Introducción a MATLAB

CEU San Pablo

Ejercicios

Samuel G. Corregidor

MATLAB es un sistema de cómputo numérico que ofrece un entorno de desarrollo integrado (IDE) con un lenguaje (M) de programación propio. Con esta lista de ejercicios se pretende familiarizar al alumno con el lenguaje y el entorno.

El entorno

(1) En la parte superior, buscar el boton *Layout*. Seleccionar las diversas opciones y elegir una.

(2) Buscar el boton *Preferences*. Seleccionar *Workspace*. Prestar atención a las opciones dentro de *Saving variables* y *MATLAB array size limit*.

(3) En *Preferences*: Seleccionar *Keyboard* y elegir *Shortcuts*. Buscar los atajos de teclado más utilizados, por ejemplo, *Run Current Test*.

(4) Crea una carpeta llamada MATLAB, será aquí donde guardaremos todos los archivos relacionados con MATLAB.

La consola

El entorno ofrece una consola sobre la que podemos ejecutar lineas de comando. Puede ser utilizada como una calculadora.

(5) Escribir sobre la consola el siguiente código.

```
a = 3*4
```

¿Qué observas? ¿Qué clase de variable has definido? Crea una nueva variable ahora añadiendo punto y coma.

```
b = 2*4;
```

¿Qué diferencia hay entre poner punto y coma y no ponerlo? ¿Como se puede mostrar por consola el valor de una de las variables asignadas? Realiza algún cálculo con las variables declaradas.

(6) Prueba a escribir los siguientes comandos.

```
clear; clc;
```

¿Qué hace *clear*? ¿Y *clc*?

(7) Declara las siguientes variables.

```
x = 1;
y = 1/(1-x);
z = 1/y;
u = z*y;
```

¿Qué observas? ¿Qué significa NaN?

(8) MATLAB acepta la notación científica. Por ejemplo;

```
x = 2e-10;
y = 3e10;
z = x*y;
```

¿Qué valor debe tener z?

Matrices

(9) Vamos a guardar como variable la matriz $\mathbf{A} = \begin{pmatrix} 1 & 0 & 1 \\ 2 & 1 & -1 \end{pmatrix}$ con el siguiente código.

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

Prueba a escribir las siguientes lineas en la consola:

```
A(1,2)
A(1,2) = 88;
A
A(5,5) = 43;
A
A(0,0)
A(:,1)
A(1,:)
```

¿Qué sucede en cada linea?

(10) Definir las siguientes matrices columna y fila en matlab:

$$\mathbf{v} = \begin{pmatrix} 1+i \\ 1-i \\ -i \end{pmatrix} \qquad \mathbf{u} = \begin{pmatrix} 1-i & 1 & -i \end{pmatrix}$$

Ejecuta las siguientes lineas e interpretalas.

```
A*v
A*u
A.*v
A.*u
v(2) - u(1)
ut = transpose(u)
u'
L = [v ut]
L2 = [u; v']
Z = zeros(5)
M = zeros(2,3)
I = eye(4)
length(u)
size(M)
```

(11) Define en matlab la matriz $A = \begin{pmatrix} 1 & -1 & 2 & 3 \\ 2 & -3 & 4 & 2 \end{pmatrix}$ y aplica las siguientes lineas de comando.

```
H = hermiteForm(A)
rank(A)
rank(H)
```

¿Qué matriz hemos obtenido? ¿Qué hace *rank*?

(12) Define en matlab las matrices $A = \begin{pmatrix} 1 & 2 & 1 \\ 2 & 1 & 2 \end{pmatrix}$, $B = \begin{pmatrix} -2 & 3 \\ 2 & 1 \end{pmatrix}$ y aplica las siguientes lineas de comando.

```
inv(A)
det(A)
inv(B)
det(B)
pinv(A) % Pseudoinversa de Moore Penrose
pinv(B)
```

Interpreta los resultados y aplica la función *sym*() sobre las matrices obtenidas. ¿Qué conclusiones obtienes? Prueba a ejecutar el siguiente código. ¿Qué clase de variable obtienes?

```
B = sym(B)
inv(B)
```

(13) Considera el siguiente sistema de ecuaciones lineales.

