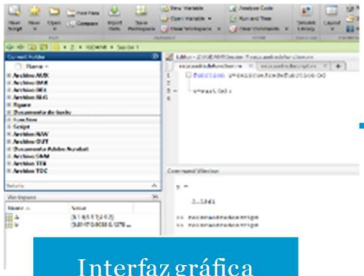
Métodos Numéricos Avanzados en Ingeniería

Introducción a Matlab

Índice

Esquema



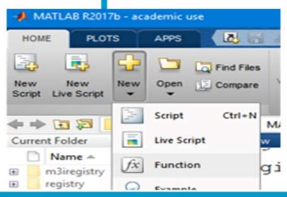
help

Instrucciones básicas

Interfaz gráfica

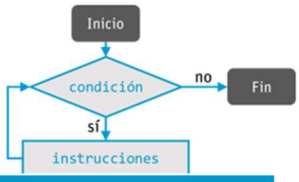
$$A = \begin{bmatrix} 1 & -1 & 0 \\ 0 & 2 & 5 \\ 1 & 3 & 1 \end{bmatrix}.$$

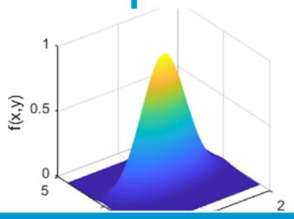
```
>> A=[1 -1 0;...
   0 2 5;...
   1 3 1];
```



Vectores y matrices

Funciones y scripts





Estructuras de control

Representaciones gráficas

```
>> syms x y
>> ec1=x+y-1;
>> ec2=2*x-3*y+2
```

Cálculo simbólico

Ideas clave

1.1. ¿Cómo estudiar este tema?

Para estudiar este tema lee las Ideas clave que encontrarás a continuación

Los métodos numéricos permiten obtener soluciones aproximadas a una serie de problemas. Antes de la revolución del transistor, que dio lugar a los circuitos integrados y, por ende, a las computadoras, los métodos numéricos requerían de un coste temporal tremendo para poder obtener las soluciones aproximadas de determinados problemas de la ciencia y la ingeniería sin conocer su solución analítica. Sin embargo, es en la segunda parte del siglo XX cuando los métodos numéricos tienen su mayor auge. El motivo fundamental es la aparición de las máquinas de computación. Si bien es cierto que no funcionaban con la velocidad que lo hace cualquier equipo doméstico de nuestros días, sí que supusieron un gran salto cualitativo para desarrollar este tipo de técnicas.

Uno de los *softwares* comerciales que nos permite optimizar el tiempo que dedicamos a la resolución de problemas de forma numérica es Matlab. Originalmente, Matlab no era un lenguaje de programación, sino una calculadora matricial interactiva. De hecho, Matlab es un acrónimo de *Matrix Laboratory*. Su primera versión data de 1981, y disponía de las funciones indicadas en la Figura 1.

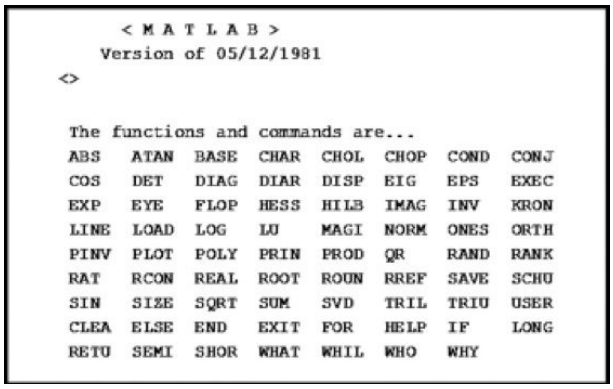


Figura 1. Funciones disponibles en la primera versión de Matlab. Fuente: https://www.mathworks.com/ .

La versión de la que disponemos hoy en día de Matlab es mucho más avanzada. De hecho, cada semestre sale una nueva versión en la que se mejora el código y se implementan nuevas funciones.

A lo largo de este tema vamos a realizar una i**ntroducción a los conceptos más básicos de Matlab** para comenzar a manejar determinadas instrucciones que nos permitan afrontar algunos problemas simples.

Los apartados de los que consta este tema son:

- La interfaz gráfica.
- Instrucciones básicas.
- Vectores y matrices.
- Funciones y scripts.
- Estructuras de control.
- Representaciones gráficas.
- Cálculo simbólico.

1.2. La interfaz gráfica

Cuando abrimos por primera vez el programa Matlab, dependiendo de la versión con la que trabajemos, vemos una interfaz gráfica u otra. En cualquier caso, esta interfaz gráfica se puede personalizar para las necesidades de cada usuario. A continuación, explicaremos el contenido de cada una de las partes que pueden aparecer en la interfaz gráfica y la información que contiene.

Directorio de trabajo

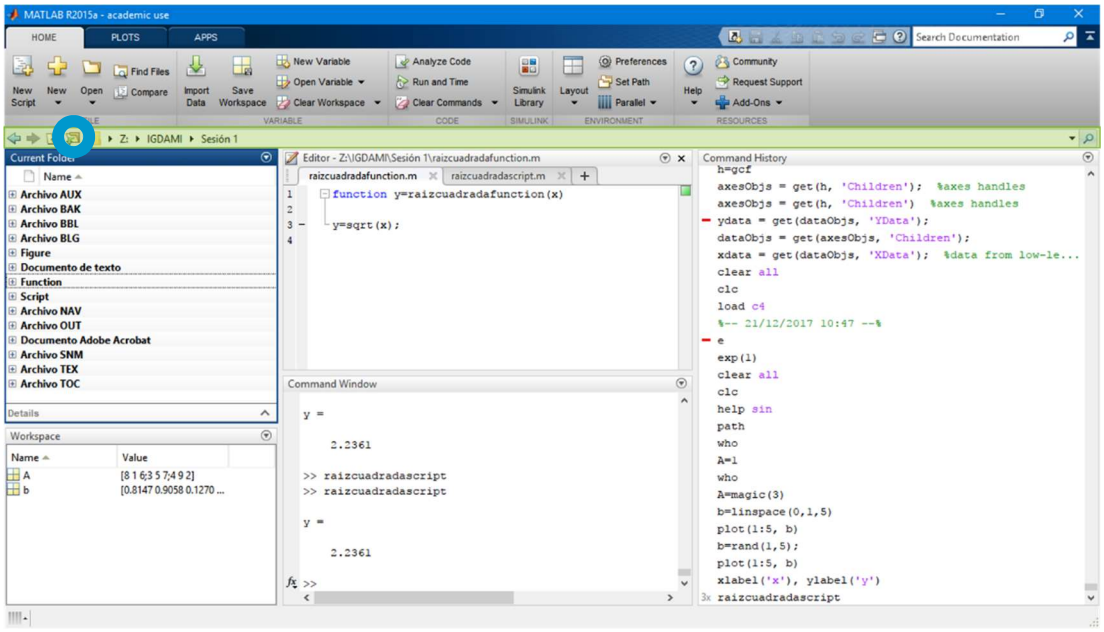


Figura 2. Ubicación del directorio de trabajo dentro de la interfaz gráfica de Matlab.

Uno de los primeros pasos que hay que dar es establecer el directorio de trabajo. Es recomendable que se genere una carpeta para esta asignatura, y dentro de esa carpeta, se generen diferentes subcarpetas para cada uno de los temas. En caso contrario, Matlab establecerá el directorio de trabajo en C:\Users\User\Documents\MATLAB (para sistemas operativos Windows). Para cambiar el directorio de trabajo hay que clicar en el punto azul de la figura 2.

