

BREVES NOTAS DE MATLAB

ENTORNO, MATRICES, GRÁFICOS, SENTENCIAS DE CONTROL DE FLUJO, FUNCIONES y SCRIPTS

Entorno Matlab

1. **Current Directory:** directorio en el disco duro sobre el que trabaja Matlab (MATrix LABoratory)
2. **Workspace:** conjunto de variables que se definen en una sesión. En la ventana *workspace* se indican las variables definidas, sus valores y tipos. Las variables se pueden guardar en un **fichero .mat**. También es posible importar un fichero .mat.
 - `save resultados.mat` = [Ventana Workspace/Botón Guardar]
 - `load resultados.mat` = [Ventana Workspace/Botón Abrir]
3. **Command Window:** ventana de comandos
Modos de ejecución de comandos o funciones:

- **modo directo**, es decir, desde la ventana de comandos

```
>> nombre_comando + <enter>
```

El carácter >> es el prompt que aparece en la ventana de comandos

Ejemplo: >> help nombre_comando

- **mediante un programa:** ejecutando un fichero .m. Los ficheros .m contienen programas escritos en lenguaje matlab

```
>> nombre_de_fichero + <enter>
```

El fichero .m debe estar en el directorio de trabajo. Para crear ficheros .m:
[Ventana Matlab/File/New/M file]

Para abortar un comando o función, en cualquier modo, pulsar CTRL+C

4. **Command History:** historial de comandos

Lenguaje Matlab

- **Algunos comandos.**

```
>> Nombre_comando % el símbolo "%" para introducir
comentarios en el código
>> Nombre_comando1 , Nombre_comando2 % el carácter ","
para separar comandos
>> type nombre_comando % muestra el código matlab del
comando nombre_comando
>> help nombre_comando % muestra información sobre el
comando
>> doc nombre_comando % muestra información sobre el
comando en un navegador
>> clear % borra todas las variables del espacio de
trabajo (workspace)
>> clc % borra la ventana de comandos
>> whos nombre_variable % muestra información sobre la
variable nombre_variable
>> close all % cierra todas las ventanas gráficas tipo
figure
>> imview close all % cierra todas las ventanas gráficas
tipo image viewer
```

Matrices y variables

- Si se define una variable sin indicar el tipo, Matlab la define por defecto de tipo *double* (punto flotante de doble precisión, 8 bytes). Las variables con nombres en mayúsculas y minúsculas denotan variables distintas. Antes de usar una variable hay que inicializarla:

```
>> A=1;      % símbolo "%" para introducir comentarios
>> B=A+1     % si no ponemos ";" el resultado se muestra
en pantalla
```

- **Todas las variables tienen carácter de matriz**

```
>> A=[1 2; 3 4];    →       $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ 
```

El tamaño de la matriz se puede modificar posteriormente a su definición

```
>> C=1;      % escalar = matriz 1x1

>> M=[]      % matriz vacía

>> sum(A)    % [4 6], suma columnas de matriz A

>> A'        % transpuesta de A

>> inv(A)    % matriz inversa de A
```

```
>> diag(A)% diagonal de A (en una columna)
```

$A(i,j)$ → elemento de la fila i y la columna j de la matriz A

```
>> all=A(1,1); % asignamos a la variable all el valor de A(1,1)
```

```
>> A(1,1)=9; % cambiamos valor de A(1,1)
```

```
>> 1:10 % vector fila [1 2 3 4 5 6 7 8 9 10]
```

```
>> 1:2:10 % vector fila [1 3 5 7 9]
```

```
>> k=4; j=2; A(1:k,j) % primeros k elementos de la columna j
```

```
>> A(2,:) % la segunda fila de A
```

```
>> A(:,end) % la última columna de A
```

```
>> A=B(:, [1 3 2 4]) % cambia las columnas 2 y 3
```

```
>> A+1; % suma la constante 1 a todos los elementos de la matriz (lo mismo para las operaciones “-“, “/” y “*”)
```

```
>> sin(A); % obtiene una matriz cuyos elementos son los senos de los elementos de A correspondientes (lo mismo para otras funciones)
```

```
>> B=[A A+1 ; A+2 A+3]; %concatenación de matrices
```

```
>> A(:,2)=[]; % elimina la columna 2
```

