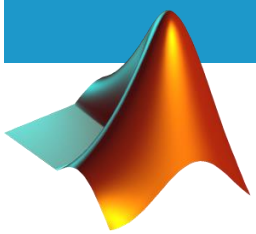
Métodos Numéricos Aplicados I

# Introducción a Matlab

# Índice

# Esquema



INTRODUCCIÓN A MATLAB

- Instrucciones básicas
- Vectores | Matrices
- Funciones anónimas
- Archivos .m
  - function | script
- Estructuras de control
  - Condicionales | Bucles
- Representaciones gráficas
- Cálculo simbólico
- Archivos .mlx

# Ideas clave

## 1.1 Introducción y objetivos

Los métodos numéricos consisten en una serie de procedimientos que nos permiten obtener soluciones aproximadas a determinados problemas, cuando no es posible obtener su solución analítica.

La potencialidad de los métodos numéricos se sustenta sobre la computación. Con las posibilidades que actualmente disponen los ordenadores de sobremesa, cualquiera puede tener a su alcance una máquina computacional. Este hecho ha supuesto que en los últimos años estos métodos hayan adquirido mayor relevancia.

A lo largo del curso trabajaremos con el software Matlab. Se trata de una herramienta matemática de cálculo numérico, aunque en sus últimas versiones también incorpora cálculo simbólico.


Accede al vídeo: ¿Qué es Matlab?

Pero antes de conocer a fondo las diferentes posibilidades de Matlab, es importante mencionar que para aprovechar al máximo su usabilidad tendrás que instalártelo. Como estudiante de UNIR, tienes acceso a una licencia para que puedas utilizarlo.


Accede al vídeo: Instalación de Matlab

Una vez instalado el software, te recomendamos que te familiarices con su interfaz gráfica para que sepas dónde está y qué contiene cada ventana. De esta manera, podrás configurar la interfaz gráfica a tu gusto.

Accede al vídeo: La interfaz gráfica

Ahora ya estamos preparados para arrancar con el uso de Matlab. Los objetivos que trataremos de alcanzar en este tema serán los siguientes:

- Conocer las instrucciones básicas
- Aprender a introducir y manejar vectores y matrices
- Definir funciones anónimas
- Distinguir entre los elementos `función` y `script`
- Conocer la sintaxis de las estructuras de control
- Aprender a realizar representaciones gráficas simples
- Identificar las instrucciones para el cálculo simbólico
- Conocer la potencialidad de los archivos `.mlx`

## 1.2 Instrucciones básicas

Situados en la consola de Matlab, conocida como `Command Window`, ejecutaremos una serie de instrucciones básicas para conocer su comportamiento.

## Comandos básicos

A continuación mostramos una serie de comandos que nos van a permitir obtener información, gestionar el `Workspace`, u otras funcionalidades.

- `help par` Muestra por pantalla la ayuda del comando `par`.
- `save par` Guarda las variables que hay en el `Workspace` dentro del archivo `par.mat`.
- `clear par` Borra la variable `par` del `Workspace`.
- `clc` Limpia la consola.
- `whos` Muestra información acerca de las variables que hay en el `Workspace`.
- `clear par` Carga en el `Workspace` la variables almacenadas en el archivo `par.mat`.

**Ejemplo 1.**

Obtengamos la ayuda del comando `save`.

```
>> help save
```

Vemos por pantalla la siguiente información.

```
save - Save workspace variables to file

    This MATLAB function saves all variables from the current
    workspace in a MATLAB formatted binary file (MAT-file)
    called filename.
```

**Ejemplo 2.**

Vamos a introducir un vector `v`, elevaremos sus elementos al cuadrado y guardaremos ambas variables en un archivo `T01Ejemplo2.mat`.

```
>> v=[1 4 7 -1 -4 3];
>> w=v.^2;
>> save T01Ejemplo2.mat
```

A continuación, eliminamos las variables, limpiamos la pantalla y volvemos a cargar las variables.

```
>> clear all
>> clc
>> load T01Ejemplo2.mat
```

Vemos cómo en el `Workspace` aparecen las variables que habíamos guardado.

## Tipos de archivos

Matlab tiene una serie de archivos con extensión propia que es necesario conocer. Si bien no los vamos a desarrollar en este punto, sí que es conveniente conocer su existencia.

- `*.m` Son ficheros que contienen texto interpretable por Matlab; en el apartado 1.5 veremos en qué consisten, diferenciando entre `function` y `script`.
- `*.mat` Son ficheros que contienen datos a volcar en el `Workspace`; en el Ejemplo 2 vimos un caso simple de uso.
- `*.fig` Son ficheros que almacenan representaciones gráficas; dedicaremos el apartado 1.7 a algunas representaciones simples.
- `*.mlx` Son ficheros que sirven para generar memorias de trabajos. En ellos, se puede utilizar texto con formato, ecuaciones, ejecuciones de comandos, ejecuciones

de la consola u otras funcionalidades; en el apartado 1.9 veremos algunos usos de estos archivos.

