Practica-1.pdf

manu_martinez03

Fundamentos de Control

2º Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Escuela de Ingenierías Industriales
Universidad de Málaga

Introducción a MATLAB

Matrices y vectores

Table of Contents

Matrices y vectores....1
1. El entorno MATLAB....1
2. Matrices y vectores....2
2.1 Definición de matrices....3
Matrices de uso frecuente....4
2.2 Definiendo vectores....5
2.3 Trabajando con matrices....5
Traspuesta de una matriz....5
Aritmétrica de matrices....6
La versatilidad tiene operador, dos puntos :....7
Un comando muy útil: find....9

MATLAB fue inicialmente diseñado como una herramienta para facilitar el cálculo matricial, de hecho el nombre MATLAB deriva de “MATrix LABoratory” (Laboratorio de Matrices).

Hoy día, MATLAB es un sistema interactivo y un lenguaje de programación de carácter científico y técnico que es utilizado, con éxito, tanto en el ámbito académico como en el ámbito industrial.

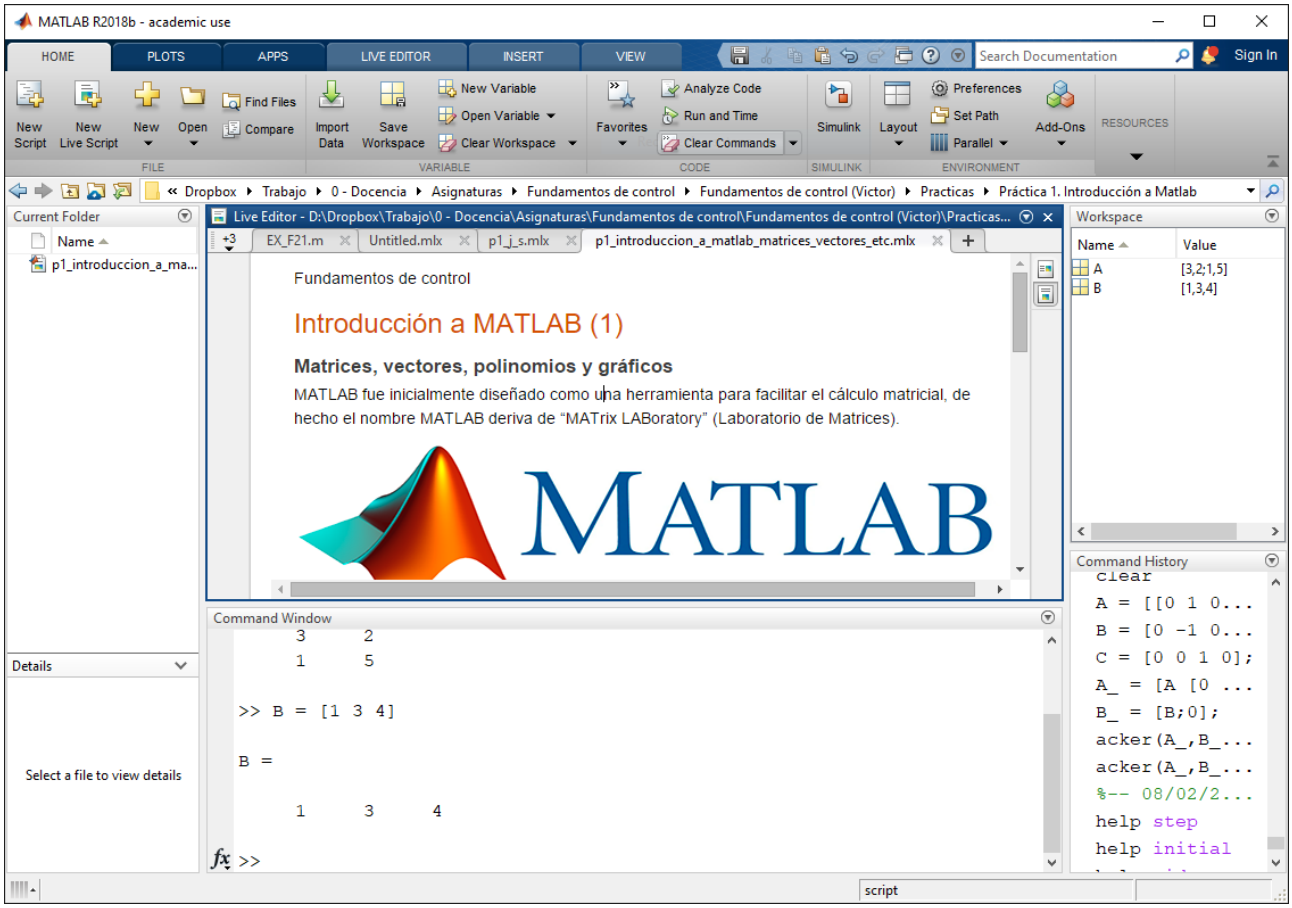
Las características principales de MATLAB son:

- Orientado al cálculo matemático y científico.
- Uso de las matrices como elemento básico.
- Potencia de representación gráfica.
- Sistema abierto.
- Facilidad de uso.
- Lenguaje de programación.
- Aplicable a multitud de campos (versátil).

1. El entorno MATLAB

La ventana de MATLAB se encuentra dividida en cuatro partes fundamentales (subventanas) que inicialmente muestran la siguiente información:

- Subventana izquierda: muestra el conteido del **directorio de trabajo actual** y, en la parte inferior, detalles del fichero que tengamos seleccionado (versión del fichero, autor, etc.).
- Subventana central-arriba: contiene el **editor de código** (ficheros con extensión *.m*) o de scripts vivos (*live scripts*, extensión *.mlx*).
- Subventana central-abajo: **ventana de comandos** (*Command Window*) donde se introducen los comandos propios de MATLAB. Esta ventana tiene capacidad de "memorizar" los comandos que han sido introducidos con anterioridad y consultarlos pulsando la tecla arriba.
- Subventana superior derecha: es el denominado **espacio de trabajo** (*Workspace*) y en ella aparecen todas las variables que se han utilizado en la ventana de comandos. Esta ventana tiene la capacidad de crear nuevas variables, eliminar variables y de visualizar y editar los valores de las variables existentes.



Además, en la parte superior se muestra la ruta del directorio de trabajo actual, la cual puede cambiarse para guardar y ejecutar comandos desde una carpeta definida por el propio usuario.

Una vez nos han presentado el entorno, vamos a empezar a jugar con él.

Pista: para cambiar esta distribución de ventanas inicial, o mostrar algunas ocultas por defecto (por ejemplo, el historial de comandos, *Command history*), puede emplearse la opción *Layout* dentro del menú *Home*.

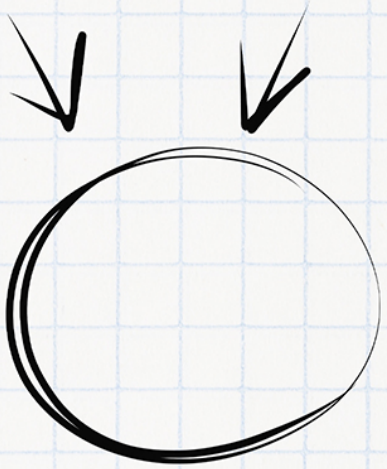
2. Matrices y vectores

Imagínate aprobando el examen

Necesitas tiempo y concentración

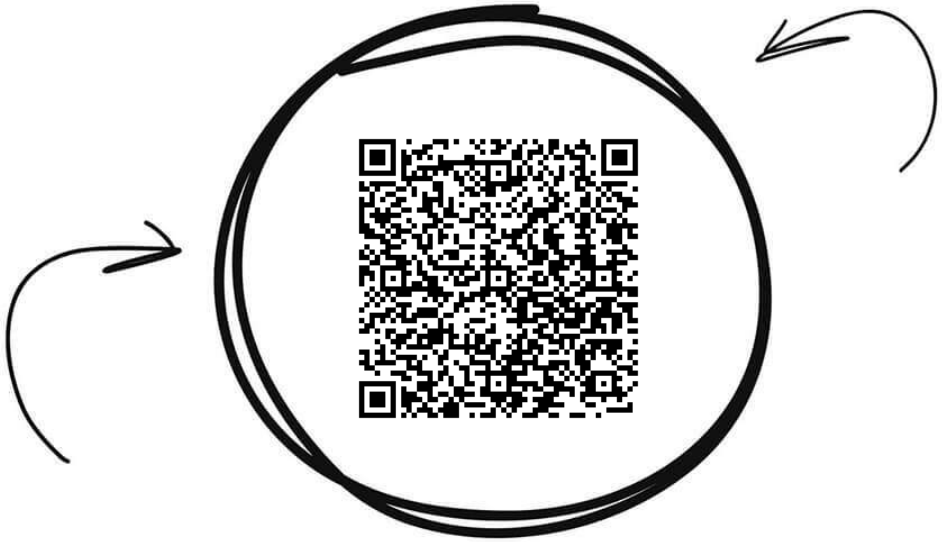
Planes	PLAN TURBO	PLAN PRO	PLAN PRO+
Descargas sin publi al mes	10	40	80
Elimina el video entre descargas	✓	✓	✓
Descarga carpetas	✗	✓	✓
Descarga archivos grandes	✗	✓	✓
Visualiza apuntes online sin publi	✗	✓	✓
Elimina toda la publi web	✗	✗	✓
Precios Anual	0,99 € / mes	3,99 € / mes	7,99 € / mes

Ahora que puedes conseguirlo, ¿Qué nota vas a sacar?



WUOLAH

Fundamentos de Control



Banco de apuntes de la

WUOLAH

Comparte estos flyers en tu clase y consigue más dinero y recompensas

1. Imprime esta hoja
2. Recorta por la mitad
3. Coloca en un lugar visible para que tus compis puedan escanar y acceder a apuntes
4. Llévate dinero por cada descarga de los documentos descargados a través de tu QR

El elemento básico o primitiva de trabajo de MATLAB son las **matrices**. Para MATLAB, por defecto, cualquier variable es considerada como una matriz rectangular, y esta no necesita ser dimensionada previamente para ser usada. La mayoría de las funciones de MATLAB están diseñadas para operar directamente con matrices.

