MÉTODOS NUMÉRICOS

Introducción a Matlab

Curso 23/24

Doble grado en Matemáticas e Informática

Facultad de Ciencias Matemáticas - UCM

Juan-Carlos Felipe-Navarro
Sergio Junquera

CONTENIDOS:

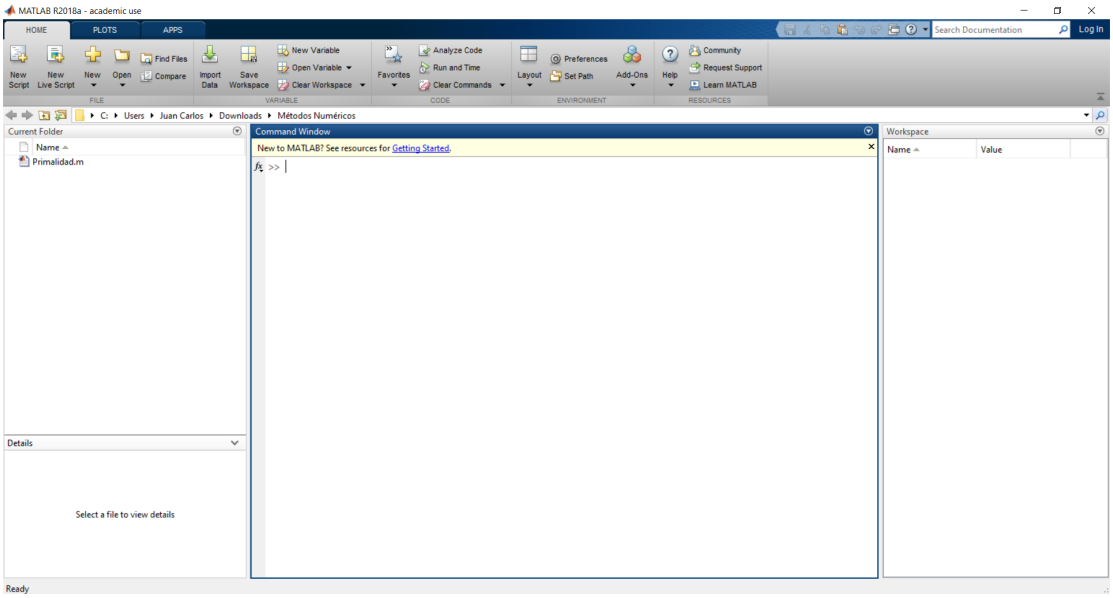
MÉTODOS NUMÉRICOS

Curso 2023-2024

INTRODUCCIÓN A MATLAB

`MATLAB` es un programa que permite realizar cálculos de forma rápida y fiable. Su nombre viene de la abreviación de "MATrix LABoratory". Tal y como su nombre indica, es un programa pensado para trabajar con matrices de forma que permite implementar cómodamente y rápidamente cualquier algoritmo que las utilice.

Al arrancar `MATLAB` se abre una ventana similar a esta:



Se pueden distinguir tres pantallas:

- Command Window: es la parte donde se ejecutan los comandos.
  El símbolo >> indica que el programa está listo para recibir instrucciones.
- Current Directory: es la ventana donde se muestran los archivos del directorio que se tiene activo.
- Workspace: muestra las variables que se han definido y su valor.

1. MATLAB COMO CALCULADORA

1.1. Operaciones básicas.

Aunque el programa está pensado para trabajar con vectores y matrices, también se puede utilizar como un lenguaje de programación escalar. Las operaciones se evalúan por orden de prioridad: primero las potencias, después las multiplicaciones y divisiones y, finalmente, las sumas y restas. Las operaciones de igual prioridad se evalúan de izquierda a derecha.

```
>> 2+3*2

ans =

     8

>> (2+3)*2

ans =

    10

>> (2/3^2*5)*(3-4^3)^2

ans =

   4.1344e+03
```

Es usual cuando se trabaja con `MATLAB` querer volver a ejecutar o modificar un comando introducido anteriormente. Esto se puede hacer de forma rápida utilizando las flechas ↑ y ↓ del teclado: si se pulsan las teclas ↑ y ↓ sobre la línea de comandos (`>>`) `MATLAB` nos va mostrando los comandos que hemos ejecutado anteriormente:

Se puede cambiar el formato de escritura de los resultados numéricos mostrados en la Command Window. La manera usual de realizar esta modificación es mediante los comandos `format short` y `format long` que se utilizan directamente desde la línea de comandos y que permiten mostrar los números con 4 o 15 decimales respectivamente.

```
>> format long
>> (2/3^2*5)*(3-4^3)^2

ans =

     4.134444444444444e+03

>> format short
```

Es importante remarcar que estos comandos no cambian la precisión con que internamente realiza los cálculos. Simplemente modifica la visualización de los mismos.