## Operadores

Las operaciones aritméticas elementales vienen definidas como sigue.

- `par1+par2` Suma de `par1` y `par2`
- `par1-par2` Resta `par2` a `par1`
- `par1*par2` Multiplica `par1` y `par2`
- `par1/par2` Divide `par1` por `par2`
- `par1^par2` Eleva `par1` a `par2`

Las operaciones anteriores son de aplicación sobre escalares, vectores y matrices, siempre y cuando cumplan con las reglas del álgebra en cuanto a dimensiones.

Hay otras operaciones adicionales en Matlab para el trabajo con vectores y matrices. Es necesario mencionar que deben tener las mismas dimensiones.

- `par1.*par2` Multiplica elemento a elemento `par1` y `par2`.
- `par1./par2` Divide elemento a elemento `par1` a `par2`.
- `par1.^par2` Eleva elemento a elemento `par1` a `par2`

**Ejemplo 3.**

Tenemos almacenados en `T01Ejemplo03.mat` las variables `d` y `t`, que representan la distancia y el tiempo que han dedicado 10 personas a su entrenamiento de carrera. A continuación, obtenemos el vector velocidad `v` para cada uno de los corredores.

```
>> load T01Ejemplo03.mat
>> v=d./t;
```

## Funciones

Identificamos como funciones a aquellos comandos que requieren de la introducción de un parámetro para actuar sobre él. A continuación listamos una serie de ejemplos cuyos nombres evidencian su funcionalidad e invitamos a probar.

- `sin`, `asin`, `sind`, `asind`.
- `cos`, `acos`, `cosd`, `acosd`.
- `tan`, `atan`, `atan2`, `tand`, `atand`, `atan2d`
- `log`, `log10`, `log2`

**Ejemplo 4.**

Calculemos $2^{3x-5} = 8$. Matemáticamente desarrollamos

$$\log_2(2^{3x-5}) = \log_2(8) \leftrightarrow 3x - 5 = \log_2(8) \leftrightarrow x = \frac{\log_2(8) + 5}{3}.$$

Ejecutamos en Matlab

```
>> x=(log2(8)+5)/3;
```

## Valores fundamentales

En este apartado, listamos las constantes más habituales.

- $\pi$: `pi`
- $e$: `exp(1)`
- $i$: `1i`
- Último elemento calculado sin asignar: `ans`
- $\frac{0}{0}$: `Nan`

**Ejemplo 5.**

Calculemos $e^{-i\frac{\pi}{2}}$.

Ejecutamos en Matlab

```
>> exp(-1i*pi/2);
```

## Formatos numéricos de salida

Cuando ejecutamos operaciones, en la consola nos pueden aparecer los resultados en diferentes formatos. La Tabla 1 recoge algunos ejemplos.

| Comando | Resultado |
|---|---|
| `format short` | 4 dígitos tras el punto decimal |
| `format long` | 15 dígitos tras el punto decimal |
| `format shortE` | 4 dígitos tras el punto decimal con notación científica |
| `format longE` | 15 dígitos tras el punto decimal con notación científica |
| `format shortG` | Ajusta automáticamente la presentación |
| `format bank` | 2 dígitos tras el punto decimal |
| `format rat` | Fracción entre dos enteros |

Tabla 1: Algunos formatos numéricos

Accede al vídeo: Elementos básicos

## 1.3 Vectores y matrices

Uno de los éxitos de Matlab es la facilidad con la que se pueden ejecutar operaciones con vectores y matrices.

### Vectores

Los vectores son conjuntos de valores en una dimensión. Pueden ser vectores fila

$$v = \begin{bmatrix} v_1 & v_2 & \cdots & v_n \end{bmatrix},$$

o vectores columna

$$v = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}.$$

**Introducción de vectores**

Para introducir un vector, este debe ir entre corchetes `[]`. Si se trata de un vector

- fila, separamos sus componentes por espacios o comas `,`
- columna, separamos sus componentes por punto y coma `;`

Para transformar un vector fila en un vector columna, y viceversa, podemos utilizar el operador `.'`.

**Ejemplo 6.**

Vamos a introducir el vector

$$v = \begin{bmatrix} 1 \\ 0 \\ -i \end{bmatrix},$$

a partir de un vector fila. Para ello, ejecutamos en la consola

```
>> v=[1  0  -1i].'
```

**Extracción de elementos de un vector**

Para extraer elementos de un vector hay que llamar al vector y poner la componente a extraer entre paréntesis.

