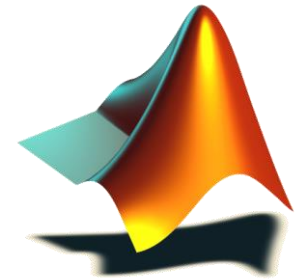


INTRODUCCIÓN A MATLAB



GIARA

Dpto. Automática y Computación

Indice

- ❑ Introducción
 - ❑ Programación
 - ❑ Números y operaciones
 - ❑ Vectores y matrices
 - ❑ Operaciones con vectores y matrices
 - ❑ Funciones para vectores y matrices
 - ❑ Gráficos 2D y 3D
-

Introducción

- ¿Qué es Matlab?, MATriX LABoratory
 - Es un lenguaje de programación (inicialmente escrito en C) para realizar cálculos numéricos con **vectores y matrices**. Como caso particular puede también trabajar con números escalares, tanto reales como complejos.
 - Cuenta con paquetes de funciones especializadas
-

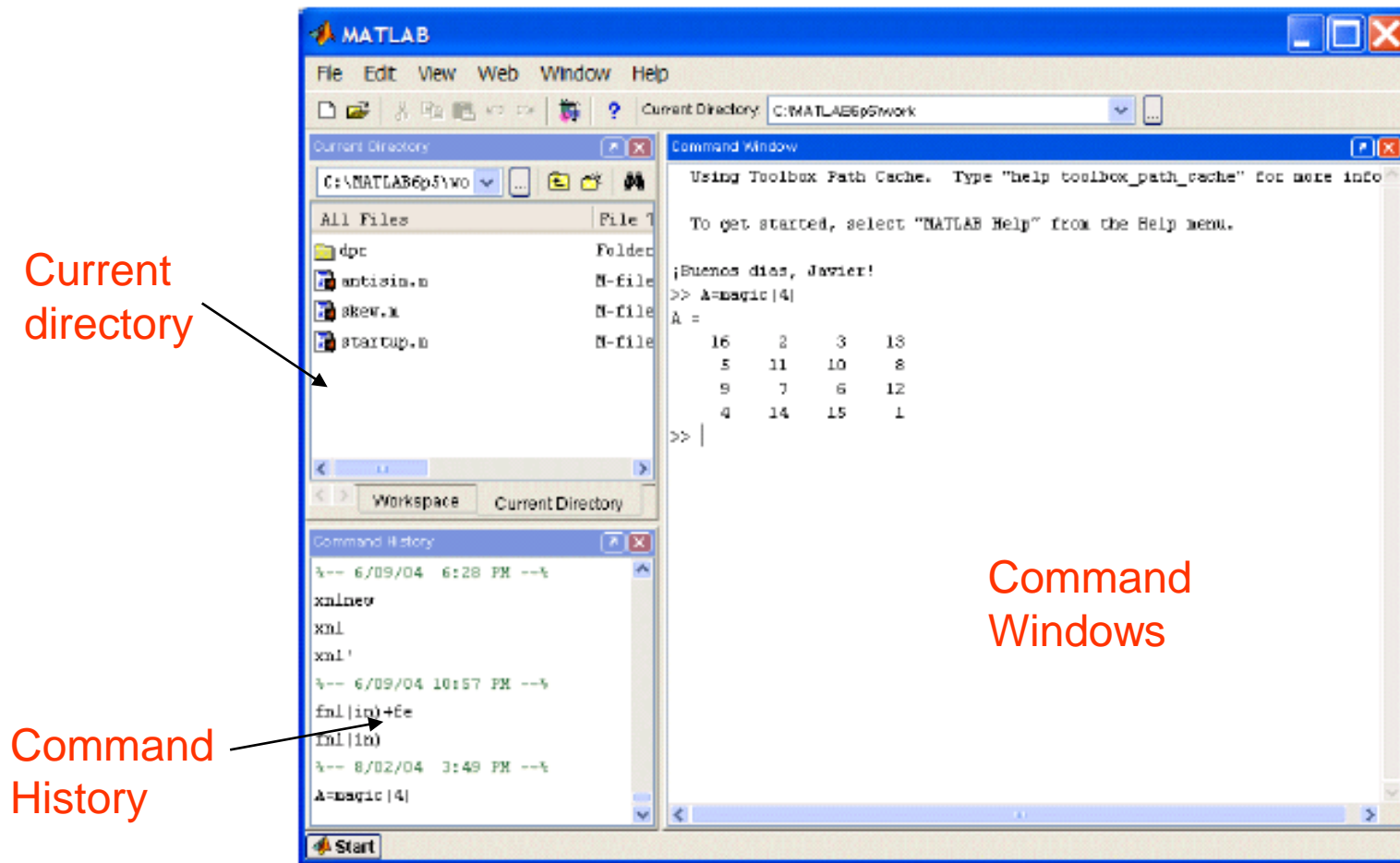
Introducción

Elementos básicos del escritorio de Matlab

- **Command Windows:** Donde se ejecutan todas las instrucciones y programas. Se escribe la instrucción o el nombre del programa y se da a Enter.
- **Command History:** Muestra los últimos comandos ejecutados en Command Windows. Se puede recuperar el comando haciendo doble click.
- **Current directory:** Situarse en el directorio donde se va a trabajar
- **Help** (también se puede usar desde comand windows)
- **Workspace:** Para ver las variables que se están usando y sus dimensiones (si son matrices)
- **Editor del Matlab:** Todos los ficheros de comandos Matlab deben de llevar la extensión .m

