Programación en MatLab

Introducción a la Informática

Ing. Soldiamar Matamoros Encalada

Matlab

- "MATrix LABoratory"
- MATLAB es un programa para realizar cálculos numéricos con ***vectores*** y ***matrices***.
- También trabajar con números escalares –tanto reales como complejos–, con cadenas de caracteres y con otras estructuras de información más complejas.
- Realiza una amplia variedad de ***gráficos*** en dos y tres dimensiones.
- Tiene un lenguaje de programación propio.

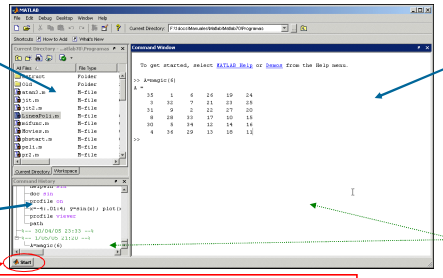
Ventana Inicial

- Ésta es la vista que se obtiene eligiendo la opción ***Desktop Layout/Default***, en el menú ***View***

Muestra los ficheros del directorio activo o actual.
Se puede alternar con ***Workspace*** clicando en la pestaña correspondiente.

Muestra los últimos comandos ejecutados en la ***Command History***.

Estos se pueden volver a ejecutar haciendo doble clic sobre ellos.



Donde se ejecutan los comandos de MATLAB, a continuación del ***prompt*** característico (>>)

Para editar uno de estos comandos hay que copiarlo antes a la ***Command Window***

función análoga a la del botón ***Inicio*** de ***Windows***

El ***Workspace*** contiene información sobre todas las variables que se hayan definido en esta sesión y permite ver y modificar las matrices con las que se esté trabajando.

Un ejercicio sencillo

- Escriba en la ***Command Window*** la siguiente línea, a continuación del *prompt y* luego pulsar ***intro***.

- Se han escrito tres instrucciones diferentes, separadas por comas.
- También la respuesta del programa tiene tres partes.

- Primera instrucción
  - se define una matriz cuadrada (6×6) llamada **A**, cuyos elementos son números aleatorios entre cero y uno (aunque aparezcan sólo 4 cifras, han sido calculados con 16 cifras de precisión).
- Segunda instrucción
  - se define una matriz **B** que es igual a la inversa de **A**.
- Tercera Instrucción
  - Finalmente se ha multiplicado **B** por **A**, y se comprueba que el resultado es la matriz unidad.

- **>> A=rand(6), B=inv(A), B*A**

```
A =
0.9501 0.4565 0.9218 0.4103 0.1389 0.0153
0.2311 0.0185 0.7382 0.8936 0.2028 0.7468
0.6068 0.8214 0.1763 0.0579 0.1987 0.4451
0.4860 0.4447 0.4057 0.3529 0.6038 0.9318
0.8913 0.6154 0.9355 0.8132 0.2722 0.4660
0.7621 0.7919 0.9169 0.0099 0.1988 0.4186
B =
5.7430 2.7510 3.6505 0.1513 -6.2170 -2.4143
-4.4170 -2.5266 -1.4681 -0.5742 5.3399 1.5631
-1.3917 -0.6076 -2.1058 -0.0857 1.5345 1.8561
-1.6896 -0.7576 -0.6076 -0.3681 3.1251 -0.6001
-3.6417 -4.6087 -4.7057 2.5299 6.1284 0.9044
2.7183 3.3088 2.9929 -0.1943 -5.1286 -0.6537
ans =
1.0000 0.0000 0 0.0000 0.0000 -0.0000
0.0000 1.0000 0.0000 0.0000 -0.0000 0.0000
0 0 1.0000 -0.0000 -0.0000 0.0000
0.0000 0 -0.0000 1.0000 -0.0000 0.0000
-0.0000 0.0000 -0.0000 -0.0000 1.0000 0.0000
-0.0000 -0.0000 -0.0000 -0.0000 -0.0000 1.0000
```

Cont...

- Con la ***Command History***, es posible recuperar comandos anteriores de MATLAB y moverse por dichos comandos con el ratón y con las teclas- flechas ↑ y ↓.
  - Al pulsar la primera de dichas flechas aparecerá el comando que se había introducido inmediatamente antes.
- De modo análogo es posible moverse sobre la línea de comandos con las teclas ← y →, ir al principio de la línea con la tecla ***Inicio***, al final de la línea con ***Fin***, y borrar toda la línea con ***Esc***.
- Recuérdese que sólo hay una línea activa (la última).
- Para borrar todas las salidas anteriores de MATLAB y dejar limpia la ***Command Window*** se pueden utilizar las funciones ***clc*** y ***home***.
  - La función ***clc*** (*clear console*) elimina todas las salidas anteriores, mientras que ***home*** las mantiene, pero lleva el ***prompt*** (**>>)** a la primera línea de la ventana.
- Si se desea salir de MATLAB basta teclear los comandos ***quit*** o ***exit***, elegir ***Exit*** MATLAB en el menú ***File*** o utilizar cualquiera de los medios de terminar una aplicación en ***Windows***.

Cont...

- Los programas de MATLAB se encuentran en ficheros con la extensión ****.m***.
  - Se ejecutan tecleando su nombre en la línea de comandos (sin la extensión), seguido de los argumentos entre paréntesis, si se trata de funciones.
- El comando ***pwd*** (*print working directory*) permite saber cuál es el ***directorio actual***.
- Para cambiar de ***directorio actual*** se puede utilizar el comando ***cd*** (de *change directory*) seguido del nombre del directorio
  - Se puede utilizar un ***path***
    - Absoluto: ***cd C:\Matlab\Ejemplos*** o
    - Relativo (***cd Ejemplos***).
  - Para subir un nivel en la jerarquía de directorios se utiliza el comando ***cd ..***, y ***cd ..\..*** para subir dos niveles.
  - Éste es el mismo sistema que en MS-DOS.
  - MATLAB permite utilizar la barra normal (/) y la barra invertida (\), indistintamente.

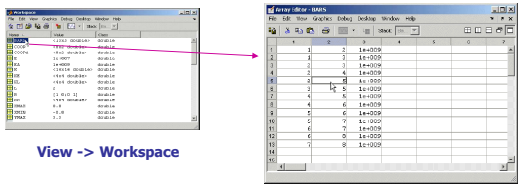
Cont...

- **Workspace** (El espacio de trabajo de MATLAB) es el conjunto de variables y de funciones de usuario que en un determinado momento están definidas en la memoria del programa o de la función que se está ejecutando.
- Para obtener información sobre el ***Workspace*** se pueden utilizar los comandos ***who*** y ***whos*** (información más detallada).
- ***Cada función tiene su propio espacio de trabajo***, con variables cuyos nombres no interfieren con las variables de los otros espacios de trabajo.

```
>> whos
     Name Size Bytes Class
     A 3x3 72 double array
     B 3x3 72 double array
     C 3x3 72 double array
     D 3x3 72 double array
     Grand total is 36 elements using 288 bytes
```


Cont...

- Al hacer doble clic en uno de los elementos del workspace aparece el Array editor, en el cual se puede modificar los datos

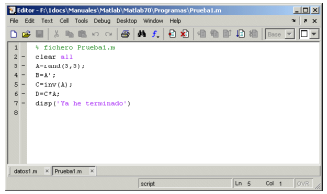


View -> Workspace

El Editor/Debugger

- El tipo de los ***ficheros-M*** (o ***M-files***)(****.m***) es texto ASCII, y contienen ***conjuntos de comandos*** o ***definición de funciones***
- Al teclear su nombre en la línea de comandos y pulsar ***Intro***, se ejecutan uno tras otro todos los comandos contenidos en dicho fichero.
- Se puede guardar instrucciones y grandes matrices (ahorra el trabajo de tecleado).
- Se pueden crear con cualquier editor de ficheros ASCII tal como ***Notepad.***
  - MATLAB dispone de un ***editor*** para crear, modificar y ejecutarlos paso a paso para ver si contienen errores (proceso de ***Debug*** o depuración).

- El ***Editor*** muestra con diferentes colores los diferentes tipos o elementos constitutivos de los comandos (en *verde* los comentarios, en *violeta* las cadenas de caracteres, etc.).
- El ***Editor*** se preocupa de que las comillas o paréntesis que se abren, se cierren.
  - Al olocar el cursor antes o después de una apertura o cierre de corchete o paréntesis y pulsar las teclas (←) o (→), se muestra con qué cierre o apertura de corchete o paréntesis se empareja el elemento considerado; si no se empareja con ninguno, aparece con una rayita de tachado.



9

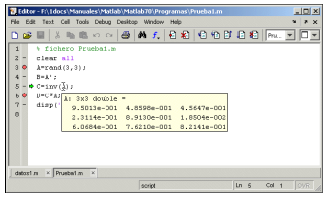
Cont… El Editor/Debugger

- Para comentar con el carácter % varias líneas se debe seleccionarlas, hacer clic derecho y elegir ***Comment.***
  - Estos comentarios pueden volver a ser código ejecutable seleccionándolos y ejecutando ***Uncomment*** en el menú contextual.
- Se puede organizar el sangrado de los bucles y bifurcaciones seleccionado ***Smart Indent*** del menú contextual.
- Para ejecutar un fichero se debe eligir el comando ***Run*** en el menú ***Debug***, pulsar la tecla **F5**, clicar en el botón ***Continue*** ( ) de la barra de herramientas del ***Editor*** o teclear el nombre del fichero en la línea de comandos de la ***Command Window***.

10

Cont… El Editor/Debugger

- Los puntos rojos que aparecen en el margen izquierdo son ***breakpoints*** (puntos en los que se detiene la ejecución de programa)
- La ***flecha verde*** en el borde izquierdo indica la sentencia en que está detenida la ejecución (antes de ejecutar dicha sentencia)
- Cuando el cursor se coloca sobre una variable (en este caso sobre **A**) aparece una pequeña ***ventana con los valores numéricos*** de esa variable, tal como se ve en la.

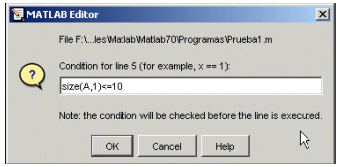


```
clear all;
A=rand(3,3);
B=A';
C=inv(A);
D=C*A;
disp('Ya he terminado');
```

11

Cont… El Editor/Debugger

- Se puede introducir ***breakpoints condicionales*** (punto amarillo)***,*** en los que el programa se para sólo si se cumple una determinada condición.
- Clicar con el botón derecho en la correspondiente línea del código en la ventana del ***Editor/Debugger*** y elegir en el menú contextual ***Set/Modify Conditional Breakpoint,*** en la ventana mostrada se escribe la condición que debe cumplirse para que el programa se detenga en dicho punto



12

El profiler

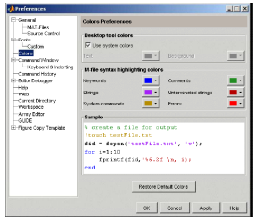
- Permite saber cómo se ha empleado el tiempo de la CPU en la ejecución de un determinado programa.
- Útil para determinar los cuellos de botella de un programa: funciones y las líneas de código que más veces se llaman.
- Esto ayuda a mejorar la eficiencia de un programa.

