

## MATLAB

### PROGRAMACIÓN:

Esta es una introducción a la programación de **scripts** y **funciones** en Matlab.

El primer interrogante que puede surgir es **¿qué es un script?**. Este término inglés significa: escrito, guión, nota; el término guión es el que más se utiliza en las traducciones al español.

Recordemos que en Matlab trabajamos sobre el Workspace que es la ventana inicial donde ingresamos comandos y los ejecutamos directamente.

Frecuentemente una serie de comandos debe ser ejecutada varias veces durante una misma sesión, para evitarnos el trabajo de ingresarlos continuamente existen los scripts.

Antes de comenzar con los aspectos intrínsecos de la programación conozcamos el entorno que ofrece Matlab para el desarrollo de funciones y scripts. Con entorno quiero referirme al editor de archivos propio.

### Editor

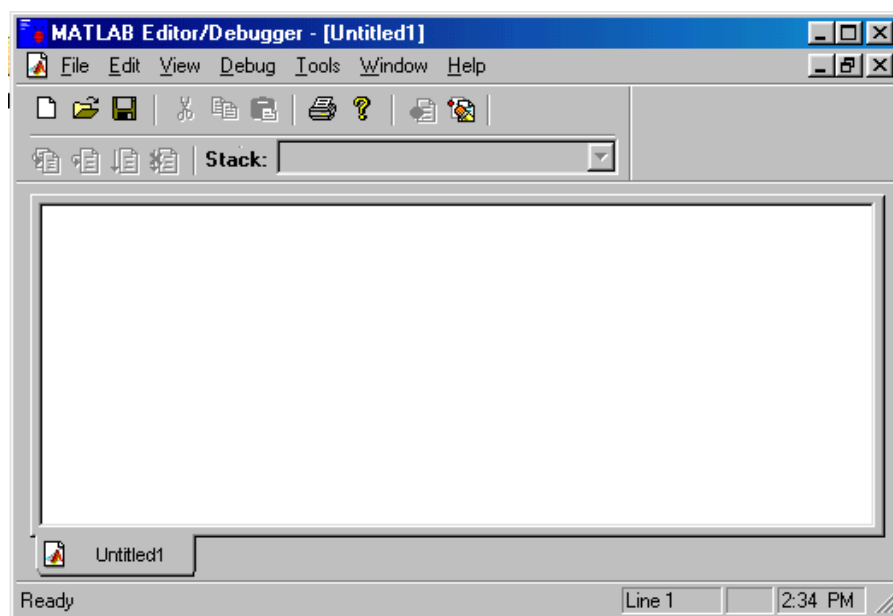
Las **funciones** y **scripts** no son más que archivos de texto ASCII, con la extensión **\*.m**, que contienen **definición de funciones** o **conjuntos de comandos** respectivamente.

El editor permite tanto crear y modificar estos archivos, como ejecutarlos paso a paso para ver si contienen errores (proceso de **Debug** o depuración, eliminar errores al programa). También Matlab permite que utilicemos cualquier editor (edit de DOS, Word, Notepad, etc.), ya que los archivos son sólo de texto. El siguiente gráfico muestra la ventana principal del **Editor/Debugger**.

Puede apreciarse que aun no se ha introducido código alguno.

El editor muestra con diferentes colores los diferentes tipos o elementos constitutivos de los comandos. (en *verde* los comentarios, en *rojo* las cadenas de caracteres, etc.). El editor además indica que las comillas o paréntesis que se abren se cierran correctamente.

En la siguiente figura se observa un script llamado **prueba.m**, (en la barra azul puede verificarse), el cual contiene un conjunto de comandos, relativos al uso de matrices y cadenas.



**NOTAS:** La función **rand(n)** retorna una matriz cuadrada **nxn**, cuyos elementos son números aleatorios entre 0 y 1, la función **magic(n)** también retorna una matriz cuadrada **nxn** pero con la particularidad de que la suma de los elementos de cada una de sus filas, columnas y diagonales tiene el mismo valor.