Current folder (carpeta actual)

Una vez establecido el directorio de trabajo, **podemos ver en la ventana marcada de la Figura 3 el contenido de la carpeta actual**. Ahí vamos a disponer de todos los archivos como si del Explorador de Windows o el Finder de IOS se tratara. Los archivos propios de Matlab se podrán ejecutar con un doble clic, mientras que el resto de los archivos se pueden abrir clicando con el botón secundario y seleccionando «Open Outside Matlab».

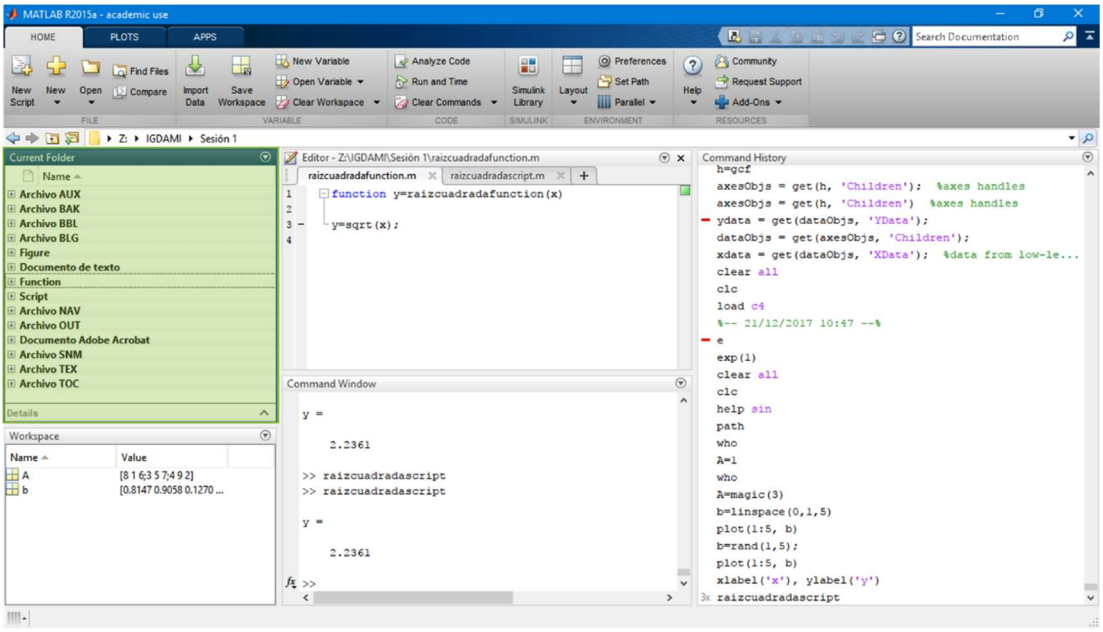


Figura 3. Ubicación de la carpeta actual dentro de la interfaz gráfica de Matlab.

Workspace

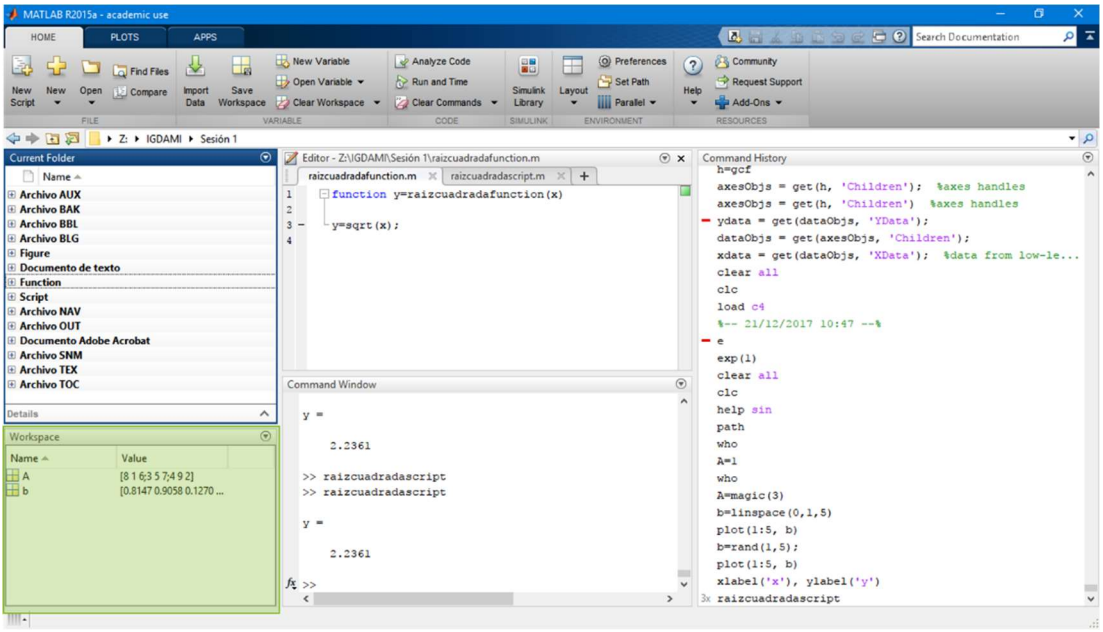


Figura 4. Ubicación del *workspace* dentro de la interfaz gráfica de Matlab.

El ***workspace*** es el lugar donde se **almacenan las variables con las que estamos trabajando en la sesión actual de Matlab.** Cada vez que cerremos el programa, cerramos la sesión actual de Matlab y, por tanto, el *workspace* se va a limpiar. Desde el *workspace* tenemos acceso a diferentes funcionalidades. Por un lado, podemos conocer qué variables tenemos por el nombre que le hemos asignado. Por otro, disponemos del tipo de variable que se trata y de su valor en caso de que sea representable.

Editor

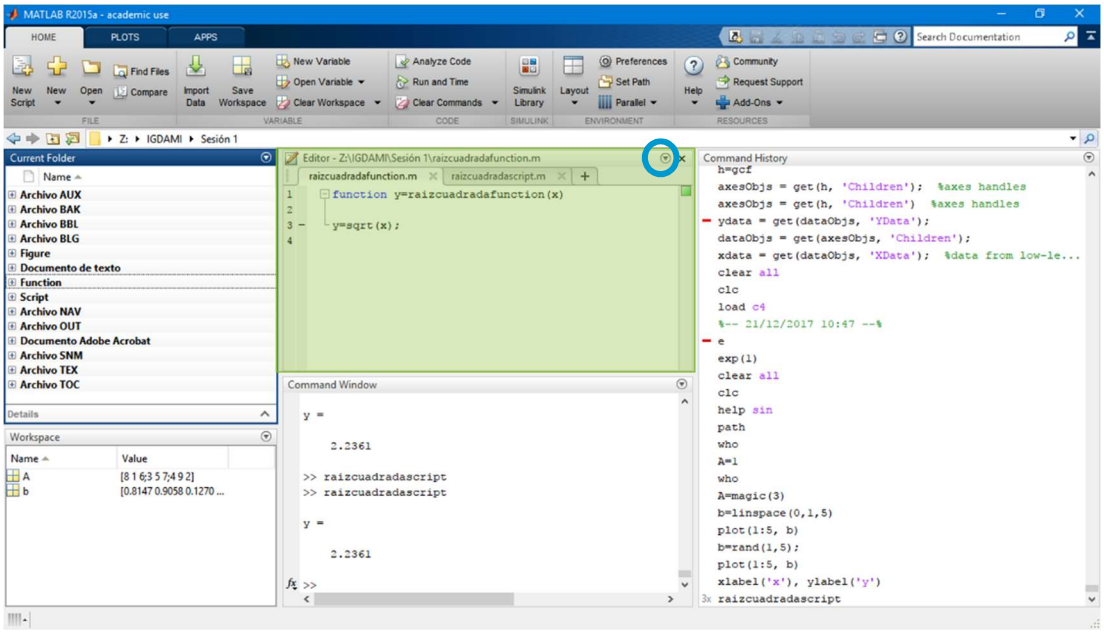


Figura 5. Ubicación del editor dentro de la interfaz gráfica de Matlab.