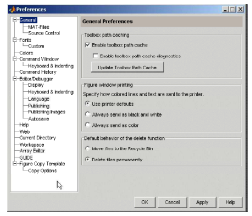
13

Preferencias

Formatos de salida y de otras opciones de MATLAB

- **File -> Preferences** muestra todas las posibilidades que ofrece MATLAB



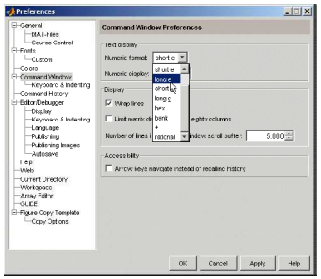


Permite elegir los colores generales del código

14

Cont... Preferencias

- MATLAB siempre calcula con doble precisión, es decir con unas 16 cifras decimales equivalentes



- Las posibilidades existentes son las siguientes:
  - **short** coma fija con 4 decimales (***defecto***)
  - **long** coma fija con 15 decimales
  - **hex** cifras hexadecimales
  - **bank** números con dos cifras decimales
  - **short e** notación científica con 4 decimales
  - **short g** notación científica o decimal, dependiendo del valor
  - **long e** notación científica con 15 decimales
  - **long g** notación científica o decimal, dependiendo del valor
  - **rational** expresa los números racionales como cocientes de enteros

Se pueden cambiar también desde la línea de comandos usando *format*.
Ejemplo: para ver las matrices en formato *long*
>> format long

15

Guardar variables y estados de una sesión: save y load

- Se puede guardar el estado de una sesión de trabajo tecleando **save** antes de abandonar el programa.
  - Esto crea en el ***directorio actual*** un fichero binario llamado ***matlab.mat*** (o ***matlab***) con el estado de la sesión (excepto los gráficos, que por ocupar mucha memoria hay que guardar aparte).
- Para recuperar el estado la siguiente vez que se arranque el programa se usa **load**
- Se pueden guardar también matrices y vectores de forma selectiva y en ficheros con nombre especificado por el usuario

>> save,

>> load

>> save filename A x y

%(sin comas entre los nombres de variables)

Las tres variables deben tener valores previamente.

>> load filename

% Si no se indica ninguna variable, se guardan todas las variables creadas en esa sesión.

% guarda las variables **A**, **x** e **y** en un fichero binario llamado ***filename.mat*** (o ***filename***).

16

Guardar variables y estados de una sesión: save y load

- Para almacenar en formato ASCII:

>> save -ascii
% almacena 8 cifras decimales

>> save -ascii -double
% almacena 16 cifras decimales

>> save -ascii -double -tab
% almacena 16 cifras separadas por tabs aunque en formato ASCII sólo se guardan los valores y no otra información tal como los nombres de las matrices y/o vectores.

- Cuando se recuperan estos ficheros con **load -ascii** toda la información se guarda en una única matriz con el nombre del fichero. Esto produce un error cuando no todas las filas tienen el mismo número de elementos.
- Con la opción **-append** en el comando **save** la información se guarda a continuación de lo que hubiera en el fichero.

Guardar sesión y copiar salidas: diary

- Existe otra forma más sencilla de almacenar.
- Con el comando ***diary*** se almacena de forma más sencilla en un fichero un texto que describa lo que el programa va haciendo (la entrada y salida de los comandos utilizados)

```
>> diary filename.txt
...
>> diary off  % suspende la ejecución de diary
...
>> diary on ... % reanuda la ejecución de diary
```

- El simple comando ***diary*** pasa de ***on*** a ***off*** y viceversa.
- Para poder acceder al fichero ***filename.txt*** con ***Notepad*** o ***Word*** es necesario que ***diary*** esté en ***off***.
- Si en el comando ***diary*** no se incluye el nombre del fichero se utiliza por defecto un fichero llamado ***diary*** (sin extensión).

Comentarios

- **Líneas de comentarios** se realizan por medio del %, todo lo que se coloca de ahí hasta el fin de la línea es comentario.
- Para comentar un bloque de sentencias, se las selecciona y del menú contextual se elige la opción ***Comment***.
- Otra forma de comentar bloques de sentencias es encerrar las líneas que se desea inutilizar entre los caracteres **%{** y **%}**.

Medida de tiempos y de esfuerzo de cálculo

- **cputime** devuelve el tiempo de CPU (con precisión de centésimas de segundo) desde que el programa arrancó.
  - **Ejemplo:** Se lo puede llamar antes y después de realizar una operación y se restan los valores devueltos
- **etime(t2, t1)** tiempo transcurrido entre los vectores **t1** y **t2** (¡atención al orden!), obtenidos como respuesta al comando ***clock***.
- **tic** ***ops*** **toc** imprime el tiempo en segundos requerido por ***ops***.
  - El comando ***tic*** pone el reloj a cero y ***toc*** obtiene el tiempo transcurrido.

Cont...

- **Ejemplo:**
  - Medir de varias formas el tiempo necesario para resolver un sistema de 1000 ecuaciones con 1000 incógnitas.

```
>> n=1000; A=rand(n); b=rand(n,1); x=zeros(n,1);
>> tiempoIni=clock; x=A\b; tiempo=etime(clock, tiempoIni)
>> time=cputime; x=A\b; time=cputime-time
>> tic; x=A\b; toc
```

  - Los tiempos pequeños (décimas o centésimas de segundo), no se pueden medir con gran precisión donde se han puesto varias sentencias en la misma línea.
  - Todas las sentencias de cálculos matriciales van seguidas de punto y coma (;) con objeto de evitar la impresión de resultados.
  - Conviene ejecutar dos o tres veces cada sentencia para obtener tiempos óptimos, ya que la primera vez que se ejecutan se emplea un cierto tiempo en cargar las funciones a memoria.

Cont... Operaciones con Matrices y Vectores

- Como en casi todos los lenguajes de programación, en MATLAB las matrices y vectores son **variables** que tienen **nombres**.
- Se sugiere (MATLAB no lo exige) que se utilicen:
  - letras **mayúsculas para matrices** y
  - letras **minúsculas para vectores y escalares**
- Para definir una matriz no hace falta declararlas o establecer de antemano su tamaño
  - 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 o introducen por filas**; los elementos de una misma fila están separados por **blancos** o **comas**, mientras que las filas están separadas por pulsaciones **intro** o por caracteres **punto y coma** (;).

Ejemplo: el siguiente comando define una matriz **A** de dimensión (3×3):

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

La respuesta del programa es la siguiente:

```
A =
1 2 3
4 5 6
7 8 9
```


Cont... Operaciones con Matrices y Vectores

- Además de valores numéricos, en la definición de una matriz o vector se pueden utilizar expresiones y funciones matemáticas.
- **Ejemplo:** una sencilla operación con **A** es hallar su **matriz traspuesta** (**A'**).
  - Como el resultado de la operación no ha sido asignado a ninguna otra matriz, MATLAB utiliza un nombre de variable por defecto (***ans***, de *answer*), que contiene el resultado de la última operación.

```
>> A'
ans =
1 4 7
2 5 8
3 6 9
```

- **ans** puede ser utilizada como operando en la siguiente expresión que se introduzca.
- **Ejemplo:** Podría haberse asignado el resultado a otra matriz llamada **B**:

```
>> B=A'
B =
1 4 7
2 5 8
3 6 9
```


Cont... Operaciones con Matrices y Vectores

- **Ejemplo:** Hacer el producto **B*A** (deberá resultar una matriz simétrica):

```
>> B*A
ans =
66 78 90
78 93 108
90 108 126
```

- Para acceder a los elementos de un vector se pone el índice entre paréntesis: **x(3)** ó **x(i)**
- Los elementos de las matrices se acceden poniendo los dos índices entre paréntesis, separados por una coma.
- **Ejemplo: A(1,2)** ó **A(i,j)**
  - Aunque en MATLAB las matrices se introducen por filas, se almacenan por columnas lo cual permite que se **puede accederse a cualquier elemento de una matriz con un sólo subíndice**.

Cont... Operaciones con Matrices y Vectores

- **Ejemplo:**
  - si **A** es una matriz (3×3) se obtiene el mismo valor escribiendo **A(1,2)** que escribiendo **A(4)**.
- Invertir una matriz es casi tan fácil como trasponerla. A continuación se va a definir una nueva matriz **A** -no singular- en la forma:

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

- **Ejemplo:**
  - Calcular la inversa de **A** y el resultado asignar a **B**.

```
>> B=inv(A)
B =
0.1803 0.2213 -0.1885
0.1311 0.0246 0.0902
-0.0984 0.1066 0.0574
```

- Para comprobar que este resultado es correcto basta pre-multiplicar **A** por **B**;

```
>> B*A
ans =
1.0000 0.0000 0.0000
0.0000 1.0000 0.0000
0.0000 0.0000 1.0000
```

25

Cont... Operaciones con Matrices y Vectores

- **Vector Fila y Vector Columna**
  - Son diferentes para MatLab

- **Vector Fila x**
  - Para definir un v**ector fila x** se lo hace de forma análoga a las matrices con los números separados por ***blancos*** o ***comas***.
- **Ejemplo:**

```
>> x=[10 20 30] % vector fila
x =
10 20 30
```

- **Vector Columna x**
  - Los números están separados por ***intros*** o ***puntos y coma*** (;).
- **Ejemplo:**

```
>> y=[11; 12; 13] % vector columna
y =
11
12
13
```

26

Cont... Operaciones con Matrices y Vectores

- **Ejemplo:** si se intenta sumar los vectores **x** e **y** se obtendrá el siguiente mensaje de error:

```
>> x+y
??? Error using ==> +
Matrix dimensions must agree.
```

- Estas dificultades desaparecen si se suma **x** con el vector transpuesto de **y**:

```
>> x+y'
ans =
21 32 43
```

- MATLAB considera ***vectores fila por defecto***, como se ve en el ejemplo siguiente:

```
>> x(1)=1, x(2)=2
x =
1   20  30
x =
1   2   30
```

27

Cont... Operaciones con Matrices y Vectores

- **Operaciones con matrices**
  - Se puede operar con matrices por medio de:
    - **Operadores:** *suma* (+), *producto* (*) y *traspuesta* (')
    - **Funciones:** *invertir* ***inv( )***
  - Los operadores matriciales son:
    - + adición o suma
    - – sustracción o resta
    - * multiplicación
    - ' traspuesta
    - ^ potenciación
    - \ división-izquierda
    - / división-derecha
    - .* producto elemento a elemento
    - ./ y .\ división elemento a elemento
    - .^ elevar a una potencia elemento a elemento
  - Los operadores anteriores se pueden aplicar también de modo ***mixto***, es decir con un operando escalar y otro matricial.

28

Cont... Operaciones con Matrices y Vectores

- **Ejemplo:**

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

>> A*2
ans =
2 4
6 8

>> A-4
ans =
-3 -2
-1 0
```

- Se puede utilizar el ***operador de división*** / para dividir por un escalar todos los elementos de una matriz o un vector.

Sistemas de Ecuaciones Lineales

- **Ejemplo:**
- **Operador división-izquierda**
  - Considérese el siguiente sistema de ecuaciones lineales, **Ax** = **b** (1)
  - donde **x** y **b** son vectores columna, y **A** una matriz cuadrada invertible.
- **Resolución 1**
  x = inv(**A**)***b**
- **Resolución 2**
  x = **A\b**
  El operador ***división-izquierda*** por una matriz (\) equivale a pre-multiplicar por la inversa de esa matriz.
- **Ejemplo:**
- Considérese el siguiente ejemplo de matriz (1×2) que conduce a un sistema de infinitas soluciones.

```
>> A=[1 2], b=[2]
A =
1 2
b =
2

>> x=A\b
x =
0
1
```


Cont... Sistemas de Ecuaciones Lineales

- **Ejemplo:**
  - Sistema de tres ecuaciones formadas por una recta que no pasa por el origen y los dos ejes de coordenadas:

```
>> A=[1 2; 1 0; 0 1], b=[2 0 0]'
A =
1 2
1 0
0 1
b =
2
0
0
```

```
>> x=A\b, resto=A*x-b
x =
0.3333
0.6667
resto =
-0.3333
0.3333
0.6667
```

- La "inteligencia" del operador barra invertida \ tiene un coste: MATLAB debe de emplear cierto tiempo en determinar las características de la matriz: triangular, simétrica, etc.

Cont... Sistemas de Ecuaciones Lineales

- **Ejemplo:**
  - **Operador División-derecha (/)**
  - Aunque no es una forma demasiado habitual, también se puede escribir un sistema de ecuaciones lineales en la forma correspondiente a la transpuesta de la ecuación (1):

    yB = **c** (3)
  - donde **y** y **c** son vectores fila (**c** conocido).
  - Si la matriz **B** es cuadrada e invertible, la solución de este sistema se puede escribir en las formas siguientes:

    y = **c***inv(**B**) (4a)

    y = **c**/**B** (4b)
  - Este operador (/) equivale a postmultiplicar por la inversa de la matriz.
  - Si se traspone la ecuación (3) y se halla la solución aplicando el operador *división-izquierda* se obtiene:

    y' = (**B'**)**c'** (5)
  - Comparando las expresiones (4b) y (5) se obtiene la relación entre los operadores *división-izquierda* y *división-derecha* (MATLAB sólo tiene implementado el operador *división-izquierda*):

    c/**B** = ((**B'**)**c'**)' (6)

Cont... Sistemas de Ecuaciones Lineales

- **Operadores elemento a elemento**
  - Se puede aplicar elemento a elemento los operadores matriciales (*, ^, \ y /).
  - Para ello basta precederlos por un punto (.).

- **Ejemplo 1:**

```
>> [1 2 3 4]^2
??? Error using ==> ^
Matrix must be square.