Introducción

Elementos básicos del escritorio de Matlab



Introducción

Algunos comentarios sobre Comand Windows

- ❑ Se pueden recuperar instrucciones con las teclas ↓↑
 - ❑ Se puede mover por la línea de comandos con las teclas → ←. Ir al comienzo de la línea con la tecla **Inicio** y al final con **Fin**. Con **Esc** se borra toda la línea.
 - ❑ **Se puede cortar la ejecución de un programa con Ctrl+C**
-

Variables

- No se declaran
- El nombre de la variable se debe iniciar con una letra. Después se pueden escribir letras, números, guiones, guiones bajos.
- Las variables son sensibles a las mayúsculas
 - ❑ No es la misma variable $x=5$ que $X=7$
- Cuando MATLAB encuentra un nuevo nombre le asigna el tamaño y el contenido asignado.
 - ❑ **IMPORTANTE:** revisar los nombres de las variables puesto que si nos equivocamos al escribir se crean variables diferentes y no se nos avisa del error!!

Variables

- Información sobre variables que se están usando y sus dimensiones (si son matrices): **Workspace**. También tecleando

>> who

>> whos (da más información)

- Para eliminar alguna variable se ejecuta

>> clear variable1 variable2

- Si se quieren borrar todas las variables

>> clear

Funciones

- Una función de MATLAB tiene valores de entrada(argumentos) y devuelve valores de salida.
 - Ejemplo:

```
>>[fil, col]=size(M);
```
- En este caso la función llamada size tiene un argumento de entrada y 2 valores de salida.
- MATLAB tiene una gran cantidad de funciones

Funciones

- Definición de una función propia:
function [s1,s2, s3] = nomfunction(e1, e2)
- se guarda con extensión .m y el nombre del fichero debe ser el mismo que el de la función
 - Ejemplo: **nomfunction.m**
- Se ejecuta en línea de comandos (o en otra función o script) asignando a variables la salida de la función de ese nombre
>>[a1, a2, a3]=nomfunction(b,c);

Programación

Ficheros de Matlab

- **Ficheros de programa (scripts):** Se construyen mediante una secuencia de comandos. El fichero principal se llamará **main_nombre.m**

Funciones útiles

- **input()** ingresa datos por pantalla.
- **disp()** muestra por pantalla (o en un archivo)

Sentencias if-else

La condición del IF debe ir entre paréntesis. La estructura de esta instrucción de control es:

```
{%if}  
if expression  
    statements  
elseif expression  
    statements  
else  
    statements  
end
```

```
a=input('ingresar el nro a: ')  
b=input('ingresar el nro b: ')  
    if a == b  
disp('son iguales')  
    else  
        disp('son distintos')  
    end
```

Estructuras de repetición: for

for *variable=valor_inicial:incremento:valor_final*
conjunto_sentencias
end

NOTA: *si no se pone incremento se hace de 1 en 1*

```
for i=1:10
    disp('El valor de i es ')
    disp(i)
end
```

```
%con una matriz:
matriz = [ 1 2 3 4; 1 2 3 4; 1 2 3 4; 1 2 3 4];
for x = matriz
    n = x(1)*x(2)*x(3)*x(4);
    disp(n)
end
```

```
%con una matriz:
matriz = [ 1 2 3 4; 1 2 3 4;
1 2 3 4; 1 2 3 4];
for x = matriz
    n = x(1)*x(2)*x(3)*x(4);
    disp(n)
end
```

Estructuras de repetición: while

while condicion (debe ir entre paréntesis)

sentencias

end

```
a=3;
while a < 5
    disp (Valor de a: ')
    disp (a)
    a = a + 1;
end
```

```
%desplegar el volumen de una
%esfera con radios de 1, 2, 3, 4 y 5
r = 0;
while r<5
    r = r+1;
    vol = (4/3)*pi*r^3;
    disp([r, vol])
end;
```