Los elementos de una matriz pueden ser números enteros, reales, complejos, o expresiones matemáticas, entre otras muchas cosas.

2.1 Definición de matrices

Las matrices en MATLAB pueden ser creadas de diversas formas, pero la más común es emplear una lista explicita de elementos. Por ejemplo:

```
A = [2 4 5; 1 3 8]
```

```
A =

     2     4     5
     1     3     8
```

Pista: si una operación de asignación (=) no incluye un punto y coma al final (;), MATLAB mostrará el contenido de la variable asignada tras ejecutarla.Si lo incluye, a esto se le denomina modo *no echo*.

Esta creación tiene varios aspectos a destacar:

- La matriz se denomina `A`,
- se incluye un símbolo de asignación `=`,
- la definición de la matriz comienza con un corchete `[`,
- cada columna se separa por una coma `,`,
- cada fila se separa empleando un punto y coma `;`, y
- la definición de la matriz finaliza con un corchete `]`.

MATLAB utiliza la notación matemática para referenciar un elemento de una matriz. Es decir, para referenciar el elemento de la fila `i` y la columna `j` de la matriz `A` se usaría la expresión `A(i,j)`.

Tarea 1: Crea la matriz `A` tal que:

$$A = \begin{bmatrix} -1 & 2 & 3 & -4 \\ 5 & 6 & -7 & 8 \\ 9 & -10 & 11 & 12 \\ 13 & -14 & -15 & 16 \end{bmatrix}$$

y consulta su elemento en la fila 2 y columna 3.

```
% Tu código aquí, para ejecutarlo puedes pulsar el botón Run o usar la
% combinación de teclas Ctrl+Enter
A=[-1 2 3 -4; 5 6 -7 8; 9 -10 11 12; 13 -14 -15 16]
```

```
A = 4×4
    -1     2     3    -4
     5     6    -7     8
     9   -10    11    12
```

```
    13   -14   -15    16
```

```
elemento=A(2,3)
```

```
elemento = -7
```

Tarea 2: Crea la matriz B tal que:

$$B = \begin{bmatrix} 1+i & 2+2i \\ 3+3i & 4+4i \end{bmatrix}$$

y consulta su elemento en la fila 2 y columna 1.

Pista: En MATLAB la unidad imaginaria básica puede representarse empleando `i` o `j` indistintamente.

```
% Tu código aquí
B=[1+i 2+2*i; 3+3*i 4+4*i]
```

```
B = 2×2
    10    20
    30    40
```

```
elemento=B(2,1)
```

```
elemento = 30
```

```

```

Matrices de uso frecuente

MATLAB incorpora una serie de comandos para crear matrices de uso frecuente. En las siguientes celdas de código puedes jugar con ellas. Fijaté en los parámetros de entrada:

```
% Matriz de ceros de dimensión 2x2
M = zeros(2)
```

```
M = 2×2
     0     0
     0     0
```

```
% Matriz de ceros de dimensión 2x3
M = zeros(2,3)
```

```
M = 2×3
     0     0     0
     0     0     0
```

```
% Matriz de unos (identidad) de dimensión 3x3
M = eye(3)
```

```
M = 3×3
     1     0     0
     0     1     0
```


```
     0     0     1
```

```
% Matriz de números aleatorios (distribución uniforme)
M = rand(3)
```

```
M = 3×3
    0.2238    0.5060    0.9593
    0.7513    0.6991    0.5472
    0.2551    0.8909    0.1386
```

```
% Matriz de números aleatorios (distribución normal)
M = randn(3)
```

```
M = 3×3
   -0.7423   -0.6156    0.8886
   -1.0616    0.7481   -0.7648
    2.3505   -0.1924   -1.4023
```

2.2 Definiendo vectores

Partiendo de la base de que la primitiva de trabajo es la matriz, MATLAB también puede trabajar con **escalares** (matrices 1x1) y con **vectores fila** (matrices 1xn) o **columna** (matrices nx1).

Tarea 3: Crea el vector columna v tal que:

$$v = \begin{bmatrix} 1 \\ -2 \\ 3 \\ 4 \end{bmatrix}$$

```
% Tu código aquí
v=[1; -2;  3; 4]
```

```
v = 4×1
     1
    -2
     3
     4
```

```

```

2.3 Trabajando con matrices

Traspuesta de una matriz

En MATALAB la traspuseta de una matriz se obtiene con el operador tilde `'`. Por ejemplo:

```
A'

ans =

     2     1
     4     3
```

```
     5     8

B = [1 2 4]'

B =

     1
     2
     4
```

Tarea 4: Calcular, a partir de la matriz `A` previamente definida, una nueva matriz `C`, tal que: $C = A^T$

```
% Tu código aquí
C=A'
```

```
C = 4×4
    -1     5     9    13
     2     6   -10   -14
     3    -7    11   -15
    -4     8    12    16
```

```
```

Aritmétrica de matrices

Los siguientes comandos implementan operaciones aritmétricas básicas:

- +: Suma de matrices.
- -: Resta de matrices.
- *: Producto de matrices.
- /: División matricial por la derecha (A/B equivale a $A \times B^{-1}$) .
- \: División matricial por la izquierda ($A\backslash B$ equivale a $A^{-1} \times B$).
- ^: Operador potencia.
- `inv`: inversa de una matriz.
- ': traspuesta de una matriz.

Tarea 5: Calcular, a partir de la matriz `A` y el vector `v` previamente definidos, el producto: $C = A \times v$. ¿Qué pasaría si intento realizar el producto $C = v \times A$? ¿Por qué?

```
% Tu código aquí
C=A*v
```

```
C = 4×1
   -12
     4
   110
```

```
    60
```

```
%no puede realizarse ya que las dimensiones de ambas matrices no coinciden,
%por lo cual el priducto no puede llevarse a cabo
```

Tarea 6: Calcular, a partir de la matriz `A` y el vector `v`, una nuevas matrices `x1` y `x2`, tales que:

$$\begin{cases} A * \mathrm{x1} = v \\ \mathrm{x2} * A = v' \end{cases}$$

```
% Tu código aquí
x1= inv(A)*v
```

```
x1 = 4×1
    0.7508
   -0.2222
    0.0303
   -0.5261
```

```
x2= v' * inv(A)
```

```
x2 = 1×4
   -0.9091    0.0000    0.2727   -0.1818
```

Tarea 7: Calcular, a partir de la matriz `A`, una nueva matriz `AA`, tal que: $\mathrm{AA} = A^2$.

Pista: Usa el operador potencia

```
% Tu código aquí
AA=A^2
```

```
AA = 4×4
   -14    36    76    -8
    66     4  -224    72
   196  -320    38   208
   -10  -132  -268   -88
```

La versatilidad tiene operador, dos puntos :

El operador dos puntos **:** es uno de los operadores más versátiles e importantes de MATLAB, ya que permite definir vectores, referencias, submatrices, etc.

Por ejemplo, puede definir un vector que posea los valores desde 1 hasta 10 de forma compacta:

```
x = 1:10
```

```
x =

     1     2     3     4     5     6     7     8     9    10
```

Nótese que la sintaxis resulta `vector = inicio:fin`. Si se desea que el paso sea disinto de 1, dicho paso se puede introducir entre los valores de inicio y de fin: `vector = inicio:paso:fin`. Por ejemplo:

```
x =

     1     3     5     7     9
```

Tarea 8. Crea un vector u que contenga toda la serie de valores reales comprendidos en el intervalo $[0, 10]$ que resultan de la consideración de incrementos de una décima entre cada dos elementos consecutivos.

```
% Tu código aquí
u=0:0.1:10
```

```
u = 1×101
         0    0.1000    0.2000    0.3000    0.4000    0.5000    0.6000    0.7000 ···
```

```

```

Como se ha introducido, el operador dos puntos **:** también puede emplearse para referenciar parte de una matriz. Por ejemplo, el siguiente código extrae la sumatriz compuesta por las dos primeras filas y columnas de `D` y la almacena en una nueva matriz `S1`:

```
D =

     2     4     5
     1     3     8

S1 = D(1:2,1:2)

S1 =

     2     4
     1     3
```

Otra manera de referenciar parte de una matriz es indicar directamente las filas o columnas a usar. En cualquier caso, el uso del operador dos puntos sin emplear inicio o fin indica que se desea referenciar todas las filas o columnas. Ejemplo:

```
S1 = D(:,[1 3])

S1 =

     2     5
     1     8
```

Tarea 9: Obtener una matriz `C` constituida por las filas 1 y 4 de la matriz `A`.

```
% Tu código aquí
C=A([1 4],:)
```

```
C = 2×4
    -1     2     3    -4
```


```
    13   -14   -15    16
```

Además de para asignar a una matriz `C` una submatriz de otra matriz `A`, también se puede usar la referncia a fila o columnas para cambiar el valor de dicha submatriz referenciada, incluso para eliminarla. Por ejemplo, el siguiente código cambia el valor de la segunda fila de la matriz `D`:

```
D =

     1     3     5
     2     4     6

D(2,:) = [3 8 3]

D =

     1     3     5
     3     8     3
```

y el siguiente elimina dicha fila:

```
D(2,:) = []

D =

     1     3     5
```

Tarea 10: Obtener la matriz resultante de la eliminación de la columna 2 de la matriz `C`.

```
% Tu código aquí
C(:,2)=[]
```

```
C = 2×3
    -1     3    -4
    13   -15    16
```

Un comando muy útil: find

El comando **`find`** permite devolver los índices de los elementos de una matriz que cumpan con una cierta condición. Esta condición puede incluir comparadores como <, <=, >, >= o ==. Los índices siguen un orden resultado de concatenar las columnas de la matriz. Por ejemplo, los elementos de una matriz `D` que sean mayores de 3:

```
D =

     5     3     2
     2     4     6

idx = find(D>3)
```

```
idx =

     1
     4
     6
```

Este comando acepta un segundo argumento que permite escificar el número de elementos máximos que queremos en la respuesta. Por ejemplo si sólo estamos interesados en los dos primeros índices que cumplan la condición:

```
idx = find (D > 3, 2)

idx =

     1
     4
```

Los índices devueltos por **`find`** se pueden usar para referenciar dichos elementos en la matriz `D` y modificarlos. Por ejemplo:

```
D(idx) = 9

D =

     9     3     2
     2     4     9
```

Si en vez de los índices que ocupan los elementos que cumplen con la condición fijada, estuvieramos interesados en la fila y columna que estos ocupan, podríamos obtenerlos empleando la siguiente sintaxis en la llamada a **`find`**:

```
D =

     5     3     2
     2     4     6

[row,col] = find (D > 3, 2)

row =

     1
     2


col =

     1
     2
```

Nótese que `row` and `col` son vectores columna que almacenan la fila y columna donde aparecen los dos primeros elementos en `D` que cumplen con la condición.