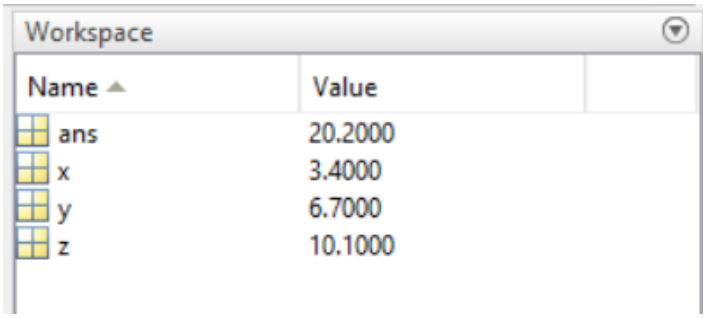
1.2. Asignación de variables.

Cuando interesa guardar un valor o el resultado de una operación para poder utilizarlo posteriormente, éstos se asignan a una variable. La instrucción básica en `MATLAB` es

```
variable = expresión;
```

```
>> x = 3.4;
>> y = 6.7;
>> z = x+y;
>> 2*z

ans =

   20.2000
```

El `;` al final de la instrucción sirve únicamente para suprimir la respuesta por pantalla. Ello no significa que no se realice la operación, si no que no nos la muestra. Al definir la varible, ésta nos aparece en el Workspace. En este entorno, podemos mirar el valor de la variable y también podemos borrarlas, renombrarlas, cambiar su valor...



Es importante destacar que si se escribe directamente una expresión sin asignarla a ninguna variable, el resultado se almacena en una variable llamada `ans`.

Como nombre de las diferentes variables se puede utilizar cualquier secuencia de caracteres alfanuméricos siempre que empiecen por una letra. `MATLAB` utiliza los primeros 31 caracteres del nombre de la variable y distingue entre mayúsculas y minúsculas. Hay algunos nombres especiales que conviene no utilizar porque corresponden a valores ya definidos en el programa (`eps`, `pi`, `realmax`, `realmin`...).

La instrucción `who` proporciona un listado de las variables que se están utilizando, y para saber más información sobre las mismas, como el espacio que ocupan en memoria, se tiene el comando `whos`.

```
>> who

Your variables are:

ans  x    y    z

>> whos
  Name      Size            Bytes  Class     Attributes

  ans       1x1                 8  double
  x         1x1                 8  double
  y         1x1                 8  double
  z         1x1                 8  double
```

Para borrar una variable se utiliza la instrucción `clear` seguida del nombre de la misma. Por el contrario, si se ejecuta únicamente `clear` o `clear all` se borrarán todas las variables. También es muy útil el comando `clc` que borra el contenido de la Command Window (no elimina el historial de comandos).

1.3. Funciones de MATLAB.

MATLAB dispone de varias funciones que ya están programadas y pueden utilizarse. Algunas de las más utilizadas en el área de matemáticas se muestran en la tabla:

`abs`	valor absoluto de un real o módulo de un complejo
`cos`, `sin`, `tan`	coseno, seno y tangente de un ángulo en radianes
`acos`, `asin`, `atan`	inversas de las funciones trigonométricas
`exp`	función exponencial
`log`, `log2`, `log10`	logaritmo natural, en base 2 y en base 10
`sqrt`	raíz cuadrada
`ceil`	redondea al siguiente entero
`floor`	redondea al entero anterior
`rem`	residuo de una división
`gcd`	máximo común divisor
`lcm`	mínimo común múltiplo

Como podemos ver a continuación, MATLAB trabaja por defecto con números complejos. La unidad imaginaria se representa como `i` o `j`, variables con dicho valor como predeterminado.

```
>> a = log(4)

a =

    1.3863

>> b = sqrt(-3)

b =

   0.0000 + 1.7321i
```

Por otro lado, el comando `@` nos permite definir funciones sencillas que podremos evaluar de forma rápida posteriormente.

```
>> f = @(x) x^2 + 8;
>> f(0)

ans =

     8

>> f(1)

ans =

     9
```

2. Vectores y matrices

Como ya se ha comentado, `MATLAB` está pensado para trabajar cómodamente con vectores y matrices. Una de las ventajas que tiene es que para definirlos no hace falta que le digamos por adelantado qué tamaño deben tener. Esto es una gran ventaja, ya que podemos ir modificando su tamaño a medida que vamos calculando.