>> [1 2 3 4].^2
ans =
1 4 9 16
```

- **Ejemplo 2:**

```
>> [1 2 3 4]*[1 -1 1 -1]
??? Error using ==> *
Inner matrix dimensions must agree.

>> [1 2 3 4].*[1 -1 1 -1]
ans =
1 -2 3 -4
```


Ejercicio de tipos Datos

- Supongamos que:
  - El estudiante de nombre **Pedro Vélez**, de **18** años, entró a la Universidad en 2001.
  - El desea registrarse en la materia de código **FIEC04341**, en el paralelo **4**.
  - Le niegan el registro pues dicen que debe **$140.35** a la Universidad,
  - El estudiante afirma que esto es completamente **falso**.

Cont... Ejercicio de tipos Datos

- En el enunciado anterior podemos identificar los datos

Pedro Velez	->	Nombre	*Texto*
24	->	Edad	*Numérico*
1998	->	Año	*Numérico*
FIEC04341	->	Código Materia	*Texto*
4	->	Paralelo	*Numérico*
FALSO	->	Es Deudor?	*Lógico*

Tipos de datos

- MATLAB es un programa preparado para trabajar con vectores y matrices, pero también trabaja con variables escalares (matrices de dimensión 1).
- MATLAB trabaja siempre en ***doble precisión***, es decir guardando cada dato en 8 bytes, con unas 15 cifras decimales exactas.
- También puede trabajar con:
  - Cadenas de caracteres (*strings*)
  - Otros tipos de datos:
    - ***Matrices de más dos dimensiones***,
    - ***matrices dispersas***,
    - ***vectores y matrices de celdas***,
    - ***estructuras*** y
    - ***clases y objetos***.

Cont... Tipos de datos

- **Números reales de doble precisión**
  - Los elementos de los vectores y las matrices son numeros reales almacenados en 8 bytes (53 bits para la mantisa y 11 para el exponente de 2; entre 15 y 16 cifras decimales equivalentes).
  - MATLAB mantiene una forma especial para los *números muy grandes* (más grandes que los que es capaz de representar), que son considerados como ***infinito***.

Ejemplo:
El ***infinito*** se representa como ***inf*** ó ***Inf***.
Obsérvese cómo responde el programa al ejecutar el siguiente comando:
>> 1.0/0.0
Warning: Divide by zero
ans =
Inf

Ejemplo:
Los resultados que no están definidos como números se representan con **NaN (Not a Number)**.
Ejecútense los siguientes comandos y obsérvense las respuestas obtenidas:
>> 0/0
Warning: Divide by zero ans =
NaN

>> inf/inf
ans =
NaN

37

Cont... Tipos de datos

- **Operaciones de Coma Flotante**
  - MATLAB dispone de tres funciones (no tienen argumentos) relacionadas con estas operaciones.
    - **Eps:** devuelve la diferencia entre 1.0 y el número de coma flotante inmediatamente superior. Da una idea de la precisión o número de cifras almacenadas. En un PC, ***eps*** vale 2.2204e-016.
    - **Realmin:** devuelve el número más pequeño con que se puede trabajar (2.2251e-308)
    - **Realmax:** devuelve el número más grande con que se puede trabajar (1.7977e+308)

38

Cont... Tipos de datos

- **Otros tipos de variables:**
  - Integer,
  - Float y
  - Logical
- Por defecto MATLAB trabaja con variables de punto flotante y doble precisión (double).
  - Con estas variables pueden resolverse casi todos los problemas prácticos y con frecuencia no es necesario complicarse la vida declarando variables de tipos distintos, como se hace con cualquier otro lenguaje de programación.
- En algunos casos es conveniente declarar variables de otros tipos porque puede ahorrarse mucha memoria y pueden hacerse los cálculos mucho más rápidamente.

39

Cont... Tipos de datos

- **Números Enteros**
  - MATLAB permite crear variables enteras con 1, 2, 4 y 8 bytes (8, 16, 32 y 64 bits).
  - Estas variables pueden tener signo o no tenerlo.
  - Las variables con signo representan números en intervalos "casi" simétricos respecto al 0.
    - Los tipos son: **int8**, **int16**, **int32** e **int64**
  - Las variables sin signo representan número no negativos, desde el 0 al número máximo.
    - Los tipos son **uint8, uint16, uint32 y uint64.**
  - Para crear una variable entera de un tipo determinado se pueden utilizar sentencias como las siguientes:
    >> **i=int32(100)**; % se crea un entero de 4 bytes con valor 100
    >> j=zeros(100); **i=int32(j)**; % se crea un entero i a partir de j
    >> **i=zeros(1000,1000,'int32')**; % se crea una mariz 1000x1000 de enteros

40

Cont... Tipos de datos

- **Ejemplo:**
  - Las funciones **intmin('int64')** e **intmax('int64')** permiten saber el valor del entero más pequeño y más grande (en valor algebraico) que puede formarse con variables enteras de 64 bits:

```
>> disp([intmin('int64'), intmax('int64')])
-9223372036854775808 9223372036854775807
```

- La función **isinteger(i)** devuelve 1 si la variable i es entera y 0 en otro caso.
- La función **class(i)** devuelve el tipo de variable que es **i** (**int8**, **int16**, ...).
- La función **isa(i, 'int16')** permite saber exactamente si la variable **i** corresponde a un entero de 16 bits.

Cont... Tipos de datos

- **Números Reales**
  - MATLAB dispone de dos tipos de variables reales o *float*: **single** (4 bytes) y **double** (8 bytes).
    - Por defecto se utilizan ***doubles***.
  - Las funciones **single(x)** y **double(y)** permiten realizar conversiones entre ambos tipos de variables.
  - Las funciones **realmin** y **realmax** permiten saber los números ***double*** más pequeño y más grande (en valor absoluto) que admite el computador.
  - Para los números de simple precisión habría que utilizar **realmin('single')** y **realmax('single')**.
  - La función **isfloat(x)** permite saber si **x** es una variable real, de simple o doble precisión.
  - Las funciones **isa(x, 'single')** ó **isa(x, 'double')** permiten saber exactamente de qué tipo de variable se trata.

- **Ejemplo:**
- Uso de variables ***single*** para reducir el tiempo de CPU y la memoria:

```
>> n=1000; AA=rand(n);
   A=single(AA);

>> tic, Bs=inv(A); toc
Elapsed time is 1.985000
   seconds.

>> tic, Bd=inv(AA); toc
Elapsed time is 4.296000
   seconds.
```


Cont... Tipos de datos

- **Variables lógicas**
  - Sólo pueden tomar los valores ***true*** (1) y ***false*** (0).
  - Surgen como resultado de los operadores relacionales (==, <, <=, >, >=, ~=) y de muchas funciones lógicas como **any** y **all** que se aplican a vectores y matrices.
  - La función **logical(A)** produce una variable lógica, con el mismo número de elementos que **A**, con valores 1 ó 0 según el correspondiente elementos de **A** sea distinto de cero o igual a cero.

Cont... Tipos de datos

- **Ejemplo:**
  Una de las aplicaciones más importantes de las variables lógicas es para separar o extraer los elementos de una matriz o vector que cumplen cierta condición, y operar luego selectivamente sobre dichos elementos.

```
>> A=magic(4)
A =
16 2 3 13
5 11 10 8
9 7 6 12
4 14 15 1
```

```
>> j=A>10
j =
1 0 0 1
0 1 0 0
0 0 0 1
0 1 1 0
>> isa(j,'logical')
ans =
1

>> A(j)=-10
A =
-10 2 3 -10
5 -10 10 8
9 7 6 -10
4 -10 -10 1
```

Cont... Tipos de datos
Operaciones Lógicas

Operador	Significado	Ejemplo	Tipo de resultado	Resultado
No(not)	Negación de un valor	No(6>10)	Entero o real	Verdadero
Y(and)	Conjunción	(1<5) y (5>10) Ecuador clasificó y Colombia no clasifico	Entero o real	Falso Verdadero
O(or)	Disyunción	(5>10) o (10<9)	Entero o real	Falso

Cont... Tipos de datos

- **Números complejos: función complex**
  - En muchos cálculos matriciales los datos y/o los resultados no son reales sino ***complejos***, pero MATLAB trabaja sin ninguna dificultad con ellos.

- **Ejemplo:**
- Vemos como se representan por defecto los números complejos:

```
>> a=sqrt(-4)
a =
0 + 2.0000i

>> 3 + 4j
ans =
3.0000 + 4.0000i
```

- En la entrada de datos de MATLAB se pueden utilizar indistintamente la **i** y la **j** para representar el *número imaginario unidad* (en la salida, sin embargo, puede verse que siempre aparece la **i**).
- Si la **i** o la **j** no están definidas como variables, puede intercalarse el signo (*).

- **Ejemplo:**
- Cuando se está trabajando con números complejos, conviene no utilizar la **i** como variable ordinaria, pues puede dar lugar a errores y confusiones.
- Obsérvense los siguientes resultados:

```
>> i=2
i =
2

>> 2+3i
ans =
2.0000 + 3.0000i

>> 2+3*i
ans =
8

>> 2+3*j
ans =
2.0000 + 3.0000i
```


Cont... Tipos de datos

- **Asignación de *valores complejos* a vectores y matrices**
  - Se puede hacerse de las dos formas.

Ejemplo:
Conviene hacer antes ***clear i***, para que **i** no esté definida como variable.

```
>> A = [1+2i  2+3i; -1+1i  2-3i]
A =
1.0000 + 2.0000i   2.0000 + 3.0000i
-1.0000 + 1.0000i  2.0000 - 3.0000i
```

Ejemplo:
Es posible definir las partes reales e imaginarias por separado usando el operador (*).