**Ejemplo 7.**

Sea el vector

$$v = \begin{bmatrix} 1 \\ 0 \\ -i \end{bmatrix}.$$

Para extraer la segunda componente, ejecutamos

```
>> v(2)
```

En caso de que queramos extraer más de una componente, entre paréntesis introduciremos un vector con las componentes a extraer.

**Ejemplo 8.**

Sea el vector

$$v = \begin{bmatrix} 1 & -1 & 0 & 3 & 1.5 & 7 & 0 \end{bmatrix}.$$

Para extraer las componentes 3, 4 y 6, ejecutamos

```
>> componentes=[3 4 6];
>> v(componentes)
```

**Generación de vectores con componentes equiespaciadas**

A lo largo de la asignatura vamos a generar en incontables ocasiones vectores cuyas componentes están equiespaciadas. Para ello, disponemos de diferentes opciones, que se utilizan en función del uso que le vayamos a dar.

- `a:b` genera un vector de componentes espaciadas una unidad desde `a` hasta `b`.
- `a:h:b` genera un vector de componente espaciadas `h` desde `a` hasta `b`.
- `linspace(a,b)` genera un vector de 100 componentes equiespaciadas entre `a` y `b`.
- `linspace(a,b,n)` genera un vector de `n` componentes equiespaciadas entre `a` y `b`.

**Ejemplo 9.**

Para generar un vector

- $v = \begin{bmatrix} 1 & 2 & 3 & \cdots & 9 & 10 \end{bmatrix}$ ejecutamos

```
>> v=1:10;
```

- $v = \begin{bmatrix} 2 & 4 & 6 & 8 & 10 \end{bmatrix}$ ejecutamos

```
>> v=2:2:10;
```

- $v = \begin{bmatrix} 10 & 8 & 6 & 4 & 2 \end{bmatrix}$ ejecutamos

```
>> v=10:-2:2;
```

- $v = \begin{bmatrix} 0 & 0.01 & 0.02 & \cdots & 0.98 & 0.99 \end{bmatrix}$ ejecutamos

```
>> v=linspace(0,0.99);
```

- $v = \begin{bmatrix} 0 & 0.01 & 0.02 & \cdots & 0.99 & 1 \end{bmatrix}$ ejecutamos

```
>> v=linspace(0,1,101);
```

**Operaciones y comandos específicos con vectores**

Previamente vimos las operaciones aritméticas y las adicionales elemento a elemento para trabajar con vectores. Otras de las operaciones algebraicas disponibles son

- el producto escalar: `dot(v1,v2)`,
- el producto vectorial: `cross(v1,v2)`,
- la norma: `norm(v)`.

Asimismo, también hay otras operaciones que serán de utilidad, como

- obtener el tamaño del vector: `length(v)`,
- voltear el vector horizontalmente: `fliplr(v)`,
- voltear el vector verticalmente: `flipud(v)`.

**Ejemplo 10.**

Calculemos el producto vectorial entre los vectores $e_1 = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$ y $e_2 = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}$. Para ello, ejecutamos

```
>> e1=[1 0 0];
>> e2=[0 1 0];
>> prodVect=cross(e1,e2)
```

## Matrices

Una vez conocemos cómo trabajar con vectores, demos un paso más con las matrices.

### Introducción de matrices

Una matriz de tamaño $m \times n$ se introduce entre corchetes `[]`. Los elementos de cada una de las $m$ filas se introducen como vectores, es decir, separados por espacio o por coma `,`, mientras que para separar cada una de las $n$ filas se introduce un punto y coma `;`.

**Ejemplo 11.**

Para introducir la matriz

$$A = \begin{bmatrix} 1 & -1 & 0 \\ 0 & 2 & 5 \\ 1 & 3 & 1 \end{bmatrix}$$

ejecutamos

```
>> A=[1 -1 0; 0 2 5; 1 3 1];
```

**Extracción de elementos de una matriz**

Para extraer elementos de una matriz se sigue un procedimiento similar al de los vectores. En este caso, pondremos entre paréntesis el elemento con los subíndices separados por una coma.

**Ejemplo 12.**

Para extraer el elemento $a_{21}$ de la

$$A = \begin{bmatrix} 1 & -1 & 0 \\ 0 & 2 & 5 \\ 1 & 3 & 1 \end{bmatrix}$$

ejecutamos

```
>> A(2,1)
```

Si lo que queremos es extraer más de una componente, pondremos entre paréntesis los vectores que contienen los subíndices a extraer.