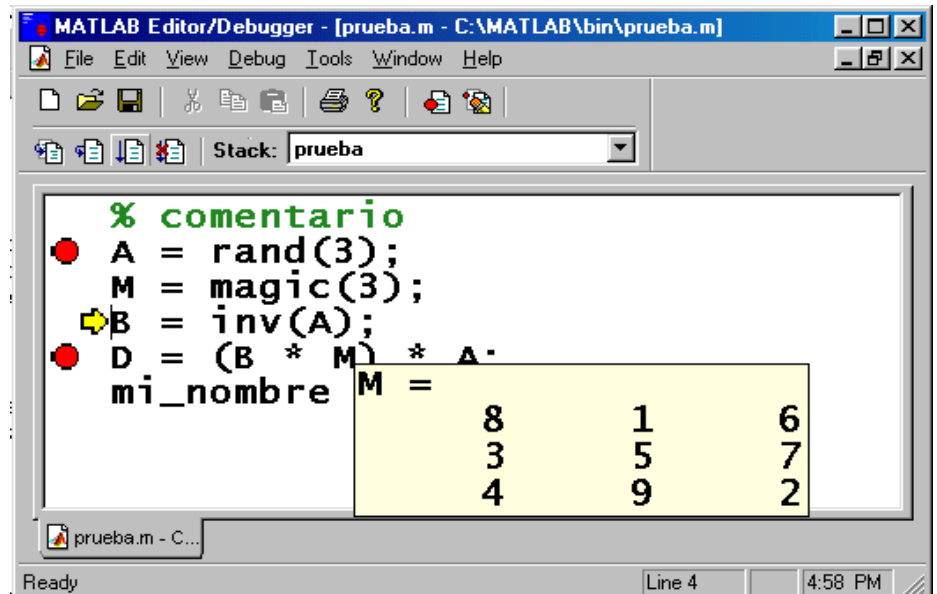
Recuerde que las cadenas de caracteres se delimitan con comillas simples y no con comillas dobles.

La función **inv(M)** retorna la matriz inversa de M.



Es importante la utilización del punto y coma (;) al final de cada sentencia, pues evita que Matlab haga eco de la misma al ejecutarla produciendo salida innecesaria al Workspace.

La siguiente figura corresponde a la ejecución de este archivo de comandos controlado con el **Debugger**. Dicha ejecución se comienza con el comando **Run** en el menú **Tools**.

Los puntos rojos que aparecen en el margen izquierdo son **breakpoints** (puntos donde se detiene la ejecución del programa). La flecha amarilla indica la sentencia en que está detenida la ejecución; cuando el cursor se coloca sobre una variable (en este caso sobre M) aparece una ventana con los valores de esa variable. Puede verse en la figura que está activa la segunda barra de herramientas que corresponde al **Debugger**. Cada botón significa lo siguiente.



Botón	Significado
	<b>Set/Clear Breakpoint.</b> Coloca o borra un <b>breakpoint</b> en una línea.
	<b>Clear all Breakpoints.</b> Elimina todos los breakpoints que hay en el archivo.
	<b>Step In.</b> Avanzar un paso, y si en ese paso hay una llamada a función de usuario, entra en dicha función.
	<b>Single Step.</b> Avanzar un paso sin entrar en las funciones de usuario que se llamen en esa línea.

	<b>Continue.</b> Continuar la ejecución hasta el siguiente <b>breakpoint</b> .
	<b>Quit Debugging.</b> Terminar la ejecución del <b>Debugger</b> .

El **Debugger** es un programa enormemente útil para detectar y corregir errores, que hay que conocer muy bien.

Por último contestamos a las siguientes preguntas:

- **¿Cómo accedemos al editor?**

Desde el Workspace: **>> edit;**

Desde el menú **File / New / M-file**.

- **¿Cómo se ejecuta un script?**

Sencillamente se debe introducir su nombre en la línea de comandos.

O mediante el editor como describimos anteriormente.

- **¿Cómo se ejecuta una función?**

De igual manera que un script pero sus argumentos deben pasarse entre paréntesis y separados por coma.

Ejemplo: **mi\_funcion**(arg1, arg2, ..., argn)

- **¿Cómo se programa en Matlab?**

La respuesta a esta pregunta no es sencilla, la responderemos en varios pasos.

Sepamos algunas cosas respecto a las variables que podemos usar en nuestros programas.

**Variables:** secuencia de caracteres que

**Funciones de entrada y salida:**

- ♦ **Input:** Esta función permite imprimir un mensaje en la línea de comandos de Matlab y recuperar como valor de retorno un valor numérico o el resultado de una expresión tecleada por el usuario.

**Ejemplo:** **n = input('Ingrese un número: ');**

- ♦ **Disp:** Esta función permite imprimir en pantalla un mensaje de texto o el valor de una matriz, pero sin imprimir su nombre. En realidad, disp siempre imprime vectores y/o matrices: las cadenas de caracteres son un caso particular de vectores.

**Ejemplo:** **disp('El programa ha terminado');**

**Comencemos con las sentencias fundamentales ....**

**Sentencia If:**

En su forma más simple, la sentencia if se describe como sigue:

```

if condicion
    sentencias
end
```

Observe que la condición no va encerrada entre paréntesis.