Tarea 11: Calcular, a partir de la matriz `A`, una nueva matriz `D` que contenga un cero en el lugar en el que la matriz `A` contenga un número par.

Pista: MATLAB provee el comando **`mod(a,b)`** que devuelve el módulo de dividir `a` entre `b`.

```
% Tu código aquí
idx=find(mod(A,2)==0);
A(idx)=0;
D=A
```

```
D = 4×4
    -1     0     3     0
     5     0    -7     0
     9     0    11     0
    13     0   -15     0
```

```

```

Tarea 12: Calcular, a partir de la matriz `A`, una nueva matriz `E` constituida por las filas completas de la matriz `A` que contengan múltiplos de `6`.

```
% Tu código aquí
A=[-1 2 3 -4; 5 6 -7 8; 9 -10 11 12; 13 -14 -15 16];
[row,col]=find(mod(A,6)==0);
E=A(row,:)
```

```
E = 2×4
     5     6    -7     8
     9   -10    11    12
```

```

```

Introducción a MATLAB

Manejo de polinomios

Table of Contents

Manejo de polinomios....1
1. Trabajando con polinomios....1
1.1 Evaluando polinomios....1
1.2 Obteniendo las raíces de un polinomio....2
1.3 Multiplicando polinomios....3
1.4 Obteniéndo el polinomio característico de una matriz....4
[BOMBA] El comando salvador: help....5
1.5 División, cociente y resto....5

A lo largo de nuestra vida académica hemos ido acumulando experiencia con **polinomios**: expresiones algebraicas en las que intervienen números (coeficientes) y letras (variables) que están relacionadas mediante operaciones como sumas, multiplicaciones y/o potencias. Los polinomios se definen en MATLAB mediante vectores fila con los coeficientes del polinomio en cuestión en orden de potencias decrecientes. Por ejemplo, si queremos definir el polinomio $p(x) = x^3 - 2x - 5$ se procedería como sigue:

```
% p = x^3 - 2*x - 5
% Polinomio de tercer grado, con lo que:
% Coeficiente que acompaña a x^3: 1
% Coeficiente que acompaña a x^2: 0
% Coeficiente que acompaña a x^1: -2
% Término independiente: -5
% Resultando en la definición:
p = [1 0 -2 -5]

p =

     1     0    -2    -5
```

De manera general, el polinomio $p(x) = p_2x^2 + p_1x + p_0$ se representa en Matlab como:

```
p = [p2 p1 p0]
```

Una vez presentados nuestros protagonistas, ¡trabajemos un poco con ellos!

1. Trabajando con polinomios

1.1 Evaluando polinomios

Para evaluar un polinomio en un cierto punto MATLAB nos ofrece el comando **`polyval`**. Por ejemplo, $p(4)$ se puede calular como:

```
res = polyval(p,4)

res =
```

```
          51
```

Alternativamente, también se puede evaluar un polinomio desde el punto de vista matricial usando **`polyvalm`**. Por ejemplo, el polinomio $p(x) = x^3 - 2x - 5$ se convertiría en la expresión matricial $p(X) = X^3 - 2X - 5I$, dónde X es una matriz cuadrada y I es la matriz identidad. Por ejemplo:

```
p = [1 0 -2 -6];
X = [3 -2; 0 3];
Y = polyvalm(p,X)

Y =

    15   -50
     0    15
```

Tarea 1: Definir el polinomio $q(x) = -2x^5 + 3x^3 - 2x + 5$ y evaluarlo en $x = 2$.

```
% Tu código aquí
q=[-2 0 3 0 -2 5]
```

```
q = 1×6
    -2     0     3     0    -2     5
```

```
x=polyval(q,2)
```

```
x = -39
```

```

```

Resultado esperado: `q_x = -39`

1.2 Obteniendo las raíces de un polinomio

Las raíces de un polinomio pueden calcularse empleando el comando **`roots`**. Por ejemplo:

```
r = roots(p)

r =

   2.0946 + 0.0000i
  -1.0473 + 1.1359i
  -1.0473 - 1.1359i
```

Tarea 2: Calcula las raíces del polinomio $q(x)$ previamente definido.

```
% Tu código aquí

r=roots(q)
```

```
r = 5×1 complex
  -1.1927 + 0.6181i
  -1.1927 - 0.6181i
```

```
1.3854 + 0.0000i
0.5000 + 0.8660i
0.5000 - 0.8660i
```

1.3 Multiplicando polinomios

Si estamos trabajando con dos polinomios, estos pueden multiplicarse empleando el comando **conv**. Por ejemplo, si se desea multiplicar el ya conocido polinomio $p(x) = x^3 - 2x - 5$ por $p_2(x) = 4x^3 + 5x + 6$

```
p2 = [4 5 6]

p2 =

     4     5     6

p3 = conv(p,p2)

p3 =

     4     5    -2   -30   -37   -30
```

Lo que da como resultado el polinomio $p_3(x) = 4x^5 + 5x^4 - 2x^3 - 30x^2 - 37x - 30.$

Este comando también es útil para construir un polinomio a partir de su factorización. Por ejemplo si $q(x) = (x-2)(x+3)(x+5i)(x-5i)$podemos obtener el polinomio del que proviene dicha factorización como sigue:

```
q = conv(conv(conv([1 -2],[1 3]),[1 5i]),[1 -5i])

q =

     1     1    19    25  -150
```

Lo que da como resutlado $q(x) = x^4 + x^3 + 19x^2 + 25x - 150.$

Nota: Nótese que la convolución de dos vectores es equivalente a la multiplicación si estos representan coeficientes polinomiales.

Tarea 3: Obtén el polinomio $q(x)$ resultante de la factorización $q(x) = (x+3)(x+2+3i)(x+2-3i)$. Comprueba que el polinomio obtenido es el correcto calulando sus raíces y comparándolas con la factorización inicial.

```
% Tu código aquí
q=conv([1 3],conv([1 2+3*i],[1 2-3*i]))
```

```
q = 1×4
     1     7    25    39
```

```
r=roots(q)
```

```
r = 3×1 complex
  -2.0000 + 3.0000i
  -2.0000 - 3.0000i
  -3.0000 + 0.0000i
```

1.4 Obteniéndo el polinomio característico de una matriz

El polinomio característico de una matriz se calcula como $p_A(\lambda) = \det(A - \lambda I)$. En álgebra, este polinomio característico es una herramienta ampliamente utilizada ya que provee gran cantidad de información sobre la matriz, como por ejemplo los valores propios, el determinante y su traza.

En MATLAB el polinomio característico de una matriz puede obtenerse con el comando **`poly`**. Por ejemplo:

```
A =

     2     3     4
     4     5     2
     1     8     6

pc = poly(A)

pc =

    1.0000  -13.0000   20.0000  -70.0000
```

Tarea 4: Calcular, a partir de la matriz `A=[2 4 3; -1 3 2; 8 3 -9]`, un vector p constituido por los coeficientes del polinomio característico de la matriz `A`.

```
% Tu código aquí
A=[2 4 3; -1 3 2; 8 3 -9]
```

```
A = 3×3
     2     4     3
    -1     3     2
     8     3    -9
```

```
pc=poly(A)
```

```
pc = 1×4
    1.0000    4.0000  -65.0000  119.0000
```

```

```

Resultado esperado:

```
pc = 1×4
     1.0000    4.0000  -65.0000  119.0000
```

Tarea 5: El comando **`poly`** tiene un segundo uso,interesante. Para ilustrarlo, calcular, a partir del vector `p = [1 0 -2 -5]`, un nuevo vector `r` constituido por las raíces de dicho polinomio.

```
% Tu código aquí
p = [1 0 -2 -5]
```

```
p = 1×4
     1     0    -2    -5
```

```
r=roots(p)
```

```
r = 3×1 complex
   2.0946 + 0.0000i
  -1.0473 + 1.1359i
  -1.0473 - 1.1359i
```

poly permite obtener el vector de coeficientes de un polinomio a partir de un vector que contiene sus raíces. Por ejemplo, empleando las raíces de nuestro polinomio de ejemplo p:

```
p4 = poly(r)

p4 =

    1.0000   -0.0000   -2.0000   -5.0000
```

De este modo, los comandos **poly** y **roots** pueden verse como inversos: con **roots** calculo raíces que puedo convertir en polinomio con **poly**, y viceversa.

Tarea 6: Calcular, a partir del vector r, un nuevo vector p4 constituido por los coeficientes de un polinomio cuyas raíces son los elementos del vector r. Comprobar que el vector p4 obtenido coincide con el vector p calculado anteriormente.

```
% Tu código aquí
p4=poly(r)
```

```
p4 = 1×4
    1.0000   -0.0000   -2.0000   -5.0000
```

[BOMBA] El comando salvador: help

Si tenemos dudas sobre el uso de algún comando o función, podemos emplear el comando **help** para obtener ayuda sobre el mismo. Por ejemplo, ejecuta la siguiente celda de código para revisar la documentación relativa al comando **poly**:

```
help poly
```

Si seguimos teniendo dudas, vistar la documentación del comando puede ser útil.