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

Almacena en MATLAB la matriz **A** de coeficientes y el vector **b** de términos independientes. Ejecuta las siguientes lineas.

```
x1 = A\b
x2 = inv(A)*b
```

¿Qué has obtenido? ¿Cuál de los dos métodos es *mejor*?

(14) Interpreta el siguiente código.

```
A = pascal(5)
null(A)
colspace(A)
A = sym(A)
null(A)
colspace(A)
```

(15) Interpreta el siguiente código.

```
A = pascal(3)
charpoly(A)
eig(A)
[P, D] = eig(A)
A = sym(A)
charpoly(A)
eig(A)
[P, D] = eig(A)
```

(16) Interpreta el siguiente código.

```
A = [1 -2 1; 0 -1 0; -1 1 3];
charpoly(A)
eig(A)
[P, D] = eig(A)
```

```
P*D*inv(P)-A
[P, J] = jordan(A)
P*J*inv(P)-A
```

Funciones

Como ya hemos visto, MATLAB tiene sus propias funciones predefinidas, las cuales tienen entradas y salidas. Por ejemplo, *inv*() admite una matriz como entrada y tiene una salida, *eig*() admite una matriz como entrada y puede devolver dos salidas.

Nosotros también podemos crear nuestras propias funciones para utilizarlas dentro de un *script*. Las funciones siempre serán declaradas en la parte más baja del código.

(17) Pincha en el boton *New Live Script*. Aquí escribiremos nuestras lineas de código que ejecutaremos más adelante. Es aconsejable empezar todos los scripts con las siguientes lineas.

```
clear;
clc;
```

¿Qué diferencia hay entre *New Script* y *New Live Script*? ¿Por qué *clear* y *clc*?

(18) Escribe el siguiente código.

```
clear;
clc;

suma = suma_resta(4,5)
[suma,resta] = suma_resta(4,5)

function [s,r] = suma_resta(x,y)
    s = x+y;
    r = x-y;
end
```

Ejecuta el código e interpreta los resultados.

(19) Crea una nueva función llamada *prop* cuya entrada sean los números (x, y) y cuya salida sea $\frac{x+y}{x-y}$. Haz uso de la función anterior.

(20) Interpreta el siguiente código.

```
clear;
clc;
h = 0.01; % Paso
x = 1:h:2;
y = f(x)

function y = f(x)
    y = x.^2;
end
```

Condicionales

Los condicionales son fundamentales en cualquier lenguaje de programación. Vamos a tratar aquí los bloques *if*, *elseif*, *else*.

(21) Prueba a dar distintos valores a la variable x en el siguiente código.

```
clear;
clc;

x = 5/2;

if (x-2) == (3-x)
    disp('¡Has acertado!')
end
```

(22) Prueba a dar distintos valores a la variable x en la siguiente implementación del anterior código.

```
clear;
clc;

x = 1;

if (x-2) == (3-x)
    disp('¡Has acertado!')
else
    disp('Intentalo de nuevo :(')
end
```

(23) Prueba a dar distintos valores a la variable x en la siguiente implementación del anterior código.

```
clear;
clc;

x = 5/2 + 0.00001;

if (x-2) == (3-x)
    disp('¡Has acertado!')
elseif abs((x-2) - (3-x)) < 0.01
    disp('¡Has estado muy cerca!')
else
    disp('Intentalo de nuevo :(')
end
```

(24) Los condicionales resultan de gran utilidad dentro de las funciones. Observa el siguiente ejemplo.

```
clear;
clc;

fib(7)

function y = fib(x)
    if x == 1
        y = 1;
    elseif x == 2
        y = 1;
    else
        y = fib(x-1)+fib(x-2);
    end
end
```

(25) Prueba esta función.

```
function y = fib(n)
    y = (((1+sqrt(5))/2)^n - ((1-sqrt(5))/2)^n)/sqrt(5);
end
```

Bucles

Al igual que en otros lenguajes de programación, en MATLAB contamos con los bucles *for* y *while*.

