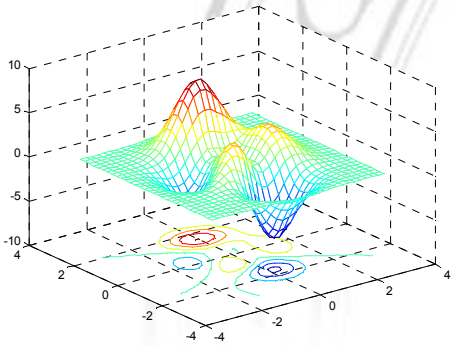


```
surf(x,y,z);
```

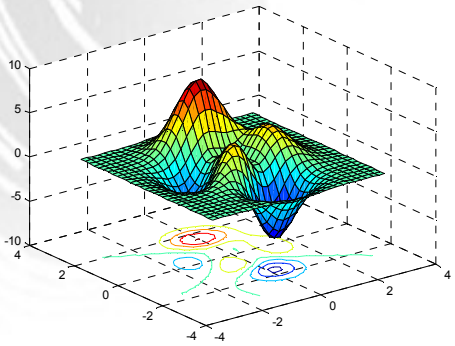


Que también tienen variantes para visualizar contornos (como los mapas de isobaras):

```
meshc(x,y,z)
```

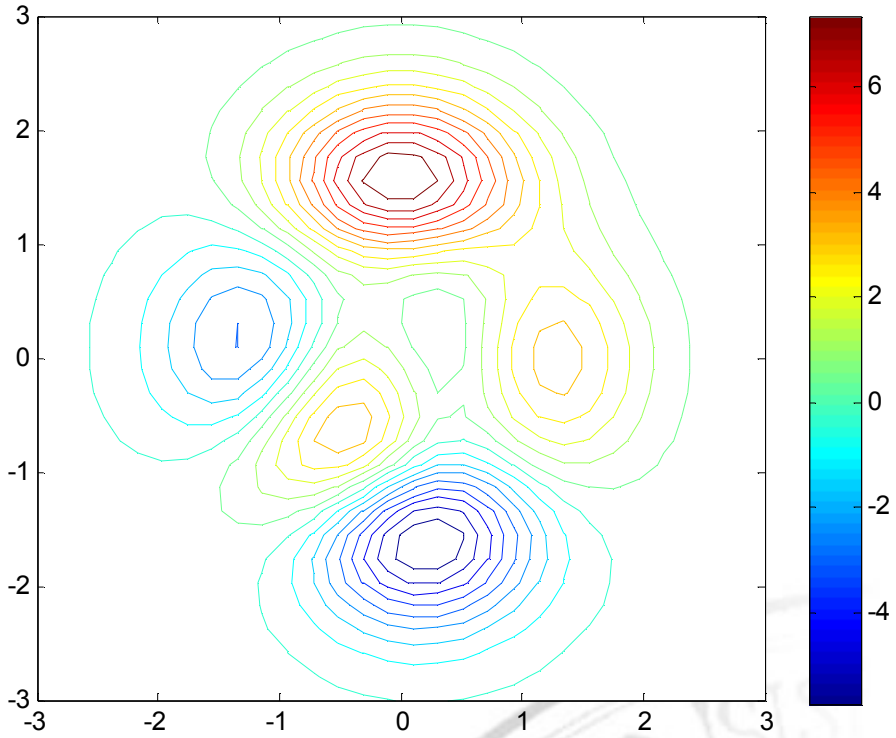


```
surfc(x,y,z);
```



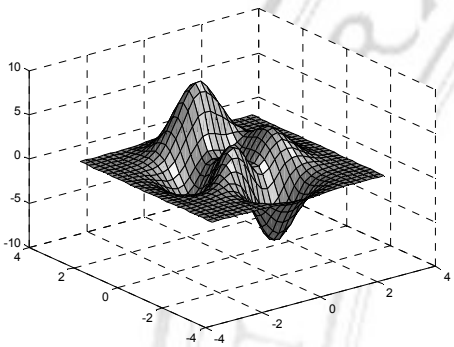
Si lo único que nos interesa es una proyección en 2D de la superficie, podemos utilizar directamente la función `contour`, a la que en el siguiente ejemplo añadimos un indicador de escala con `colorbar`:

```
contour(x,y,z,20);
colorbar;
```