1.5 División, cociente y resto

El cociente y el resto de la división de dos polinomios puede obtenerse mediante el comando **deconv**. Por ejemplo:

```
[q,r] = deconv(p,p2)
```


```
q =

    0.2500   -0.3125


r =

         0         0   -1.9375   -3.1250
```

Tarea 7: Calcular, a partir de los vectores `p = [1 0 -2 -5]` y `p2 = [1 4 5 6]`, unos nuevos vectores `coc` y `res` constituidos, respectivamente, por los coeficientes de los polinomios cociente y resto de la división del polinomio cuyos coeficientes son los términos del vector `p2` entre el polinomio cuyos coeficientes son los términos del vector `p`.

¿Un poco de ayuda sobre **deconv**?

```
help deconv
```

```
  deconv Deconvolution and polynomial division.
     [Q,R] = deconv(B,A) deconvolves vector A out of vector B. The result
     is returned in vector Q and the remainder in vector R.

     The outputs satisfy B = conv(A,Q) + R when length(A) <= length(B);
     otherwise, Q = 0 and R = B. With K = min(length(A),length(B)), these
     two cases can be written as B = conv(A(1:K),Q) + R.

     If A and B are vectors of polynomial coefficients, deconvolution is
     equivalent to polynomial division. The result of dividing B by A is
     quotient Q and remainder R.

     Class support for inputs B,A:
        float: double, single

     See also conv, residue.

     Documentation for deconv
```

```
% Tu código aquí
p = [1 0 -2 -5]
```

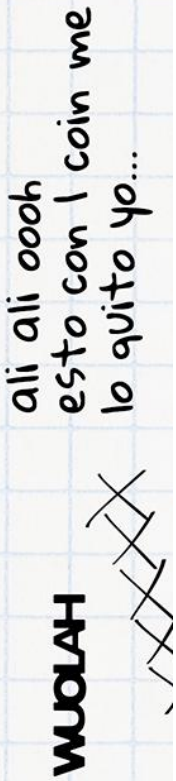
```
p = 1×4
     1     0    -2    -5
```

```
p2 = [1 4 5 6]
```

```
p2 = 1×4
     1     4     5     6
```

```
[coc,res]=deconv(p2,p)
```

```
coc = 1
res = 1×4
     0     4     7    11
```



Comprueba que el resultado es correcto calculando el producto del cociente por p y sumandole el resto.

```
% Tu código aquí

p2=coc*p + res
```

```
p2 = 1×4
     1     4     5     6
```

Resultado esperado:

```
p2 = 1×4
     1     4     5     6
```

Introducción a MATLAB

Funciones matemáticas top

Table of Contents

Funciones matemáticas top....1
1.Biblioteca de funciones predefinidas....1
2. Variables predefinidas/Constantes....3

1.Biblioteca de funciones predefinidas

Para facilitarnos la vida MATLAB incorpora una serie de funciones matemáticas. Vamos a revisar algunas de las más usadas. Algunas os sonarán de scripts anteriores, como **`find`**, **`conv`** o **`deconv`**:

Operaciones con matrices/números:

- **`abs:`** Si se aplica a un número real calcula su valor absoluto y si se aplica a un número complejo calcula su módulo.
- **`all:`** Devuelve verdad (uno) si todos los elementos del vector al que se aplica esta función son verdad (distintos de cero).
- **`any:`** Devuelve verdad (uno) si algún elemento del vector al que se aplica esta función es verdad (distinto de cero).
- **`exp:`** Aplica la función exponencial.
- **`find:`** Devuelve un vector con los índices de todos los elementos no nulos del vector al que se aplica. find(a>100) devuelve los índices de todos **`los`** elementos del vector a que son mayores que 100.
- **`log:`** Calcula el logaritmo neperiano.
- **`log10:`** Calcula el logaritmo decimal.
- **`max:`** Calcula el valor máximo de los elementos de un vector. Si se desea, permite conocer en qué posición del vector se encuentra el elemento de valor máximo.
- **`min:`** Calcula el valor mínimo de los elementos de un vector. Si se desea, permite conocer en qué posición del vector se encuentra el elemento de valor mínimo.
- **`mod / rem:`** Calcula el resto de una división.
- **`sqrt:`** Calcula la raíz cuadrada.

Operaciones con polinomios:

- **`conv:`** Calcula el producto de dos polinomios.
- **`deconv:`** Calcula el cociente y el resto de la división de dos polinomios.
- **`poly:`** Genera el polinomio característico de una matriz. Genera el vector de coeficientes de un polinomio a partir de un vector que contiene sus raíces.
- **`roots:`** Calcula las raíces de un polinomio a partir de su vector de coeficientes.

Trabajando con funciones:

- **`fminsearch:`** Calcula el valor de la variable independiente para el cual una función matemática alcanza el valor mínimo. Hay que especificarle como parámetro de entrada el punto en el que empieza a buscar el mínimo.
- **`quad:`** Obtiene la integral definida de una función matemática por aproximación cuadrática (método de Simpson). La función considerada debe indicarse entre comillas simples.
- **`quadl:`** Obtiene la integral definida de una función matemática por aproximación cuadrática (método de Lobatto). La función considerada debe indicarse entre comillas simples.

Trigonometría:

- **`cos:`** Calcula el coseno de una cantidad expresada en radianes.
- **`sin:`** Calcula el seno de una cantidad expresada en radianes.

Tarea 1: Calcular el seno de 60º.

Pista: MATLAB, en su eterna sabiduría, nos provee de los comandos **rad2deg** y **deg2rad**. Puedes emplear **help** para comprobar como se comportan.

```
% Tu código aquí
help deg2rad
```

```
  deg2rad Convert angles from degrees to radians.
    deg2rad(X) converts angle units from degrees to radians for each
    element of X.

    See also rad2deg.

    Documentation for deg2rad
    Other uses of deg2rad
```

```
rad=deg2rad(60)
```

```
rad = 1.0472
```

```
s_rad=sin(rad)
```

```
s_rad = 0.8660
```

```

```

Resultado esperado:

```
s_rad = 0.8660
```

Tarea 2: El siguiente código define una función `fun` con dos variables independientes `x(1)` y `x(2)`, y un punto de partida `x0`. Calcula cual es el valor de dichas variables para el cual la función alcanza el valor mínimo.

```
fun = @(x)100*(x(2) - x(1)^2)^2 + (1 - x(1))^2;
x0 = [-1.2,1];
% Tu código aquí
```


```
fminsearch(fun,x0)
```

```
ans = 1×2
    1.0000    1.0000
```

Resultado esperado:

```
x = 1×2
      1.0000    1.0000
```

2. Variables predefinidas/Constantes

MATLAB también incorpora una serie de variables predefinidas que se podrían interpretar como valores constantes, y que pueden ser directamente añadidos a expresiones. Juega un poco con ellas ejecutando estas celdas de código:

```
% La famosa constante pi
pi
% Valor infinito
Inf
% Valor no un número (not a number)
NaN
% Parte imaginaria de un número complejo
1i
% Parte imaginaria de un número complejo
1j
```

Tarea 3: Calcula la circunferencia de un círculo que tiene como diámetro 4 metros, empleando la constante `pi`.

```
% Tu código aquí
diametro=4
```

```
diametro = 4
```

```
c=diametro*pi
```

```
c = 12.5664
```

Resultado esperado: `c = 12.5664`

Introducción a MATLAB

Gráficos

Table of Contents

Gráficos 1
1. Uno para gobernarlos a todos: plot 1
Algunas funcionalidades interesantes 7
2. Subgráficas 9
3. Control sobre los ejes 13

Uno de los puntos fuertes de MATLAB es el amplio abanico de posibilidades que ofrece a la hora de mostrar visualmente, en forma de gráficas, los datos con los que se está trabajando. Vamos a ver alguna de las opciones más comunes, ya que el estudio de la totalidad de funcionalidades ofrecidas requeriría de una asignatura completa para ello. A lo largo de la asignatura iremos descubriendo más posibilidades.

1. Uno para gobernarlos a todos: plot

El comando **plot** se postula como uno de los más usados a la hora de generar gráficos bidimensionales. Veamos su documentación:

```
help plot
```

```
plot   Linear plot.
   plot(X,Y) plots vector Y versus vector X. If X or Y is a matrix,
   then the vector is plotted versus the rows or columns of the matrix,
   whichever line up.  If X is a scalar and Y is a vector, disconnected
   line objects are created and plotted as discrete points vertically at
   X.

   plot(Y) plots the columns of Y versus their index.
   If Y is complex, plot(Y) is equivalent to plot(real(Y),imag(Y)).
   In all other uses of plot, the imaginary part is ignored.

   Various line types, plot symbols and colors may be obtained with
   plot(X,Y,S) where S is a character string made from one element
   from any or all the following 3 columns:

          b     blue          .     point              -     solid
          g     green         o     circle             :     dotted
          r     red           x     x-mark             -.    dashdot
          c     cyan          +     plus               --    dashed
          m     magenta       *     star             (none)  no line
          y     yellow        s     square
          k     black         d     diamond
          w     white         v     triangle (down)
                              ^     triangle (up)
                              <     triangle (left)
                              >     triangle (right)
                              p     pentagram
                              h     hexagram

   For example, plot(X,Y,'c+:') plots a cyan dotted line with a plus
   at each data point; plot(X,Y,'bd') plots blue diamond at each data
```