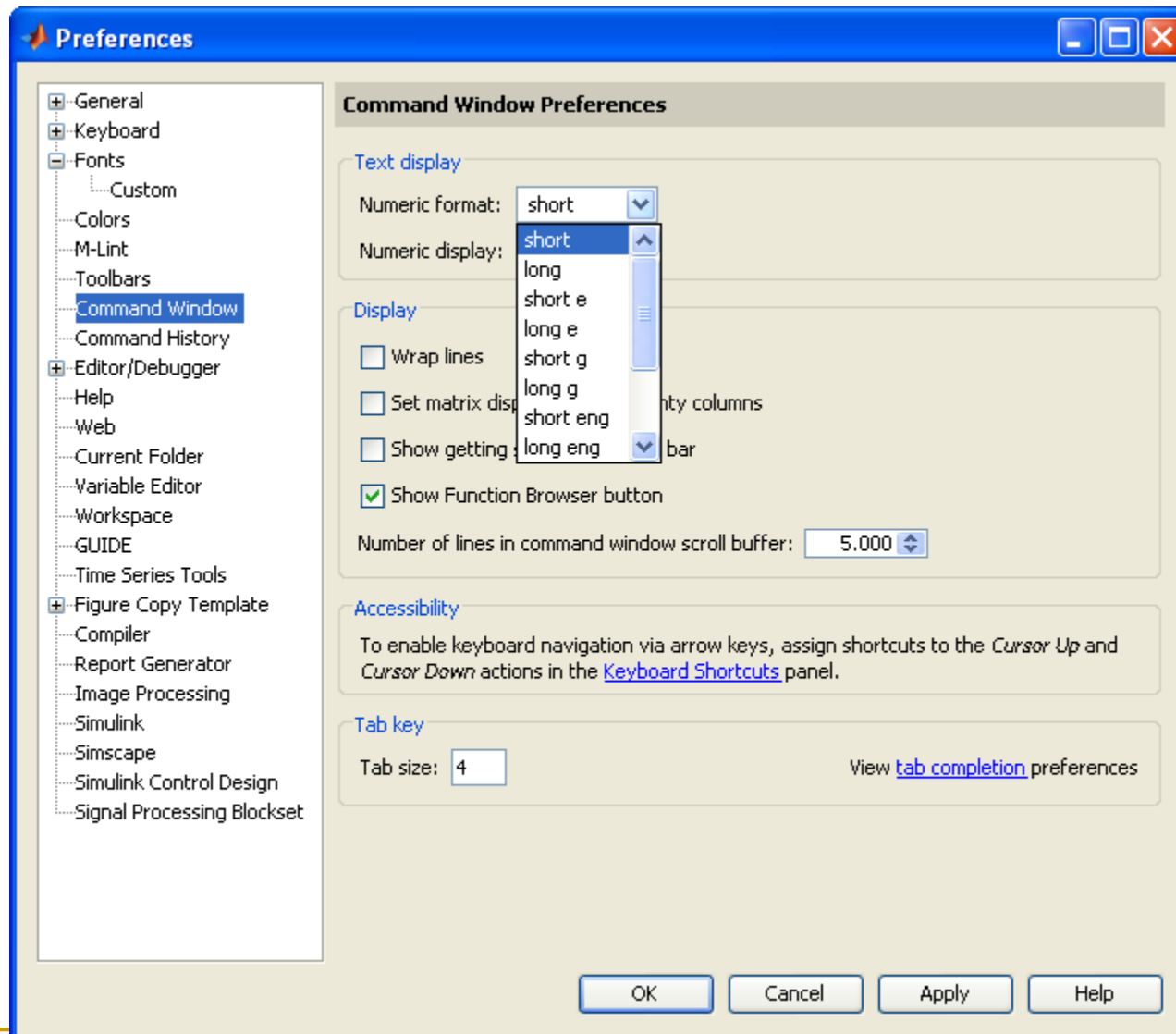
```
r = 1;
while r>0
    r = input('Teclee el radio (o -1 para terminar): ');
    if r>0
        vol = (4/3)*pi*r^3;
        disp([r, vol])
    end
end
```

Números y operaciones

Datos numéricos:

- No hace falta definir variables enteras, reales, etc. como en otros lenguajes
 - Números enteros: $a=2$
 - Números reales: $x=-35.2$
 - Máximo de 19 cifras significativas
 - $2.23e-3=2.23*10^{-3}$
- **Precisión y formatos:** Por defecto tiene un formato corto, pero se pueden usar otros
 - >> format long (14 cifras significativas)
 - >> format short (5 cifras significativas)
 - >> format short e (notación exponencial)
 - >> format long e (notación exponencial)
 - >> format rat (aproximación racional)

File → Preferences → Command Windows



Números y operaciones

Datos numéricos:

■ Constantes características:

- $\pi = \pi$
- NaN (not a number, 0/0)
- $\text{Inf} = \infty$.