**Ejemplo 13.**

Para extraer los elementos $a_{21}$ y $a_{22}$ de la

$$A = \begin{bmatrix} 1 & -1 & 0 \\ 0 & 2 & 5 \\ 1 & 3 & 1 \end{bmatrix}$$

ejecutamos

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

En caso de que queramos acceder a todos los elementos de una fila o de una columna, podemos utilizar los dos puntos `:`.

**Ejemplo 14.**

Para extraer los elementos $a_{21}$, $a_{22}$ y $a_{23}$ de la matriz

$$A = \begin{bmatrix} 1 & -1 & 0 \\ 0 & 2 & 5 \\ 1 & 3 & 1 \end{bmatrix},$$

es decir, la segunda fila, ejecutamos

```
>> A(2,:)
```

**Matrices especiales**

Existen una serie de matrices especiales que se pueden generar de una forma más rápida utilizando una serie de comandos de Matlab que describimos a continuación.

- Matriz cuadrada de $n \times n$ ceros: `zeros(n)`
- Matriz rectangular de $m \times n$ ceros: `zeros(m,n)`
- Matriz cuadrada de $n \times n$ unos: `ones(n)`
- Matriz rectangular de $m \times n$ unos: `ones(m,n)`
- Matriz identidad de tamaño $n$: `eye(n)`

**Operaciones específicas con matrices**

Además de las operaciones aritméticas y las adicionales elemento a elemento para trabajar con matrices, también encontramos otras como las que se describen a continuación.

- Determinante: `det`
- Matriz inversa: `inv`
- Rango: `rank`
- Eliminación de Gauss-Jordan: `href`

Una de las operaciones más habituales con matrices es la de la resolución del sistema de ecuaciones lineales

$$Ax = b \leftrightarrow A^{-1}Ax = A^{-1}b \leftrightarrow x = A^{-1}b.$$

La implementación en Matlab podría ser

```
>> x=inv(A)*b;
```

El cálculo de la matriz inversa implica un coste computacional bastante elevado, aunque no seamos capaces de percibirlo. Es por ello que Matlab tiene implementado un operador que nos permite resolver este sistema de una forma más eficiente. Se trata del operador contrabarra \. Matlab nos propone que, para resolver el sistema, utilicemos la instrucción

```
>> x=A\b;
```

**Ejemplo 15.**

Vamos a resolver el sistema

$$\left.\begin{array}{rcl} x_1 + x_2 - 2x_3 & = & 4 \\ x_1 - x_2 & = & 1 \\ x_1 + 3x_3 & = & 0 \end{array}\right\},$$

por lo que tenemos un sistema $Ax = b$, donde

$$A = \begin{bmatrix} 1 & 1 & -2 \\ 1 & -1 & 0 \\ 1 & 0 & 3 \end{bmatrix}, x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}, b = \begin{bmatrix} 4 \\ 1 \\ 0 \end{bmatrix}.$$

Para ello, ejecutamos

```
>> A=[1 1 -2;1 -1 0;1 0 3];
>> b=[4 1 '0];
>> x=A\b;
```

Accede al vídeo: Vectores y matrices

## 1.4 Funciones anónimas

Cuando queremos trabajar con una función $f(x)$ podemos trabajar con funciones anónimas. Estas funciones se almacenan en el `Workspace`. Su definición comienza con una arroba `@` seguida de la variable entre paréntesis, poniendo fuera del paréntesis la expresión de $f(x)$.

**Ejemplo 16.**

Vamos a introducir la función anónima

$$f(x) = e^{-x} + \frac{x^2}{\sin(\pi x)}.$$

Para ello, ejecutamos

```
>> f=@(x) exp(-x)+x^2/sin(pi*x);
```

Para ejecutar la función $f(x)$, es suficiente con escribir el nombre de la función introduciendo entre paréntesis el escalar o el vectorial sobre el que se quiere evaluar.

**Ejemplo 17.**

Queremos evaluar la función

$$f(x) = e^{-x} + \frac{x^2}{\sin(\pi x)}$$

sobre los valores de la matriz

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 3 & 5 \\ 1 & 4 & 7 \end{bmatrix}.$$

Para ello, ejecutamos

```
>> A=[1 2 3;1 3 5;1 4 7];
>> f(A)
```

Las funciones anónimas no solo permiten un parámetro de entrada, sino que también son útiles para funciones de varias variables. En este caso, la estructura es similar, pero las variables de las que depende se definen dentro del paréntesis separadas por

comas.