El **editor** es una herramienta tremendamente utilizada dentro del programa Matlab. Es el l**ugar en que se generan las funciones, los *scripts*, se escribe texto**, etc. En algunas versiones no sale encastrado dentro de la interfaz principal, y para acceder a él hay que clicar en la parte superior de la pantalla, dentro del botón New Script. Por otro lado, si nos interesa tener el editor como una ventana independiente para poder ponerlo a pantalla completa, es suficiente con clicar en el círculo azul de la figura 5 u seleccionar «Undock».

Command Window (ventana de comandos)

La ventana de comandos, también denominada «consola», es el lugar en el que vamos a ejecutar las instrucciones. Vamos a poder asignar un valor a una variable, un valor a un vector, un valor a una matriz, efectuar operaciones, aplicar funciones. También va a ser el lugar desde el que indiquemos las instrucciones para generar gráficas que representen funciones, diagramas de barras, funciones de error. Asimismo, vamos a poder acceder a la ayuda de las funciones que tiene implementadas Matlab por defecto.

En definitiva, la ventana de comandos es el centro de operaciones de Matlab, desde el que se dan las órdenes para ejecutar todo el abanico de posibilidades que nos ofrece.

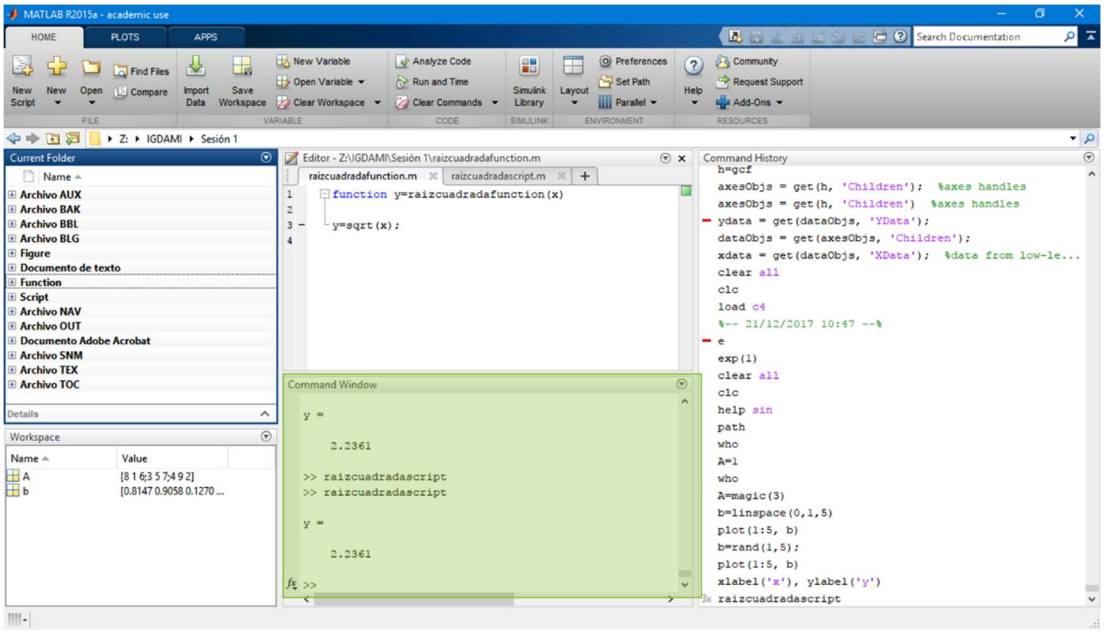


Figura 6. Ubicación de la ventana de comandos dentro de la interfaz gráfica de Matlab.

Command History (historial de comandos)

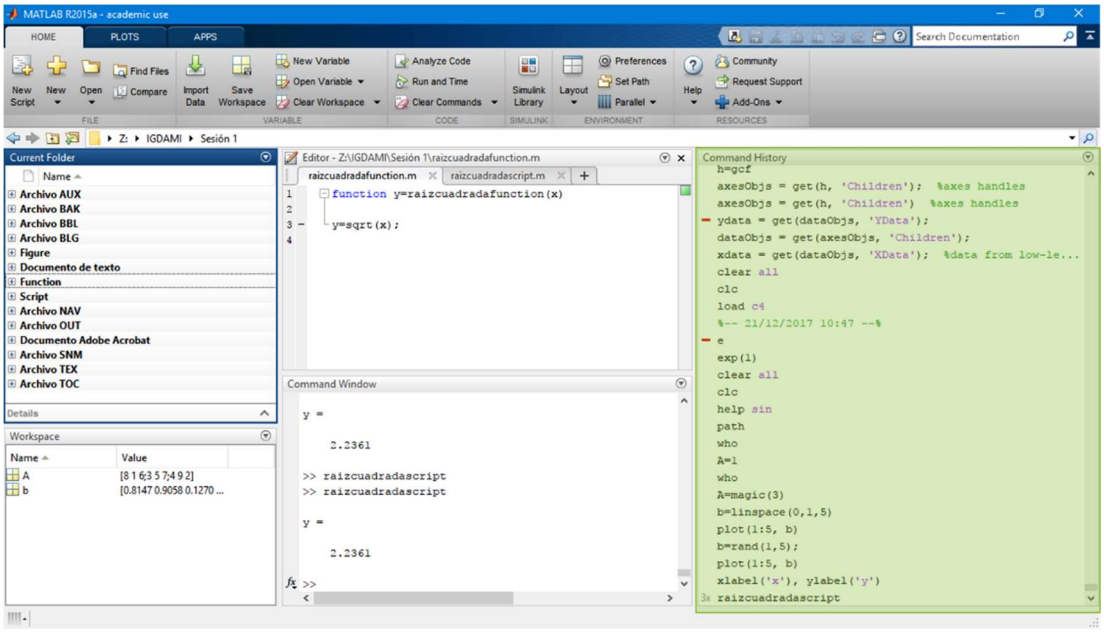


Figura 7. Ubicación del historial de comandos dentro de la interfaz gráfica de Matlab.

El historial de comandos contiene el histórico de las instrucciones ejecutadas desde la consola de Matlab. Es importante destacar que solo incluye las instrucciones y no sus resultados.

1.3. Instrucciones básicas

A continuación, describiremos una serie de instrucciones básicas para el manejo de Matlab. Es cierto que, cuanto mayor es el conocimiento que un usuario tiene de Matlab, mayor es su gama de instrucciones básicas. No obstante, planteamos esta serie de pautas para que el nuevo usuario se familiarice.

Comandos

- `help`
  - Es el **comando para solicitar la ayuda** dentro de la consola de Matlab. Escribiendo `help` nombre_del_comando, nos aparecerá la ayuda correspondiente al comando nombre_del_comando.

