PRÁCTICA 1. INTRODUCCIÓN A MATLAB

MATLAB es el nombre abreviado de "MATrix LABoratory". MATLAB es un programa para realizar cálculos numéricos con ***vectores*** y ***matrices***. Como caso particular puede también trabajar con números escalares, tanto reales como complejos. Una de las capacidades más atractivas es la de realizar una amplia variedad de ***gráficos*** en dos y tres dimensiones.

Variables:

1. NO pueden comenzar con un número, aunque si pueden tener números (variable1 es un nombre válido).
2. Las mayúsculas y minúsculas se diferencian en los nombres de variables. (A y a son dos variables diferentes)
3. Los nombres de variables no pueden contener operadores ni puntos. (No es válido usar /, *, -, +, ...)
4. No es necesario definir el tipo de variable o tamaño (si se usa un vector y después se expande, no hay problema)

1. Comenzando a Trabajar

Dado que MATLAB es un lenguaje *interpretado* comenzaremos hablando de expresiones. Supongamos que nos interesa obtener el valor de multiplicar 2 por 3; tenemos dos formas de hacerlo según el uso que queramos darle al resultado de la multiplicación. La primera:

```
>> 2*3
```

(las instrucciones terminan con el retorno de carro)
entonces obtendremos la respuesta

```
ans = 6
```

asignando el resultado de la expresión (6) a una variable por defecto llamada **ans** (por answer), que contiene el resultado de la última operación. La variable ***ans*** puede ser utilizada como operando en la siguiente expresión que se introduzca.

Evalúa:

```
>>  ans*2
```

Si lo que queremos es almacenar el resultado de la multiplicación en una variable:

```
>> a=2*3
```

y entonces MATLAB asignará a la variable **a** el valor 6.

```
>> a
```

Para utilizar cadenas de texto, con la comilla simple:

```
>> Cadena1= 'Inteligencia Artificial'
```

El uso de la comilla doble

```
 >> cadena2="IA"
```

produce:

??? cadena2="IA"
Error: The input character is not valid in MATLAB statements or expressions.

Para MATLAB el ***infinito*** se representa como ***inf*** ó ***Inf***.

```
» 1.0/0.0
Warning: Divide by zero
ans = Inf
```

MATLAB tiene también una representación especial para los resultados que no están definidos como números:

```
» 0/0
Warning: Divide by zero
ans = NaN
» inf/inf
ans =   NaN
```

En ambos casos la respuesta es ***NaN***, que es la abreviatura de ***Not a Number***.

Órdenes del Intérprete

Orden	**Descripción**
clc	Limpia la consola
clear	Borra las variables
close	Cierra las ventanas de gráficos
who / whos	Muestra las variables
help	Muestra la ayuda
quit	Cierra Matlab

2. VECTORES Y MATRICES

Las matrices y vectores son ***variables*** que tienen ***nombres***. Para definir una matriz *no hace falta establecer de antemano su tamaño* (de hecho, se puede definir un tamaño y cambiarlo posteriormente). MATLAB determina el número de filas y de columnas en función del número de elementos que se proporcionan (o se utilizan).
Las matrices se definen por filas; los elementos de una misma fila están separados por ***blancos*** o ***comas***, mientras que las filas están separadas por ***intro*** o por caracteres ***punto y coma*** (;).
Define una matriz **A** de dimensión (3x3):

```
» A=[1 2 3; 4 5 6; 7 8 9]
» B=[]  %% matriz vacía
```

A partir de este momento las matrices **A y B** están disponibles para hacer cualquier tipo de operación con ella. Por ejemplo, una sencilla operación con **A** es hallar su ***matriz traspuesta***. En MATLAB el apóstrofo (') es el símbolo de *trasposición matricial*. Para calcular **A'** (traspuesta de **A**) basta teclear lo siguiente:

```
» A'
ans =
1 4 7
2 5 8
3 6 9
```