○ **Generando matrices**

```
>> A=zeros(2,4); % matriz 2x4 de ceros
```

```
>> B=ones(2,3); % matriz 2x3 de unos
```

```
>> B=eye(3,3); % matriz identidad 3x3
```

`rand`, `randn` → números aleatorios según distribución uniforme o normal respectivamente

```
>> load matriz.dat % siendo matriz.dat un fichero de datos, matlab crea la matriz de nombre “matriz” para recoger los datos del fichero. Igual con un fichero .txt
```

○ **Operaciones entre matrices**

`A+B;` % suma matrices, igual con resta (-)

`A*B;` % producto matricial

`inv(A);` % inversa de A

`A^k ;` % potencia k

Operaciones elemento a elemento en matrices: “.*“ , “./“ , “.^“ , “.^“ , “.^“

`mean(A);` % vector fila con media de cada columna

`std(A) ;` % lo mismo pero con desviación típica

- **Tipos de datos simples** (Matlab dispone de otros tipos más avanzados como estructuras, matrices de celdas, clases, etc..). Consultar las páginas 28 a 31 del documento *Aprenda Matlab como si estuviera en primero*, publicado por la E.T.S. de Ingenieros Industriales de la U.P.M.

Por defecto, Matlab trata todas las variables como datos de punto flotante y doble precisión (8 bytes). Sin embargo, para conseguir una mayor velocidad en los cálculos y ahorro de memoria se pueden declarar variables de otros tipos:

- `logical` ; un bit
- `int8` ; entero con signo de 8 bits
- `int16` ; entero con signo de 16 bits
- `int32` ; entero con signo de 32 bits
- `int64` ; entero con signo de 64 bits
- `uint8` ; entero sin signo de 8 bits
- `uint16` ; entero sin signo de 16 bits
- `uint32` ; entero sin signo de 32 bits
- `uint64` ; entero sin signo de 64 bits
- `single` ; real de punto flotante con 4 bytes
- `double` ; real de punto flotante con 8 bytes, empleados por Matlab por defecto

Para realizar declaraciones de variables en otros tipos y llevar a cabo conversiones entre tipos vea el siguiente ejemplo:

```
>> A=5.5 % se crea por defecto una variable A de tipo double
>> B=uint8(6) % se crea una variable B de tipo uint8 de valor 6
>> C=int32(A) % se convierte el valor de la variable A de tipo double en tipo int32 y se almacena en la variable C
```

Si las variables son matrices el procedimiento para la definición y conversión de tipos es el mismo.

Otros tipos de datos (para detectar errores):

- `inf` % infinito
- `NaN` % not a number

Probar con: `0/0` y `1/0`

○ **Números complejos:**

```
>> z=1+2i % número complejo
>> real(z) % devuelve parte real
>> imag(z) % devuelve parte imaginaria
>> conj(z) % devuelve el conjugado de z
```

○ **Cadenas de texto:**

```
>> s='cadena de caracteres'
```

Gráficos

```
>> plot(x) % valores de x frente al índice (siendo x un vector)
>> plot(x,y) % vector y frente a vector x
```

Ejemplo:

```
>> x=0:pi/100:2*pi;
>> y=sin(x);
>> figure(número_figura) % crea una ventana gráfica y es la
actual
>> plot(x,y); % dibuja gráfico en ventana actual (y frente a
x)
>> xlabel('texto') % texto para el eje x
>> ylabel('texto') % texto para el eje y
>> title('texto') % texto para el título
>> y1=sin(x-0.25); % otra función
>> y2=sin(x-0.5); % otra función
>> hold on % mantiene la ventana anterior para que se dibuje
en ella el próximo gráfico sin borrar dicha ventana
>> plot(x,y1,x,y2) % dos curvas
>> legend('sen(x)', 'sen(x-0.25)', 'sen(x-0.5)') % leyendas
>> grid on % para dibujar rejilla en el gráfico
>> figure; % otra ventana
>> plot(x,y);
```

Si `Z` es complejo `plot(Z)` equivale a `plot(real(Z), imag(Z))`

`axis` → para modificar el rango de los ejes

La ventana gráfica tiene opciones para modificar títulos, leyendas, rangos de ejes, etc. Además se pueden aplicar *zooms* y giros a los gráficos.