> **Ejemplo 1.** Obtén la ayuda del comando `save`.
>
> Ejecutamos en la consola de Matlab:
> `>> help save`
>
> Y obtenemos:
>
> **save** Save workspace variables to file.
> **save**(FILENAME) stores all variables from the current workspace in a MATLAB formatted binary…

- `save, load`
  - El **comando save** nos permite guardar algunas o todas las variables que tenemos en el *workspace* dentro de un archivo .mat. Esto nos permite disponer

de los datos que estaban en el *workspace* aunque cierre la sesión de Matlab. Para cargar datos .mat dentro del directorio actual de trabajo, utilizaríamos el comando `load`.

> **Ejemplo 2.** Estamos haciendo una encuesta dentro de un equipo de dos personas. La persona P1 se encarga de registrar los datos, y la persona P2 es la responsable de analizarlos. Ambos trabajan con Matlab, pero se encuentran a 1.000km de distancia. ¿Cómo podrían realizar el trabajo?
>
> La persona P1 introduce todos los datos en Matlab, y una vez introducidos ejecuta:
>
> ```
> >> save datosP1
> ```
>
> En el directorio actual de trabajo se genera una variable `datosP1.mat`. La persona P1 le envía un mail a P2 con `datosP1.mat` como adjunto. La persona P2 recibe el archivo, lo guarda en su directorio de trabajo y ejecuta:
>
> ```
> >> load datosP1
> ```

- `clear`
  - El comando `clear` borra el contenido del *workspace*.

- `clc`
  - El comando `clc` limpia la consola de Matlab.

- `diary`
  - El comando `diary` nos sirve para guardar en un archivo que se puede abrir con el bloc de notas toda la información que ha aparecido en la consola. Para ello, cuando queremos empezar a guardar escribimos `diary on`, y cuando finalizamos escribimos `diary off`. En la carpeta actual de trabajo se habrá guardado un archivo con el nombre `diary` sin extensión, que se puede abrir con el bloc de notas o cualquier editor.

Tipos de archivos

- `*.m`
  - Los archivos `*.m` son ficheros que contienen funciones o scripts. Más adelante veremos en qué consiste cada uno de estos archivos.
- `*.mat`
  - Los archivos `*.mat` sirven para almacenar variables. En el Ejemplo 2 vimos un caso de uso.
- `*.fig`
  - Los archivos `*.fig` son ficheros que contienen figuras tales como representaciones de funciones, diagramas de barras, curvas de nivel, etc.

Funciones

A continuación, listamos una serie de funciones cuyo nombre ya da una idea de qué realizan. Debemos indicar que, aquellas funciones trigonométricas que aparecen a continuación trabajan siempre con radianes.

- `sin, asin`
- `cos, acos`
- `tan, atan, atan2`
- `log, log10, log2, exp`

Ejemplo 3. Calcula $2^{3x-5} = 8$.

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

Ejecuto en Matlab:

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

Valores

- π: `pi`
- e: `exp(1)`
- i: `1i`
- Último elemento calculado sin asignar: `ans`
- `0/0`: `Nan`

> **Ejemplo 4**. Calcula $e^{-\frac{i\pi}{2}}$.
>
> Ejecuto en Matlab:
>
> ```
> >> exp(-1i*pi/2)
> ```

Formatos numéricos de salida

Podemos establecer el formato numérico de salida con el comando `format` seguido del formato que deseemos. La Tabla 1 recoge algunos de los formatos más destacables.

Comando	Resultado	Ejemplo
`format short`	4 dígitos tras el punto decimal	`3.1416`
`format long`	15 dígitos tras el punto decimal	`3.141592653589793`
`format shortE`	4 dígitos tras el punto decimal con notación científica	`3.1416e+00`
`format longE`	15 dígitos tras el punto decimal con notación científica	`3.141592653589793e+00`

Tabla 1. Algunos formatos numéricos de salida.

1.4. Operaciones con vectores y matrices

Uno de los grandes potenciales que tiene Matlab es su capacidad para operar con vectores y matrices de una forma muy sencilla para el usuario.

Introducción de vectores

Un vector fila en Matlab se introduce entre corchetes (`[ ]`), y separando sus componentes por comas (`,`) o por espacios.

> **Ejemplo 5**. Introduce el vector $u = [-3 \quad 1 \quad 5]$.
>
> Ejecuto en Matlab:
>
> ```
> >> u=[-3 1 5];
> ```

Un vector columna en Matlab se introduce entre corchetes, y separando sus componentes por puntos y coma (`;`). Una opción más utilizada es introducir el vector fila y trasponerlo.

La operación de trasposición se realiza con el apóstrofe (`'`), aunque debemos ir con cuidado si estamos trabajando con números complejos. Realmente, al utilizar el apóstrofe estamos trasponiendo y conjugando el vector o la matriz. Así que, si trabajamos con vectores o matrices reales, podremos utilizar sin temor el apóstrofe, pero si trabajamos con vectores o matrices complejos, será mejor utilizar el punto apóstrofe (`.'`) para su trasposición sin conjugación.

Ejemplo 6. Introduce el vector $v = \begin{bmatrix} -1 \\ 0 \\ 4 \end{bmatrix}$

Ejecuto en Matlab:

```
>> v=[-1 0 4]';
```

Para extraer componentes de un vector, debemos indicar el vector y, entre paréntesis, la componente que queremos extraer. En caso de querer extraer más de una componente, entre paréntesis introduciremos un vector con las posiciones a extraer.

Ejemplo 7. Introduce el vector $w = [0 \quad 3 \quad -1 \quad -1 \quad 4 \quad 2]$ y obtén las componentes 3 y 5.

Ejecuto en Matlab:

```
>> w=[0 3 -1 -1 4 2];
>> w([3 5])
```

Vectores con componentes equiespaciadas

En general, los vectores los vamos a utilizar mucho sin darnos cuenta, para diferentes propósitos: acceder a los elementos de un vector cuyo contenido sean datos, generar una serie de nodos sobre los que evaluar una función, etc. En muchas ocasiones necesitamos que los elementos del vector estén equiespaciados, para lo cual Matlab nos permite utilizar diferentes estructuras.

- `valor_inicial:intervalo:valor_final`
  - Si no ponemos el intervalo, se toma por defecto el valor 1
  - Si `valor_inicial` es superior a `valor_final`, `intervalo` debe ser negativo
- `linspace(valor_inicial, valor_final, número_de_puntos)`
  - Si no ponemos el número de puntos, tomará el valor 100 por defecto.

Ejemplo 8. Introduce los vectores $a = [0.1 \quad 0 \quad -0.1 \quad -0.2 \quad -0.3 \quad -0.4]$, $b = [3 \quad 4 \quad 5]$ y $c = [1 \quad 1.1 \quad 1.2 \quad \cdots \quad 2]$.

Ejecuto en Matlab:

```
>> a=.1:-.1:-.4;
>> b=3:5;
>> c=linspace(1,2,11);
```

Operaciones con vectores

Para la realización de operaciones con vectores, debemos tener en cuenta el álgebra matricial. En este sentido, solo se podrán:

- sumar (+) vectores de las mismas componentes.
- multiplicar (*) vectores fila por columna o columna por fila, del mismo número de componentes.

En base a estas restricciones, se pueden aplicar también las siguientes operaciones:

- Producto escalar: `dot`
- Producto vectorial: `cross`
- Norma: `norm`
- Longitud: `length`

Asimismo, en Matlab aparece una nueva operación que nos permite trabajar con los vectores de una manera algo diferente. Es la operación elemento a elemento y se puede aplicar sobre los operadores producto (*), cociente (/) o potencia (^). Basta con añadir un punto (`.`) delante del operador correspondiente y hará las operaciones elemento a elemento.