```
>> A = [1 2; -1 2] + [2 3; 1 -3]*i  % En este caso el * es necesario
A =
1.0000 + 2.0000i   2.0000 + 3.0000i
-1.0000 + 1.0000i  2.0000 - 3.0000i
```

Ejemplo:
Crear un número complejo a partir de dos argumentos que representan la parte real e imaginaria:

```
>> complex(1,2)
ans =
1.0000 + 2.0000i
```

- Es importante advertir que el *operador de matriz traspuesta* ('), aplicado a matrices complejas, produce la ***matriz conjugada y traspuesta***.
- Existe una función que permite hallar la matriz conjugada (***conj( )***) y el operador punto y apóstrofo (**.'**) que calcula simplemente la matriz traspuesta.

Cont... Tipos de datos

- **Cadenas de Caracteres**
  - MATLAB puede definir variables que contengan cadenas de caracteres.
  - Las cadenas de texto van entre apóstrofos o comillas simples (Nótese que en C van entre comillas dobles: "cadena").

  - **Ejemplo:**
    s = 'cadena de caracteres'

Cont... Tipos de datos

- **Variables y expresiones matriciales**
  - Una ***variable*** es un nombre que se da a una entidad numérica, que puede ser una matriz, un vector o un escalar.
    - El valor de esa variable, e incluso el tipo de entidad numérica que representa, puede cambiar a lo largo de una sesión de MATLAB o a lo largo de la ejecución de un programa.
  - Una expresión de MATLAB puede tener dos formas:
    1. Asignando su resultado a una variable:
       - variable = expresión
    2. Evaluando simplemente el resultado del siguiente modo:
       - expresión
         - El resultado se asigna a la variable interna ***ans*** (de *answer*) que almacena el último resultado.

Cont... Tipos de datos

- **Ejemplo de variables:**
  - El valor del radio, y el valor del Area, pueden cambiar
  - Una variable es
    - Un dato cuyo valor puede cambiar durante un cálculo, o
    - En la resolución de un problema.
  - Ejemplo:
    - El lado del cuadrado, para calcular perímetro o área del mismo.
    - El código de una materia, el número de cédula de una persona, etc.
  - Las variables pueden ser de cualquier tipo de dato.
  - La computadora representa a las variables como
    - Una porción de memoria.

Cont... Tipos de datos

- Por defecto una expresión termina cuando se pulsa ***intro***.
  - Si se desea que continúe en la línea siguiente, se debe introducir ***tres puntos*** (...) antes de pulsar ***intro***.
- Se pueden incluir varias expresiones en una misma línea separándolas por:
  - *comas* (,) o
  - *puntos y comas* (;) - su resultado se calcula, pero no se escribe en pantalla.
- Evita la escritura de resultados intermedios y de grandes cantidades de números para matrices de gran tamaño.

Cont... Tipos de datos

- **Reglas para escribir variables:**
  - Debe tener un nombre que la identifique:
    - Lado, nombre, radio, area, etc.
  - **MATLAB distingue entre mayúsculas y minúsculas** los nombres de variables.
  - **Los nombres** deben empezar siempre por una letra
    - **Bien:** lado, total
    - **Mal:** 89lado, 1total, *nombre.
  - Pueden constar de hasta 63 letras y números (función ***namelengthmax)***.
  - El carácter guión bajo (_) se considera como una letra.
  - No pueden contener ningún otro carácter especial
    - **Bien:** lado_cuadrado, total1, total2
    - **Mal:** lado cuadrado, total$, total#
  - A diferencia del C, no hace falta declarar las variables que se vayan a utilizar.
    - Se deba tener especial cuidado con no utilizar nombres erróneos en las variables, porque no se recibirá ningún aviso del ordenador.

Cont... Tipos de datos

- El comando **clear** tiene varias formas posibles:

clear	sin argumentos, ***clear*** elimina todas las variables creadas previamente (excepto las variables globales).
clear A, b	borra las variables indicadas.
clear global	borra las variables globales.
clear functions	borra las funciones.
clear all	borra todas las variables, incluyendo las globales, y las funciones.

Cont... Tipos de datos

- **Constantes**
  - Pi es una constante
  - Su valor esta establecido, y no varía
  - Ejemplo
    - El perímetro de un cuadrado es **4** veces el valor cualquiera de sus lados
  - Las constantes, no tienen porque solo ser números, pueden ser datos de todo tipo.
    - 'a', 'casa' son constantes de tipo carácter y cadena

Operador de Asignación

- Una variable puede cambiar su valor
- Se debe efectuar una asignación
  - El valor se asigna a la variable
- Usamos el operador de asignación(=)
- El formato para asignar un valor a una variable
  - Nombre de la variable = expresion

- **Ejemplo:**
  - A = 4
  - B = 8
  - B = A+B
  - A = A+1
  - B = B+3
- La asignación es de izq. A derecha
- Si la variable tenía otro valor, este se pierde
  - A = 5
  - A = A*5

Tipos de Matrices Predefinidos

eye(4)	forma la matriz unidad de tamaño (4×4)
zeros(3,5)	forma una matriz de *ceros* de tamaño (3×5)
zeros(4)	idem de tamaño (4×4)
ones(3)	forma una matriz de *unos* de tamaño (3×3)
ones(2,4)	idem de tamaño (2×4)
linspace(x1,x2,n)	genera un vector con **n** valores igualmente espaciados entre **x1** y **x2**
logspace(d1,d2,n)	genera un vector con **n** valores espaciados logarítmicamente entre 10^d1 y 10^d2. Si **d2** es **pi9**, los puntos se generan entre 10^d1 y **pi**
rand(3)	forma una matriz de números aleatorios entre 0 y 1, con distribución uniforme, de tamaño (3×3)
rand(2,5)	idem de tamaño (2×5)
randn(4)	forma una matriz de números aleatorios de tamaño (4×4), con distribución normal, de valor medio 0 y varianza 1.
magic(4)	crea una matriz (4×4) con los números 1, 2, ... 4*4, con la propiedad de que todas las filas y columnas suman lo mismo
hilb(5)	crea una matriz de Hilbert de tamaño (5×5). La matriz de Hilbert es una matriz cuyos elementos (i,j) responden a la expresión (1/(i+j-1)). Esta es una matriz especialmente difícil de manejar por los grandes errores numéricos a los que conduce
invhilb(5)	crea directamente la inversa de la matriz de Hilbert
kron(x,y)	produce una matriz con todos los productos de los elementos del vector **x** por los elementos del vector **y**. Equivalente a **x'*y**, donde **x** e **y** son vectores fila
compan(pol)	construye una matriz cuyo polinomio característico tiene como coeficientes los elementos del vector **pol** (ordenados de mayor grado a menor)
vander(v)	construye la matriz de ***Vandermonde*** a partir del vector **v** (las columnas son las potencias de los elementos de dicho vector)

Formación de una Matriz a Partir de Otras

- MATLAB ofrece la posibilidad de crear una matriz a partir de matrices previas ya definidas, por varios posibles caminos:
  - recibiendo alguna de sus propiedades (como por ejemplo el tamaño),
  - por composición de varias submatrices más pequeñas,
  - modificándola de alguna forma.
- Un caso especialmente interesante es el de crear una nueva matriz ***componiendo como submatrices*** otras matrices definidas previamente.
- **Ejemplo:**

```
>> A=rand(3)
>> B=diag(diag(A))
>> C=[A, eye(3); zeros(3), B]
```

  - la matriz **C** de tamaño **(6×6)** se forma por composición de cuatro matrices de tamaño (3×3).
  - Los tamaños de las submatrices deben de ser coherentes.

Cont... Formación de una Matriz a Partir de Otras

[m,n]=size(A)	devuelve el número de filas y de columnas de la matriz **A. Si** la matriz es cuadrada basta recoger el primer valor de retorno
n=length(x)	calcula el número de elementos de un vector **x**
zeros(size(A))	forma una matriz de *ceros* del mismo tamaño que una matriz **A previamente creada**
ones(size(A))	ídem con *unos*
A=diag(x)	forma una matriz diagonal **A cuyos elementos diagonales son los elementos de un vector ya existente x**
x=diag(A)	forma un vector **x a partir de los elementos de la diagonal de una matriz ya existente A**
diag(diag(A))	crea una matriz diagonal a partir de la diagonal de la matriz **A**
blkdiag(A,B)	crea una matriz diagonal de submatrices a partir de las matrices que se le pasan como argumentos
triu(A)	forma una matriz triangular superior a partir de una matriz **A (no tiene por qué ser cuadrada). Con un segundo argumento puede controlarse que se mantengan o eliminen más diagonales por encima o debajo de la diagonal principal.**
tril(A)	ídem con una matriz triangular inferior
rot90(A,k)	Gira k*90 grados la matriz rectangular **A en sentido antihorario. k es un entero que puede ser negativo. Si se omite, se supone k=1**
flipud(A)	halla la matriz simétrica de **A respecto de un eje horizontal**
fliplr(A)	halla la matriz simétrica de **A respecto de un eje vertical**
reshape(A,m,n)	Cambia el tamaño de la matriz **A devolviendo una matriz de tamaño** *m*×*n* cuyas columnas se obtienen a partir de un vector formado por las columnas de **A puestas una a continuación de otra. Si** la matriz **A tiene menos de** *m*×*n* elementos se produce un error.

Direccionamiento de Vectores y Matrices a partir de Vectores

- Los elementos de un vector **x** se pueden direccionar a partir de los de otro vector **v**.
- En este caso, **x**(**v**) equivale al vector **x(v(1))**, **x(v(2))**, ... Considérese el siguiente ejemplo:

```
>> v=[1 3 4]
v =
1 3 4

>> x=rand(1,6)
x =
0.5899    0.4987   0.7351   0.9231   0.1449   0.9719

>> x(v)
ans =
0.5899    0.7351   0.9231
```


Cont... Direccionamiento de Vectores y Matrices a partir de Vectores

- **Ejemplo:**
- De forma análoga, los elementos de una matriz **A** pueden direccionarse a partir de los elementos de dos vectores **f** y **c**.

```
>> f=[2 4]; c=[1 2];
>> A=magic(4)
A =
16 2 3 13
5 11 10 8
9 7 6 12
4 14 15 1

>> A(f,c)
ans =
5 11
4 14
```

```
>> A(f,c)
ans =
5 11
4 14
```

- **Ejemplo:** continuación del anterior
- Comprobar cómo los elementos de una matriz se pueden direccionar con un sólo índice, considerando que las columnas de la matriz están una a continuación de otra formando un vector:

```
>> f=[1 3 5 7];
>> A(f), A(5), A(6)
ans =
16 9 2 7
ans =
2
ans =
11
```

Operador dos Puntos (:)

- Puede usarse de varias formas.
- Para empezar, defínase un vector **x** con el siguiente comando:

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

- Se podría decir que el operador (:) representa un ***rango***:
  - En este caso, los números enteros entre el 1 y el 10.
  - El incremento por defecto es 1,
- Este operador puede también utilizarse con otros valores enteros y reales, positivos o negativos.
  - En este caso el incremento va entre el valor inferior y el superior.

Operador dos Puntos (:)

- **Ejemplo:**

```
>> x=1:2:10
x =
1 3 5 7 9

>> x=1:1.5:10
x =
1.0000 2.5000 4.0000 5.5000 7.0000 8.5000 10.0000

>> x=10:-1:1
x =
10 9 8 7 6 5 4 3 2 1
```

- Por defecto, este operador produce vectores fila.
- Si se desea obtener un vector columna basta trasponer el resultado.

- **Ejemplo:**
  - Generar una tabla de funciones *seno* y *coseno*.
    - Recuérdese que con (;) después de un comando el resultado no aparece en pantalla.

```
>> x=[0.0:pi/50:2*pi]';
>> y=sin(x); z=cos(x);
>> [x y z]
```


Cont... Operador dos Puntos (:)

- **Ejemplo:**

```
>> A=magic(6)
A =
35 1 6 26 19 24
3 32 7 21 23 25
31 9 2 22 27 20
8 28 33 17 10 15
30 5 34 12 14 16
4 36 29 13 18 11
```

- MATLAB accede a los elementos de una matriz por medio de los índices de fila y de columna.