Existe también la "bifurcación múltiple", en la que pueden concatenarse tantas condiciones como se desee, y que tiene la forma:

```

if condicion1
    bloque1
elseif condicion2
    bloque 2
elseif condicion3
    bloque 3
else % opción por defecto para cuando no se cumplan las otras conds.
    bloque 4
end

```

**NOTA:** El uso de **else** es opcional.

Para crear condiciones debemos conocer tanto los operadores relacionales como los operadores lógicos.

Operadores relacionales		Operadores lógicos	
<b>igualdad</b>	<b>==</b>	<b>and</b>	& (ampersand)
<b>desigualdad</b>	<b>~=</b>	<b>or</b>	(pipe)
<b>menor</b>	<b>&lt;</b>	<b>not</b>	~ (tilde)
<b>mayor</b>	<b>&gt;</b>	<b>xor</b>	o exclusivo
<b>menor o igual</b>	<b>&lt;=</b>	<b>any</b>	True si <b>algún</b> elemento del vector es true
<b>mayor o igual</b>	<b>&gt;=</b>	<b>all</b>	True si <b>todos</b> los elementos del vector son true

#### Ejemplo de uso de if:

```
nro = input('Ingrese un número positivo: ');
```

```

if mod(nro,2)==0 disp('es par');
else disp('es impar');
end

```

#### Ejemplo de uso de if/elseif:

```
nro = input('Ingrese un número positivo: ');
```

```

if nro==1 disp('Ud. ingreso 1');
elseif nro==2 disp('Ud. ingreso 2');
elseif nro==3 disp('Ud. ingreso 3');
else disp('El número es mayor que 3');
end

```

#### Sentencia Switch:

Esta sentencia realiza la función análoga a un conjunto **if...elseif** concatenados. Su forma general es:

```

switch switch_expresion
    case case_expr1,
        bloque 1
    case {case_expr2, case_expr3, case_expr4, ...},

```

```

        bloque 2
    ...
    otherwise,
        bloque 3
end

```

**NOTA:** Al igual que con **else**, el uso de **otherwise** es opcional.

Al principio se evalúa la **switch\_expresion**, cuyo resultado debe ser un número escalar o una cadena de caracteres. Este resultado se compara con las **case\_expr**, y se ejecuta el bloque de sentencias que corresponda con ese resultado. Si ninguno es igual a **switch\_expresion** se ejecutan las sentencias correspondientes a **otherwise** (que significa de otra manera). Es posible poner varias **case\_expr** dentro de llaves, como se ve arriba.

### Ejemplo de uso switch:

```

nro = input('Ingrese un número positivo: ');

switch nro
    case 1, disp('Ud. ingreso 1'),
    case 2, disp('Ud. ingreso 2')
    case 3, disp('Ud. ingreso 3')
    otherwise, disp('El número es mayor que 3')
end

```

### Sentencia For:

La sentencia for repite un conjunto de sentencias un número predeterminado de veces. La siguiente construcción ejecuta sentencias con valores de **i** de **1** a **n**, variando en uno:

```

    for i=1:n
        sentencias
    end

```

For anidados:

```

    for i=1:n
        for j=1:m
            sentencias
        end
    end

```

***¿Cuántas veces se ejecutará este bloque de sentencias?***

### Ejemplos de uso de for:

```

nro = input('Ingrese un número positivo: ');

```

```

for i=1:nro
    disp(i)
end

```

OTROS CASOS

```

for i=nro:-0.2:1
    disp(i)
end

```

```

for i=A
    disp(i)
end

```

## **Sentencia While:**

La sintaxis de la estructura while es la siguiente.

```
while condicion
    sentencias
end
```

donde **condicion** puede ser una expresión vectorial o matricial. Las sentencias se siguen ejecutando mientras haya elementos distintos de cero en **condicion**, es decir, mientras haya algún o algunos elementos **true**. El bucle termina cuando todos los elementos de **condicion** son **false** (es decir, cero).

### **Ejemplo de uso de while:**

```
nro = input('Ingrese un número positivo: ');
i=0;

while i<=nro
    disp(i);
    i=i+1;
end;
```

### **Ejemplo de uso de while con break (corta el bucle):**

```
nro = input('Ingrese un número positivo: ');
i=0;

while i<=nro
    disp(i);
    i=i+1;
    if i==2
        break;
    end;
end;
```

## **FUNCIONES:**

Los scripts sólo aportan a Matlab la manera de realizar tareas de una manera más rápida. Las **funciones** permiten definir funciones enteramente análogas a las de Matlab, con su **nombre**, sus **argumentos** y sus **valores de retorno**. Los archivos \*.m que definen funciones permiten extender las posibilidades de Matlab; de hecho existen bibliotecas de archivos \*.m que se venden (**toolkits**) o se distribuyen gratuitamente.