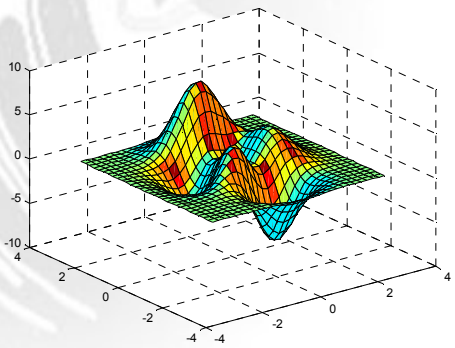


Podemos incluso visualizar diagramas en los que el color no indique valores individuales, sino que se utilice para simular diferentes condiciones de iluminación:

```
colormap gray;
surfl(x,y,z);
```



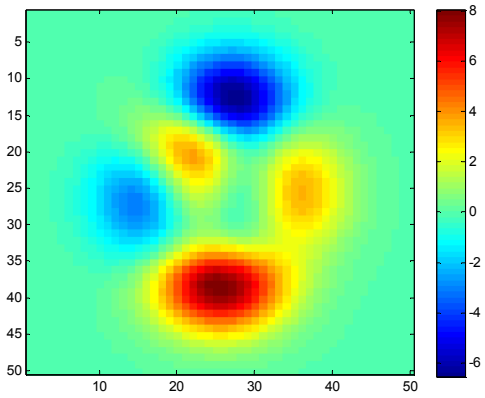
```
colormap jet;
surfl(x,y,z);
```



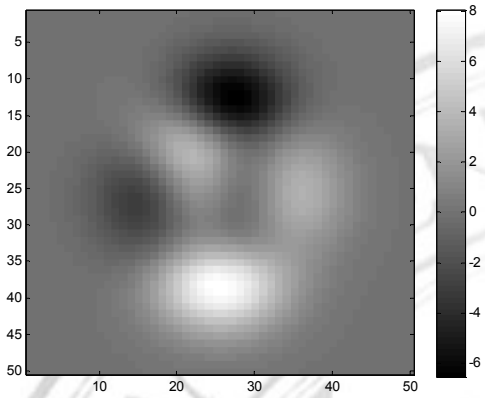
Matrices e imágenes

Cuando tengamos una matriz bidimensional y queremos verla en forma de imagen, podemos utilizar la función `imagesc`:

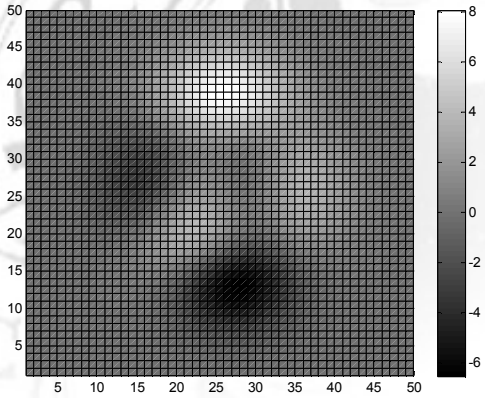
```
[x y z] = peaks(50);
imagesc(z);
colorbar;
```



En blanco y negro con `colormap gray`:

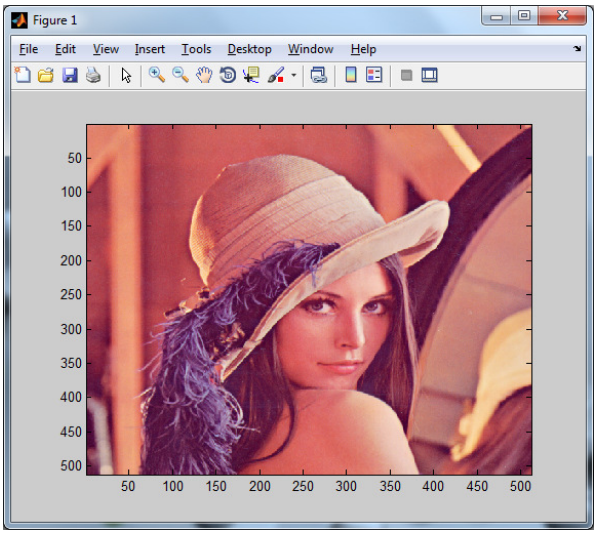


Como alternativa, también podemos usar `pcolor`:



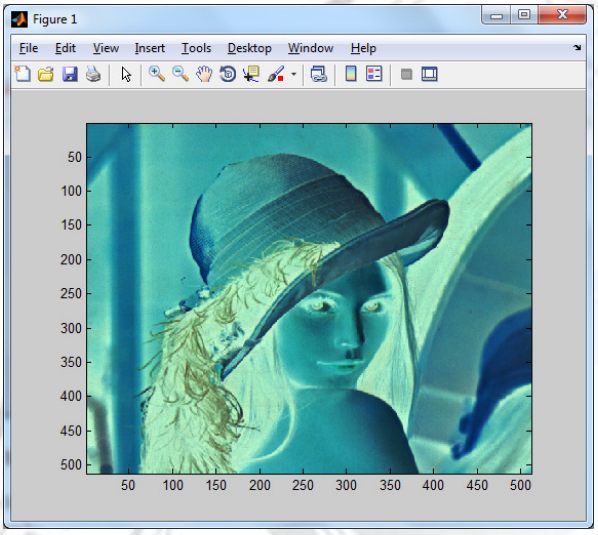
Cuando los datos de la matriz los queramos leer de una imagen, podemos hacerlo directamente utilizando la función `imread`:

```
img = imread('lena.png');
image(img);
```



Procesar la imagen en MATLAB es tan sencillo como hacer:

```
negativo = 256-img;
image(negativo);
```



Y, por supuesto, podemos guardar la imagen cuando queramos con `imwrite`:

```
imwrite(negativo,'negativo.png');
```

UNA CURIOSIDAD:
La historia detrás de esta imagen es un tanto peculiar y puede encontrarla en Internet…

Programación en MATLAB

MATLAB incluye un lenguaje de programación propio, algunos de cuyos detalles se mencionan a continuación:

Estructuras de control

Como cualquier otro lenguaje de programación, MATLAB nos permite utilizar estructuras condicionales de control, como la sentencia `if`:

```
if v==1,
  disp('El valor de la variable es uno.');
elseif v(1)==2,
  disp('El valor de la variable es dos. ');
else
  disp('El valor ni es uno ni es dos :-(');
end
```

También podemos utilizar la sentencia `switch`, similar a la de C, cuando tengamos que diferenciar distintos casos (siempre y cuando la expresión del `switch` corresponda a un valor de tipo escalar, p.ej. entero, o cadena):

```
switch lower(method)
   case {'linear','bilinear'}
     disp('Method is linear')
   case 'cubic'
     disp('Method is cubic')
   case 'nearest'
     disp('Method is nearest')
   otherwise
     disp('Unknown method.')
end
```

NOTA: A diferencia de C, no hay que terminar cada caso con un `break`.

En cuanto a las estructuras iterativas de control, disponemos de bucles `for` y `while`:

```
% Bucle for

for i=1:10,
    v(i) = 2^i;
end

% Bucle while

i = 1;
while i <= 5,
  v(i) = 100;
  i = i+1;
end
```

EJERCICIO: Prepare una macro o *script* en MATLAB que haga lo siguiente:

1. Haciendo uso de rangos, en una única sentencia de asignación, rellene un vector llamado `factores` con los valores de 1 al 7.

2. Utilizando un bucle `for`, calcule el resultado de multiplicar cada uno de los valores del vector `factores` por 142,857 (aunque se pueda hacer, no utilice en este apartado las operaciones aritméticas que MATLAB permite realizar sobre matrices). El resultado de las multiplicaciones deberá almacenarse en el vector `productos`.

3. Utilizando un bucle `while` y accediendo a los valores de los vectores `factores` y `productos`, muestre la siguiente tabla de multiplicar por pantalla:

$1 \times 142{,}857 = 142{,}857$
$2 \times 142{,}857 = 285{,}714$
$3 \times 142{,}857 = 428{,}571$
$4 \times 142{,}857 = 571{,}428$
$5 \times 142{,}857 = 714{,}285$
$6 \times 142{,}857 = 857{,}142$
$7 \times 142{,}857 = 999{,}999$

http://en.wikipedia.org/wiki/142857

Almacene el resultado de realizar este ejercicio en el fichero `macro.m` y no olvide entregarlo junto con el resto de la práctica.

Funciones

Normalmente, en vez de utilizar macros o *scripts*, encapsularemos nuestro código en funciones.