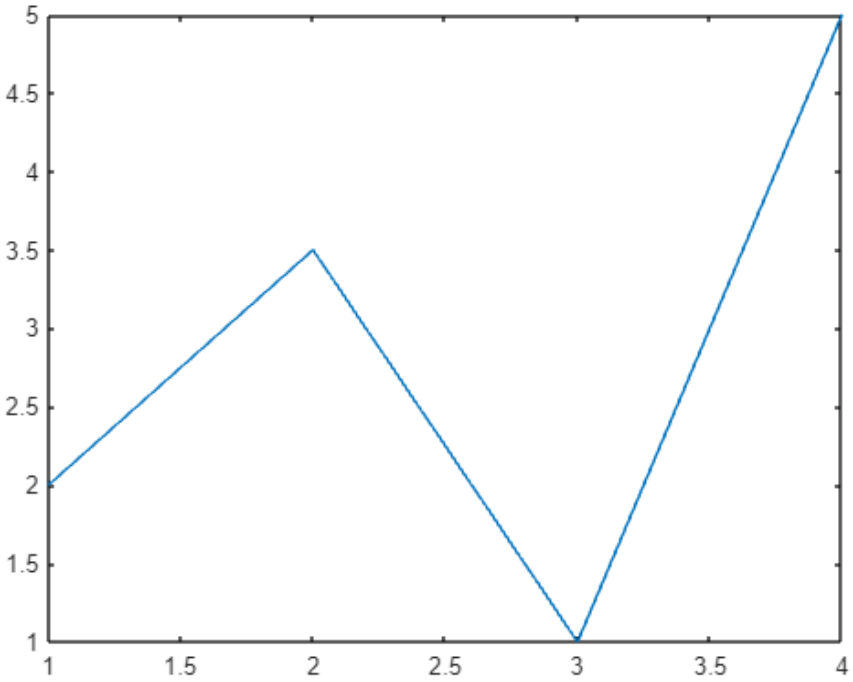
```
point but does not draw any line.

plot(TBL,XVAR,YVAR) plots the variables xvar and yvar from the table
tbl. To plot one data set, specify one variable for xvar and one
variable for yvar. To plot multiple data sets, specify multiple
variables for xvar, yvar, or both. If both arguments specify multiple
variables, they must specify the same number of variables

plot(TBL,YVAR) plots the specified variable from the table against the
row indices in the table. If the table is a timetable, the specified
variable is plotted against the row times from the timetable.

plot(X1,Y1,S1,X2,Y2,S2,X3,Y3,S3,...) combines the plots defined by
the (X,Y,S) triples, where the X's and Y's are vectors or matrices
and the S's are strings.

For example, plot(X,Y,'y-',X,Y,'go') plots the data twice, with a
solid yellow line interpolating green circles at the data points.

The plot command, if no color is specified, makes automatic use of
the colors specified by the axes ColorOrder property.  By default,
plot cycles through the colors in the ColorOrder property.  For
monochrome systems, plot cycles over the axes LineStyleOrder property.

Note that RGB colors in the ColorOrder property may differ from
similarly-named colors in the (X,Y,S) triples.  For example, the
second axes ColorOrder property is medium green with RGB [0 .5 0],
while plot(X,Y,'g') plots a green line with RGB [0 1 0].

If you do not specify a marker type, plot uses no marker.
If you do not specify a line style, plot uses a solid line.

plot(AX,...) plots into the axes with handle AX.

plot returns a column vector of handles to lineseries objects, one
handle per plotted line.

The X,Y pairs, or X,Y,S triples, can be followed by
parameter/value pairs to specify additional properties
of the lines. For example, plot(X,Y,'LineWidth',2,'Color',[.6 0 0])
will create a plot with a dark red line width of 2 points.

Example
   x = -pi:pi/10:pi;
   y = tan(sin(x)) - sin(tan(x));
   plot(x,y,'--rs','LineWidth',2,...
                   'MarkerEdgeColor','k',...
                   'MarkerFaceColor','g',...
                   'MarkerSize',10)

See also plottools, semilogx, semilogy, loglog, plotyy, plot3, grid,
title, xlabel, ylabel, axis, axes, hold, legend, subplot, scatter.

Documentation for plot
Other uses of plot
```

Como podemos comprobar permite, entre otras cosas:

- dibujar las columnas de un vector Y frente a sus índices (orden en el que aparecen en el vector, comenzando por el índice 1):

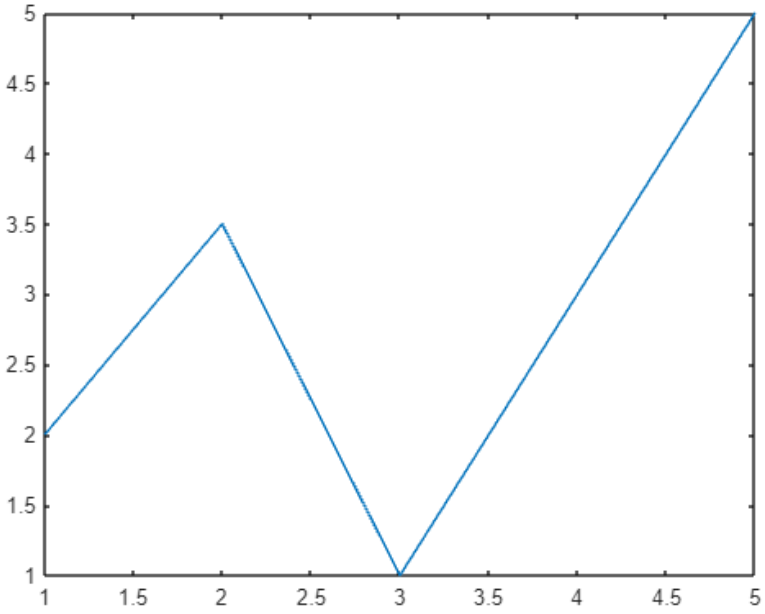
```
Y = [2 3.5 1 5];
```

```
plot(Y)
```



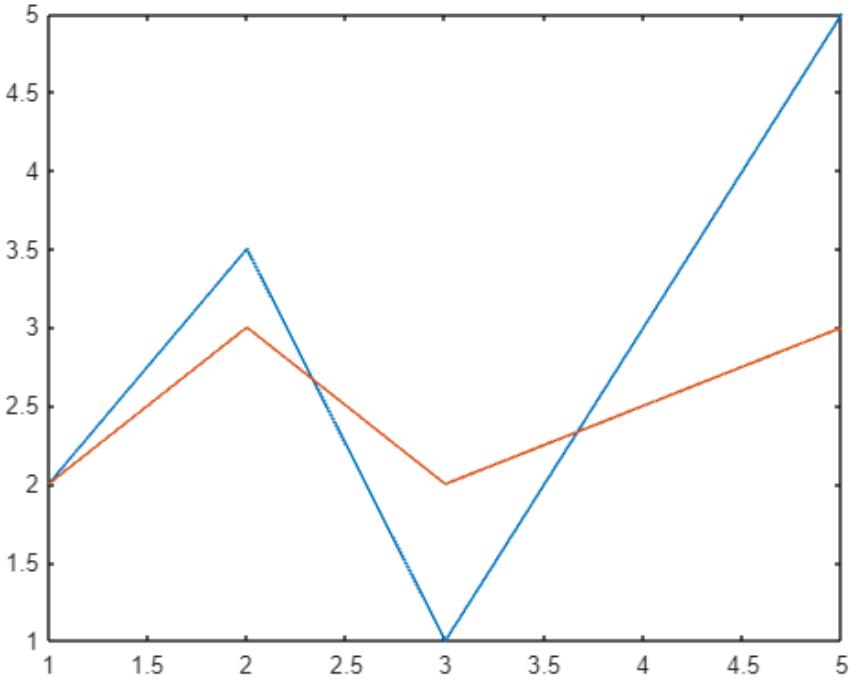
- dibujar un vector X frente a otro Y. Ejecuta el siguiente ejemplo para comprobarlo:

```
X = [1 2 3 5];
Y = [2 3.5 1 5];
plot(X,Y)
```



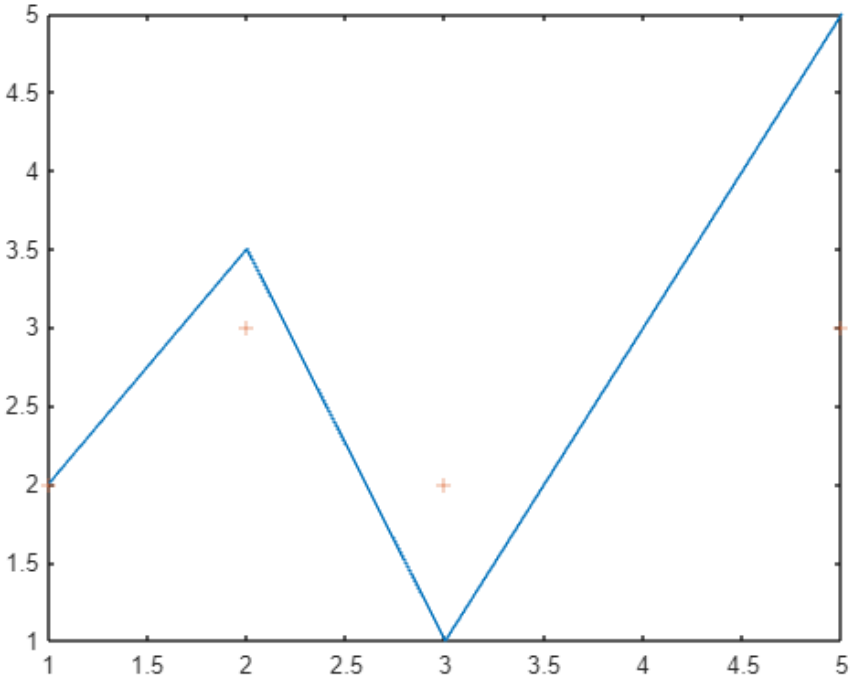
- mostrar varias gráficas distintas en una misma figura:

```
X2 = [1 2 3 5];
Y2 = [2 3 2 3];
plot(X,Y,X2,Y2)
```