3. OPERADOR DOS PUNTOS (:)

Este operador es muy importante en MATLAB y puede usarse de varias formas.
Para especificar **rangos de valores** el formato básico sería:

ValorMínimo:Incremento:ValorMáximo,

Por ejemplo:

```
>> 2:0.5:4
```

denota al vector fila

```
[2 2.5 3 3.5 4]
```

Esta notación de dos puntos es particularmente útil, entre otras cosas, para introducir matrices de grandes dimensiones. Si por ejemplo, se necesita un vector **A** formado por los cien primeros enteros (del 1 al 100), bastará escribir:

****** Si no se especifica el valor del incremento se toma como 1**

```
>> A=1:100;
```

Las siguientes instrucciones generan una "tabla de senos", (**sin** es la función seno).

```
>> x=[0.0:0.1:2*pi];
>> y=sin(x);
```

****

Utiliza **plot** para mostrar gráficamente la función seno(x):
(help plot para estudiar más opciones de esta orden)

```
plot(x,y,'-r') %%% dibuja la función y=seno(x) trazando una línea roja: '-r'
title('Funcion seno(x)')  %%% título de la figura
legend ('y=seno(x)')  %% leyenda
```

4. ACCESO A LOS ELEMENTOS DE UNA MATRIZ

Para acceder a un elemento de la matriz las filas y columnas se indexan de 1 a N filas y de 1 M columnas. Para referenciar una columna o fila completa se utiliza el operador **:** (2 puntos)

EJERCICIO 1

Comprueba que son ciertas las siguientes afirmaciones con la matriz

A=[10 22 34; 43 55 64; 76 86 96; 89 90 91]

A(4,3) : elemento de la 4ª fila y la 3ª columna

A(1,**:**) : todos los elementos de la 1ª fila

A(**:**,2) : todos los elementos de la 2ª columna

A(2:3, :) : todos los elementos de la 2ª a la 3ª fila y todas las columnas

¿Qué instrucción es necesaria para acceder a los elementos de las filas 1 y 2 de las columnas 2 y 3?

5. TRATAMIENTO DE IMÁGENES

El primer comando que se empleará será la lectura de ficheros de imágenes utilizando la función **imread**, por ejemplo:

```
>> A = imread('cameraman.tif');
```

Se puede comprobar que la imagen es de niveles de grises con 256 valores diferentes (de la clase uint8), con un tamaño de 256 x256 píxeles. La visualización de la imagen se realiza mediante **imshow**:

```
>> imshow(A)
```

EJERCICIO 2
Repite estas operaciones con la imagen **mri.tif**. ¿Cuál es su tamaño?¿Cuántos niveles de grises tiene?.Si la imagen fuese en color, normalmente quedará definida por tres matrices correspondiente a los tres colores básicos (rojo, verde y azul). Vuelva a realizar las mismas

operaciones de: a) lectura, b) tamaño y clase de la imagen y c) visualización sobre la imagen de color **board.tif**.

Usando la notación de matrices de Matlab se puede visualizar las tres componentes de color. El nivel de gris para cada parte del espectro de la luz será definido por (:,:,i). Indica que todas las filas y las columnas para la componente i, i=1,2 ó 3 (rojo,verde,azul)

```
>> imshow([A(:,:,1) A(:,:,2) A(:,:,3)])
```

El operador [ ] permitirá construir una matriz de N x (3*M), siendo N el número de filas y M el número de columnas.