■ Números complejos

- $i = \sqrt{-1}$
 >> $i = \sqrt{-1}$
 i =
 0 + 1.0000i
- $z = 2 + i \cdot 4$
 >> $z = 2 + i \cdot 4$
 z =
 2.0000 + 4.0000i
- $z = 2 + 4i$
 >> $z = 2 + 4i$
 z =
 2.0000 + 4.0000i

Números y operaciones

Funciones de Matlab:

- **exp(x), log(x), log2(x)** (en base 2), **log10(x)** (en base 10), **sqrt(x)**
- **Funciones trigonométricas:** sin(x), cos(x), tan(x), asin(x), acos(x), atan(x)
- **Funciones hiperbólicas:** sinh(x), cosh(x), tanh(x), asinh(x), acosh(x), atanh(x)
- Otras funciones: **abs(x)** (valor absoluto), **int(x)** (parte entera), **round(x)** (redondea al entero más próximo), **sign(x)** (función signo)
- **Funciones para números complejos:** real(z) (parte real), imag(z) (parte imaginaria), abs(z) (módulo), angle(z) (ángulo), conj(z) (conjugado)

Vectores

Definición de vectores

- **Vector fila**; elementos separados por blancos o comas
`>> v = [1 2 3 4]`
- **Vector columna**: elementos separados por **punto y coma** (;)
`>> u = [1;2;3;4]`
- Para obtener la dimensión de un vector
`>> longitud = length(u)`
ans
 longitud = 4
- La traspuesta de un vector (o una matriz) se hace con '
`>> w = u'`
w =
 1 2 3 4

Vectores

- Generación automática de vectores fila.
 - Especificar el primer y último valor del vector a generar, a y b respectivamente.
 - Posibilidades:
 - Especificando el incremento **h** de sus componentes
>> v=a:h:b
 - Especificando su dimensión **n**
>> linspace(a,b,n) (por defecto n=100)
 - Componentes logarítmicamente espaciadas
>> logspace(a,b,n)
(n puntos logarítmicamente espaciados entre 10^a y 10^b .
Por defecto n=50)

Vectores

- **Acceso a los elementos de un vector**

```
>> a=v(2)
```

```
a =
```

```
2
```

- **Acceder a un conjunto de elementos de un vector**

- **Escribir el nombre de la variable, la posición inicial y la final**

```
>> a=v(2:4)
```

```
a =
```

```
2 3 4
```

- **A la última posición se puede acceder con el comando end**

```
>> a=v(2:end)
```

```
a =
```

```
2 3 4
```

Vectores

- **Mostrar los elementos de un vector en orden no continuo.**
 - **Dado el vector v de la siguiente forma**

```
>>v = [10 9 8 7 6 5 4 3 2 1]
v =
10 9 8 7 6 5 4 3 2 1
```
 - **Si queremos acceder a todos los elementos de las posiciones impares**
 - **Comenzamos de la primera posición impar (1), realizamos incrementos de 2 y vamos hasta la última posición**

```
>> v(1:2:end)
ans =
10 8 6 4 2
```
 - **De forma análoga podemos acceder a las posiciones pares**

```
>> v(2:2:end)
ans =
9 7 5 3 1
```

Vectores

■ Acceder a los elementos de las posiciones que queramos

- Si queremos acceder a los elementos de las posiciones 1, 4 y 7

```
>> v([1 4 7])
```

```
ans =
```

```
10 7 4
```

- Podemos definir un vector con las posiciones deseadas y utilizarlo

```
>> a = [1 4 7];
```

```
>> v(a)
```

```
ans =
```

```
10 7 4
```

Matrices

Definición de matrices

- No hace falta establecer de antemano su tamaño
 - Se puede definir un tamaño y cambiarlo posteriormente
- **Las matrices se definen por filas:** los elementos de una misma fila están separados por blancos o comas. Las filas están separadas por punto y coma (;).

» M=[3 4 5; 6 7 8; 1 -1 0]

M =

3	4	5
6	7	8
1	-1	0

- Para generar una matriz vacía

>> M=[]

M =

[]

Matrices

- **Acceso a los elementos de una matriz: similar a los vectores pero con dos índices (fila, columna)**

- **Acceso a un elemento**

```
>> M(2, 3)
```

```
ans =
```

```
8
```

- **Acceso a una fila entera de la matriz**

```
>> M(2,:)
```

```
ans =
```

```
6    7    8
```

- **Acceso a una columna entera de la matriz**

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

```
ans =
```

```
5
```

```
8
```

```
0
```

Matrices

■ Cambiar el valor de algún elemento

```
>> M(2,3)=1;
```

```
M =
```

```
3  4  5
6  7  1
1 -1  0
```

■ Eliminar una columna

```
>> M(:,1)=[ ]
```

```
M =
```

```
4  5
7  8
-1 0
```

■ Eliminar una fila

```
>> M(2,:)=[ ]
```

```
M =
```

```
3  4  5
1 -1 0
```

Matrices

- **Para seleccionar una submatriz**

- **Ejemplo: acceder a las dos primeras filas**

```
>> M(1:2,:)
```

```
ans =
```

```
3  4  5
6  7  8
```