**Ejemplo 18.**

Vamos a introducir la función anónima

$$f(x,y) = e^{-x}\left(\cos(\pi y) + \sin(\pi y)\right).$$

Para ello, ejecutamos

```
>> f=@(x,y) exp(-x)*(cos(pi*y)+sin(pi*y));
```

## 1.5 Archivos `.m`

Los archivos de extensión `.m` son ficheros cuyo contenido es capaz de interpretar Matlab como si fueran comandos. Estos ficheros se pueden generar con cualquier editor de texto, aunque habitualmente se hacen con el editor de Matlab. Dentro de los ficheros `.m` podemos distinguir entre `función` y `script`.

### Script

Un script es un conjunto de líneas de código que se ejecutan secuencialmente. Su funcionalidad es similar a la de la consola. La ventaja que tiene es que se puede acceder a las líneas de código sin necesidad de teclearlas en cada ocasión.

Los scripts no tienen parámetros de entrada ni de salida. Actúan con las variables del `Workspace` tal y como se encuentran al iniciar la ejecución, y las modifican si necesitan acceder a ellas.

**Ejemplo 19.**

Vamos a generar un script para resolver el sistema

$$\left.\begin{array}{rcl} x_1 + x_2 - 2x_3 & = & 4 \\ x_1 - x_2 & = & 1 \\ x_1 + 3x_3 & = & 0 \end{array}\right\}.$$

```
A=[1 1 -2; 1 -1 0;1 0 3];
b=[4 1 0]';
Ab=[A b];
sol=href(Ab);
x=sol(:,4);
```

Como este script lo tenemos guardado como un archivo `.m`, cuando queramos recordar cómo resolver el sistema de ecuaciones, podemos acceder a él.

## Función

Una función es, al igual que los scripts, un conjunto de líneas de código que se ejecutan secuencialmente.

La mayor diferencia está en la posibilidad de introducir parámetros de entrada y/o de salida. Una vez iniciada la ejecución de la función, esta trabaja con un `Workspace` local, por lo que no accede a las variables que hubiera previamente. En el caso en que tuviera parámetros de salida, las variables quedarían almacenadas en el `Workspace` una vez finalizara la ejecución de la función.

Un requisito de las funciones es que queden definidas en la primera línea. Para ello, la estructura que se sigue es

```
function [out1,out2]=nombre_funcion(in1,in2,in3)
```

Las funciones pueden tener tantos parámetros de entrada y de salida como se quiera, siempre y cuando cumplan con la estructura mencionada anteriormente.

Habitualmente, bajo la primera línea se introduce un comentario en el que se describe qué hace la función. Los comentarios en Matlab se introducen precedidos de un signo porcentual %.

**Ejemplo 20.**

Vamos a generar una función para resolver cualquier sistema de tres ecuaciones con tres incógnitas. Para ello, implementamos el siguiente programa.

```
function x=sistemaAxb(A,b)
% La función x=sistemaAxb(A,b) resuelve el sistema ...
   de ecuaciones lineales Ax=b, tomando como ...
   parámetros de entrada la matriz A y el vector ...
   b. Devuelve como salida el vector solución x
Ab=[A b];
sol=href(Ab);
x=sol(:,4);
```

Una vez implementado el programa, resolvemos el sistema de ecuaciones lineales

$$\left.\begin{array}{rcl} x_1+x_2-2x_3 & = & 4 \\ x_1-x_2 & = & 1 \\ x_1+3x_3 & = & 0 \end{array}\right\}.$$

ejecutando desde la línea de comandos

```
A=[1 1 -2; 1 -1 0;1 0 3];
```

```
b=[4  1  0]';
x=sistemaAxb(A,b)
```

De esta manera, con las funciones podemos reutilizar el código para diferentes valores de entrada.

Accede al vídeo: Funciones y scripts

## 1.6 Estructuras de control

Cuando ejecutamos unas líneas de código, esta ejecución se realiza secuencialmente. Sin embargo, las estructuras de control nos permiten modificar el flujo de ejecución. En este apartado, distinguiremos entre las estructuras condicionales y los bucles.

Pero un elemento común que aparece en las estructuras de control es el uso de condiciones.

### Condiciones

Un booleano es un tipo de dato que ocupa 1 byte y tiene como posibles estados falso y verdaderno. En Matlab, los booleanos se llaman datos de tipo `logical`, y los estados falso y verdadero se representan por `0` y `1`, respectivamente.