2.1. Definiciones y operaciones básicas.

Para definir un vector fila se escriben sus componentes entre corchetes, separadas por comas o por espacios en blanco:

```
>> v1 = [1, 2, 3]

v1 =

     1     2     3

>> v2 = [6 8 23]

v2 =

     6     8    23
```

Por el contrario, para definir un vector columna se escriben sus componentes entre corchetes, separadas por puntos y coma o mediante saltos de línea:

```
>> w1 = [31; 4; 0; -3]

w1 =

    31
     4
     0
    -3

>> w2 = [-9
12
7.3]

w2 =

   -9.0000
   12.0000
    7.3000
```

Una matriz puede definirse fila por fila en líneas separadas, o bien en una sola línea, si bien en tal caso hay que separarlas con punto y coma:

```
>> A = [1, 2, -5, 7, -12; 4, 6, 1, -2, 33; -5, 21, 6, 19, -93; 0, 1, -2, -7, 99]

A =

     1     2    -5     7   -12
     4     6     1    -2    33
    -5    21     6    19   -93
     0     1    -2    -7    99
```

Cabe destacar que en general para MATLAB los vectores son también matrices con una única fila (o una única columna).

Un comando muy útil cuando se trabaja con matrices y vectores es ' para la transposición (en realidad trasposición y conjugación):

```
>> B = A'

B =

     1     4    -5     0
     2     6    21     1
    -5     1     6    -2
     7    -2    19    -7
   -12    33   -93    99
```

Una vez definido un vector o matriz en MATLAB podemos acceder a sus elementos especificando entre paréntesis, a continuación del nombre del vector o matriz, la posición a que queremos acceder:

```
>> A(2,4)

ans =

    -2
```

También podemos seleccionar fácilmente una submatriz, indicando las filas y columnas iniciales y finales:

```
>> A(2:3,3:4)

ans =

     1    -2
     6    19
```

Selecciones más complicadas también son posibles. Por ejemplo, si queremos seleccionar la submatriz formada por las filas 1 y 3 y las columnas 2 y 5:

```
>> A([1,3],[2,5])

ans =

     2   -12
    21   -93
```

Para seleccionar una fila completa especificamos : como índice de columna. Por ejemplo, para mostrar las dos primeras filas completas:

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

ans =

     1     2    -5     7   -12
     4     6     1    -2    33
```

Por otro lado, para borrar una fila (o columna) de una matriz la cambiamos por la matriz vacía `[]`. Por ejemplo, para borrar la tercera columna de la matriz anterior:

```
>> A(:,3)=[]

A =

     1     2     7   -12
     4     6    -2    33
    -5    21    19   -93
     0     1    -7    99
```

`MATLAB` es también capaz de concatenar o "pega" matrices o vectores en una nueva matriz o vector siempre que las dimensiones originales encajen adecuadamente:

```
>> C = [A, A'; -1, 2, 3, -4, 42, -3, -6, 0]

C =

     1     2     7   -12     1     4    -5     0
     4     6    -2    33     2     6    21     1
    -5    21    19   -93     7    -2    19    -7
     0     1    -7    99   -12    33   -93    99
    -1     2     3    -4    42    -3    -6     0
```

En diferentes escenarios puede ser necesario conocer el tamaño de una matriz (número de filas y columnas). Para ello usamos el comando `size`:

```
>> size(C)

ans =

     5     8
```

También podemos obtener simplemente el número de filas o de columnas de la matriz:

```
>> size(C,1)

ans =

     5

>> size(C,2)

ans =

     8
```

Las operaciones básicas entre vectores y matrices se representan con los operadores habituales:

```
>> v1+2*v2

ans =

    13    18    49
```

Además de las operaciones básicas con vectores y matrices, `MATLAB` proporciona algunas operaciones componente a componente. Por ejemplo, para elevar al cuadrado todos los componentes de un vector/matriz utilizamos un punto delante del símbolo `^`:

```
>> v1.*v2

ans =

     6    16    69

>> A.^2

ans =

           1           4          49         144
          16          36           4        1089
          25         441         361        8649
           0           1          49        9801
```

O para multiplicar dos vectores componente a componente utilizamos un punto frente al símbolo `*`:

```
>> v1.*v2

ans =

     6    16    69
```

Por último, si aplicamos una de las funciones internas de `MATLAB` a un vector, automáticamente ésta se aplica sobre todas las componentes del mismo. Del mismo modo, también podemos aplicar una función definida por el usuario a un vector, siempre y cuando pueda ser aplicada al mismo:

```
>> log(v1)

ans =

         0    0.6931    1.0986

>> f(v1)
Error using  ^
Incorrect dimensions for raising a matrix to a power. Check that the matrix is square and the power is
a scalar. To perform elementwise matrix powers, use '.^'.