- **Ejemplo: crear una matriz B igual a M, y hacer que todos los elementos de la 3era columna de B sean igual a cero**

```
>> B=M;
```

```
>> M(:,3)=0
```

```
B =
```

```
3  4  0
6  7  0
1 -1  0
```

- **Acceder a elementos no continuos**

- **Ejemplo: elementos de las columnas impares**

```
>> M(1:end,1:2:end)
```

```
ans =
```

```
3  5
6  8
1  0
```

Matrices

- El operador `:` también nos devuelve todos los elementos de una matriz en un único vector columna:

```
>> M=(:)
```

```
ans =
```

```
3
```

```
6
```

```
1
```

```
4
```

```
7
```

```
-1
```

```
5
```

```
8
```

```
0
```

- La función `size(A)` devuelve las dimensiones de la matriz `A` y asigna a variables dichas dimensiones:

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

```
A =
```

```
1 1 1
```

```
1 2 3
```

```
>> [M, N] = size(A)
```

```
M =
```

```
2
```

```
N =
```

```
3
```

Matrices

Definición de matrices:

■ Generación de matrices:

- ❑ Generación de una matriz de ceros, **zeros(n,m)**
- ❑ Generación de una matriz de unos, **ones(n,m)**
- ❑ Inicialización de una matriz identidad **eye(n,m)**
- ❑ Generación de una matriz de elementos aleatorios **rand(n,m)**

■ Añadir matrices a matrices:

- ❑ $[X \ Y]$ se añaden por columnas,
 - ❑ $[X; Y]$ se añaden por filas
- siendo X e Y matrices ya definidas.

Números y operaciones

Operaciones aritméticas elementales:

- Suma: +
- Resta -
- Multiplicación: *
- División: /
- Potencias: ^
- *.Operacion* Operación elemento a elemento
- Prioridad:
 - Potencias,
 - divisiones y multiplicaciones
 - y por último sumas y restas.
 - Usar () para cambiar la prioridad

Operaciones con vectores y matrices

Operaciones de vectores y matrices con escalares:

v: vector/matriz

k: escalar:

- $v+k$ se suma k a cada elemento de v
- $v-k$ se resta k a cada elemento de v
- $v*k$ se multiplica cada elemento de v por k
- v/k se divide cada elemento de v entre k

- Operaciones elemento a elemento
 - $k./v$ divide k por cada elemento de v
 - $v.^k$ potenciación de cada componente de v a k
 - $k.^v$ potenciación k elevado a cada componente de v

Operaciones con vectores y matrices

Operaciones con vectores y matrices:

- + suma
- − resta
- * multiplicación matricial
- .* producto elemento a elemento
- ^ potenciación
- .^ elevar a una potencia elemento a elemento
- \ división-izquierda
- / división-derecha
- ./ y .\ división elemento a elemento
- matriz traspuesta: $\mathbf{B}=\mathbf{A}'$

Funciones para vectores y matrices

Funciones de matlab para vectores

- **sum(v)** suma los elementos de un vector
- **prod(v)** producto de los elementos de un vector
- **dot(v,w)** producto escalar de vectores
- **mean(v)** hace la media de un vector
- **diff(v)** vector cuyos elementos son la resta de los elementos de v
- **[y, k] = sort(v)** ordena los elementos de un vector de menor a mayor y lo almacena en y, mientras que en k se almacenan los índices ordenados
 - **[y, k] = sort(v, 'descend')** ordena los elementos de mayor a menor
- **[y,k]=max(v)** en y se guarda el valor máximo de las componentes de un vector y k indica la posición de dicho máximo
- **[y,k]= min(v)** en y se guarda el valor mínimo de las componentes de un vector y k indica la posición de dicho mínimo

Funciones para vectores y matrices

Funciones de matlab para matrices

- **sum(v)** suma los elementos de una matriz por columnas
 - **sum(v,2)** suma los elementos de una matriz por filas
- **prod(v)** producto de los elementos de cada columnas
 - **prod(v,2)** multiplica los elementos de cada fila
- **mean(v)** hace la media de una matriz por columnas
 - **mean(v, 2)** hace la media de una matriz por filas
- **[y,k]=max(v)** en y se guarda el valor máximo de cada columna de una matriz y en k las posiciones de los máximos en cada columna
 - El valor máximo de una matriz M se obtendría como **max(max(M))**
- **[y,k]=min(v)** en y se guarda el valor mínimo de cada columna de una matriz y en k las posiciones de los mínimos en cada columna
 - El valor mínimo de una matriz M se obtendría como **min(min(M))**