```
>> A(2,3)
ans =
7
```

- **Ejemplo:**
  - Extraer los 4 primeros elementos de la 6ª fila:

```
>> A(6, 1:4)
ans =
4 36 29 13
```


Cont... Operador dos Puntos (:)

- **Ejemplo:**
  - Extraer todos los elementos de la 3ª fila:

```
>> A(3, :)
ans =
31 9 2 22 27 20
```

- **Ejemplo:**
  - Para acceder a la última fila o columna puede utilizarse la palabra ***end***, en lugar del número correspondiente.
  - Extraer la sexta fila (la última) de la matriz:

```
>> A(end, :)
ans =
4 36 29 13 18 11
```

- **Ejemplo:**
  - Extraer todos los elementos de las filas 3, 4 y 5:

```
>> A(3:5,:)
ans =
31 9 2 22 27 20
8 28 33 17 10 15
30 5 34 12 14 16
```

- **Ejemplo:**
  - Se puede extraer conjuntos disjuntos de filas utilizando *corchetes* [ ]
  - Extraer las filas 1, 2 y 5:

```
>> A([1 2 5],:)
ans =
35 1 6 26 19 24
3 32 7 21 23 25
30 5 34 12 14 16
```

Cont... Operador dos Puntos (:)

- Todo lo que se dice para filas vale para columnas y viceversa:
  - Basta cambiar el orden de los índices.
- El operador dos puntos (:) puede utilizarse en ambos lados del operador (=).

- **Ejemplo:**
- Definir una matriz identidad **B** de tamaño 6×6 y reemplazar filas de **B** por filas de **A**.
  - La siguiente secuencia de comandos sustituye las filas 2, 4 y 5 de **B** por las filas 1, 2 y 3 de **A**,

```
>> B=eye(size(A));
>> B([2 4 5],:)=A(1:3,:)
B =
1 0 0 0 0 0
35 1 6 26 19 24
0 0 1 0 0 0
3 32 7 21 23 25
31 9 2 22 27 20
0 0 0 0 0 1
```


Cont... Operador dos Puntos (:)

- Se pueden realizar operaciones aún más complicadas:

```
>> B=eye(size(A));
>> B(1:2,:)=[0 1; 1 0]*B(1:2,:)
```

- **Ejemplo:**
- Como invertir el orden de los elementos de un vector:

```
>> x=rand(1,5)
x =
0.9103 0.7622 0.2625 0.0475 0.7361
>> x=x(5:-1:1)
x =
0.7361 0.0475 0.2625 0.7622 0.9103
```

- Por haber utilizado paréntesis –en vez de corchetes– los valores generados por el operador (:) afectan a los índices del vector y no al valor de sus elementos.

- **Ejemplo:**
- Para invertir el orden de las columnas de una matriz se puede hacer lo siguiente:

```
>> A=magic(3)
A =
8 1 6
3 5 7
4 9 2
>> A(:,3:-1:1)
ans =
6 1 8
7 5 3
2 9 4
```


Definición de Vectores y Matrices a Partir de un Fichero

- MATLAB acepta como entrada un fichero **nombre.m** que contiene instrucciones y/o funciones.
  - Dicho fichero se llama desde la línea de comandos tecleando su nombre, sin la extensión.
- Un fichero ***.m** puede llamar a otros ficheros ***.m**, y puede llamarse a sí mismo (funciones recursivas).
- Las variables definidas dentro de un fichero de comandos ***.m** que se ejecuta desde la línea de comandos son variables del **espacio de trabajo base**, esto es, pueden ser accedidas desde fuera de dicho fichero; no sucede lo mismo si el fichero ***.m** corresponde a una función.
- Si un fichero de comandos se llama desde una función, las variables que se crean pertenecen al espacio de trabajo de dicha función.
- **Ejemplo:**
  - Crear un fichero llamado **unidad.m** que construya una matriz unidad de tamaño 3×3 llamada **U33** en un directorio llamado **c:\matlab**. Este fichero deberá contener la línea siguiente:

```
U33=eye(3)
```

  - Desde MATLAB llámese al comando **unidad** y obsérvese el resultado.
  - Entre otras razones, es muy importante utilizar ficheros de comandos para poder utilizar el **Debugger** y para evitar teclear muchas veces los mismos datos, sentencias o expresiones.

Definición de Vectores y Matrices Mediante Funciones y Declaraciones

- Se pueden definir las matrices y vectores por medio de
  - **funciones de librería** y
  - **funciones programadas por el usuario**
- **Operadores relacionales**
  - El lenguaje de programación de MATLAB dispone de los siguientes operadores relacionales:

<	menor que
>	mayor que
<=	menor o igual que
>=	mayor o igual que
==	igual que
~=	distinto que

- En MATLAB los operadores relacionales pueden aplicarse a vectores y matrices.
- Al igual que en C, si una comparación se cumple el resultado es 1 (***true***), si no se cumple es 0 (***false***).
- Recíprocamente, cualquier valor distinto de cero es considerado como ***true*** y el cero equivale a ***false***.

- La diferencia con C está en que cuando los operadores relacionales de MATLAB se aplican a dos matrices o vectores del mismo tamaño, **la comparación se realiza elemento a elemento**, y **el resultado es otra matriz de unos y ceros del mismo tamaño**, que recoge el resultado de cada comparación entre elementos.

- **Ejemplo:**

```
>> A=[1 2;0 3]; B=[4 2;1 5];
>> A==B
ans =
0 1
0 0
>> A~=B
ans =
1 0
1 1
```

Cont... Definición de Vectores y Matrices Mediante Funciones y Declaraciones

- **Operadores lógicos**
  - Los operadores lógicos de MATLAB son los siguientes:
  - Los operadores lógicos se combinan con los relacionales para poder comprobar el cumplimiento de condiciones múltiples.
  - Los ***operadores lógicos breves*** (&&) y (||) se utilizan para simplificar las operaciones de comparación evitando operaciones innecesarias, pero también para evitar ciertos errores que se producirían en caso de evaluar incondicionalmente el segundo argumento.
- **Ejemplo:**
- Sentencia que evita una división por cero

```
r = (b~=0) && (a/b>0);
```

&	and (función equivalente: **and(A,B)**). Se evalúan siempre ambos operandos, y el resultado es **true** sólo si ambos son **true**.
&&	and breve: si el primer operando es **false** ya no se evalúa el segundo, pues el resultado final ya no puede ser más que **false**.
|	or (función equivalente: **or(A,B)**). Se evalúan siempre ambos operandos, y el resultado es **false** sólo si ambos son **false**.
||	or breve: si el primer operando es **true** ya no se evalúa el segundo, pues el resultado final no puede ser más que **true**.
~	negación lógica (función equivalente: **not(A)**)
xor(A,B)	realiza un "or exclusivo", es decir, devuelve 0 en el caso en que ambos sean 1 ó ambos sean 0.

Funciones de Librería

- MATLAB tiene un gran número de funciones incorporadas.
- Algunas son **funciones intrínsecas:** funciones incorporadas en el propio código ejecutable del programa.
  - Estas funciones son particularmente rápidas y eficientes.
- Existen además funciones definidas en ficheros ***.m** y ***.mex12** que vienen con el propio programa o que han sido aportadas por usuarios del mismo.
- Dispone también de ficheros ***.p**, que son los ficheros ***.m** pre-compilados con la función **pcode**.
- Para que MATLAB encuentre una determinada función de usuario el correspondiente fichero-M debe estar en el **directorio actual** o en uno de los directorios del **search path**.

Características generales de las funciones

- Al igual que en C:
  - Una función tiene **nombre**, **valor de retorno** y **argumentos**.
  - Una función se llama utilizando su nombre en una expresión o utilizándolo como un comando más.
- Las funciones se pueden definir en ficheros de texto ***.m**.
- **Ejemplos de llamada a funciones:**

```
>> [maximo, posmax] = max(x);
>> r = sqrt(x^2+y^2) + eps;
```

```
>>Alfa=0.3
>> a = cos(alfa) - sin(alfa);
```

  - Se han usado algunas funciones matemáticas como el cálculo del valor máximo, el seno, el coseno y la raíz cuadrada.
  - Los **nombres** de las funciones se han puesto en negrita.
- Los **argumentos** de cada función van a continuación del nombre entre paréntesis (y separados por comas si hay más de uno).

Cont... Características generales de las funciones

- **Valores de retorno** son el resultado de la función y sustituyen a ésta en la expresión donde la función aparece.
  - Las funciones pueden tener **valores de retorno matriciales**, como en el primero de los ejemplos anteriores.
- En este caso se calcula el elemento de máximo valor en un vector, y se devuelven dos valores: el valor máximo y la posición que ocupa en el vector.
  - Los 2 valores de retorno **se recogen entre corchetes**, separados por comas.
- En MATLAB las funciones que no tienen argumentos no llevan paréntesis, por lo que a simple vista no siempre son fáciles de distinguir de las simples variables:
  - Ejemplo la función **eps**, que devuelve la diferencia entre 1.0 y el número de coma flotante inmediatamente superior.

Cont... Características generales de las funciones

- Los nombres de las funciones de MATLAB **no son palabras reservadas** del lenguaje.
  - Es posible crear una variable llamada **sin** o **cos**, que ocultan las funciones correspondientes.
- Para poder acceder a las funciones hay que eliminar (**clear**) las variables del mismo nombre que las ocultan, o bien haber definido previamente una **referencia a función** (function handle).
- MATLAB permite que una función tenga un **número variable de argumentos y valores de retorno**.

Funciones Internas

- Hay operaciones complejas
- En ocasiones, los operadores no son suficientes
- Una función es
  - Una expresión
  - Toma un numero n de argumentos
  - Efectúa una o varias operaciones sobre los mismos
  - Devuelve un resultado
- La sintaxis de uso es la siguiente:
  - Variable = **nombre_funcion**(argumento1, argumento2, ...)
- Todos los lenguajes tienen un conjunto de funciones en común

Cont... Funciones Internas

Función	Operación	Argumentos	Resultado	Ejemplo
abs(x)	Valor Absoluto de x	X es numérico	Igual que el argumento	X = -9 R = abs(X) R tiene ahora 9
arctan(x)	Arco tangente de x	X es numérico	Retorna un real	
cos(x)	Coseno de x	X es numérico	Retorna un real	
exp(x)	Exponencial de x	X es numérico	Retorna un real	
ln(x)	Logaritmo neperiano de x	X es numérico	Retorna un real	
log10(x)	Logaritmo decimal de x	X es numérico	Retorna un real	
round(x)	Redondeo de x	X es real	Retorna un entero	X = 9.56 R = round(x) R tiene ahora 10
sen(x)	Seno de x	X es numérico	Retorna un real	

Equivalencia entre comandos y funciones

- Existe una equivalencia entre las funciones y los comandos con argumentos de MATLAB. Así, un comando en la forma,
  >> comando arg1 arg2
- es equivalente a una función con el mismo nombre que el comando a la que los argumentos se le pasan como cadenas de caracteres,
  >> comando('arg1', 'arg2')
- Esta dualidad entre comandos y funciones es sobre todo útil en programación, porque permite "construir" los argumentos con las operaciones propias de las cadenas de caracteres.

Funciones matemáticas elementales que operan de modo escalar

sin(x)	seno
cos(x)	coseno t
an(x)	tangente
asin(x)	arco seno
acos(x)	arco coseno
atan(x)	arco tangente (devuelve un ángulo entre -π/2 y +π/2)
atan2(x)	arco tangente (devuelve un ángulo entre -π y +π), se le pasan 2 argumentos, proporcionales al seno y al coseno
sinh(x)	seno hiperbólico
cosh(x)	coseno hiperbólico
tanh(x)	tangente hiperbólica
asinh(x)	arco seno hiperbólico
acosh(x)	arco coseno hiperbólico
atanh(x)	arco tangente hiperbólica
log(x)	logaritmo natural
log10(x)	logaritmo decimal
exp(x)	función exponencial

sqrt(x)	raíz cuadrada
sign(x)	devuelve -1 si <0, 0 si =0 y 1 si >0. Aplicada a un número complejo, devuelve un vector unitario en la misma dirección
rem(x,y)	resto de la división (2 argumentos que no tienen que ser enteros)
mod(x,y)	similar a **rem (Ver diferencias con el Help)**
round(x)	redondeo hacia el entero más próximo
fix(x)	redondea hacia el entero más próximo a 0
floor(x)	valor entero más próximo hacia -∞
ceil(x)	valor entero más próximo hacia +∞
gcd(x)	máximo común divisor
lcm(x)	mínimo común múltiplo
real(x)	partes reales
imag(x)	partes imaginarias
abs(x)	valores absolutos
angle(x)	ángulos de fase

77

Funciones que actúan sobre vectores

- No sobre matrices
- Cuando se aplican sobre matrices **se aplican por separado a cada columna de la matriz**, dando como valor de retorno un vector resultado de aplicar la función a cada columna de la matriz considerada como vector.
  - Si estas funciones se quieren aplicar a las filas de la matriz basta aplicar dichas funciones a la matriz transpuesta.

[xm,im]=max(x)	máximo elemento de un vector. Devuelve el valor máximo **xm y la posición que ocupa im**
min(x)	mínimo elemento de un vector. Devuelve el valor mínimo y la posición que ocupa
sum(x)	suma de los elementos de un vector
cumsum(x)	devuelve el vector suma acumulativa de los elementos de un vector (cada elemento del resultado es una suma de elementos del original)
mean(x)	valor medio de los elementos de un vector
std(x)	desviación típica
prod(x)	producto de los elementos de un vector
cumprod(x)	devuelve el vector producto acumulativo de los elementos de un vector
[y,i]=sort(x)	ordenación de menor a mayor de los elementos de un vector **x. Devuelve el vector ordenado y, y un vector i con las posiciones iniciales en x de los elementos en el vector ordenado y.**

78

Determinación de la fecha y la hora

Clock	devuelve un vector fila de seis elementos que representan el **año, el mes, el día, la hora, los minutos y los segundos, según el reloj interno del computador. Los cinco primeros son valores enteros, pero la cifra correspondiente a los segundos contiene información hasta las milésimas de segundo.**
Now	devuelve un número (**serial date number) que contiene toda la información de la fecha y hora actual. Se utiliza como argumento de otras funciones.**
Date	devuelve la fecha actual como cadena de caracteres (por ejemplo: 24-Aug-2004).
datestr(t)	convierte el serial date number **t en cadena de caracteres con el** día, mes, año, hora, minutos y segundos. Ver en los manuales on-line los formatos de cadena admitidos.
datenum()	convierte una cadena ('mes-día-año') o un conjunto de seis números (año, mes, día, horas, minutos, segundos) en serial date number.
datevec()	convierte serial date numbers o cadenas de caracteres en el vector de seis elementos que representa la fecha y la hora.
calendar()	devuelve una matriz 6×7 con el calendario del mes actual, o del mes y año que se especifique como argumento.
weekday(t)	devuelve el día de la semana para un serial date number **t.**

79

Otros Tipos de Datos de Matlab

- MATLAB puede también trabajar con otros tipos de datos:
  1. **Conjuntos o cadenas de caracteres**, fundamentales en cualquier lenguaje de programación.
  2. **Hipermatrices**, o matrices de más de dos dimensiones.
  3. **Estructuras**, o agrupaciones bajo un mismo nombre de datos de naturaleza diferente.
  4. **Vectores o matrices de celdas** (cell arrays), que son vectores o matrices cuyos elementos pueden ser cualquier otro tipo de dato.
  5. **Matrices dispersas**, que son matrices que pueden ser de muy gran tamaño con la mayor parte de sus elementos cero.

80

Cadenas de caracteres

- MATLAB trabaja con **cadenas de caracteres**, con ciertas semejanzas y diferencias respecto a C/C++ y Java.
- Las funciones para cadenas de caracteres están en el sub-directorio **toolbox\matlab\strfun** del directorio en que esté instalado MATLAB.
- Los caracteres de una cadena se almacenan en un vector, con un caracter por elemento.
  - Cada carácter ocupa dos bytes.
  - Las cadenas de caracteres van entre **apóstrofos** o **comillas simples**, como por ejemplo: 'cadena'.
  - Si la cadena debe contener comillas, éstas se representan por un doble carácter comilla, de modo que se pueden distinguir fácilmente del principio y final de la cadena.
    - **Por ejemplo,** para escribir la cadena **ni 'idea'** se escribiría **'ni''idea'''**.
- Una **matriz de caracteres** es una matriz cuyos elementos son caracteres, o bien una matriz cuyas filas son cadenas de caracteres.
  - Todas las filas de una **matriz de caracteres** deben tener el **mismo número de elementos**.
    - Si es preciso, las cadenas (filas) más cortas se completan con blancos.

81

Cont... Cadenas de caracteres

- **Ejemplos:**