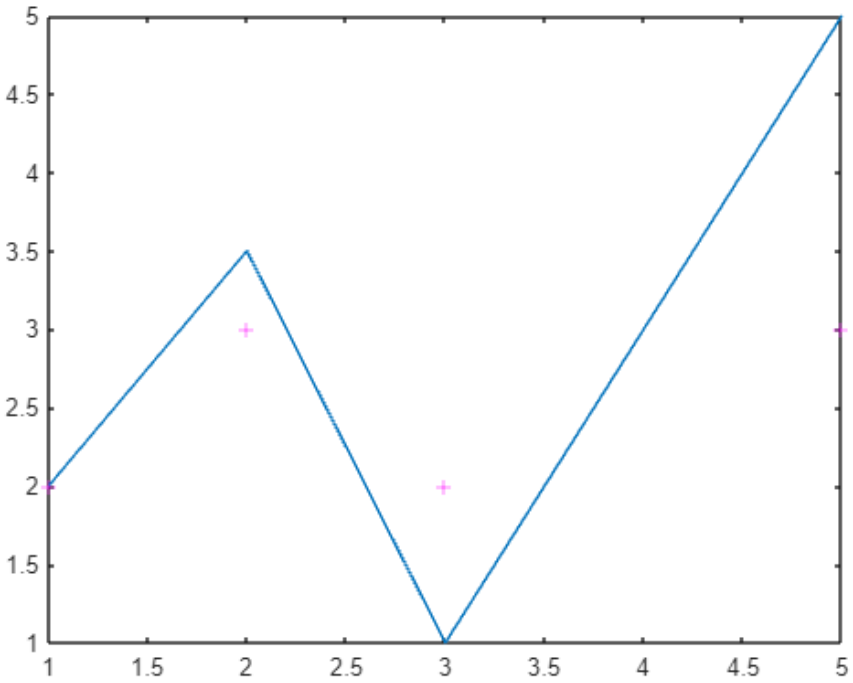
Por defecto MATLAB conecta cada uno de los puntos en los vectores anteriores con una línea, este comportamiento se puede modificar incluyendo un **símbolo** que represente dichos puntos:

```
plot(X,Y,X2,Y2,'+')
```



Otra posibilidad de personalización de las gráficas es elegir el **color** de estas. Por ejemplo para pintar en magenta:

```
plot(X,Y,X2,Y2,'+m')
```



Puedes consultar la documentación de MATLAB para comprobar los símbolos y colores que hay disponibles.

Algunas funcionalidades interesantes

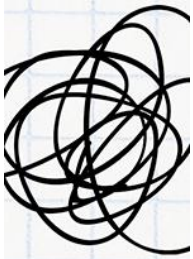
De manera genérica, MATLAB ofrece una serie de comandos para trabajar con gráficos:

- **`clf`**: Borra la ventana gráfica activa.
- **`figure`**: Abre una ventana gráfica y le asigna el papel de ventana gráfica activa.
- **`ginput`**: Permite extraer, utilizando el ratón, las coordenadas de una serie de puntos situados sobre un gráfico. La activación de la tecla return finaliza el proceso
- **`grid`**: Muestra una rejilla sobre la gráfica situada en la ventana gráfica activa.
- **`hold`**: hold on hace que todos los gráficos sucesivos cuyo trazado sea ordenado se incluyan en la ventana gráfica activa. hold off deshace este efecto.

Tarea 1: Obtener una representación gráfica en el color magenta de la función $y = \sin(\theta)$para los valores de θ comprendidos en el intervalo $[-2\Pi, 8\Pi]$ con una resolución (paso) de un grado. Además, marcar en la misma con un '*' negro el punto en el que ésta alcanza un mínimo en el intervalo de θ comprendido entre 0 y 10. Asimismo, obtener la integral definida de dicha función en el intervalo de θ comprendido entre 1 y 2.

Pista: necesitarás usar los comandos **`sin`**, **`fminsesarch`**, **`hold`**, y **`quad`**.

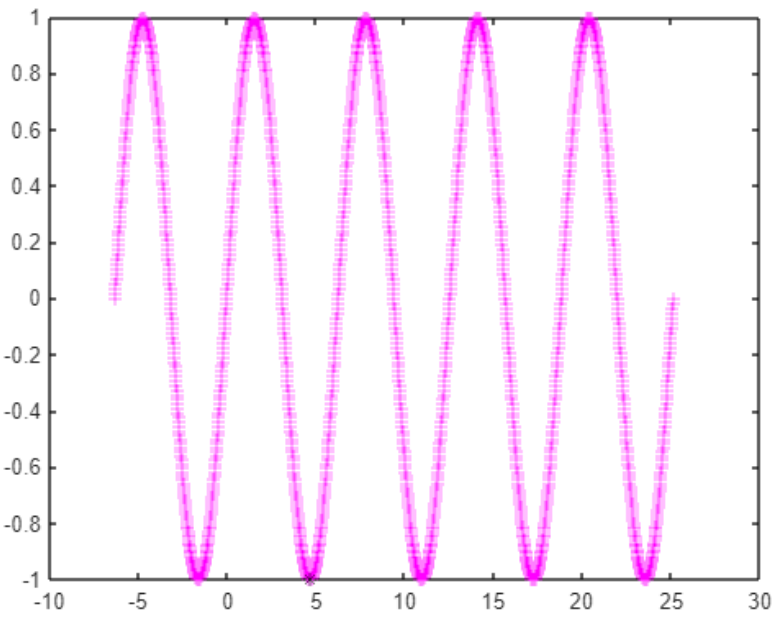
```
% Siempre que se trabaja con figuras es buena idea abrir una
% ventana gráfica para evitar continuar "pintando" sobre una
% creada anteriormente
```




```
figure
% Tu código aquí
paso=deg2rad(1);
```

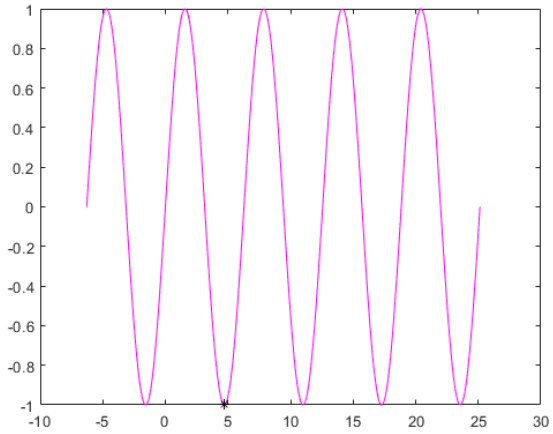
```
paso = 0.0175
```

```
X=-2*pi:paso:8*pi;
Y=sin(X);
plot(X,Y,'+m');
hold on
x=fminsearch(@sin,5);
plot(x,sin(x),'k*')
```




```

```

Resultado esperado:



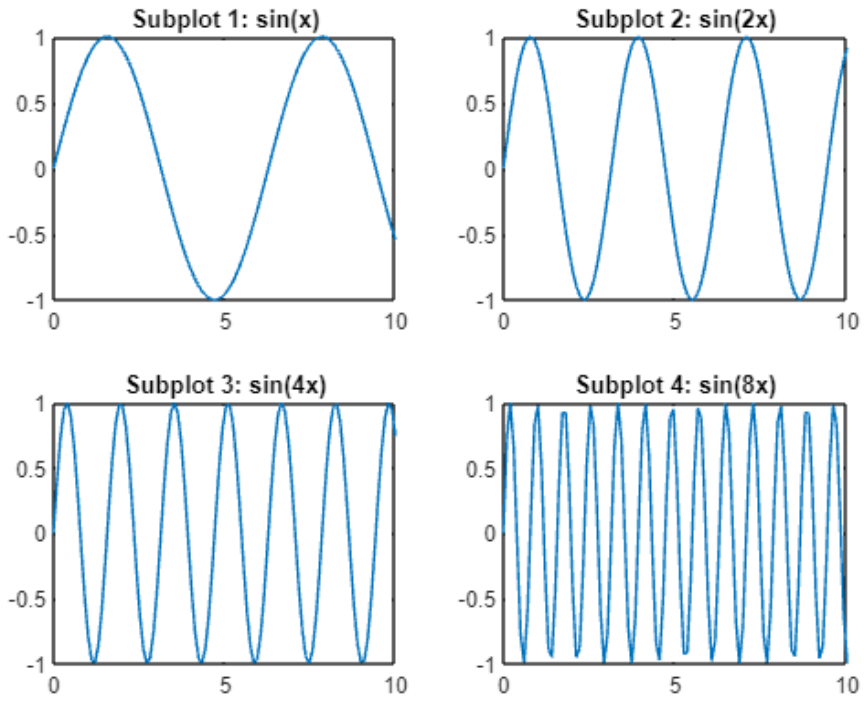

```
integral = 0.9564
```

2. Subgráficas

Pero, ¿qué pasa si necesito mostrar gráficas distintas simultáneamente? No hay problema, MATLAB ha pensado en todo y mediante el comando **`subplot`** podemos dividir la ventana gráfica activa en una serie de particiones horizontales y verticales, activando una de ellas. Si no hay ventana gráfica activa, la crea.

Veamos el siguiente código de ejemplo que divide la ventana gráfica en 2 filas con 2 gráficos cada una (el tercer argumento de **`subplot`** indica el índice de la subgráfica que se va a activar para *pintar* en ella):

```
subplot(2,2,1)
x = linspace(0,10);
y1 = sin(x);
plot(x,y1)
title('Subplot 1: sin(x)')

subplot(2,2,2)
y2 = sin(2*x);
plot(x,y2)
title('Subplot 2: sin(2x)')

subplot(2,2,3)
y3 = sin(4*x);
plot(x,y3)
title('Subplot 3: sin(4x)')

subplot(2,2,4)
y4 = sin(8*x);
plot(x,y4)
title('Subplot 4: sin(8x)')
```

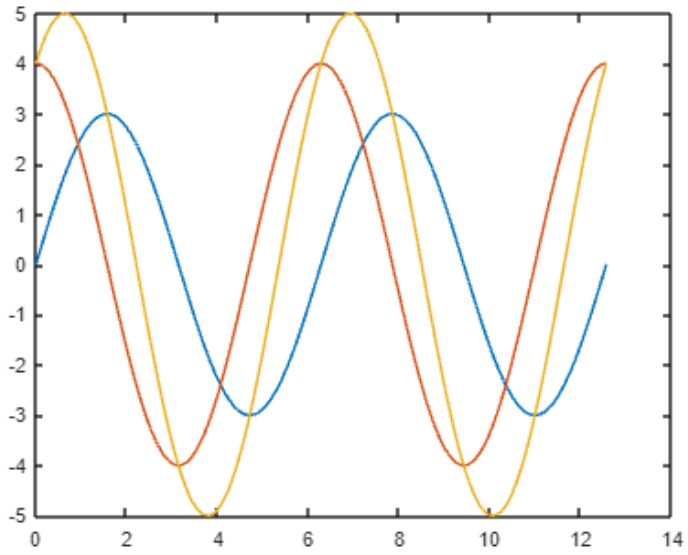


Tarea 2: Obtener una representación gráfica de las funciones $y_1 = 3\sin(\theta)$, $y_2 = 4\cos(\theta)$ y la suma de ambas $y_3 = y_1 + y_2$ para los valores de θ comprendidos en el intervalo $[0, 4\Pi]$, con una resolución de un grado, en cada una de las formas siguientes:

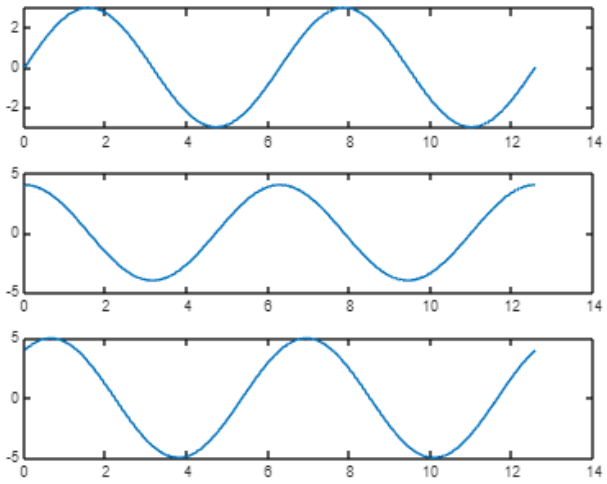
- a) En una única ventana gráfica utilizando los mismos ejes y colores diferentes.
- b) En una única ventana gráfica (figura) donde podrán visualizarse las tres gráficas alineadas verticalmente.

Pista: Entre los comandos a utilizar necesitarás recurrir a **`figure`**.para ir creando las nuevas figuras.

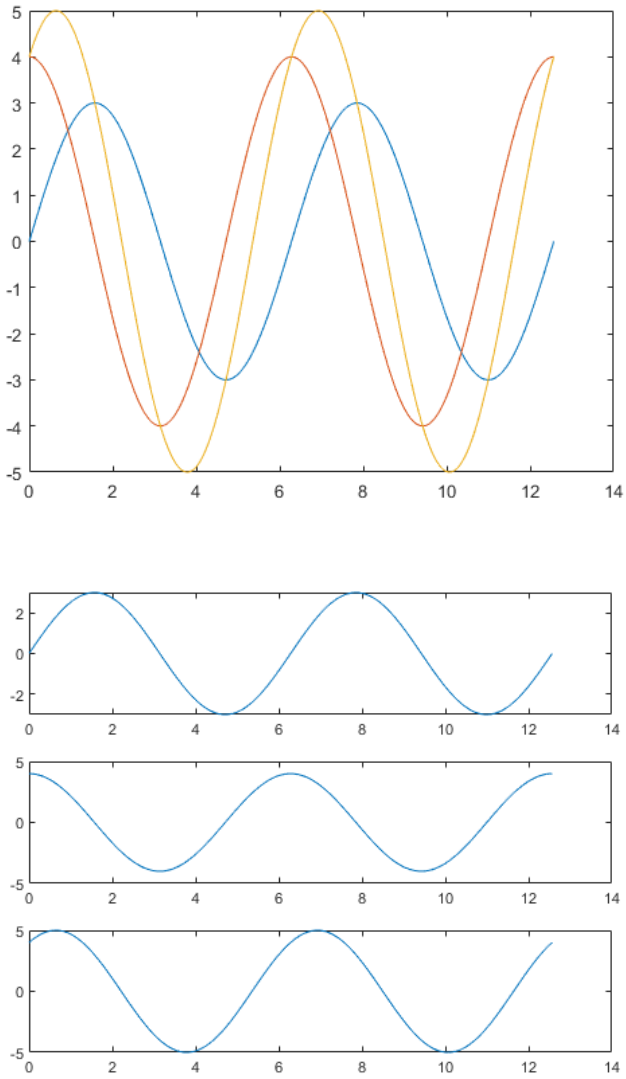
```
figure
% Tu código aquí
paso=deg2rad(1);
% a)
intervalo=[0:paso:4*pi];
plot(intervalo,3*sin(intervalo));
hold on
plot(intervalo,4*cos(intervalo));
plot(intervalo,4*cos(intervalo)+3*sin(intervalo))
```




```
figure
%b)
subplot(3,1,1)
x=[0:paso:4*pi];
y1=3*sin(x);
plot(x,y1)
subplot(3,1,2)
x=[0:paso:4*pi];
y2=4*cos(x);
plot(x,y2)
subplot(3,1,3)
x=[0:paso:4*pi];
y3=y1+y2;
plot(x,y3)
```



Resultado esperado:



3. Control sobre los ejes

MATLAB provee de una serie de funciones, equivalentes a **`plot`**, que permiten especificar la escala de los ejes. Por ejmplo:

- **`loglog`**: tanto el eje de abcisas como el de ordenadas se representan en escala logarítmica decimal.
- **`semilogx`**: el eje de abcisas se representa en escala logarítmica decimal.
- **`semilogy`**: el eje de ordenadas se representa en escala logarítmica decimal.

Además, también es posible poner un título a la gráfica con el comando **`title`** (habrá que usar **`sgtitle`** si se le quiere poner título a una figura que contiene **`subfigure`**), así como un texto junto al eje de abcisas (**`xlabel`**) o de ordenadas (**`ylabel`**).

Para finalizar con los ejes, también está disponible la función **`axis`**. Vamos a ver en qué consiste:

```
help axis
```

```
 axis  Control axis scaling and appearance.
    axis([XMIN XMAX YMIN YMAX]) sets scaling for the x- and y-axes
       on the current plot.
    axis([XMIN XMAX YMIN YMAX ZMIN ZMAX]) sets the scaling for the
       x-, y- and z-axes on the current 3-D plot.
    axis([XMIN XMAX YMIN YMAX ZMIN ZMAX CMIN CMAX]) sets the
       scaling for the x-, y-, z-axes and color scaling limits on
       the current axis (see CLIM).
    V = axis returns a row vector containing the scaling for the
       current plot.  If the current view is 2-D, V has four
       components; if it is 3-D, V has six components.

    axis AUTO  returns the axis scaling to its default, automatic
       mode where, for each dimension, 'nice' limits are chosen based
       on the extents of all line, surface, patch, and image children.
    axis MANUAL  freezes the scaling at the current limits, so that if
       HOLD is turned on, subsequent plots will use the same limits.
    axis TIGHT  sets the axis limits to the range of the data.
    axis FILL  sets the axis limits and PlotBoxAspectRatio so that
       the axis fills the position rectangle.  This option only has
       an effect if PlotBoxAspectRatioMode or DataAspectRatioMode are
       manual.

    axis IJ  puts MATLAB into its "matrix" axes mode.  The coordinate
       system origin is at the upper left corner.  The i axis is
       vertical and is numbered from top to bottom.  The j axis is
       horizontal and is numbered from left to right.
    axis XY  puts MATLAB into its default "Cartesian" axes mode.  The
       coordinate system origin is at the lower left corner.  The x
       axis is horizontal and is numbered from left to right.  The y
       axis is vertical and is numbered from bottom to top.

    axis EQUAL  sets the aspect ratio so that equal tick mark
       increments on the x-,y- and z-axis are equal in size. This
       makes SPHERE(25) look like a sphere, instead of an ellipsoid.
    axis IMAGE  is the same as axis EQUAL except that the plot
       box fits tightly around the data.
    axis SQUARE  makes the current axis box square in size.
    axis NORMAL  restores the current axis box to full size and
        removes any restrictions on the scaling of the units.
        This undoes the effects of axis SQUARE and axis EQUAL.
    axis VIS3D  freezes aspect ratio properties to enable rotation of
        3-D objects and overrides stretch-to-fill.

    axis OFF  turns off all axis labeling, tick marks and background.
    axis ON  turns axis labeling, tick marks and background back on.

    axis(H,...) changes the axes handles listed in vector H.

    See also axes, grid, subplot, xlim, ylim, zlim, rlim

    Documentation for axis
```

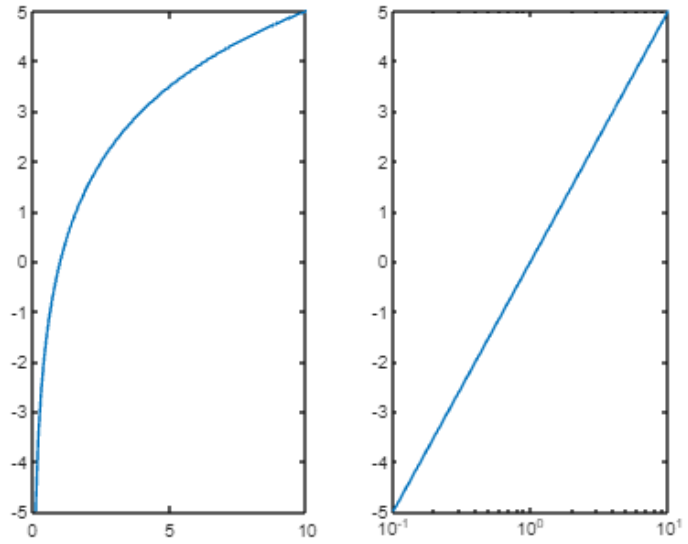
Tarea 3: Se desea representar gráficamente la función $y = 5\log_{10}x$ para los valores de x comprendidos en el intervalo $[0.1, 10]$, con una resolución de una décima. Obtener en una única ventana gráfica, utilizando ejes diferentes alineados verticalmente, una representación de dicha función en cada una de las formas siguientes:

- a) Utilizando escalas lineales en ambos ejes.