Tanto en scripts como en funciones la primer línea de comentarios conforman el help.

### **Ejemplo:**

Dado el script llamado **cuadrado.m**:

```
% este es un script que eleva un número al cuadrado
nro = input('Ingrese un número positivo: ');
nro = nro * nro;
disp('El número ingresado elevado al cuadrado es: ');
disp(nro);
```

Y cuando escribimos en el Workspace:

**>> help cuadrado**

***este es un script que eleva un número al cuadrado***

### **Construcción de funciones:**

La **primer línea** de un archivo llamado **ejemplo.m** que define una función tiene la forma:

**function [lista de valores de retorno] = ejemplo (lista de argumentos)**

donde **ejemplo** es el nombre de la función. Entre corchetes y separados por coma van los **valores de retorno** (siempre que haya más de uno), y entre paréntesis también separados por comas los **argumentos**. Puede haber funciones sin valores de retorno y también sin argumentos. Si no hay valores de retorno se omiten los corchetes. Y el signo igual (=); si sólo hay un valor de retorno no hacen falta poner corchetes. Tampoco hace falta poner paréntesis si no hay argumentos.

Una diferencia importante entre Matlab y los lenguajes de programación convencionales, como C/C++/Java es que una función no modifica nunca los argumentos que recibe.

Las variables definidas dentro de una función son variables locales, en el sentido de que son inaccesibles desde otras partes del programa y en el que no interfieren con variables del mismo nombre definidas en otras funciones o partes del programa. Para que la función tenga acceso a variables que no han sido pasadas como argumentos es necesario declarar dichas variables como variables globales, tanto en el programa principal como en distintas funciones que deben acceder a su valor.

Dentro de la función, los valores de retorno deben ser calculados en algún momento (no hay sentencia **return**). Se utiliza return sólo para salir forzosamente del cuerpo de la función.

### **Ejemplos de funciones:**

% función que retorna la división y el resto de la misma de  
% dos números dados

%  
% USO: [div, resto] = divmod(nro1, nro2);

**function** [division, modulo] = divmod(x,y);  
division = x/y;  
modulo = mod(x,y);

**Los restantes ejemplos no contienen las primeras líneas de comentarios que conforman el help de las mismas, queda como ejercicio para el lector crear dicho help. El uso de comentarios explicativos del uso y propósito de la función es una buena práctica de programación.**

♦ **La siguiente función termina abruptamente al alcanzar return.**

**function** [division, modulo] = divmod(x,y);

```

division = 1;
modulo = 1;
return;
division = x/y;
modulo = mod(x,y);

```

- ♦ **El cálculo del factorial. Observe como se capturan los casos particulares para el 0 y el 1. Y el caso de entrada inválida,  $n < 0$ .**

% retorna el factorial del número dado

```

function [] = factorial(n);

if n < 0
    disp('Ingrese un número positivo !')
    return;
end;

if n==0 | n==1
    disp('El factorial de ');
    disp(n);
    disp('es 1');
    return;
end;

mult = 1;
for i = 1:n;
    mult = mult * i;
end;

disp('El factorial de '); disp(n); disp('es: '); disp(mult);

```

- ♦ **Ejemplo de uso de una función definida por el usuario para definir una nueva función. El primer código conforma una función que calcula el número combinatorio de dos valores dados (el lector debe agregar código para impedir cálculos erróneos, como cuando  $n$  y  $m$  sean negativos,  $n < m$ , etc.)**

% Función número combinatorio

```

function [] = comb(n,m);
comb = ones(1,3);
for i=1:n, comb(1) = comb(1) * i; end;
for i=1:m, comb(2) = comb(2) * i; end;
for i=1:(n-m), comb(3) = comb(3) * i; end;

disp('El combinatorio es ');
comb(1)/(comb(2)*comb(3));

```

Se puede observar que estamos calculando en los tres for los números factoriales de  $n$ ,  $m$  y  $n-m$ , en dicho orden. Pero como anteriormente creamos la función factorial utilizamos a la misma (uno de los principios de programación es la reutilización de código) y en lugar de los for tendremos llamadas a función.



```
% Función número combinatorio  
function [] = comb(n,m);  
disp('El combinatorio es ');  
disp(factorial(n)/(factorial(m)*factorial(n-m)));
```

**En clases crearemos funciones relacionadas con los ejercicios de las prácticas de la materia.**

**Diego A. Bottallo**