Ejemplo 9. Sean $u = [3 \quad 0 \quad -1 \quad 4]$ y $v = [1 \quad 1 \quad -2 \quad 2]$. Obtén el vector w cuyas componentes son $w_i = u_i \cdot v_i, i = 1, \dots, 4$ y el escalar $z = \langle u, v \rangle = u \cdot v$.

Ejecuto en Matlab:

```
>> u=[3 0 -1 4];
>> v=[1 1 -2 2];
>> w=u.*v;
>> z=dot(u,v);
```

Introducción de matrices

El otro elemento fundamental para el trabajo en Matlab son las matrices. En este sentido, vamos a describir la manipulación de matrices.

Para empezar, una matriz se introduce en Matlab entre corchetes (`[ ]`), separando los elementos de cada fila por espacios o comas (`,`), y separando las filas por punto y coma (`;`).

Ejemplo 10. Introduce la matriz.

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

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

Para acceder a un elemento de la matriz, introduciremos el nombre de la matriz y pondremos entre paréntesis dos valores separados por una coma. El primer valor se corresponderá con la fila y el segundo elemento se corresponderá con la columna.

Ejemplo 11. Dada la matriz A del Ejemplo 10, obtén los elementos a_{13} y a_{32}.

```
>> a13=A(1,3);
>> a32=A(3,2);
```

En caso de que queramos acceder a toda una fila, el valor introducido será el número de la fila y dos puntos (`:`) en el valor de la columna. Recíprocamente, para acceder a los elementos de una columna, el valor en el índice de las filas será dos puntos (`:`) y en el índice de las columnas el número de la columna a extraer.

Ejemplo 12. Dada la matriz A del Ejemplo 10, obtén la fila 1 y la columna 3.

```
>> fila1=A(1,:);
>> columna3=A(:,3);
```

Otros elementos interesantes a los que podemos acceder son los elementos de la diagonal a_{ii}, de la diagonal superior $a_{i+1,i}$ o de la diagonal inferior $a_{i-1,i}$. Para ello, se utiliza el comando `diag`.

Ejemplo 13. Dada la matriz del Ejemplo 10, obtén la diagonal y la diagonal inferior de A.

```
>> d=diag(A);
>> dInferior=diag(A,-1);
```

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 de ceros: `zeros`
  - Matriz cuadrada de $n \times n$ ceros: `zeros(n)`
  - Matriz rectangular de $m \times n$ ceros: `zeros(m,n)`

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

Operaciones con matrices

De forma análoga al caso de los vectores, para realizar las diferentes operaciones con matrices es necesario respetar las dimensiones. Recordemos que para poder sumar (+) dos matrices $A_{m\times n}$ y $B_{p\times q}$, se debe cumplir que $m = p$ y $n = q$, es decir, que tengan las mismas dimensiones. Asimismo, para poder realizar un producto (*) entre dos matrices $A_{m\times n}$ y $B_{p\times q}$, se debe cumplir que $n = p$ y la matriz resultante será de dimensiones $m \times q$. Recordemos también que el producto de matrices no cumple la propiedad conmutativa, de forma que $A \cdot B \neq B \cdot A$.

Además, se pueden obtener determinadas características de las matrices:

- Determinante: `det`
- Matriz inversa: `inv`
- Rango: `rank`

La operación elemento a elemento también se puede aplicar sobre matrices. De este modo, vamos a poder realizar el producto elemento a elemento (`.*`), el cociente elemento a elemento (`./`) o la potencia elemento a elemento (`.^`).

Ejemplo 14. Sean $A = \begin{bmatrix} 1 & 0 & -2 \\ 0 & 0 & 1 \\ 1 & 3 & -2 \end{bmatrix}$ y $B = \begin{bmatrix} 2 & 1 & 0 \\ 3 & 2 & 1 \\ -1 & 0 & -1 \end{bmatrix}$. Obtén la matriz C cuyas componentes son $c_{ij} = a_{ij} \cdot b_{ij}, i, j = 1,2,3$ y el determinante $|A|$.

Ejecuto en Matlab:

```
>> A=[1 0 -2;0 0 1;1 3 -2];
>> B=[2 1 0;3 2 1;-1 0 -1];
>> C=A.*B;
>> detA=det(A);
```

El sistema $Ax = b$

Para resolver el sistema $Ax = b$, donde A es una matriz y x y b son vectores, siendo el objetivo conocer el valor de x, matricialmente operamos como:

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

De forma que la implementación en Matlab de esta operación es:

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

El cálculo de la matriz inversa supone un coste computacional bastante elevado, aunque no seamos capaces de percibirlo. 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. Resuelve el sistema:

$$\begin{cases} x_1 + x_2 - 2x_3 = 4 \\ x_1 - x_2 = 1 \\ x_1 + 3x_3 = 0 \end{cases}$$

El sistema que nos piden resolver, expresado matricialmente, es:

$$Ax = b \leftrightarrow \begin{bmatrix} 1 & 1 & -2 \\ 1 & -1 & 0 \\ 1 & 0 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 4 \\ 1 \\ 0 \end{bmatrix}$$

Ejecuto en Matlab:

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

1.5. Funciones y *scripts*

El editor de Matlab nos permite generar una serie de archivos de diferentes características. Entre ellos están las funciones y los *scripts*. Son archivos muy parecidos, pero con una diferencia fundamental, basada en los parámetros de entrada y salida. Para ello, comenzaremos definiendo las funciones y, posteriormente, los *scripts*.

Funciones

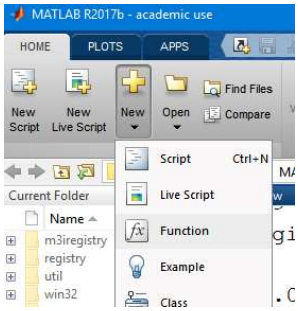


Figura 8. Cómo comenzar con una nueva función.

La figura 8 ilustra cómo introducir una nueva función en Matlab. Al seleccionar `Function`, se abrirá el editor de Matlab.

Una función tiene parámetros de entrada y de salida. La sintaxis de la primera línea de una función es:

```
function [salida1, salida2]=nombre_función(entrada1, entrada2)
```

Los parámetros de entrada van entre paréntesis (`()`) y los parámetros de salida entre corchetes (`[]`). Una función puede tener tantos parámetros de entrada y de salida como necesitemos. Los parámetros de entrada pueden ser escalares, vectores, matrices, cadenas de caracteres, … En las siguientes líneas del código, implementaremos la función. El Ejemplo 16 muestra la implementación de una función.

Ejemplo 16. Implementa una función `e2g.m` que, dados los coeficientes a, b y c de una ecuación de segundo grado de la forma $ax^2 + bx + c = 0$, obtenga sus raíces.