Funciones para vectores y matrices

Funciones de Matlab para matrices

- matriz inversa: **$B = \text{inv}(M)$** , rango: **$\text{rank}(M)$**
- **$\text{diag}(M)$** : Obtencion de la diagonal de una matriz.
 - **$\text{sum}(\text{diag}(M))$** calcula la traza de la matriz A.
 - **$\text{diag}(M,k)$** busca la k-ésima diagonal.
- **$\text{norm}(M)$** norma de una matriz (máximo de los valores absolutos de los elementos de A)
- **$\text{flipud}(M)$** reordena la matriz, haciendo la simétrica respecto de un eje horizontal.
 - **$\text{fliplr}(M)$**) reordena la matriz, haciendo la simétrica respecto de un eje vertical
- **$[V, \text{landa}] = \text{eig}(M)$** devuelve una matriz diagonal **landa** con los autovalores y otra **V** cuyas columnas son los autovectores de M

Funciones para vectores y matrices

- Creación de matrices a partir de otras
 - `repmat(A, dimensiones)`
 - A: matriz original a replicar
 - dimensiones: establece el número de copias de A

A =

```
100    0    0
   0  200    0
   0    0  300
```

```
B = repmat(A,2)
```

B =

```
100    0    0  100    0    0
   0  200    0    0  200    0
   0    0  300    0    0  300
100    0    0  100    0    0
   0  200    0    0  200    0
   0    0  300    0    0  300
```

A =

```
100    0    0
   0  200    0
   0    0  300
```

```
B = repmat(A,2,3)
```

B =

```
100    0    0  100    0    0  100    0    0
   0  200    0    0  200    0    0  200    0
   0    0  300    0    0  300    0    0  300
100    0    0  100    0    0  100    0    0
   0  200    0    0  200    0    0  200    0
   0    0  300    0    0  300    0    0  300
```

```
A = 1:4;
B = repmat(A,4,1)
```

B =

```
1    2    3    4
1    2    3    4
1    2    3    4
1    2    3    4
```

Funciones para vectores y matrices

■ Operaciones entre matrices elemento a elemento

□ bsxfun(fun,A,B)

- fun: función a aplicar
- A y B: matrices involucradas en la operación

```
A = [ 1 2 3 4 5;  
      1 2 3 4 5;  
      1 2 3 4 5 ];
```

```
B = [ 1 10 100 1000 10000];
```

```
C = bsxfun(@times,A,B)
```

```
C =  
  
      1      20     300     4000     50000  
      1      20     300     4000     50000  
      1      20     300     4000     50000
```

```
>> x = 1:10;  
>> y = x';  
>> miFun = @(x, y) x.*y;  
>> z = bsxfun(miFun, x, y)
```

```
z =
```

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

Operadores lógicos

- == Igualdad
- > Mayor
- >= Mayor o igual
- < Menor
- <= Menor o igual
- ~ not
- & and
- | or

Operadores lógicos

- En sentencias para comparar y entre vectores y matrices.

```
>> A = [1 2 3; 4 5 6; 7 8 9];  
>> B = [0 2 4 ; 3 5 6; 3 4 9];  
>> A==B  
ans=  
0 1 0  
0 1 1  
0 0 1  
>> A>=B  
ans=  
1 1 0  
1 1 1  
1 1 1
```

```
>> A = [1 2 0; 0 4 5];  
>> B = [1 -2 3 ; 0 1 1];  
>> A&B  
ans=  
1 1 0  
0 1 1  
%vale 1 si los dos operandos  
son distinto de cero y vale 0  
en los demás casos
```

Operadores lógicos

- Si queremos asignar un valor a todos los elementos que cumplan una condición

```
>> indicesLogicos = M>3;
```

```
>> M(indicesLogicos) = -3
```

M =

3	-3	-3
-3	-3	-3
1	-1	0

Funciones para vectores y matrices

- Si queremos obtener los índices de los elementos que cumplan una condición

- `find(condicion)`

- Ejemplo: buscar los índices de los elementos de la matriz M que sean mayores que 3

```
>> indices = find(M>3)
```

```
indices =
```

```
2
```

```
4
```

```
5
```

```
7
```

```
8
```

- Se puede usar la variable anterior para operar con los elementos de esas posiciones. Siguiendo con el ejemplo anterior

```
>> M(indices) = -3
```

```
M =
```

```
3  -3  -3
```

```
-3  -3  -3
```

```
1  -1  0
```

Funciones para vectores y matrices

- Comprobación de si el vector/matriz tiene elementos que sean infinito
 >>isinf(M)
- Comprobación de si el vector/matriz tiene elementos que sean NaN
 >> isnan(M)
- Guardar en ficheros y recuperar datos:
 - **save** nombre_fichero nombre_matriz1, nombre_matriz2
 - **load** nombre_fichero nombre_matriz1, nombre_matriz2
 - **save** nombre_fichero nombre_matriz1 –ascii (guarda 8 cifras decimales)
 - **save** nombre_fichero nombre_matriz1 –ascii –double (guarda 16 cifras decimales)

Gráficos 2D y 3D

Funciones gráficas 2D y 3D elementales

- **2D: plot()** crea un gráfico a partir de vectores con escalas lineales sobre ambos ejes,

>> plot(X,Y,'opción') (opción: permite elegir color y trazo de la curva)

- **hold on:** permite pintar más gráficos en la misma figura (se desactiva con **hold off**)
 - **grid** activa una cuadrícula en el dibujo. Escribiendo de nuevo grid se desactiva.
- **2D: loglog()** escala logarítmica en ambos ejes, **semilogx()**: escala lineal en el eje de ordenadas y logarítmica en el eje de abscisas, **semilogy()**: escala lineal en abscisas y logarítmica en ordenadas.