- b) Utilizando escala logarítmica decimal para el eje de abcisas y escala lineal para el eje de ordenadas.

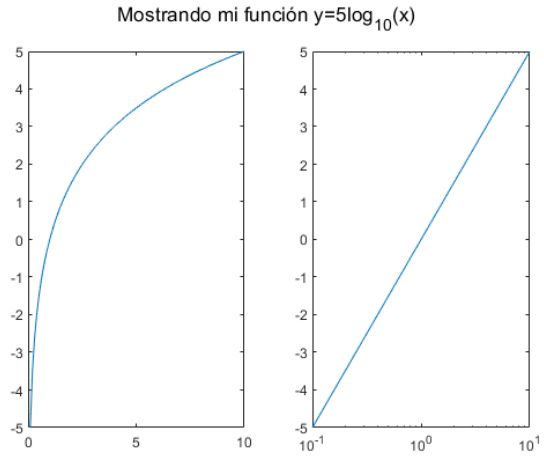
Además, ponle el título que desees.

```
figure
% Tu código aquí
x=(0.1:0.1:10);
f=5*log10(x);
subplot(1,2,1)
plot(x,f)
subplot(1,2,2)
semilogx(x,f)
```




```

```

Resultado esperado:



Introducción a MATLAB

Scripts, bucles e instrucciones condicionales

Table of Contents

Scripts, bucles e instrucciones condicionales....1
1. Archivos de Comandos....1
2. Bucles....3
2.1 For....3
2.2 While....4
3. Instrucciones condicionales....4
3.1 Si yo tuviera... (if)....5
3.2 El caso (switch)....6

Hasta el momento hemos venido trabajando con **scripts vivos** que según MATLAB pueden definirse como:

"Archivos de programa que contienen código, salidas y texto con formato, y conviven en un solo entorno interactivo conocido como Live Editor. En los scripts en vivo, puede escribir código y ver las salidas y las gráficas generadas junto con el código que las produjo. Añada texto con formato, imágenes, hipervínculos y ecuaciones para crear una narrativa interactiva que puede compartir con otros."

Esta manera de trabajar con MATLAB es bastante reciente, sin embargo, existe un enfoque más tradicional, que aún sigue siendo el adecuado cuando se necesita definir funciones complejas que alargarían en exceso los scripts vivos: los **archivos de comando** o **scripts** a secas.

En este cuaderno también vamos a ver unos commandos que nos permiten controlar el flujo de nuestros scripts: los **bucles** y las **instrucciones condicionales**.

1. Archivos de Comandos

Los archivos de comandos (archivos *script*) almacenan secuencias de sentencias. Se utilizan para llevar a cabo operaciones que involucran a múltiples sentencias, con objeto de evitar que cuando exista un error en alguna de ellas haya que reescribir y reejecutar todas las que le siguen. Asimismo, este tipo de archivos se puede utilizar simplemente para almacenar una secuencia de sentencias para la que se prevé una futura reutilización. Otro importante uso de los *script* es el de la definición de funciones. Por ejemplo, los comandos que hemos visto están definidos en este tipo de ficheros.

La siguiente imagen muestra un ejemplo de un script que para un cierto vector v calcula su media y su desviación típica:

```
EDITOR  PUBLISH  VIEW
1 -   v = [1 3 4 5 6];
2 -   n = length( x );
3 -   media = sum( x ) / n;
4 -   desvtip= sqrt( sum( (x - media).^2 ) / n );
5     % salida formateada
6 -   fprintf( 'Media : %f \n', media );
7 -   fprintf( 'Desviación Típica : %f \n', desvtip );
```

UTF-8 | script | Ln 7 | Col 49

Mientras que la siguiente ilustra una función (comando) denominada **desv** que permite ejecutar el mismo código con distintos vectores de entrada:

```
EDITOR  PUBLISH  VIEW
1   function [ media, desvtip ] = desv( x )
2   %desv Calcula la media y desviación típica de un vector
3   %   [media, desvtip] = desv(v) calcula la media y la desviación típica d
4   %   vector v.
5   %
6   %   Ejemplos:
7   %   v = [1 3 4 5 6];
8   %   [media, desvtip] = desv(v);
9   %   Media : 3.800000
10  %   Desviación Típica : 1.720465
11
12 -    n = length( x );
13 -    media = sum( x ) / n;
14 -    desvtip= sqrt( sum( (x - media).^2 ) / n );
15      % salida formateada
16 -    fprintf( 'Media : %f \n', media );
17 -    fprintf( 'Desviación Típica : %f \n', desvtip );
18 -  end
19
```

UTF-8 | desv | Ln 19 | Col 1

Los archivos de comandos tienen una extensión '.m' y pueden crearse en el editor que incorpora el entorno de MATLAB ('Home/New script' o 'Ctrl+N'). Las líneas de comentarios ubicadas por delante de la primera línea de programa de un archivo de comandos se muestran en la ventana de comandos de Matlab cuando en ésta se ejecuta el comando **help** seguido del nombre del archivo de comandos (excluyendo su extensión).

Un archivo de comandos puede ejecutarse de distintas formas:

- o bien desde la ventana de comandos (escribiendo una sentencia con el nombre del fichero sin extensión),
- directamente desde el editor del entorno de Matlab,
- o en una celda de código de un script vivo.

Para que dicha ejecución sea viable, el directorio en el que el que está almacenado el archivo de comandos considerado debe estar incluido en la lista de caminos de búsqueda de MATLAB ('Home/Set path'), el cual incluye el directorio de trabajo actual. Básicamente con esto nos aseguramos de que MATLAB conoce su existencia.

2. Bucles

2.1 For

El bucle por excelencia. Una instrucción **`for`** se repite un número especifico de veces, realizando un seguimiento de cada iteración con una variable en aumento. Su sintaxis es la siguiente:

```
for index = values

statements

end
```

El siguiente código a ejecutar muesta un ejemplo en el que un bucle **`for`** recorre ciertas posiciones de un vector y suma sus elementos:

```
x = [1 2 1 2 1 2];
suma = 0;
for i=3:5
    suma = suma + x(i);
    disp(suma)
end
```

Un bucle puede terminar bien sea por haber alcanzado el último valor de la variable en aumento a comprobar, o bien por la ejecución de la instrucción **`break`**. Por ejemplo:

```
suma = 0;
for i=3:5
    suma = suma + x(i);
    disp(suma)
    if i == 4
        break
    end
end
```

Por último, también podemos saltar a la siguiente iteración del bucle con el comando **`continue`**. Ejemplo:

```
suma = 0;
```


```
for i=3:5
    if i == 4
        continue
    end
    suma = suma + x(i);
    disp(suma)
end
```

Tarea 1: Define un vector fila de números aleatorios (generados a partir de una distribución de probablidad normal gaussiana) v con 10 elementos e implementa un bucle **`for`** que sume los que se encuentren en el rango de posiciones $[2, 7]$, mostrando el valor de la suma una vez concluido el bucle.

```
% Tu código aquí
v=randn(1,10)
```

```
v = 1×10
    0.8884   -1.1471   -1.0689   -0.8095   -2.9443    1.4384    0.3252   -0.7549 ···
```

```
suma=0;
for i=2:7
    suma=suma+v(i);
end
suma
```

```
suma = -4.2062
```

2.2 While

La otra joya de la corona a nivel de bucle de control es el bucle **`while`**. Este bucle continúa realizando una serie de instrucciones mientras que la expressión que evalúa sea cierta. Su sintaxis es:

`while` *`expression`*

`statements`

`end`

Un ejemplo de su uso:

```
i = 3;
suma = 0;
while i < 6
    suma = suma + x(i);
    disp(suma)
    i = i+1;
end
```

De igual modo que con el bucle **`for`**, usando **`while`** también se puede salir de dicho bucle con el comando **`break`**, o saltar a la siguiente iteración con **`continue`**.

3. Instrucciones condicionales

3.1 Si yo tuviera... (if)

Volvemos a encontrarnos con una instrucción popular: **`if`**. Esta instrucción condicional nos permite ejecutar una serie de instrucciones cuando una cierta condición es verdadera. En MATLAB una expresión es verdadera cuando su resultado no está vacío y contiene solo elementos no nulos (numéricos reales o lógicos). De lo contrario, la expresión es falsa.

Su sintaxis es:

```
if expression

statements

elseif expression

statements

else

statements

end
```

Los bloques **`elseif`** y **`else`** son opcionales, y permiten indicar que ocurre si no se cumple la primera condición condicional pero sí una segunda (o un número de `n` de condiciones subsecuentes) y que ejecutar en cualquier otro caso, respectivamente.

Veamos un ejemplo del uso de **`if`** dentro de un bucle **`for`** que sólo suma los elementos en una posición par de un vector:

```
x = [1,2,1,2,1,2,1,2];
suma = 0;
n = length(x);
for i = 1:n
    if mod(i,2)
        suma = suma + x(i);
        disp(suma)
    end
end
```

Tarea 2: Usar el bucle **`for`** y la instrucción **`if`** para, una vez definido un vector v que contenga valores tanto negativos como positivos (definidor por ti), sume sólo los elementos positivos del vector. Muestra el resultado de la suma al salir del bucle.

```
% Tu código aquí
v=[5 -8 7 -45 8 63 2 87 -100];

suma=0;

for i=1:9
    if v(i)>0
        suma=suma+v(i);
```

```
    end
end
suma
```

```
suma = 172
```

```

```

3.2 El caso (switch)

Y concluímos nuestro repaso a las instrucciones condicionales de MATLAB con la sentencia switch, la cual permite evaluar una condición y ejecutar uno de varios grupos de instrucciones. Su sintaxis es:

```
switch switch_expression

case case_expression

statements

case case_expression

statements

...

otherwise

statements

end
```

El siguiente código a ejecutar muestra un ejemplo de su uso, donde se nos va a pedir que introduzcamos un número y se va a ejecutar un caso distinto en función de este:

```
n = input('Enter a number: ');

switch n
    case -1
        disp('negative one')
    case 0
        disp('zero')
    case 1
        disp('positive one')
    otherwise
        disp('other value')
end
```