Emplea el comando **imtool** para ver el nivel de gris de la imagen de **cameraman.tif** y los colores en **board.tif**.

```
>> imtool('cameraman.tif');

>> imtool('board.tif');
```

Otro tipo interesante de imágenes son las binarias. Normalmente se emplea el '0' para indicar el fondo y '1' el objeto. Se empleará una técnica de umbralización para convertir las imágenes en binarias mediante la función **im2bw**:

```
>> IG = imread('rice.png');

>> figure, imshow(IG);

>> BW = im2bw(IG);

>> figure, imshow(BW);

>> impixelinfo;
```

EJERCICIO 3
Realice la misma operación de binarizado con la imagen coins.png

Se desea construir una imagen binaria de 120 x 200 píxeles que tenga franjas horizontales de 20 píxeles de anchura, distanciada por cada 20 píxeles:

```
>> BW = false([120,200]);

>> for i=1:40:200, BW(i:i+19,:)=true; end

>> imshow(BW)
```

Si se desea que las franjas sean verticales sólo habría que emplear el operador traspuesta de las matrices.

```
>> imshow(BW')
```

EJERCICIO 4
Realizar una función que construya y visualice dos imágenes de 256x256 con variación del nivel de gris en filas y columnas.

6. HISTOGRAMA DE UNA IMAGEN

Con un histograma es posible ver la distribución de intensidades en una imagen basta con utilizar la función **imhist**. Si nos interesa ecualizar el histograma de una imagen (para extender los valores de intensidad) bastaría con utilizar la función **histeq**. El código Matlab para visualizar en una misma pantalla una imagen y su histograma podría ser:

```
>> clear all

>> close all

>> A = imread('lena.jpg');

>> B = rgb2gray(A);  %Pasa la imagen a escala de grises

>> subplot(2,1,1),subimage(B),title('Imagen original');

>> subplot(2,1,2),imhist(B),title('Histograma');
```

EJERCICIO 5
Mostrar en una misma pantalla los siguientes gráficos e imágenes:

1. Imagen pout.tif
2. Histograma de la imagen
3. Ecualización de la imagen
4. Histograma de la imagen ecualizada

7. OPERACIONES DE SUAVIZADO

Estas operaciones tienen por objeto reducir el ruido y/o efectos espurios que pueden presentarse en una imagen a consecuencia del proceso de captura, digitalización y transmisión. Su utilización es normalmente necesaria antes de la aplicación de un detector de bordes. Existen distintos tipos de algoritmos que permiten la reducción del ruido como los filtros lineales (convolución de una imagen con una máscara predefinida) y no lineales (operación no lineal con los píxeles del entorno de vecindad).

Filtros lineales

Este tipo de filtros se realiza una operación de convolución entre la imagen a ser filtrada y una máscara. El principal inconveniente de estas técnicas es el emborronamiento que se produce en la imagen, provocando el difuminado de los bordes, por lo que tienen que ser utilizados con cierta precaución.

El primero de los filtros lineales es el de la media, el cual a partir de una imagen *I* genera una nueva imagen *G* cuya intensidad para cada píxel se obtiene promediando los valores de intensidad de los píxeles *I* incluidos en un entorno de vecindad predefinido. En Matlab la función que permite aplicar un filtro a una imagen es **imfilter.** Existe la función **fspecial** que permite crear un filtro.

Por ejemplo, apliquemos el filtro de la media sobre la imagen cameraman.tif a la que se le ha introducido ruido aleatorio del tipo sal y pimienta, con una máscara de 3X3 y otra de 9X9.

```
>> A = imread('cameraman.tif');

>> R = imnoise(A,'salt & pepper',0.05);

>> h1 = fspecial('average');
```

```
>> h2 = fspecial('average',[9,9]);

>> M1 = imfilter(R, h1);

>> M2 = imfilter(R, h2);

>> subplot(2,2,1),subimage(A),title('Imagen original')

>> subplot(2,2,2),subimage(R),title('Imagen con ruido');

>> subplot(2,2,3),subimage(M1),title('Filtro Media 3x3');

>> subplot(2,2,4), subimage(M2),title('Filtro Media 9x9');
```

En esta imagen puede apreciarse la importancia de definir el tamaño de la máscara. Cuanto mayor sea ésta se consigue una mayor reducción del ruido, pero a cambio se produce una mayor difuminación de los bordes.