```
>> c='cadena'
c =
cadena

>> size(c)                          % dimensiones del array
ans =
     1     6

>> double(c)                        % convierte en números ASCII cada carácter ans =
    99    97   100   101   110    97

>> char(abs(c))                     % convierte números ASCII en caracteres ans =
cadena

>> cc=char('más','madera')          % convierte dos cadenas en una matriz
cc =
más madera

>> size(cc)                         % se han añadido tres espacios a 'más'
ans =
     2     6
```

82

Cont... Cadenas de caracteres

funciones más importantes para manejo de cadenas de caracteres

double(c)	convierte en números ASCII cada carácter
char(v)	convierte un vector de números **v en una cadena de caracteres**
char(c1,c2)	crea una matriz de caracteres, completando con blancos las cadenas más cortas
deblank(c)	elimina los blancos al final de una cadena de caracteres
disp(c)	imprime el texto contenido en la variable **c**
ischar(c)	detecta si una variable es una cadena de caracteres
isletter()	detecta si un carácter es una letra del alfabeto. Si se le pasa un vector o matriz de caracteres devuelve un vector o matriz de unos y ceros
isspace()	detecta si un carácter es un espacio en blanco. Si se le pasa un vector o matriz de caracteres devuelve un vector o matriz de unos y ceros
strcmp(c1,c2)	comparación de cadenas. Si las cadenas son iguales devuelve un uno, y si no lo son, devuelve un cero (funciona de modo diferente que la correspondiente función de C)
strcmpi(c1,c2)	igual que **strcmp(c1,c2), pero ignorando la diferencia entre mayúsculas y minúsculas**
strncmp(c1,c2,n)	compara los **n primeros caracteres de dos cadenas**
c1==c2	compara dos cadenas carácter a carácter. Devuelve un vector o matriz de unos y ceros

83

Cont... Cadenas de caracteres

funciones más importantes para manejo de cadenas de caracteres

s=[s,' y más']	concatena cadenas, añadiendo la segunda a continuación de la primera
findstr(c1,c2)	devuelve un vector con las posiciones iniciales de todas las veces en que la cadena más corta aparece en la más larga
strmatch(cc,c)	devuelve los índices de todos los elementos de la matriz de caracteres (o vector de celdas) **cc, que empiezan por la cadena c**
strrep(c1,c2,c3)	sustituye la cadena **c2 por c3, cada vez que c2 es encontrada en c1**
[p,r]=strtok(t)	separa las palabras de una cadena de caracteres **t. Devuelve la primera palabra p y el resto de la cadena r**
int2str(v)	convierte un número entero en cadena de caracteres
num2str(x,n)	convierte un número real **x en su expresión por medio de una cadena de caracteres, con cuatro cifras decimales por defecto (pueden especificarse más cifras, con un argumento opcional n)**
str2double(str)	convierte una cadena de caracteres representando un número real en el número real correspondiente
vc=cellstr(cc)	convierte una matriz de caracteres **cc en un vector de celdas vc, eliminando los blancos adicionales al final de cada cadena. La función char() realiza las conversiones opuestas**
sprintf	convierte valores numéricos en cadenas de caracteres, de acuerdo con las reglas y formatos de conversión del lenguaje C.

84

Cont... Cadenas de caracteres
Uso de funciones

- **Ejemplo:**

  >> num2str(pi) % el resultado es una cadena de caracteres, no un número ans =

  3.142

  >> num2str(pi,8)

  ans =

  3.1415927

- **Ejemplo:**
  - Es habitual convertir los valores numéricos en cadenas de caracteres para poder imprimirlos como títulos en los dibujos o gráficos.

  >> fahr=70; grd=(fahr-32)/1.8;

  >> title(['Temperatura ambiente: ',num2str(grd),' grados centígrados'])

Hipermatrices (arrays de más de dos dimensiones)

- Una posible aplicación es almacenar con un único nombre distintas matrices del mismo tamaño (resulta una hipermatriz de 3 dimensiones).
- Los elementos de una hipermatriz pueden ser números, caracteres, estructuras, y vectores o matrices de celdas.
- El tercer subíndice representa la tercera dimensión: la "profundidad" de la hipermatriz.
- Las funciones para trabajar con estas hipermatrices están en el sub-directorio **toolbox\matlab\datatypes**.

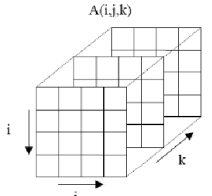


Figura 27. Hipermatriz de tres dimensiones.

Cont... Hipermatrices

- **Ejemplo:**
  - Las funciones que operan con matrices de más de dos dimensiones son análogas a las funciones vistas previamente, aunque con algunas diferencias.
  - Las siguientes sentencias generan, en dos pasos, una matriz de 2×3×2:

```
>> AA(:,:,1)=[1 2 3; 4 5 6] % matriz inicial
AA =
   1     2     3
   4     5     6

>> AA(:,:,2)=[2 3 4; 5 6 7] % se añade una segunda matriz
AA(:,:,1) =
   1     2     3
   4     5     6
AA(:,:,2) =
   2     3     4
   5     6     7
```


Cont... Hipermatrices
Uso de funciones

- Algunas funciones para generar matrices admiten más de dos subíndices y pueden ser utilizadas para generar hipermatrices.
- Entre ellas están **rand()**, **randn()**, **zeros()** y **ones()**.
- **Ejemplo:**

```
>> BB=randn(2,3,2)
BB(:,:,1) =
-0.4326    0.1253   -1.1465
-1.6656    0.2877    1.1909
BB(:,:,2) =
 1.1892    0.3273   -0.1867
-0.0376    0.1746    0.7258
```

- **Ejemplo:**
- La función **cat()** permite concatenar matrices según las distintas "dimensiones".

```
>> A=zeros(2,3); B=ones(2,3);
i j
k
A(i,j,k)
>> cat(1,A,B)
ans =
0   0   0
0   0   0
1   1   1
1   1   1
>> cat(2,A,B)
ans =
0   0   0   1   1   1
0   0   0   1   1   1
>> cat(3,A,B)
ans(:,:,1) =
0   0   0
0   0   0
ans(:,:,2) =
1   1   1
1   1   1
```

Cont... Hipermatrices
Uso de funciones

- Las siguientes funciones de MATLAB se pueden emplear también con hipermatrices:

size()	devuelve tres o más valores (el nº de elementos en cada dimensión)
ndims()	devuelve el número de dimensiones
squeeze()	elimina las dimensiones que son igual a uno
reshape()	distribuye el mismo número de elementos en una matriz con distinta forma o con distintas dimensiones
permute(A,v)	permuta las dimensiones de **A según los índices del vector v**
ipermute(A,v)	realiza la permutación inversa

- Para el resto de las funciones de MATLAB, se pueden establecer reglas para aplicar a hipermatrices:
  1. Todas las funciones de MATLAB que operan sobre escalares (**sin()**, **cos()**, etc.) se aplican sobre hipermatrices elemento a elemento.
     - Las operaciones con escalares también se aplican de la misma manera.
  2. Las funciones que operan sobre vectores (**sum()**, **max()**, etc.) se aplican a matrices e hipermatrices según la primera dimensión, resultando un array de una dimensión inferior.
  3. Las funciones matriciales propias del Álgebra Lineal (**det()**, **inv()**, etc.) no se pueden aplicar a hipermatrices.
     - Para poderlas aplicarlas se debe extraer primero las matrices correspondientes (por ejemplo, con el operador dos puntos (:)).

Estructuras

- **Struct:** es una agrupación de datos de tipo diferente bajo un mismo nombre.
- Estos datos se llaman **miembros** (members) o **campos** (fields).
- Una estructura es un nuevo tipo de dato, del que luego se pueden crear muchas variables (**objetos** o instances).
- **Ejemplo:**
- La estructura **alumno** puede contener los campos:
  - **nombre** (una cadena de caracteres) y
  - **carnet** (un número)

- En MATLAB la estructura **alumno** se crea creando un objeto de dicha estructura.
  - A diferencia de otros lenguajes de programación, no hace falta definir previamente el modelo o patrón de la estructura.
- **Ejemplo:**
  - Una posible forma de hacerlo es crear uno a uno los distintos campos

```
>> alu.nombre='Miguel'
alu =
    nombre: 'Miguel'

>> alu.carnet=75482
alu =
    nombre: 'Miguel'
    carnet: 75482

>> alu alu =
    nombre: 'Miguel'
    carnet: 75482
```


Cont... Estructuras

- Se accede a los miembros o campos de una estructura por medio del **operador punto** (.), que une el nombre de la estructura y el nombre del campo (por ejemplo: **alu.nombre**).
- **Ejemplo:**
  - Puede crearse la estructura por medio de la función **struct().**

```
>> alu = struct('nombre', 'Ignacio', 'carnet', 76589)
alu =
    nombre: 'Ignacio'
    carnet: 76589
```

- Los **nombres de los campos** se pasan a la función **struct()** entre apóstrofos ('), seguidos del valor que se les quiere dar. Este valor puede ser la cadena vacía ('') o la matriz vacía ([]).

Cont... Estructuras

- **Ejemplo:**
  - Pueden crearse vectores y matrices (e hipermatrices) de estructuras.
  - Crear un vector de 10 elementos cada uno de los cuales es una estructura tipo **alumno**.