Una condición es una operación cuya salida es un booleano. Para ello, se requiere del uso de operadores relacionales. La Tabla 2 recoge alguno de los operadores relacionales más comunes.

| Operador | Significado |
|---|---|
| == | Igual |
| >= | Mayor o igual |
| > | Estrictamente mayor |
| <= | Menor o igual |
| < | Estrictamente menor |
| ~= | Diferente |

Tabla 2: Operadores relacionales

**Ejemplo 21.**

Para conocer si las raíces de un polinomio de segundo grado $ax^2 + bx + c = 0$ son reales o complejas, necesitamos saber si el discriminante $\Delta = b^2 - 4ac$ es mayor o igual a cero, o menor que cero. Para conocerlo, ejecutamos

```
>> b^2-4*a*c<0
```

Una pareja de elementos booleanos se pueden relacionar a través de las operaciones lógicas. En la Tabla 3 se recogen las funciones de Matlab relacionadas y sus resultados.

| A | B | and(A,B) | or(A,B) | xor(A,B) |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 |

Tabla 3: Funciones y tablas de verdad

**Ejemplo 22.**

Para conocer si las raíces de un polinomio de segundo grado $ax^2 + bx + c = 0$ son reales o complejas, necesitamos saber si el discriminante $\Delta = b^2 - 4ac$ es mayor o igual a cero, o menor que cero. Para conocerlo, ejecutamos

```
>> or(b^2-4*a*c<0, b^2-4*a*c==0)
```

## Estructuras condicionales

Dentro de la programación podemos encontrar diferentes estructuras condicionales. En este apartado veremos la más común: la estructura `if-else-end`.

### Estructura condicional `if-else-end`

Esta estructura ejecuta unas instrucciones u otras dependiendo si se cumple la condición, tal y como muestra la Figura 1.

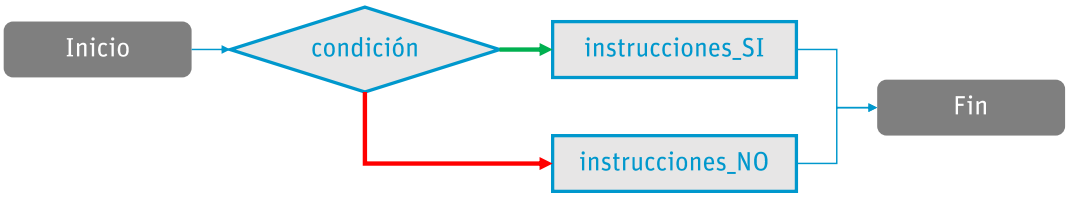



Figura 1: Flujograma de la estructura `if-else-end`

La estructura en código de `if-else-end` es

```
if condicion
    instrucciones_SI
else
    instrucciones_NO
end
```

**Ejemplo 23.**

Escribamos una función que calcule la nota final de una asignatura. Para ello,

debemos introducir la nota de la evaluación continua `e` y la nota del examen final `f`. Si el examen final está suspendido, la nota final de la asignatura es la nota del examen final. Si el examen está aprobado, la nota final de la asignatura es el 60 % del examen final más el 40 % de la evaluación continua. Una posibilidad se recoge a continuación.

```
function nf=notaFinal(e,f)
% La función nf=notaFinal(e,f) calcula la nota ...
   final nf de la asignatura a partir de las notas ...
   de la evaluación continua e y la nota del ...
   examen final f.
if f<5
    nf=f;
else
    nf=.6*f+.4*e;
end
```

La estructura `if-else-end` se puede anidar, de modo que de cada `else` puede salir otra estructura `if-else-end`.

## Bucles

Los bucles repiten la ejecución de unas líneas de código. A continuación nos centraremos en los bucles `while` y `for`.

### Bucle `while`

La estructura `while` ejecuta unas líneas de código mientras se cumpla la condición. El flujograma de la Figura 2 representa esta estructura.

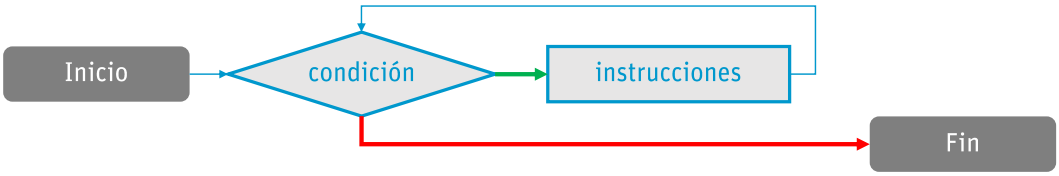



Figura 2: Flujograma de la estructura `while`

La estructura en código de `while` es

```
while condicion
    instrucciones
end
```

Es importante que en las `instrucciones` se vaya modificando alguna variable que intervenga en la `condicion`; en caso contrario, nunca se saldría del bucle.