Error in @(x)x^2+8

>> g = @(x) x.^2 + 8;
>> g(v1)

ans =

     9    12    17
```

2.2. Secuencias y matrices especiales.

Existen comandos que permiten crear vectores y matrices especiales.

La sintaxis `a:b:c` genera la secuencia de valores que van desde a hasta c en incrementos de magnitud b. Si se omite el valor de b se genera una secuencia de valores de uno en uno. Esta secuencia es considerada por `MATLAB` como un vector:

```
>> v3 = 0:2:20

v3 =

     0     2     4     6     8    10    12    14    16    18    20
```

Equivalentemente, si lo que conocemos del vector es que la primera coordenada vale a, la última c y que tiene n componentes en total se utiliza el comando linspace:

```
>> v4 = linspace(0,20,11)

v4 =

     0     2     4     6     8    10    12    14    16    18    20
```

Para definir de forma sencilla la matriz identidad de dimensión n en `MATLAB` se usa el comando `eye`:

```
>> I = eye(3)

I =

     1     0     0
     0     1     0
     0     0     1
```

De forma muy sencilla también podemos generar matrices con todas las entradas igual a cero o uno:

```
>> M0 = zeros(2,3)

M0 =

     0     0     0
     0     0     0

>> M2 = ones(3,2)

M2 =

     1     1
     1     1
     1     1
```

Una matriz diagonal se construye fácilmente a partir de un vector con el comando `diag`:

```
>> diag(v1)

ans =

     1     0     0
     0     2     0
     0     0     3
```

Este mismo comando aplicado a una matriz (no necesariamente cuadrada) crea un vector columna con los elementos de la diagonal:

```
>> diag(A)

ans =

     1
     6
    19
    99
```

También se pueden generar matrices con los términos no nulos en alguna diagonal distinta de la principal:

```
>> diag(v1,3)

ans =

     0     0     0     1     0     0
     0     0     0     0     2     0
     0     0     0     0     0     3
     0     0     0     0     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0
```

Es importante tener presente que podemos conocer más detalles y ejemplos de cómo se usa la función `diag` (o cualquier otra) a través del comando `help`:

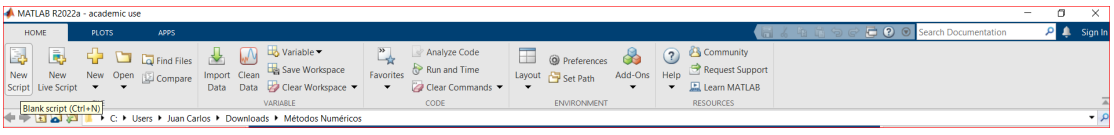
```
>> help diag
diag - Create diagonal matrix or get diagonal elements of matrix

    This MATLAB function returns a square diagonal matrix with the elements of
    vector v on the main diagonal.

    D = diag(v)
    D = diag(v,k)
    x = diag(A)
    x = diag(A,k)

    See also blkdiag, isdiag, istril, istriu, spdiags, tril, triu

    Reference page for diag
    Other functions named diag
```

Otras funciones útiles actuando en matrices son `triu` y `tril`, que extraen el triángulo superior e inferior respectivamente:

```
>> triu(A)

ans =

     1     2     7   -12
     0     6    -2    33
     0     0    19   -93
     0     0     0    99

>> tril(A)