En primer lugar, debemos conocer que la ecuación de segundo grado va a aportar dos soluciones dadas por:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

De forma que tendremos tres parámetros de entrada (a, b, c) y dos parámetros de salida (sol1, sol2). La primera línea de la función será:

```
1 function [sol1,sol2]=e2g(a,b,c)
```

A continuación, vamos a calcular la primera de las raíces. Para ello, escribimos en la siguiente línea:

```
2 sol1=(-b+sqrt(b^2-4*a*c))/(2*a);
```

Y, por último, en la tercera línea escribimos la segunda de las raíces:

```
3 Sol2=(-b-sqrt(b^2-4*a*c))/(2*a);
```

Guardamos el archivo con Save as, y le damos el mismo nombre que tiene la función, es decir, `e2g.m`. Si queremos resolver la ecuación $x^2 - x + 2 = 0$, deberemos ejecutar en la consola de Matlab:

```
>> [sol1,sol2]=e2g(1,-1,2)
```

Scripts

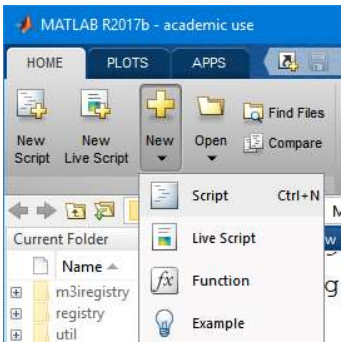


Figura 9. Cómo comenzar con un nuevo *script*.

Los *scripts*, cuyo acceso se muestra en la figura 9, también ejecutan de forma secuencial una serie de instrucciones, pero no tienen parámetros de entrada y de salida. Es el entorno ideal para escribir una memoria de operaciones realizadas. Veamos en el Ejemplo 17 un caso de aplicación.

Ejemplo 17. Calcula el determinante de la matriz.

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

Podemos generar un *script* para recordar los pasos necesarios en el cálculo de un determinante. El símbolo porcentaje (%) al principio de la línea indica que vamos a realizar un comentario. Podríamos generar el siguiente *script*.

```
1  % Script para la obtención del determinante de A
2  % Introducimos la matriz
3  A=[-3 2 1;1 3 0;0 1 2];
4  % Para calcular el determinante, utilizamos el comando
det
5  determinanteA=det(A);
```

Guardamos el archivo como `calculoDeterminante.m` y lo podremos abrir cuando deseemos. Si queremos ejecutar su contenido, escribimos en la consola:

```
>> calculoDeterminante
```

En este punto es importante destacar el uso del punto y coma (`;`) al final de cada instrucción. Si no podemos el punto y coma, se mostrará en la consola el resultado de la operación. En cambio, si lo ponemos, no se mostrará.

1.6. Estructuras de control

Dentro de la programación de algunos métodos o funciones podemos utilizar lo que se conoce como estructuras de control. Sirven para ejecutar una serie de instrucciones si se cumplen unas condiciones o de forma iterativa. A continuación, presentamos las estructuras condicionales y los bucles.

Condicionales

La estructura condicional por excelencia es el `if-else-end`. El

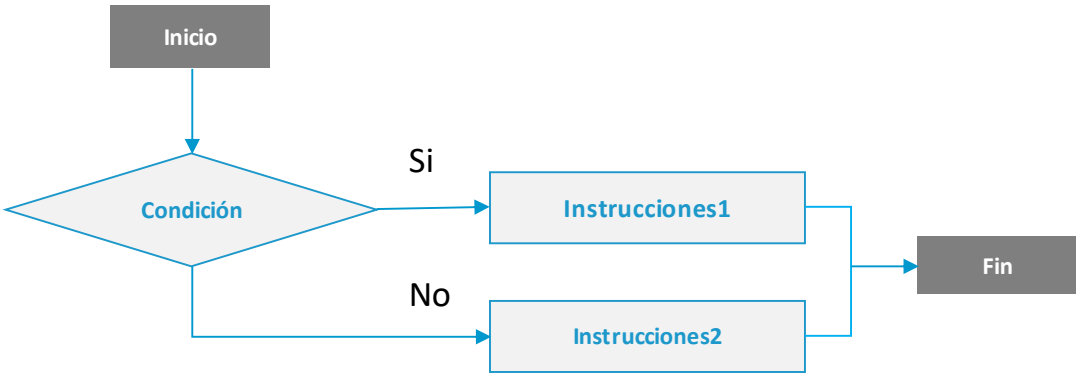


Figura 1siguiente muestra el esqueleto de esta estructura.

```
1  if condición
2       instrucciones1
3  else
4       instrucciones2
5  End
```

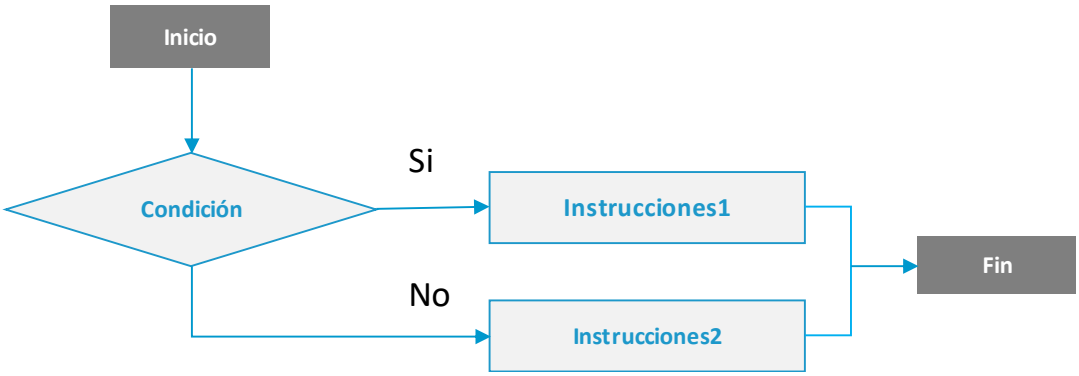


Figura 10. if-else-end. Flujograma.

Si se cumple la `condición` de la línea 1, se ejecutarán las `instrucciones1` de la línea 2 y saltará a la línea 5. En caso de que no se cumpla la `condición`, se ejecutarán las `instrucciones2` de la línea 4 y saltará a la línea 5.

Si no se incluyen las líneas 3 y 4, tendríamos una estructura `if-end`. En ese caso, si no se cumple la `condición`, saltaría directamente a la línea 5.

Dentro de la estructura `if-else-end` se pueden anidar nuevas estructuras del mismo tipo. El Ejemplo 18 muestra un caso.

> **Ejemplo 18.** Escribe una función `notaCualitativa.m` que, dado como parámetro de entrada la calificación de un alumno n entre 0 y 10, escriba en la consola su calificación cualitativa. Es decir, si $0 \leq n < 5$, suspenso; si $5 \leq n < 7$, aprobado; si $7 \leq n < 9$, notable; si $9 \leq n \leq 10$, sobresaliente.
>
> El comando que escribe por la consola una cadena de caracteres es `disp`. Planteamos la siguiente función:

```
1 function notaCualitativa(n)
2 if n<5
3    disp('Suspenso')
4 else if n<7
5         disp('Aprobado')
6      else if n<9
7              disp('Notable')
8           else
9              disp('Sobresaliente')
10          end
11     end
12 End
```

> Si quisiéramos ejecutar sobre la consola de Matlab, deberíamos introducir.

```
>> notaCualitativa(7);
```

Bucles

El segundo bloque de estructuras de control son los bucles. Vamos a comenzar con el bucle *while*. El Figura 11.muestra la estructura del bucle *while*.

```
1  while condición
```

```
2       instrucciones1
3  end3
```

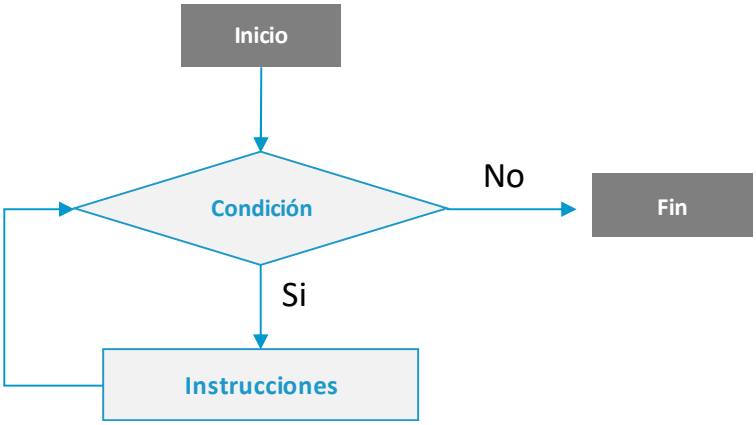


Figura 11. while-end. Flujograma.