Otro de los filtros lineales es el filtro Gaussiano, donde el valor de cada punto es el resultado de promediar con distintos pesos los valores vecinos a ambos lados de dicho punto. Este tipo de filtro también tiene el problema del difuminado de los bordes, pero no es tan acusado como el caso de la media simple. Este tipo de filtro reduce especial mente el ruido tipo gaussiano.

Ruido gaussiano: produce pequeñas variaciones en la imagen. Tiene su origen en diferencias de ganancias del sensor, ruido en la digitalización, etc.

El código de Matlab y las imágenes obtenidas de aplicar este tipo de filtro, comparándolo con uno de la media son:

```
>> A = imread(‘cameraman.tif’);

>> R = imnoise(A,’gaussian');

>> h1 = fspecial('gaussian');

>> h2 = fspecial('average');

>> G = imfilter(R, h1);

>> M = imfilter(R, h2);

>> subplot(2,2,1),subimage(A),title('Imagen original')

>> subplot(2,2,2),subimage(R),title('Imagen con ruido');

>> subplot(2,2,3),subimage(G),title('Filtro Gaussiano');

>> subplot(2,2,4), subimage(M),title('Filtro Media 3x3');
```

Puede apreciarse cómo el filtro gaussiano elimina el ruido mejor y además emborrona menos los bordes.

Filtros no lineales

Dentro de los filtros no lineales podemos encontrar el filtro de la mediana, en el cual los píxeles de la nueva imagen se generan calculando la mediana del conjunto de píxeles del entorno de vecindad del píxel correspondiente a la imagen origen. De esta forma se homogeneizan los píxeles de intensidad muy diferente con respecto a la de los vecinos. Este tipo de filtro es

bastante indicado cuando se tiene ruido aleatorio. En Matlab la instrucción empleada para realizar el filtro de la mediana es **medfilt2**. A continuación se muestra un ejemplo de una imagen afectada por ruido gaussiano y aleatorio y filtrada por el filtro de la media y de la mediana.

```
>> A = imread('lena.jpg');

>> B = rgb2gray(A);

>> G = imnoise(B,'gaussian');

>> S = imnoise(B,'salt & pepper', 0.1);

>> h1 = fspecial('average');

>> media1 = imfilter(G,h1);

>> media2 = imfilter(S,h1);

>> mediana1 = medfilt2(G);

>> mediana2 = medfilt2(S);

>> subplot(2,3,1),subimage(G),title('Con ruido Gaussiano')

>> subplot(2,3,4),subimage(S),title('Con ruido aleatorio');

>> subplot(2,3,2),subimage(media1),title('Filtro media');

>> subplot(2,3,5), subimage(media2),title('Filtro media');

>> subplot(2,3,3),subimage(mediana1),title('Filtro
mediana');

>> subplot(2,3,6), subimage(mediana2),title('Filtro
mediana');
```

EJERCICIOS PROPUESTOS

1. Dibujar la siguiente función definida por partes:

   $x^2 - x + 3$, $-2 <= x < 2$
   $2x + 1$, $2 <= x < 4$

   En la primera parte, x debe variar a intervalos de 0.05 y en la segunda parte, a intervalos de 0.01

2. Dibujar en una sola ventana con dos subventanas las funciones $y = x^2 - 3x - 2$ y $z = x^3 - 2x + 1$ con x variando entre -5 y 5 a intervalos de 0.02. La primera en líneas verdes continuas, la segunda en líneas azules discontinuas. Marcar con '+' rojo el punto (2, 1) en ambas gráficas.

3. Abre la imagen spine.tif, añádele ruido del tipo sal y pimienta y aplícale los filtros aprendidos anteriormente ¿Qué filtro elimina mejor el ruido?

4. Abre la imagen peppers.png, añádele ruido gaussiano y aplícale los filtros aprendidos anteriormente ¿Qué filtro elimina mejor el ruido? Observa qué ocurre con sus histogramas.