**Ejemplo 24.**

Escribamos una función que calcule obtenga un número aleatorio que se encuentre entre 0.5 y 1. Una posibilidad se recoge a continuación.

```
function x=aleatorioMayor05
% La función x=aleatorioMayor05 obtiene un número ...
   aleatorio mayor que 0.5 de una distribución ...
   uniforme
x=rand(1);
while and(x<0.5,x>1)
    x=rand(1);
end
```

**Bucle `for`**

La estructura `for` recorre un conjunto de elementos; en cada elemento, ejecuta unas instrucciones. La estructura en código de `for` es

```
for indice=vector
    instrucciones
end
```

**Ejemplo 25.**

Escribamos una función que dado un vector `v` obtenga la diferencia entre sus elementos. Una posibilidad se recoge a continuación.

```
function difv=diferenciaVector(v)
% La función difv=diferenciaVector(v) obtiene la ...
   diferencia entre los elementos de v
for indice=1:length(v)-1
    difv(indice)=v(indice)-v(indice+1);
end
```

Accede al vídeo: Estructuras de control

## 1.7 Representaciones gráficas

Matlab también dispone de un amplio conjunto de herramientas de representaciones gráficas. En este apartado, recogemos la representación de funciones de una y dos variables, así como la edición de algunos elementos de la gráfica.

Cuando realizamos una representación, aparece una ventana emergente. El contenido de esa ventana se puede guardar como un archivo `.fig`.

## Representación de funciones de una variable

Para representar la función $f(x)$ tenemos diferentes posibilidades. La función `plot(a,b)` representa en el eje de abscisas los valores de `a` y en el eje de ordenadas los valores de `b`.

**Ejemplo 26.**

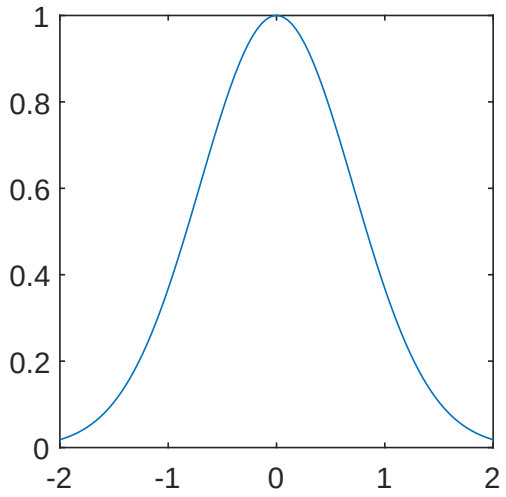
Para representar la función

$$f(x) = e^{-x^2}, \quad x \in [-2, 2],$$

vamos a utilizar un vector de 100 puntos para la variable independiente y, a continuación, evaluamos la función sobre dichos puntos.

```
>> x=linspace(-2,2);
>> f=exp(-x.^2);
>> plot(x,f)
```

El resultado es




## Representación de funciones de dos variables

Cuando tenemos una función de dos variables $f(x, y)$, también tenemos diferentes posibilidades. Es necesario mencionar que las variables `X` e `Y` deben ser matrices, por lo que habitualmente se genera un mallado a partir de los vectores `x` e `y` utilizando

```
>> [X,Y]=meshgrid(x,y);
```

En las siguientes posibilidades, se representa en los ejes X, Y y Z el contenido de las matrices `A`, `B` y `C`, respectivamente.

- `plot3(A,B,C)` hace la representación con líneas para cada valor de la variable `A`.
- `mesh(A,B,C)` hace la representación con un mallado de las variables `A` y `B`.
- `surf(A,B,C)` hace la representación con un mallado de las variables `A` y `B`, generando una superficie.