Gráficos 2D y 3D

Funciones gráficas 2D y 3D elementales

- **2D: subplot(n,m,k)** subdivide una ventana gráfica se puede en **m** particiones horizontales y **n** verticales y **k** es la subdivisión que se activa.
- **2D: polar(angulo,r)** para pintar en polares
- **2D: fill(x,y,'opción')** dibuja una curva cerrada y la rellena del color que se indique en 'opción'
- **3D: plot3** es análoga a su homóloga bidimensional **plot**.
 - » plot3(X,Y,Z, 'opción')

Gráficos 2D y 3D

Elección de la escala de los ejes

- **axis**([xmin xmax ymin ymax]) (2D),
- **axis**([xmin xmax ymin ymax zmin zmax]) (3D)
- **axis auto**: devuelve la escala a la de defecto
- **axis off**: desactiva los etiquetados de los ejes desapareciendo los ejes, sus etiquetas y la malla, **axis on**: lo activa de nuevo
- **axis equal**: los mismos factores de escala para los dos ejes
- **axis square**: cierra con un cuadrado la región delimitada por los ejes de coordenadas actuales.
- Para elegir las etiquetas que aparecen en los ejes:
 - `set(gca, 'XTick', -pi:pi/2, pi) %gca:get current axis`
 - `set(gca, 'XTicklabel', {'-pi', '-pi/2', 0, 'pi/2', 'pi'})`

Gráficos 2D y 3D

Funciones para añadir títulos a la gráfica

- **title('título')** añade un título al dibujo. Para incluir en el texto el valor de una variable numérica es preciso transformarla mediante :
 - **int2str(n)** convierte el valor de la variable entera n en carácter
 - **num2str(x)** convierte el valor de la variable real o compleja x en carácter. Ejemplo: `title(num2str(x))`
- **xlabel('texto')** añade una etiqueta al eje de abscisas. Con **xlabel off** desaparece.
 - Lo mismo **ylabel('texto')** o **zlabel('texto')**
- **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.

Gráficos 2D y 3D

Funciones de Matlab para gráficos 2D y 3D

- Imprimir gráficos: **Print** (botón File en ventana gráfica)
- Guardar gráficos: **Save** (botón File en ventana gráfica): Se crea un fichero .fig que podrá volver a editarse y modificarse
- Exportar gráficos: **Export** (botón File en ventana gráfica)
- **figure(n)**: Llamar una nueva figura o referirnos a una figura ya hecha
- **close all** borra todas las figuras, **close(figure(n))** una en concreto

Gráficos 2D y 3D

Representación gráfica de superficies

- Creación de una malla a partir de vectores **[X, Y]=meshgrid(x,y)**
- Gráfica de la malla construida sobre la superficie $Z(X,Y)$:
mesh(X,Y,Z), **meshc(X,Y,Z)** (dibuja además líneas de nivel en el plano $z=0$)
- Gráfica de la superficie $Z(X,Y)$: **surf(X,Y,Z)**, **surfc(X,Y,Z)**
- **pcolor(Z)** dibuja proyección con sombras de color sobre el plano (la gama de colores está en consonancia con las variaciones de Z)
- **contour(X,Y,Z,v)** y **contour3(X,Y,Z,v)** generan las líneas de nivel de una superficie para los valores dados en **v**. Para etiquetar las líneas, primero **cs=contour(Z)** (para saber los valores del contorno) y luego **clabel(cs)** o directamente **clabel(cs,v)**
- Ver en Demos: Graphics

Debugger



Set/Clear breakpoint: Coloca o borra un punto de ruptura en la línea en que está colocado el cursor



Clear all breakpoints: Borra todos los puntos de ruptura



Step: Avanza un paso en la ejecución del programa (F10)



Step in: Avanza un paso en la ejecución del programa y si en ese paso se llama a una función, entra en dicha función (F11)



Step out: Si hemos entrado en una función se avanza hasta haberla acabado, es decir, se sale de dicha función (Mayus+F11)



Continue: Continúa ejecutando hasta el siguiente breakpoint (F5)



Quit debugging: Termina la ejecución del debugger

Debugger

■ Ejemplo:

- ❑ Vemos un breakpoint en la línea 49 (punto rojo)
- ❑ La ejecución está parada en dicho breakpoint (flecha verde)

```
38
39
40
41 -
42 -
43
44
45 -
46 -
47
48 -
49 -
50 -
51 -
52 -
53 -
54 -
55 -
56 -
57 -
58 -
59 -
60 -
61 -
62 -
63 -
64
```

```
% TRATAMIENTO
% Obtenemos las dimensiones a tratar
dimx = ceil(3/2);
dimy = ceil(3/2);

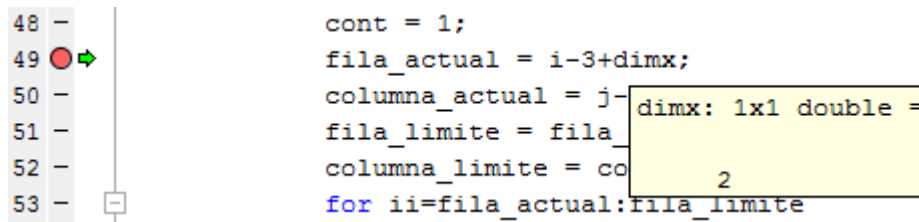
% Recorremos la IMG ...
for i = dimx:m-dimx+1
    for j = dimy:n-dimy+1
        % ... creando las submatrices RGB alrededor del pixel central
        cont = 1;
        fila_actual = i-3+dimx;
        columna_actual = j-3+dimy;
        fila_limite = fila_actual+2;
        columna_limite = columna_actual+2;
        for ii=fila_actual:fila_limite
            for jj=columna_actual:columna_limite
                submatriz(cont,1) = Img_Orig(ii,jj,1);
                submatriz(cont,2) = Img_Orig(ii,jj,2);
                submatriz(cont,3) = Img_Orig(ii,jj,3);
                r(cont) = Img_Orig(ii,jj,1);
                g(cont) = Img_Orig(ii,jj,2);
                b(cont) = Img_Orig(ii,jj,3);
                cont=cont+1;
            end
        end
    end
end
```

Debugger

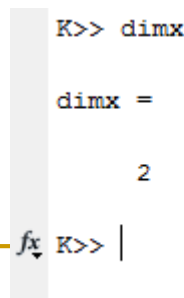
- En este momento podemos evaluar el valor de:

- Variables

- Poniendo el cursor encima de la variable

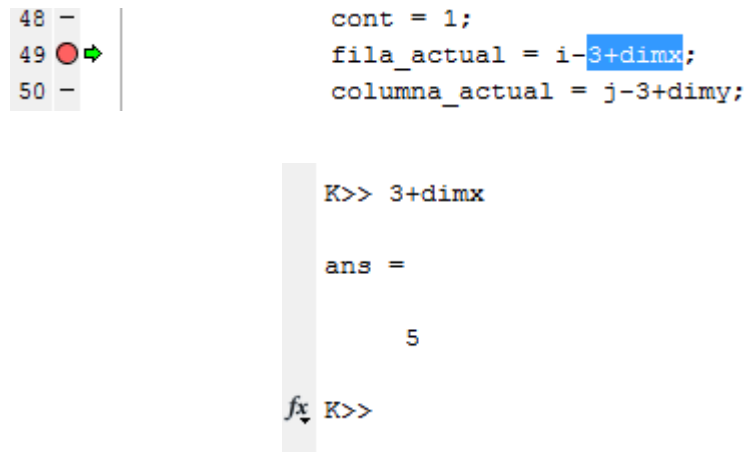


- Seleccionando la variable y presionando F9: en este caso el valor de la variable se mostrará en la línea de comandos



Debugger

- En este momento podemos evaluar el valor de:
 - Operaciones: seleccionar la operación a evaluar y presionar F9 (el resultado aparece en la línea de comandos)



The image shows a debugger interface. On the left, a code editor displays three lines of code: line 48 with a minus sign, line 49 with a red circle and a green arrow pointing right, and line 50 with a minus sign. The code on line 49 is `fila_actual = i-3+dimx;`, where the expression `3+dimx` is highlighted in blue. On the right, a command window shows the result of evaluating the selected expression: `K>> 3+dimx` followed by `ans =` and the value `5`. At the bottom of the command window, there is a prompt `f> K>>`.

```
48 -  
49 ● → fila_actual = i-3+dimx;  
50 -  
  
K>> 3+dimx  
  
ans =  
  
5  
  
f> K>>
```

Debugger

- Podemos cambiar la operación y evaluar el resultado de la modificación
 - ❑ Sin volver a ejecutar de nuevo
 - ❑ El punto rojo aparece en gris por haber cambiado el programa
 - ❑ Muy útil para corregir operaciones que fallan

```
9 ● →  
0 -  
1 -
```

```
fila_actual = i-4*dimx;  
columna_actual = j-3+dimy;  
fila_limite = fila_actual+2;
```

```
K>> 4*dimx
```

```
ans =
```

```
8
```

```
fx K>>
```