Cuando queramos crear una función llamada `f`, incluiremos nuestra implementación de la función `f` en un fichero llamado `f.m`. Cuando dicho fichero esté en el `path` de MATLAB (por ejemplo, en el directorio en el que nos encontremos, `pwd`), bastará con utilizar el nombre de la función para invocarla, pasándole como argumentos los valores que queramos darles a sus parámetros.

Por ejemplo, la siguiente función, que deberemos guardar en un fichero llamado `cuadrado.m`, calcular el cuadrado de un número:

```
function c = cuadrado(x)
  c = x * x;
end
```

Cuando queramos calcular el cuadrado de un número, no tenemos más que escribir

```
cuadrado(11)
```

Si MATLAB le da un mensaje de error (*Undefined function or variable 'cuadrado'*), compruebe su directorio de trabajo usando `pwd` y cámbielo si es necesario con `cd`. Si lo desea, también puede modificar el `path` con la función `addpath`, de forma que podrá utilizar sus funciones independientemente del directorio de trabajo en el que se encuentre:

```
addpath('C:/proyectos/iaio');
```

Cuando queramos que una función devuelva varios valores, podemos hacerlo fácilmente utilizando la siguiente sintaxis:

```
function [m M] = minmax (datos)

m = min(datos);
M = max(datos);
```

Al utilizar la función, si queremos obtener todos sus resultados, escribiremos algo similar a lo siguiente:

```
datos = rand(1,1000);
[vmin, vmax] = minmax(datos)
```

EJERCICIO:
Implemente en MATLAB una función que calcule la cuota mensual de una hipoteca.

La función debe tener la siguiente cabecera:

```
function [cuota capital intereses] = hipoteca (importe, tipo, años)
```

donde cantidad corresponde al capital inicial del préstamo, años al período de amortización de la hipotca y tipo es el tipo de interés anual al que se ha suscrito.

1. Calcule la cuota mensual que deberá pagar de la hipoteca. La fórmula necesaria para calcular la cuota mensual de una hipoteca viene dada por:

$$\delta = interésMensual = interésAnual / 12$$

$$cuotaMensual = \frac{importe \times \delta}{1 - \frac{1}{(1+\delta)^{años \times 12}}}$$

2. Redondee correctamente el resultado en céntimos de euro. Para realizar el redondeo en céntimos puede utilizar el siguiente algoritmo:

Paso	Descripción	Resultado
`100*cuota`	Cuota en céntimos de euro	`double`
`round(…)`	Redondeo al entero más cercano	`int`
`… / 100.0`	Cuota en euros (con céntimos)	`double`

3. A continuación, rellene los vectores `capital` e `intereses` con el capital amortizado cada mes y los intereses devengados en dicho mes (el porcentaje correspondiente al capital pendiente de amortizar cuando se satisface la cuota correspondiente del préstamo).

NOTA: No olvide redondear sus resultados al céntimo.

Implemente su función `hipoteca` en un fichero llamado `hipoteca.m`. Compruebe los resultados que se obtienen con esta función para distintos valores de sus parámetros y visualice gráficamente cómo evoluciona la cantidad mensual destinada a pagar intereses con respecto al capital amortizado cada mes.

La batería de pruebas `test.m` comprueba el correcto funcionamiento de esta función para algunos casos particulares. Asegúrese de completar satisfactoriamente los casos de prueba antes de entregar el fichero `hipoteca.m` con el resto de su práctica.

Referencias

Machine Learning Course
Andrew T. Ng, Stanford University
http://www.ml-class.org/

Using MATLAB to Visualize Scientific Data
Ray Gasser, Boston University
http://www.bu.edu/tech/research/training/tutorials/visualization-with-matlab/

Making Pretty Graphs
Jiro Doke, The MathWorks
http://blogs.mathworks.com/loren/2007/12/11/making-pretty-graphs/

EVALUACIÓN DE LAS PRÁCTICAS

Esta práctica inicial no es evaluable, si bien se recomienda que se entreguen los ficheros indicados a través del acceso identificado de DECSAI (`https://decsai.ugr.es/`) para familiarizarse con el sistema de entrega de prácticas utilizado a lo largo del curso.

Para comprobar el funcionamiento de su implementación de la práctica, ejecute `test`, que lanza una batería automatizada de casos de prueba sobre las funciones que haya definido.

Tras remitir su fichero de resultados, se realizará una evaluación automática de los valores indicados en su fichero de resultados de la práctica, `resultados.m`, tal como usted mismo puede comprobar al ejecutar `master` (sólo en esta práctica no evaluable).