(26) Interpreta el siguiente código en el que se utiliza el bucle *for*.

```
clear;
clc;

for i = 1:10
    disp(['Contando: ',num2str(i)])
    pause(0.5)
end
```

Cambia 1:10 por 1:0.51:10, ¿Qué observas?

(27) Compara el siguiente código con el anterior.

```
clear;
clc;

i=1;
while i <= 10
    disp(['Contando: ',num2str(i)])
    i = i + 1;
    pause(0.5)
end
```

¿Qué diferencias observas?

(28) En algunas ocasiones el comando *break* resulta útil. Interpreta el siguiente código.

```
clear;
clc;

i=1;
while true
    disp(['Contando: ',num2str(i)])
    i = i + 1;
    pause(0.5)
    if i>10
        break
    end
end
```

Plot

Vamos a considerar los siguientes puntos en $\mathbb{R}^2$:

$$P_1 = (x_1, y_1), \ldots, P_n = (x_n, y_n)$$

En MATLAB vamos a definir las matrices fila x e y que contengan las coordenadas de los puntos. El comando plot(x,y) crea una gráfica de líneas en 2D uniendo los puntos en el orden especificado.

(29) Vamos a dibujar un cuadrado escribiendo el siguiente código.

```
clear;
clc;

x = [0 1 1 0 0];
y = [0 0 1 1 0];
plot(x,y)
xlabel("x")
ylabel("y")
title("Cuadrado azul")
```

(30) Interpreta el siguiente código.

```
clear;
clc;

h = 0.01;
x = 0:h:2;
y = x.^2;
plot(x,y,'--r')
xlabel("Tiempo (ídas)")
ylabel("Beneficios (euros)")
title("Estrategia anual de 'Mudanzas Pepa'")
```

Modifica el código eliminando la linea $y = x^2$ por $y = f(x)$ y crear una función f que devuelva la misma gráfica.

(31) Interpreta el siguiente código.

```
clear;
clc;

h = 0.01;
x = 0:h:5;

subplot(2,2,1);
plot(x,x,'g')
xlabel('x (segundos)')
ylabel('y (m)')

subplot(2,2,2);
plot(x,x.^2,'k')
xlabel('x (segundos)')
ylabel('z (Hz)')

subplot(2,2,4);
plot(x,x.^3)
xlabel('u (minutos)')
ylabel('v (km)')

subplot(2,2,3);
plot(x,x.^4,'r')
xlabel('y (minutos)')
ylabel('w (Ohm)')
```

(32) Comparar con el siguiente código.

```
clear;
clc;

h = 0.01;
x = 0:h:5;

hold on
    plot(x,x,'g')
    plot(x,x.^2,'k')
    plot(x,x.^3)
    plot(x,x.^4,'r')
hold off
```

(33) Podemos aprovechar los bucles junto con el comando hold para dibujar indefinidamente una gráfica sobre una misma pantalla.

```
clear;
clc;

hold on
    n=1;
    while true
        x(n) = n;
        y(n) = (-1)^n;
        plot(x,y)
        n = n+1;
        pause(0.5)
    end
hold off
```

¿Qué problemas observas? Mejora el código.

Problemas propuestos

(34) Crea una función para calcular la suma de los inversos de los n primeros cuadrados utilizando $sum()$.

(35) Crear una función que se llame a sí misma para calcular el factorial de un número n.

(36) Crear una función para determiar si un número es primo o no. Para ello utilizar el comando $rem(m, n)$, que devuelve el resto de dividir m entre n.

(37) Aprovechar la anterior función para crear un script en el que se muestren los 100 primeros números primos.

(38) Crea una función que calcule el ángulo entre dos vectores.

(39) Crea una función cuya entrada sea una matriz $\mathbf{A}$ y la salida sea una matriz $\mathbf{Q}$ cuyas columnas están formadas por aplicar el método de Gram Schmidt sobre las columnas de $\mathbf{A}$. Además, implementa la función para que los vectores obtenidos salgan ya normalizados.

(40) Crea una función cuyas entradas sean las coordenadas de un vector $\mathbf{v}$ y una matriz $\mathbf{A}$. La salida de la función debe de ser la proyección de $\mathbf{v}$ sobre el espacio de columnas de $\mathbf{A}$.