Mientras que se cumpla la `condición`, se ejecutarán las `instrucciones1` de la línea 2 y saltará a la línea 3. De nuevo, volverá a la línea 1 para comprobar si se sigue cumpliendo la `condición`.

Es importante que, dentro de las `instrucciones1`, se vaya modificando alguna de las variables que intervienen en la `condición`; en caso contrario, no se saldría nunca del bucle. Veamos un caso de aplicación en el Ejemplo 19.

Ejemplo 19. Escribe una función `sumaCuadrado.m` que, dado como parámetros de entrada un vector v de 3 componentes y un valor de n, obtenga:

$$w = \sum_{i=1}^{3} v_i^n.$$

Vamos a generar un bucle *while* cuya condición de parada sea que el índice `ind` del vector sea mayor que 3. Dentro del bucle, tomaremos cada una de las componentes y la elevaremos a `n`:

```
1 function w=sumaCuadrado(v,n)
2 ind=1;
```

```
3 w=0;
4 while ind<=3
5     w=w+v(ind)^n;
6     ind=ind+1;
7 end
```

En las líneas 2 y 3 inicializamos los valores. Inicialmente, `w` vale 0 e `ind` vale 1. En la línea 4 entramos dentro del bucle porque `ind` es igual a 1, es decir, es menor o igual a 3. A continuación, tomamos el valor de `w` que es 0, y le sumamos el valor de v_1^n. Incrementamos el valor de `ind` y volvemos a la línea 4.

En este momento, `ind=2`. Entramos en el bucle porque `ind` es menor o igual a 3. En `w` teníamos el valor v_1^n, al cual ahora le sumamos v_2^n. Volvemos a incrementar el valor de `ind` y saltamos de nuevo a la línea 4.

Estamos con `ind=3`. Como `ind` es menor o igual a 3, entramos en el bucle. A `w` le vamos a sumar v_3^n, incrementamos `ind` y saltamos a la línea 4.

En este momento `ind=4`. Como no es menor o igual a 3, saltamos a la línea 7 y finalizamos.

Otra estructura clave es el bucle for. En este caso, no vamos a evaluar una condición, sino que vamos a dar los valores de los índices para los cuales queremos que se ejecuten las instrucciones.

```
1 for vector=índices
2     instrucciones1
3 end
```

Repitamos el Ejemplo 19 utilizando un bucle `for`.

Ejemplo 20. Escribe una función `sumaCuadradoFor.m` que realice lo mismo que el Ejemplo 19 utilizando un bucle `for`.

```
function w=sumaCuadradoFor(v,n)
```

```
w=0;
for ind=1:3
    w=w+v(ind)^n;
    ind=ind+1;
End
```

El vector `ind` va a recorrer los valores 1, 2 y 3, en ese orden. En cada uno de los pasos por el bucle, irá sumándose al valor de `w` el correspondiente valor de v_i^n.

1.7. Representaciones gráficas

Matlab nos permite realizar diferentes tipos de representaciones gráficas. En los siguientes apartados veremos algunas de las formas más utilizadas.

Plot

El comando `plot(a,b)` representa en el eje de abscisas el vector «a» y en el eje de ordenadas el vector «b». Por defecto, los puntos se unen a través de una línea azul; sin embargo, existen opciones para personalizar las representaciones. Veamos un caso muy utilizado para representar funciones de una variable.

Ejemplo 21. Representa gráficamente la función $f(x) = e^{-x^2}$.

Debemos generar un vector de valores de x en un intervalo representativo. Tomaremos el intervalo $x \in [-2,2]$, con 100 puntos.

```
>> x=linspace(-2,2);
```

A continuación, generamos el valor de $f(x)$.

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

Por último, ejecutamos el comando `plot` para representar la función.

```
>> plot(x,f)
```

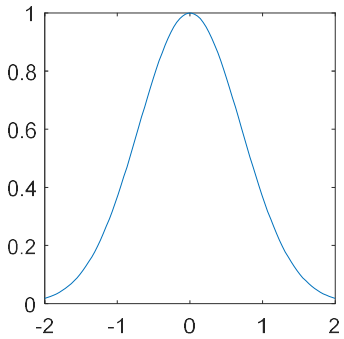


Figura 12. Función $f(x) = e^{-x^2}$.

Surf, mesh

Las instrucciones `surf(a,b,C)` y `mesh(a,b,C)` sirven para realizar representaciones en 3 dimensiones. Representan en el eje de abscisas el vector a, de dimensión m, en el eje de ordenadas el vector b de dimensión n, y en el eje de altura la matriz C de dimensión $m \times n$. El Ejemplo 22. Representa gráficamente la función $f(x,y) = e^{-(x^2+y^2)}$. ilustra un caso habitual de uso.

Ejemplo 22. Representa gráficamente la función $f(x,y) = e^{-(x^2+y^2)}$.

Generamos los vectores independientes x e y. Por ejemplo, tomamos $x \in [-2,2]$ con 100 puntos e $y \in [-4,4]$ con 200 puntos.

```
>> x=linspace(-2,2);
>> y=linspace(-4,4,200);
```

Generamos una matriz con los elementos combinados de x e y.

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

Evaluamos la función f.

```
>> f=exp(-X.^2-Y.^2);
```

Representamos la función.

```
>> surf(x,y,f)
```

Para que no se vea el mallado negro, introducimos la expresión:

```
>> shading interp
```

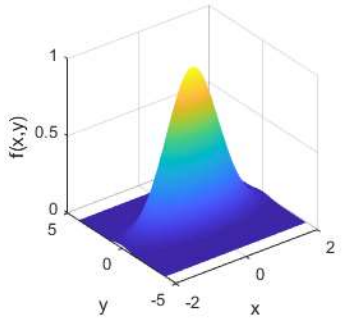


Figura 13. Función $f(x,y) = e^{-(x^2+y^2)}$.

Contour

La instrucción `contour(a,b,C)` tiene la misma sintaxis que las instrucciones `surf` y `mesh`. En este caso, en lugar de hacer una representación en 3 dimensiones, se representan las curvas de nivel. Repitamos el Ejemplo 22 con curvas de nivel.

Ejemplo 23. Representa las curvas de nivel de la función $f(x,y) = e^{-(x^2+y^2)}$.

La generación de los vectores y las matrices es la misma que la del ejemplo anterior.

```
>> x=linspace(-2,2);
>> y=linspace(-4,4,200);
>> [X,Y]=meshgrid(x,y);
>> f=exp(-X.^2-Y.^2);
```

La representación se realiza con la función `contour`.

```
>> contour(x,y,f)
```

Para interpretar mejor los valores, se utiliza el comando: `>> colorbar`

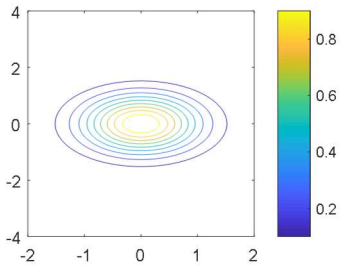


Figura 14. Función $f(x,y) = e^{-(x^2+y^2)}$.