```
>> alum(10) = struct('nombre', 'Ignacio', 'carnet', 76589)
```

  - Sólo el elemento 10 del vector es inicializado con los argumentos de la f**unción struct()**
  - El resto de los campos se inicializan con una cadena vacía o una matriz vacía.
- **Ejemplo:**
  - Para dar valor a los campos de los elementos restantes se puede utilizar un bucle **for** con sentencias del tipo:

```
>> alum(i).nombre='Noelia', alum(i).carnet=77524;
```

Cont... Estructuras

- **Ejemplo:**
  - MATLAB permite añadir un nuevo campo a una estructura en cualquier momento.
  - La siguiente sentencia añade el campo **edad** a todos los elementos del vector **alum**, aunque sólo se da valor al campo del elemento 5:

    >> alum(5).edad=18;
  - Para ver el campo **edad** en los 10 elementos del vector puede teclearse el comando:

    >> alum.edad

Cont... Estructuras

Uso de funciones

fieldnames()	devuelve un vector de celdas con cadenas de caracteres que recogen los nombres de los campos de una estructura
isfield(ST,s)	permite saber si la cadena **s es un campo de una estructura ST**
isstruct(ST)	permite saber si **ST es o no una estructura**
rmfield(ST,s)	elimina el campo **s de la estructura ST**
getfield(ST,s)	devuelve el valor del campo especificado. Si la estructura es un array hay que pasarle los índices como cell array (entre llaves {}) como segundo argumento ↵Esta forma de crear arrays de estructuras da error si la estructura ha sido previamente declarada **global.**
setfield(ST,s,v)	da el valor **v al campo s de** la estructura **ST. Si** la estructura es un array, hay que pasarle los índices como cell array (entre llaves {}) como segundo argumento

Cont... Estructuras

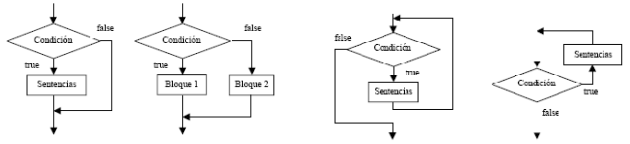
- **Estructuras anidadas:** una estructura con campos que sean otras estructuras.
  - Para acceder a los campos de la estructura más interna se utiliza dos veces el operador punto (.)
- **Ejemplo:**
  - en este la estructura **clase** contiene un campo que es un vector **alum** de alumnos

```
>> clase=struct('curso','primero','grupo','A',...
'alum', struct('nombre','Juan', 'edad', 19))
clase =
      curso: 'primero'
grupo: 'A'
      alum: [1x1 struct]
```

```
>> clase.alum(2).nombre='María';
>> clase.alum(2).edad=17;
>> clase.alum(2)
ans =
      nombre: 'María'
      edad: 17
>> clase.alum(1)
ans =
      nombre: 'Juan'
      edad: 19
```


Programación en MATLAB

- Matlab dispone de las siguiente estructuras:



- Muchos lenguajes de programación disponen de bucles con control al principio (***for*** y ***while*** en C/C++/Java) y al final (***do ... while*** en C/C++/Java).
- En MATLAB no hay bucles con control al final del bucle, es decir, no existe construcción análoga a ***do ... while***.

Cont... Programación en MATLAB

- **Sentencia if**
  - **Bifurcación simple:**
    - A diferencia de C/C++/Java: la condición no va entre paréntesis, aunque se pueden poner si se desea

```
if condicion
   Sentencias
end
```

- La condición del if puede ser una condición matricial, del tipo A==B, donde A y B son matrices del mismo tamaño.
  - Para que se considere que la condición se cumple, es necesario que sean iguales dos a dos todos los elementos de las matrices A y B (aij=bij, 1≤i≤m, 1≤j≤n).

- **Bifurcación múltiple:**
  - Pueden concatenarse tantas condiciones como se desee

```
if condicion1
   bloque1
elseif condicion2
   bloque2
elseif condicion3
   bloque3
else % opción por defecto para cuando no se cumplan las condiciones 1,2,3
   bloque4
End
```


Cont... Programación en MATLAB
Uso de la Sentencia if

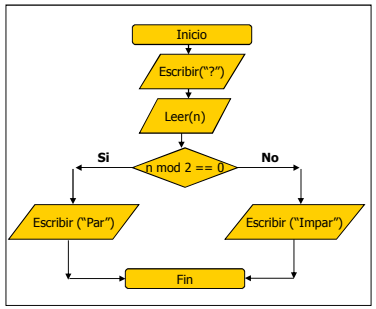
```
if condicion
   Sentencia 1;
   Sentencia 2;
   ...
end
```

```
if condicion
   Sentencia 1;
   Sentencia 2;
   ...
else
   Sentencia 1;
   Sentencia 2;
   ...
end
```

```
If condicion
    Sentencia 1;
    Sentencia 2;
    ...
elseif condicion2
    Sentencia 1;
    Sentencia 2;
    ...
elseif condicion3
    Sentencia 1;
    Sentencia 2;
    ...
else
    Sentencia 1;
    Sentencia 2;
    ...
end
```


Cont... Programación en MATLAB
Ejercicio

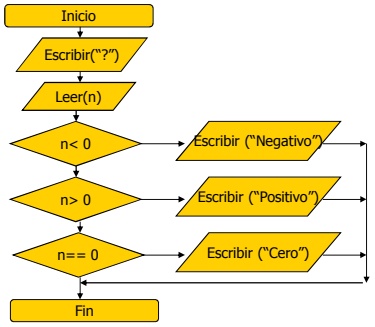
- Se desea determinar si un número entero, ingresado por teclado es par.




```
n=input('Ingrese un numero:');
if mod(n,2) == 0
   fprintf('%d es par', n);
else
   fprintf( '%d es impar', n);
end
```


Cont... Programación en MATLAB
Ejercicio

- Se desea determinar si un número ingresado por teclado es positivo, negativo, o nulo.




```
n = input('Ingrese numero:');
if n> 0
   fprintf( '%d es positivo', n);
elseif n < 0
   fprintf( '%d es negativo', n);
elseif n ==0
   fprintf( '%d es cero', n);
end
```

Otras maneras de resolver el ejercicio…

```
n = input('Ingrese numero:');
  if n> 0
     fprintf( '%d es positivo', n);
  else
        if n < 0
           fprintf( '%d es negativo', n);
        else
           if n ==0
              fprintf( '%d es cero', n);
           end
        end
  end
```

```
n = input('Ingrese numero:');
  if n ==0
     fprintf( '%d es cero', n);
  elseif n < 0
     fprintf( '%d es negativo', n);
  else
     fprintf( '%d es positivo', n);
  end
```


Cont… Programación en MATLAB
Ejercicio en clase

- Realice un programa que le permita determinar si un año es bisiesto o no.
- Las condiciones para que un año sea bisiesto son:
  - Si es divisible para 400 o,
  - Si es divisible para 4, excepto los que terminan en 00(divisibles para 100).
- Ejemplos:
  - El año 1988
    - Es divisible para 4
    - No termina en 00
    - Es año bisiesto.
  - El año 1900
    - Es divisible para 100
    - No es divisible para 400: NO BISIESTO
  - El año 2000
    - Es divisible para 400: BISIESTO

Cont… Programación en MATLAB
Solución

```
fprintf('Programa para determinar si un año es bisiesto\n');
y = input('Ingrese el año:');
if (mod(y,4) == 0 & mod(y,100)~=0) | (mod(y,400) == 0)
   fprintf('%d es bisiesto\n', y);
else
   fprintf('%d no es bisiesto\n', y);
end
```


Cont… Programación en MATLAB
Ejercicio en clase

- Escriba un programa que dadas dos notas (parcial y final), indique si el alumno aprueba o no la materia (aprueba con un promedio de 60).
  - Solo si no aprueba, el programa debe
    - Pedir una tercera nota (mejoramiento) e indicar si ahora aprueba o no.
    - Solo si aprueba debe mostrarse el promedio.

Dos maneras de resolverlo…

```
n=input('Ingrese nota parcial:'),
n1=input('Ingrese nota final:'),
Promedio=(n+n1)/2,
if Promedio>=60
   fprintf('Aprobado'),
elseif Promedio<60
       fprintf('Reprobado'),
       n2=input('Ingrese nota de mejoramiento:'),
      if n>=n1
          Promedio=(n+n2)/2,
      else
          Promedio=(n1+n2)/2,
      end
      if Promedio>=60
          fprintf('Aprobado'),
      end
end
```

```
n=input('Ingrese nota parcial:'),
n1=input('Ingrese nota final:'),
Promedio=(n+n1)/2,
if Promedio>=60
   fprintf('Aprobado'),
else
       fprintf('Reprobado'),
       n2=input('Ingrese nota de mejoramiento:'),
      if n>=n1
          Promedio=(n+n2)/2,
      else
          Promedio=(n1+n2)/2,
      end
      if Promedio>=60
          fprintf('Aprobado'),
      end
end
```


Cont… Programación en MATLAB

- Sentencia switch
  - Su forma general es la siguiente:

```
switch switch_expresion case case_expr1, bloque1
case {case_expr2, case_expr3, case_expr4,...}
   bloque2
...
otherwise, % opción por defecto
   bloque3
End
```

- A diferencia de C/C++/Java, en MATLAB sólo se ejecuta uno de los bloques relacionado con un case.

Cont… Programación en MATLAB

- **Sentencia for**
  - Repite un conjunto de sentencias un número predeterminado de veces.
  - En MATLAB es muy diferente y no tiene la generalidad de la sentencia **for** de C/C++/Java. La siguiente construcción ejecuta sentencias con valores de **i** de **1** a **n**, variando de uno en uno.

```
for i=1:n
   sentencias
end
```

o bien,

```
for i=vectorValores
   sentencias
end
```

```
for indice=expr
    sentencia1;
    sentencia2;
    ...
end
```

- donde **vectorValores** es un vector con los distintos valores que tomará la variable **i**.
- **Ejemplo:**
  - Caso más general para la variable del bucle (valor_inicial: incremento: valor_final).
  - El bucle se ejecuta por primera vez con **i=n**, y luego **i** se va reduciendo de 0.2 en 0.2 hasta que llega a ser menor que 1, en cuyo caso el bucle se termina:

```
for i=n:-0.2:1
   sentencias
end
```


Cont… Programación en MATLAB

- **Ejemplo:**
  - Presenta una estructura correspondiente a dos **bucles anidados**.
  - La variable **j** es la que varía más rápidamente (por cada valor de **i**, **j** toma todos sus posibles valores):

```
for i=1:m
   for j=1:n
     sentencias
   end
end
```

- **Ejemplo:**
  - Otra forma del bucle **for** (**A** es una matriz):

```
for i=A
   sentencias
End
```

  - variable **i** es un vector que va tomando en cada iteración el valor de una de las columnas de **A**.
  - Cuando se introducen interactivamente en la línea de comandos, los bucles **for** se ejecutan sólo después de introducir la sentencia **end** que los completa.

Cont... Programación en MATLAB

- **Ejercicios en clase**
  - Escriba un programa que reciba un texto y lo imprima n veces en pantalla.
    - tex = input('Ingrese un texto:','s');
    - n=input('Ingrese el numero de repeticiones:');
    - for i=1:n
    - fprintf(tex),
    - fprintf('\n'),
    - end
  - Escriba un programa que dado un número n imprima todos los números impares del 1 hasta n.

Cont... Programación en MATLAB

- **Ejercicios en clase**
  - Escriba un programa que calcule el factorial de un número.
- Resolución
  - f=1;
  - n=input('Ingrese numero:');
  - if n<0
  - n=input('Ingrese un numero entero mayor a cero:'),
  - end
  - if (n==0)||(n==1)
  - fprintf('El factorial del numero es 1');
  - else
  - for i=1:n;
  - f=f*i;
  - end
  - fprintf('%d es el factoria del numero %d ', f, n);
  - end

Cont... Programación en MATLAB

- **Sentencia while**
  - La estructura del bucle **while** es muy similar a la de C/C++/Java.
  - Su sintaxis es la siguiente:

```
while condicion
   sentencias
End
```

  - donde **condicion** puede ser una expresión vectorial o matricial.
- Las sentencias se siguen ejecutando mientras haya elementos distintos de cero en **condicion**, es decir, mientras haya algún o algunos elementos **true**.
- El bucle se termina cuando todos los elementos de **condicion** son **false** (es decir, cero).
- **Sentencia break**
  - Al igual que en C/C++/Java, la sentencia **break** hace que se termine la ejecución del bucle **for** y/o **while** más interno de los que comprenden a dicha sentencia.
- **Sentencia continue**
  - La sentencia **continue** hace que se pase inmediatamente a la siguiente iteración del bucle **for** o **while**, saltando todas las sentencias que hay entre el **continue** y el fin del bucle en la iteración actual.

Cont... Programación en MATLAB

- **Sentencias try...Catch...End**
  - La construcción **try...catch...end** permite gestionar los errores que se pueden producir en tiempo de ejecución.
  - Su forma es la siguiente:

```
try
   sentencias1
catch
   sentencias2
End
```

  - En el caso de que durante la ejecución del bloque **sentencias1** se produzca un error, el control de la ejecución se transfiere al bloque **sentencias2**.
  - Si la ejecución transcurriera normalmente, **sentencias2** no se ejecutaría nunca.
- MATLAB dispone de una función **lasterr** que devuelve una cadena de caracteres con el mensaje correspondiente al último error que se ha producido.
  - En la forma **lasterr('')** pone a cero este contador de errores, y hace que la función **lasterr** devuelva la matriz vacía [] hasta que se produzca un nuevo error.

Condiciones

- Las condiciones se construyen usando:
  - Operadores lógicos
    - AND (&)
    - OR (|)
    - NOT (~)
  - Operadores relacionales (<, >=, >, >=, ==, ~=)
- No confundir
  - El operador de asignación = , usado para asignar un valor a una variable.
  - Con el operador de equivalencia ==, para comparar dos valores

```
if(x = 0)
```
Operación de asignación, asigna el valor 0 a la variable x. No compara.

```
if(x == 0)
```
Operación relacional, se pregunta si x es igual a 0

Evaluación de Condiciones

- Las expresiones lógicas:
  - Son evaluadas de izquierda a derecha
  - En MATLAB pueden tomar valores de 0 y 1.
    - 0 cuando la condición es falsa.
    - 1 cuando la condición es verdadera.
  - La evaluación termina cuando se puede deducir el resultado. Ejemplo: **A) exp1 & exp2** **B) exp1 | exp2**
    - En A): Si exp1 es falso en el caso entonces toda la expresión es falsa.
    - En B): Si exp1 es verdadero, toda la expresión es verdadera.
  - Escriba una condición para la siguiente expresión
    - Que **y** sea divisible para **x** y que **x** sea diferente de 0.

(mod(y,x) == 0) & (x~=0)

Cont... Programación en MATLAB

- **Ejercicio en Clase:**
  - Escriba un programa que dado un número n, indique para cuantos números es divisible.
    - Añada una condición para saber si el numero n es o no primo (solo es divisible para 1 y para n).

Cont... Programación en MATLAB

- **Función input**
  - Imprime un mensaje en la línea de comandos y recuperar como valor de retorno un valor numérico o el resultado de una expresión tecleada por el usuario.
  - El usuario puede teclear simplemente un vector o una matriz.
- **Ejemplo:**

>> n = input('Teclee el número de ecuaciones')

  - Otra posible forma de esta función es la siguiente (obsérvese el parámetro 's')

>> nombre = input('¿Cómo te llamas?','s')

  - En este caso el texto tecleado como respuesta se lee y se devuelve sin evaluar, con lo que se almacena en la cadena **nombre**. Así pues, en este caso, si se teclea una fórmula, se almacena como texto sin evaluarse.

Cont... Programación en MATLAB

- **Función disp**
  - Permite imprimir en pantalla un mensaje de texto o el valor de una matriz, pero sin imprimir su nombre.
  - Siempre imprime vectores y/o matrices: las cadenas de caracteres son un caso particular de vectores.
- **Ejemplos:**

```
>> disp('El programa ha terminado')
>> A=rand(4,4)
>> disp(A)
```


Caracteres especiales comunes usados en la función **fprintf**

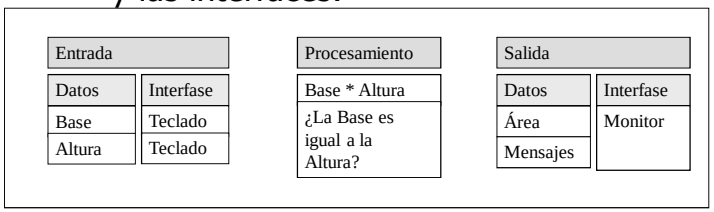
Formato	Resultado
%d	Muestra un valor como entero.
%e	Muestra un valor en formato exponencial.
%f	Muestra un valor con formato de punto flotante.
%g	Muestra un valor con punto flotante o formato exponencial, dependiendo de cual sea más corto.
\n	Posiciona el cursor en la siguiente línea.

Ejercicios en clase

- Escriba un programa en MATLAB para calcular el área de un círculo.
- Dados 2 puntos en el eje cartesiano (x1,y1) y (x2,y2), escriba un programa en MATLAB que calcule la distancia entre los puntos.
- Dados la base y la altura de un polígono, determinar el área del mismo e indicar si se trata de un rectángulo o un cuadrado.

Un Problema con Cálculos

- Definamos los datos de entrada, salida y las interfaces:



OPERACIONES MATEMATICAS

Precedencia

Operad.	Significado	Operando	Ejemplo	Resultado	Ejemplo
+	Suma	Entero o real	4+2 4.35+ 2 4.35 + 2.5	Entero o real	6 6.35 6.85
-	Resta	Entero o real	4-2 4.35- 2 4.35 - 2.5	Entero o real	2 2.35 1.85
Div	División entera	Entero	10 div 6	Entero	1
Mod	Modulo	Entero	10 mod 6	Entero	4
*	Multiplicación	Entero o real	4*2 4.35* 2 4.35 * 2.5	Entero o real	8 8.7 10.875
/	División	Real	4.00/2.00 4.35/ 2.5	Real	2.00 1.74
^	Exponenciación	Entero o real	4^2 4.5 ^ 2 4.5 ^ 0.5	Entero o real	16 20.25 2.121

121

OPERACIONES RELACIONALES

Las operaciones lógicas usualmente se combinan con las operaciones relacionales
Las operaciones relacionales SIEMPRE dan como resultado un valor LÓGICO

Operador	Significado	Ejemplo	Resultado
<	Menor que	El total de estudiantes de la Estatal es menor que el de la Politécnica 4<10	Falso Verdadero
>	Mayor que	El área del Campus Prosperina es mayor que la del Campus Peñas 8>10	Verdadero Falso
==	Igual que	El nombre del autor de 100 años de soledad es igual que el de Crónica de una muerte Anunciada	Verdadero
<=	Mayor o igual que	6 <= 10	Verdadero
>=	Menor o igual que	10 >= 8	Falso
~=	Diferente de	5 ~= 5 El precio de una entrada en el Albocine es diferente de el de las entradas en el Cinemark	Falso Verdadero

122

EJERCICIO

- Calcule el área de un círculo.
- Que puede variar en el problema?.
- Que es fijo?.

123

Contar y Acumular

- Al usar repeticiones de acciones
  - Muchas veces se necesita contar cuantas veces se repitieron ciertas acciones o ir sumando nuevos valores en cada repetición.
- Para esto se usan variables enteras
  - Que se incrementan o decrementan con un valor fijo (contadores).
    - c = c + 1;
  - O que se incrementen o decrementen con un valor variable.
    - t = t + v;
- La sentencia que efectúa el incremento se coloca dentro de un lazo.
  - Para que la variable en cuestión aumente.

124

Tarea

- Escriba un programa, tal que dado un valor n entero positivo, calcule y muestre los elementos correspondientes a la conjetura de Ullman, que consiste en lo siguiente:
  - Empiece con cualquier entero positivo
  - Si es par, divídalo para 2
  - Si es impar, multiplíquelo por 3 y agréguele 1
  - Obtenga enteros sucesivamente repitiendo el proceso
- Al final se obtendrá el número 1, independientemente del entero inicial.
- Por ejemplo, cuando el entero es 52, la secuencia será:
  - `52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1`

125

Plotting

- La función en Matlab que permite crear gráficos de funciones es:
  - plot(x,y)
- El comando plot(x,y) permite graficar los elementos del vector x en el eje x y los elementos del vector y en el eje y.

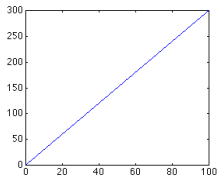
126

Ejemplo

- Si quisiéramos graficar la función: y=3x. Escribimos un programa en Matlab que tenga las siguientes sentencias:

```
x = 0:0.1:100;
y = 3*x;
plot(x,y);
```

- Genera el siguiente gráfico:



127

Plotting

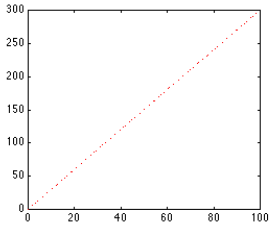
El tercer parámetro de la función plot puede tener de 1 a 3 caracteres que especifican el color y/o el tipo point marker que se usará para realizar el gráfico.

Colores	Point marker
y amarillo	. point
m magenta	o circle
c cyan	x x-mark
r rojo	+ símbolo mas
g verde	- línea sólida
b blue	* star
w white	: dotted

128

Ejemplo

```
x = 0:0.1:100;
y = 3*x;
plot(x,y,'r:')
```

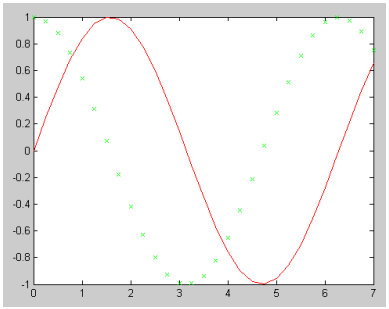


Graficando más de una función en la ventana de Gráficos de Matlab

- Usar la función hold.
  - hold on: retiene el gráfico actual en la ventana de gráficos de Matlab.
  - hold off: Es el valor por defecto, hace que se realice un nuevo gráfico y borra el(los) anterior(es).

Ejemplo

- Grafique la funciones seno y coseno en función del tiempo.




```
t=0:0.25:7;
y = sin(t);
plot(t,y,'r');
z = cos(t);
hold on
plot(t,z,'gx');
hold off
```


Etiquetas para los gráficos

- title('Función seno de x');
- xlabel('abscisa x');
- ylabel('sen(x)');

- Ver funcioncuadratica.n

Tarea

Suponga que una bola es lanzada a una altura ho sobre la superficie de la tierra, con una velocidad inicial Vo, la posición y la velocidad de la bola como función del tiempo está dada por las siguientes ecuaciones:

- $h(t) = (1/2)gt^2 + Vot + ho$
- $V(t) = gt + Vo$

Donde g es la aceleración debida a la gravedad (-9.81 m/s^2), h es la altura sobre la superficie de la tierra (asumiendo que no hay fricción en el aire), y V es el componente vertical de la velocidad.

Escriba un programa en MATLAB que pida al usuario la altura inicial de la bola en metros y la velocidad inicial de la bola en metros por segundo, y permita al usuario seleccionar si desea ver el gráfico de altura en función del tiempo o el de velocidad en función del tiempo. Incluir las etiquetas apropiadas en los gráficos.

133