ans =

     1     0     0     0
     4     6     0     0
    -5    21    19     0
     0     1    -7    99
```

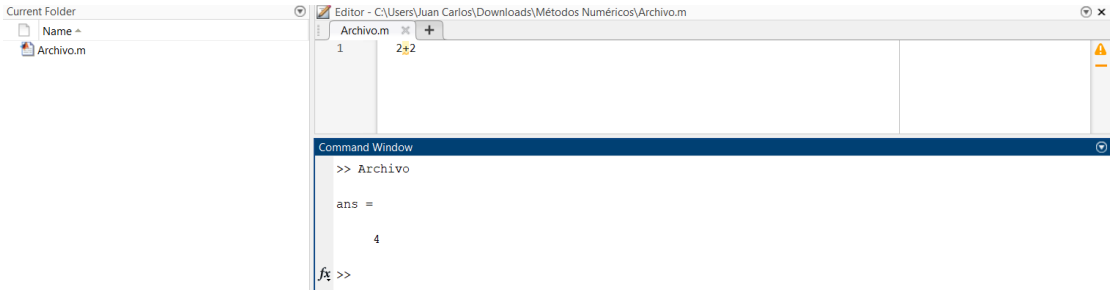
3. Archivos `.m` (scripts)

`MATLAB` permite tener archivos con instrucciones, los archivos `.m`, que se pueden ejecutar posteriormente. Estos archivos pueden ser de tres tipos: program script, function script y live script. La forma más fácil de crearlos y trabajar con ellos es mediante el editor de `MATLAB`. Para abrirlo haremos clic sobre New Script en la barra superior:

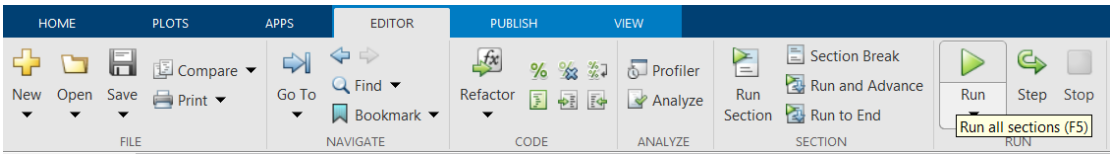


3.1. Program script.

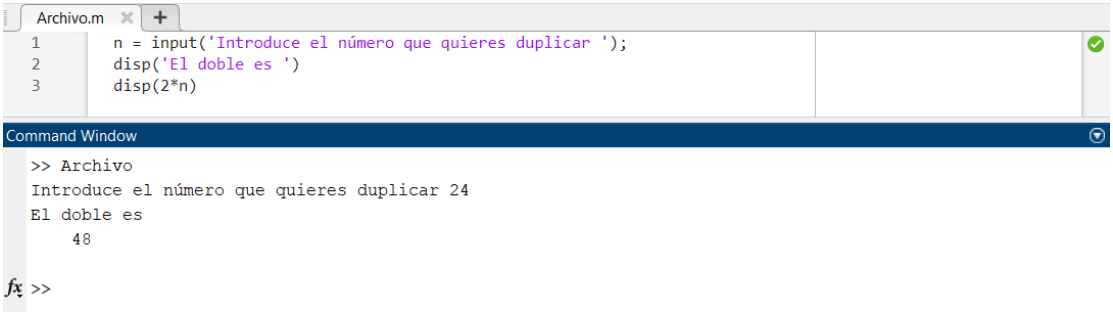
En lugar de trabajar en la Comand Window, se pueden crear programas con las instrucciones que se quieren ejecutar y después llamarlos con `MATLAB` mediante su nombre. Una vez guardado un programa script existen varias formas de ejecutarlo. La primera es escribiendo el nombre en la Command Window. Para ello es importante que el archivo esté en el directorio de trabajo:



Alternativamente podemos hacer clic en Run con el fichero abierto:



En un archivo `.m` se pueden leer valores por el teclado mediante el uso del comando `input`. Del mismo modo se pueden escribir cosas por pantalla usando `disp`:



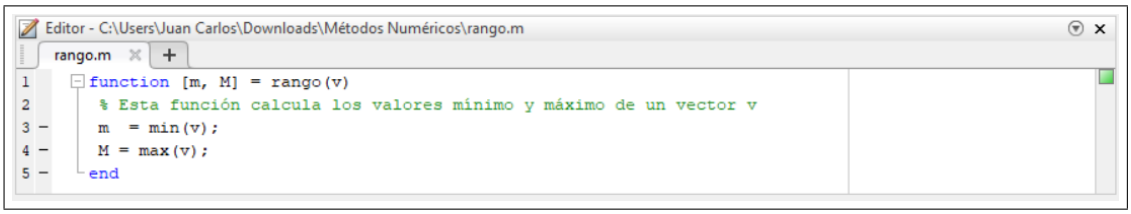
3.2. Function script.

La sintaxis para definir una función en `MATLAB` es la siguiente:

```
function [output1, output2] = nombre(input1, input2)
    %Descripcion de la funcion
    Cuerpo de la funcion;
end
```

La mejor manera de programar funciones en `MATLAB` es utilizando la ventana de edición, que además ofrece ventajas como distintos colores según que se incluyan comentarios, funciones propias de `MATLAB`, etc, o indentación, que hace más claras las distintas partes de la función. Una vez hayamos terminado de editar la función, debe guardarse en un archivo con el mismo nombre de la función.

Por ejemplo, podemos construir una función que reciba como entrada un vector de longitud arbitraria y devuelva como salida un vector formado por los valores mínimo y máximo del vector de entrada:



Al ejecutarla:

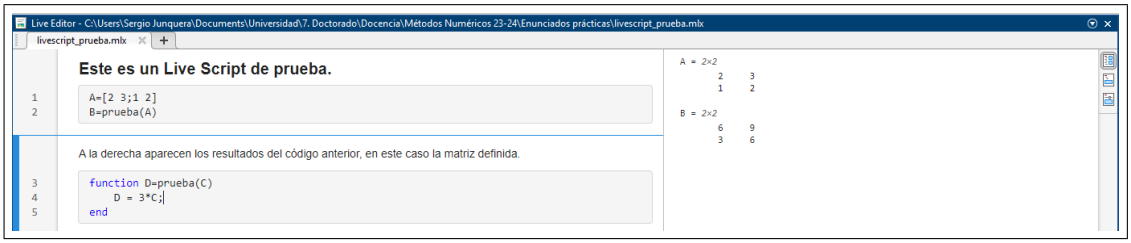
```
>> v = [-3, 1, 10, -5, 5, 0, -4];
>> [m, M] = rango(v)

m =

    -5


M =

    10
```

Finalmente, comentar que la descripción que se pone en la función se puede consultar con el comando `help`:

```
>> help rango
  Esta función calcula los valores mínimo y máximo de un vector v
```

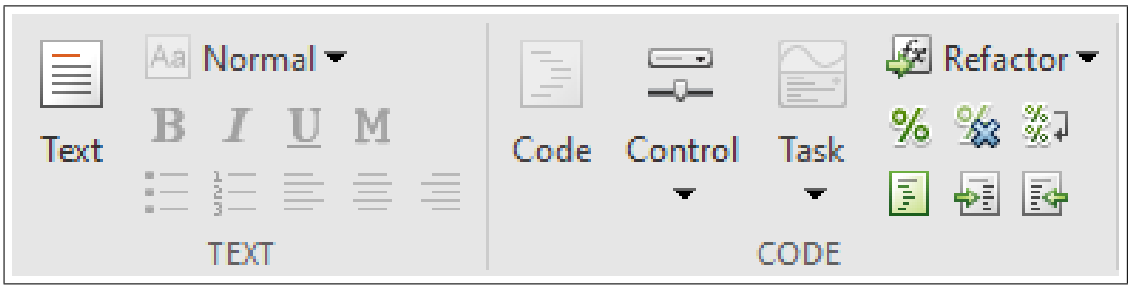
3.3. Live script (archivos .mlx).

Estos son archivos especiales en los que podemos combinar de manera sencilla texto, funciones y líneas de código normales. Serán el tipo de archivos que manejaremos durante el curso.



Como podemos ver, en un mismo archivo hemos podido incluir los tres tipos de contenido y, al ejecutarlo, el resultado del código aparece a la derecha en el editor.

Al trabajar con este tipo de scripts, en la parte superior de MATLAB aparece una nueva sección en la que podemos alternar entre texto y código, así como formatear el texto a nuestra elección.



A la hora de trabajar con estos scripts, debemos tener en cuenta una particularidad importante: las funciones que queramos incluir en el script para utilizarlas a lo largo del código deben ir al final del todo.

4. Sentencias de control de flujo

Cuando ejecutamos los archivos, se hace de manera secuencial. Este tipo de programación es muy limitada y por ello `MATLAB`, al igual que todos los lenguajes de programación, incorpora instrucciones que permiten seguir caminos no secuenciales. Tres de los más importantes son: `if`, `for` y `while`.

4.1. Sentencia condicional if-else.

La estructura es

```
if condicion 1
    bloque de instrucciones 1
elseif condicion 2
    bloque de instrucciones 1
else
    bloque de instrucciones final
end
```

Para escribir las condiciones se pueden usar los operadores relacionales:

`>`	mayor que	`>=`	mayor o igual que
`<`	menor que	`<=`	menor o igual que
`==`	igual a	`~=`	diferente de

que se pueden combinar con los diferentes operadores lógicos:

&	y
|	o
~	negación

4.2. Sentencia repetitiva for.

La estructura es

```
for variable = inicio : paso : fin
    instrucciones
end
```

O bien,

```
for variable = [vector]
    instrucciones
end
```

Se van ejecutando las instrucciones dentro del bucle hasta que el contador llega al final.

4.3. Sentencia repetitiva while.

La estructura es

```
while condicion
    instrucciones
end
```

Se ejecutan las instrucciones del interior del bucle mientras la condición sea cierta.

5. Polinomios

En MATLAB se puede trabajar con polinomios. Basta tener en cuenta que un polinomio se puede representar como un vector de los coeficientes. El orden de los coeficientes en MATLAB es de mayor a menor grado. Por ejemplo, el vector $(1, 2, 3)$ representaría el polinomio $x^2 + 2x + 3$.

Una vez se tiene en cuenta esta representación, podemos presentar algunas de las funciones de MATLAB más interesantes para trabajar con polinomios. Con el comando `polyval` evaluamos el polinomio en un punto:

```
>> polyval(v1,2) %Evaluación de x^2+2x+3 en x=2

ans =

    11
```

Podemos operar con los polinomios para calcular el producto y el cociente:

```
>> v12 = conv(v1,v2)

v12 =

     6    20    57    70    69

>> [q12, r12] = deconv(v1,v2)

q12 =

    0.1667


r12 =

         0    0.6667   -0.8333
```

También podemos calcular las raíces con el comando `roots`, que nos genera un vector columna con todas las raíces (tanto reales como complejas):

```
>> roots(v12)