1.8. Cálculo simbólico

A pesar de que Matlab es un *software* orientado al cálculo numérico, también permite trabajar con cálculo simbólico. Existen otros *softwares* con mucha mayor potencia para el cálculo simbólico, como es Mathematica. No obstante, aportaremos unas pinceladas del cálculo simbólico en Matlab.

Para cualquier caso, es necesario declarar las variables de trabajo como simbólicas con el comando `syms`. Una vez tenemos la solución simbólica, para poder obtener la solución numérica haremos uso del comando `double`.

Resolución de ecuaciones

Matlab permite obtener la solución de una ecuación. Para ello, es necesario utilizar el comando `solve(expresión, variable)`. En `expresión` pondremos la ecuación a resolver, y en `variable` la variable independiente que queremos obtener. En el Ejemplo 24 tenemos un caso de uso.

Ejemplo 24. Resuelve la ecuación $f(x) = x^3 - 3x + 1 = 0$.

En primer lugar, declaramos la variable x como simbólica:

```
>> syms x
```

A continuación, escribimos la función:

```
>> f=x^3-3x+1;
```

Por último, resolvemos la ecuación:

```
>> sol=solve(f==0,x)
```

Para obtener la solución numérica, ejecutamos:

```
>> solnum=double(sol);
```

Resolución de sistemas de ecuaciones

De forma similar al caso de la resolución de ecuaciones, tenemos la solución de sistemas de ecuaciones de forma simbólica. El comando para resolver el sistema es `solve`, y en este caso, las ecuaciones y los vectores independientes se introducen como vector. El Ejemplo 25 clarifica este caso.

Ejemplo 25. Resuelve el sistema de ecuaciones:

$$\begin{cases} x + y = 1 \\ 2x - 3y = -2 \end{cases}$$

Definimos las variables x e y como simbólicas:

```
>> syms x y
```

Escribimos las ecuaciones:

```
>> ec1=x+y-1;
>> ec2=2*x-3*y+2
```

Resolvemos el sistema:

```
>> sol=solve([ec1==0,ec2==0],[x y])
```

Para obtener la solución de cada variable, accedemos a un elemento de tipo estructura:

```
>> xsol=sol.x;
>> ysol=sol.y;
```

Para acceder a los valores numéricos, utilizamos el comando double:

```
>> xnum=double(xsol);
>> ynum=double(ysol);
```

Lo + recomendado

No dejes de leer

Curso básico de programación en Matlab®

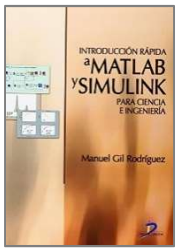
Bravo, J. L., Souto, A. y, Cantón, A. (2012). *Curso básico de programación en Matlab®*. Madrid: Tébar Flores.

En este libro puedes encontrar más instrucciones de las que hemos visto a lo largo del tema, además de profundizar en diferentes aspectos de la programación y uso de Matlab.

Accede al libro a través de la Biblioteca Virtual de UNIR

Introducción rápida a Matlab y Simulink para ciencia e ingeniería

Gil, M. (2015). *Introducción rápida a Matlab y Simulink*. Madrid: Díaz de Santos.



En este libro puedes encontrar una descripción de los comandos más básicos de Matlab a lo largo de este libro, así como una introducción a Simulink.

Accede al libro a través de la Biblioteca Virtual de UNIR

No dejes de ver

Cuenta de Matlab en Youtube

Mathworks, la empresa del software Matlab, tiene un canal en Youtube. Dentro de ese canal se puede acceder a diferentes listas de reproducción relacionadas con diferentes temáticas a tratar con Matlab, tales como Introducción al Deep Learning, Modelado Físico o Integración entre Matlab y C/C++, entre otros.

Accede al canal a través del aula virtual o desde la siguiente dirección web:

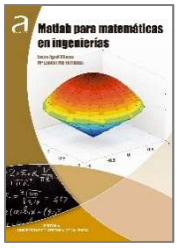
https://www.youtube.com/user/MATLAB/playlists

+ Información

A fondo

Matlab para matemáticas en ingenierías

Agud, L. y Pla, M. L. (2015). *Matlab para matemáticas en ingenierías*. Valencia: Universitat Politècnica de València.



Si quieres ampliar conceptos básicos sobre Matlab, en este libro se desarrollan con más profundidad algunos de los conceptos vistos en este tema, fundamentalmente los capítulos 1, 2 y 6.

Accede al libro a través de la Biblioteca Virtual de UNIR

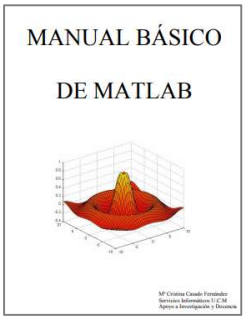
Métodos numéricos con Matlab

Cordero, A., Hueso, J. L., Martínez, E., Torregrosa, J. R. (2005). *Métodos numéricos con Matlab*. Valencia: Universitat Politècnica de València.

En los primeros capítulos de este libro se desarrollan los conceptos más básicos a la hora de utilizar el software Matlab. Este libro supone un complemento a lo descrito a lo largo de este tema.

Manual básico de Matlab

En este manual, disponible en la web de la Universidad Complutense de Madrid, puedes encontrar un listado de los primeros comandos necesarios para afrontar un curso con Matlab.



Accede a la página web a través del aula virtual o desde la siguiente dirección web:
http://webs.ucm.es/centros/cont/descargas/documento11541.pdf

Bibliografía

Cordero, A., Hueso, J. L., Martínez, E. y Torregrosa, J. R. (2004). *Cálculo Numérico*. Valencia: Universitat Politècnica de València.

Test

1. ¿Cuál de las siguientes instrucciones da como resultado un vector columna?

A. $[1\ \ 3\ \ 5\ \ 7]'$.

B. $[1,\ 3,\ 5,\ 7]$.

C. $[1;\ \ 3,\ 5;\ \ 7]$.

2. ¿Cuál de los siguientes comandos guarda las variables que hay en el *workspace* en un archivo *.mat?

A. `load.`

B. `save.`

C. `double.`

3. Un *script* de Matlab:

A. No tiene parámetros de entrada.

B. No tiene parámetros de salida.

C. Todas las anteriores son correctas.

4. ¿Con qué comando podemos representar una función $f(x)$?

A. `Plot`.

B. `Surf`.

C. `Contour`.

5. Si queremos elevar al cuadrado los elementos de la matriz A, utilizaremos la expresión:

A. A*A.

B. A.^2.

C. A^2.

6. En cálculo simbólico, ¿para qué se utiliza la instrucción `solve`?

A. Para resolver ecuaciones.

B. Para resolver sistemas de ecuaciones.

C. Las dos respuestas anteriores son correctas.

7. ¿Cuándo salimos de un bucle *while*?

A. Cuando la condición deja de cumplirse.

B. Cuando se llega al último elemento del vector de índices.

C. Todas las anteriores son correctas.

8. ¿Cuántas componentes de salida debe tener como mínimo una función?

A. 1.

B. 2.

C. Puede no tener ninguna componente de salida.

9. Indica qué saldrá por pantalla al ejecutar

```
>> 3+5;
```

A. Nada.

B. 8.

C. 3+5.

10. ¿Cómo accedemos a la segunda columna de una matriz A?

A. A(2,:)

B. A(:,2)

C. A(2;:)