**Ejemplo 27.**

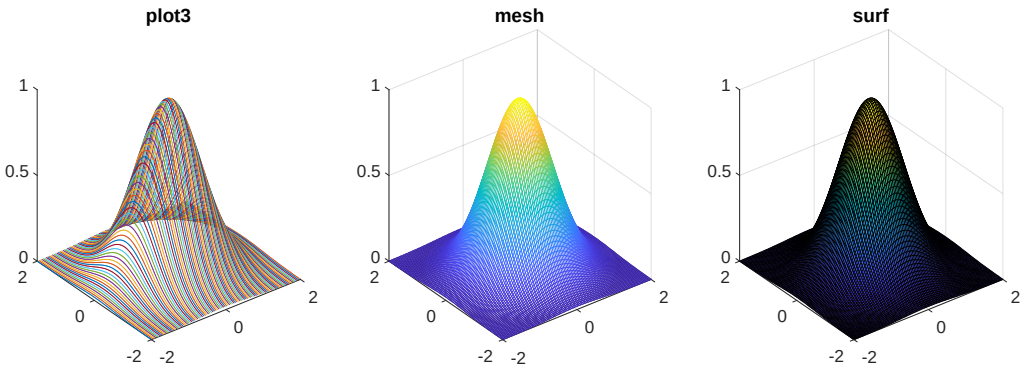
Para representar la función

$$f(x,y) = e^{-x^2-y^2}, \quad (x,y) \in [-2,2] \times [-2,2],$$

vamos a utilizar un vector de 100 puntos para las variables independientes y, a continuación, evaluamos la función sobre dichos puntos.

```
>> x=linspace(-2,2); y=linspace(-2,2)
>> [X,Y]=meshgrid(x,y);
>> f=exp(-X.^2-Y.^2);
>> figure
>> subplot(1,3,1), plot3(X,Y,f), title('plot3')
>> subplot(1,3,2), mesh(X,Y,f), title('mesh')
>> subplot(1,3,3), surf(X,Y,f), title('surf')
```

El resultado es




## Edición de elementos de la gráfica

A continuación se listan una serie de elementos que dan un mejor aspecto a las representaciones gráficas.

- `grid`: introduce un mallado de las variables

- `xlabel('texto'), ylabel ('texto'), zlabel('texto')`: introduce títulos en los ejes
- `title('texto')` introduce texto en el título de la figura

Accede al vídeo: Representaciones gráficas

## 1.8 Cálculo simbólico

A pesar de que Matlab es un software orientado al cálculo numérico, también dispone de herramientas para el cálculo simbólico. Para ello, si queremos trabajar con este tipo de cálculo, es necesario definir la variable de manera simbólica con el comando `syms`. De este modo, si queremos que la variable $x$ sea simbólica, escribiríamos

```
>> syms x
```

Una vez obtengamos el resultado, este será simbólico. Para poder recuperar el valor numérico, utilizaremos el comando `double`.

### Raíces de ecuaciones

Para resolver raíces de ecuaciones, basta con utilizar el comando `solve`.

**Ejemplo 28.**

Queremos obtener las raíces de la ecuación

$$f(x) = x^2 - 1,$$

que sabemos que son $x = \pm 1$. Para desarrollar esta operación en Matlab, ejecutamos

```
>> syms x;
>> f=x^2-1;
>> sol=solve(f==0)
```

## Funciones derivadas

El comando destinado a obtener la derivada de una función es `diff`.

**Ejemplo 29.**

Queremos obtener la derivada de la función

$$f(x) = x^2 - 1,$$

que sabemos que es

$$f'(x) = 2x.$$

Para desarrollar esta operación en Matlab, ejecutamos

```
>> syms x;
>> f=x^2-1;
>> df=diff(f,x)
```

## Integral de una función

Para obtener la integral de una función, utilizaremos el comando `int`.

**Ejemplo 30.**

Queremos obtener la derivada de la función

$$f(x) = x^2 - 1,$$

que sabemos que es

$$F(x) = \frac{x^3}{3} - x.$$

Para desarrollar esta operación en Matlab, ejecutamos

```
>> syms x;
>> f=x^2-1;
>> F=int(f,x)
```

Todas estas funciones tienen su versión para varias variables.

Accede al vídeo: Cálculo simbólico

## 1.9 Archivos `.mlx`

Los archivos Live Script son documentos interactivos en los que se pueden incluir comandos, texto formateado o ecuaciones, entre otros. Son archivos con extensión `.mlx`. Permiten visualizar las diferentes salidas de las ejecuciones que se incluyen.

A continuación, presentamos los elementos más destacados.

## Texto

Dentro de este tipo de archivos podemos introducir texto con diferentes formatos. Aparecen como formatos predefinidos **Título**, **Encabezado** y **Normal**. Además, sobre estos formatos podemos seleccionar texto en negrita, cursiva o subrayado. También podemos utilizar listas con viñetas o enumeradas.

## Ecuaciones

Los archivos Live Script permiten la introducción de ecuaciones, tanto en formato normal como en formato LaTeX. Para el formato normal, aparece un editor de ecuaciones similar al de otros programas como Microsoft Word.

## Código

La introducción de código es otra de las grandes ventajas de este tipo de archivos. Localmente se convierte en una consola de Matlab sobre la que ejecutar instrucciones y visualizar las salidas, tanto de operaciones como de gráficos.

Una vez introducido el código, para poder ejecutar la salida es necesario pulsar el botón `Run`.

Si lo que queremos es ejecutar una parte del archivo Live Script, deberemos generar una sección con dicha parte del código. Para ello, introduciremos un `Section Break` antes y después de esas líneas de código, y pulsaremos `Run Section`.

**Ejemplo 31.**

En el archivo adjunto puedes encontrar un archivo Live Script que obtiene las raíces de diferentes polinomios utilizando el comando `roots`.

T01Raices.mlx

Accede al vídeo: Archivos Live Script