ans =

  -0.6667 + 1.8409i
  -0.6667 - 1.8409i
  -1.0000 + 1.4142i
  -1.0000 - 1.4142i
```

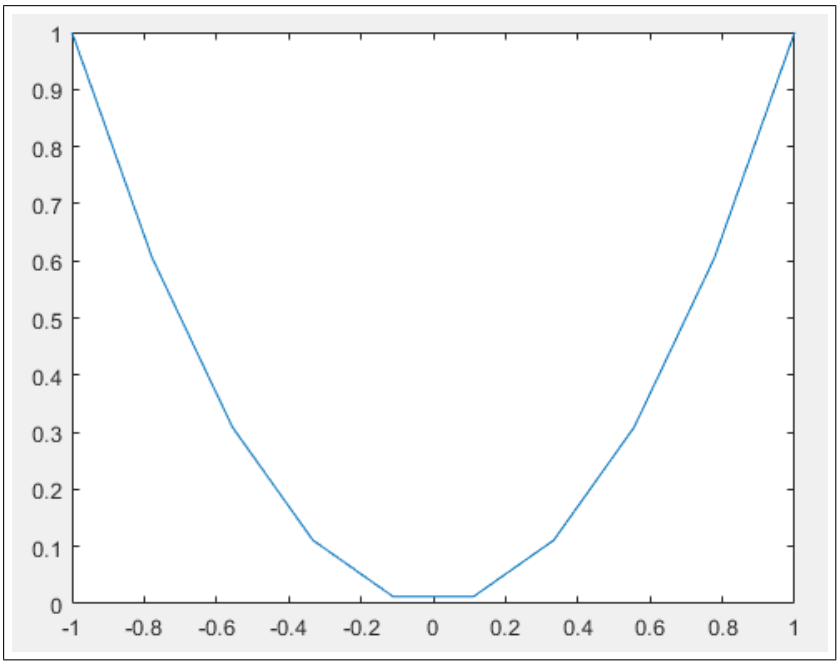
Por otro lado es posible generar un polinomio a partir de sus raices:

```
>> poly([1, 2+i, 2-i, -4]) %Polinomio con raices 1, 2+i, 2-i, -4

ans =

     1    -1   -11    31   -20
```

6. Gráficas de funciones

Otra característica importante de MATLAB es su capacidad para representar funciones y conjuntos de datos. La forma más sencilla en la que podemos hacer esto es con el comando `plot`, representando los valores de un vector frente al otro:

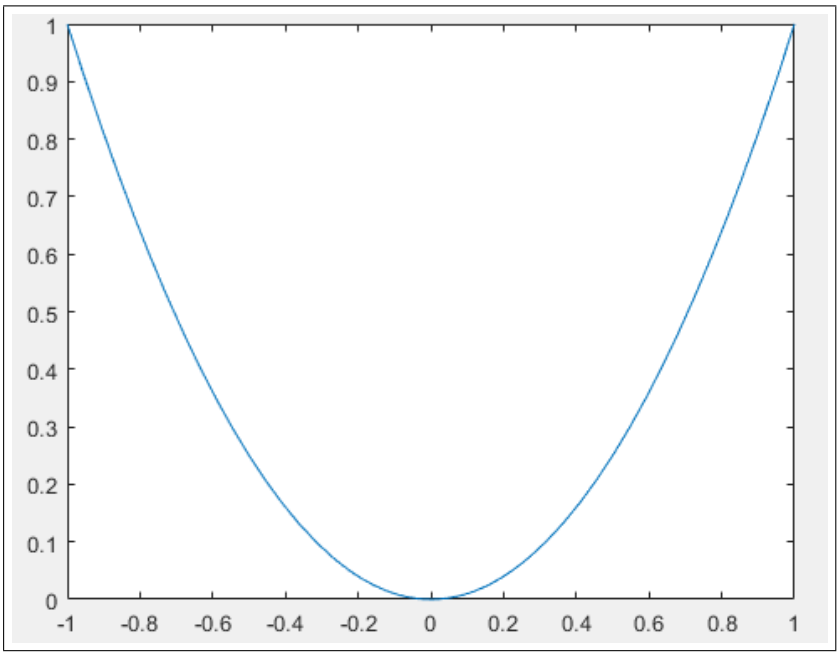
```
Command Window
>> x=linspace(-1,1,10)

x =

   -1.0000   -0.7778   -0.5556   -0.3333   -0.1111    0.1111    0.3333    0.5556    0.7778    1.0000

>> y=x.^2

y =

    1.0000    0.6049    0.3086    0.1111    0.0123    0.0123    0.1111    0.3086    0.6049    1.0000

>> plot(x,y)
>>
```



En este ejemplo se nota el aspecto lineal de la gráfica por haber tomado pocos puntos. Esto lo podemos solucionar tomando más puntos o utilizando la función `fplot`, que nos permite graficar directamente funciones:

```
Command Window
>>
>>
>>
>>
>>
>> f= @(x) x^2

f =

  function_handle with value:

    @(x)x^2

>> fplot(f,[-1 1])
```

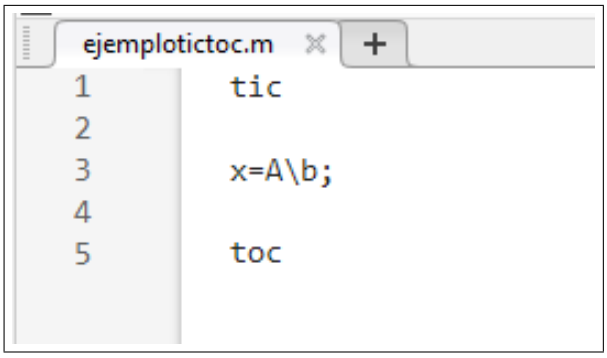
18



7. Otros conceptos

7.1. Tiempos de ejecución.

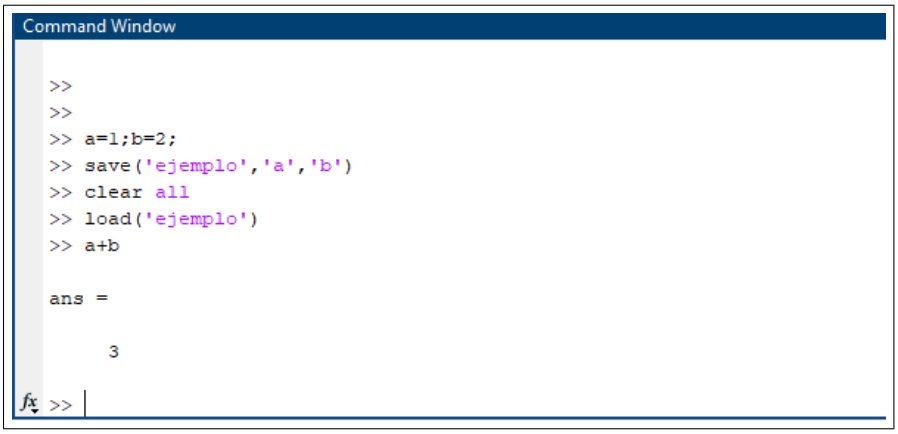
A veces es interesante ver cuánto tarda en ejecutarse una instrucción de MATLAB o un código hecho por nosotros mismos, con la intención de optimizarlo o comparar con otros métodos. Para ver el tiempo de ejecución, se utilizan las instrucciones `tic` y `toc` como en el siguiente ejemplo.



En este caso, hemos dividido una matriz $A \in GL_{200}(\mathbb{R})$ por un vector $b \in \mathbb{R}^{200}$, y el tiempo de ejecución ha sido de 0,02589 segundos.

7.2. Cargar y guardar archivos en MATLAB.

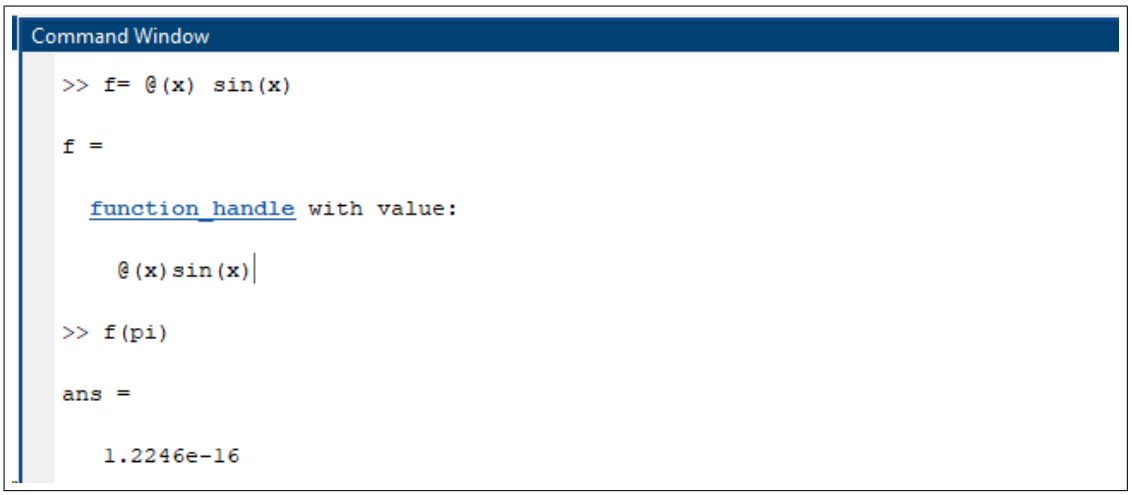
Una manera sencilla de enviar y recibir datos es guardarlas en archivos `.mat`, en vez de en scripts. Este tipo de archivos están más comprimidos y por tanto son más convenientes. Para ello, se utilizan los comandos `save` y `load`:



Por defecto, estas instrucciones guardan y cargan los archivos en nuestro directorio de trabajo actual, aunque se pueden usar otras rutas. Para más información sobre la sintaxis, podéis ver la documentación de MATLAB.

7.3. Definición de funciones y variables simbólicas.

Las funciones en MATLAB se pueden definir de varias maneras. Para nuestros propósitos la más útil es la definición por funciones anónimas como en el siguiente ejemplo:



En algunas ocasiones, como para utilizar el comando `diff`, necesitamos que las funciones estén definidas como funciones simbólicas. Para utilizar este tipo de funciones, necesitamos tener instalado el paquete `Symbolic Math Toolbox`. Para pasar de funciones anónimas a funciones simbólicas se utiliza la instrucción `sym`.