Sentencias de control de flujo

La sentencia `help nombre_comando` muestra información sobre la funcionalidad y la sintaxis de cualquier comando.

Relational operators.

<code>eq</code>	- Equal	<code>==</code>
<code>ne</code>	- Not equal	<code>~=</code>
<code>lt</code>	- Less than	<code><</code>
<code>gt</code>	- Greater than	<code>></code>
<code>le</code>	- Less than or equal	<code><=</code>
<code>ge</code>	- Greater than or equal	<code>>=</code>

Logical operators.

<code>and</code>	- Logical AND	<code>&</code>
<code>or</code>	- Logical OR	<code> </code>
<code>not</code>	- Logical NOT	<code>~</code>
<code>xor</code>	- Logical EXCLUSIVE OR	
<code>any</code>	- True if any element of vector is nonzero	
<code>all</code>	- True if all elements of vector are nonzero	

Sentencia *if*

```
if <condición>
    <sentencia>
elseif <condición>
    <sentencia>
else <sentencia>
end
```

Sentencia *switch*

```
switch <expresion>
    case 0 ...
    case 1 ...
    ...
    otherwise ...
end
```

Sentencia *for*

```
for i = 1 : 2 :10
...
end
```

Sentencia *while*

```
while <condición>
...
end
```

Sentencias *continue* y *break*

Para modificar el flujo en bucles

- *continue* → para saltar al siguiente ciclo del bucle
- *break* → para salir del bucle

Algunas funciones Matlab importantes

abs(x) → valor absoluto

size(x) → rango de una matriz

sin(x), *cos(x)* → funciones trigonométricas

sqrt(x) → raíz cuadrada

length(x) → longitud de un vector

unique(x) → valores sin repetición de un vector o matriz

mat2gray(x) → transformación lineal de los valores de x al rango 0-1

Scripts y funciones definidos por el usuario

Los ficheros *.m* contienen código Matlab creado por el usuario. Lo editamos con cualquier editor. Se pueden usar opciones del menú *File* para crear, cargar y editar estos ficheros. Se ejecutan como cualquier comando o función Matlab desde la ventana de comandos, o bien desde la ventana de edición del fichero *.m* accediendo la orden *Run* en el menú *Debug*.

Para ejecutarlos deben ser guardados previamente en el *current directory*

Hay dos tipos:

- *Scripts*
Operan sobre variables del entorno o crean nuevas variables del entorno (espacio de trabajo, también llamado *workspace*). Los valores de las variables y éstas permanecen una vez ejecutado el *script*, ya que son variables del *workspace*.

- Funciones

Aceptan argumentos de entrada y devuelven variables de salida. Las variables son locales. El nombre de la función y del fichero .m debe ser el mismo. Por tanto, el fichero .m de una función debe nombrarse nombre_funcion.m,

El contenido del fichero .m debe seguir la siguiente estructura:

```
function y=nombre_funcion(a,b)
% esta función tiene dos variables de entrada y una de
salida

% cuerpo de la funcion:
.....

y = .....% se calcula el resultado y se asigna a la
variable y, que fue definida como la variable de salida
```

Para ejecutar una función desde la ventana de comandos, recibiendo, por ejemplo, los parámetros 1 y 4:

```
>> nombre_funcion(1,4)
```

Si se quiere recoger el valor devuelto por la función en una variable:

```
>> resultado=nombre_funcion(1,4)
```