MÉTODOS NUMÉRICOS
GUÍA DE LABORATORIO NRO. 3
INTRODUCCIÓN AL MATLAB (Parte 3)

OBJETIVOS:

Conocer la declaración de archivos de comandos en MATLAB.
Realizar ejercicios de archivos de comandos en MATLAB.
Conocer la declaración de funciones en MATLAB.
Resolver problemas con archivos y funciones.

MEDIOS Y MATERIALES EDUCATIVOS:

Guía de laboratorio, computadora, software de Matlab, tutoriales y manuales de Matlab, apuntes, Internet y flash memory.

INFORME:

Realizar un informe del laboratorio realizado, puede ser individual o de un máximo de dos estudiantes.

TAREA 1. ARCHIVO DE COMANDOS.

Ficheros de comandos (scripts)

Los ficheros de comandos, conocidos también como scripts, son archivos de texto sin formato (ASCII) con la extensión característica de los archivos de MATLAB (*.m), se utilizan para almacenar una serie de comandos o instrucciones que se ejecutan sucesivamente y que habrán de realizar una tarea específica. Los scripts de MATLAB pueden editarse utilizando cualquier editor de texto sin formato (Bloc de Notas, Notepad++, Sublime Text, etc…), aunque es más recomendable utilizar el editor de MATLAB, puesto que proporciona herramientas que facilitan la corrección de errores, el control sobre la ejecución del código y la capacidad de autocompletado y sugerencias cuando se utilizan funciones nativas de MATLAB.
Para crear un nuevo script puede pulsar la combinación Ctrl + N (bajo

SO Windows), o buscar en la interfaz de MATLAB la opción New y enseguida seleccionar Script; si prefiere hacerlo desde la ventana de comandos puede introducir el comando edit que le abrirá un nuevo script.
Para guardar un fichero de comandos utilice la opción Save de la barra de herramientas o bien mediante la combinación de teclas Ctrl +S en Windows. Debe tomarse en cuenta que al guardar un script se le proporcione un nombre que no entre en conflicto con las funciones nativas de MATLAB o las palabras reservadas del lenguaje. Algunas recomendaciones que deben seguirse para nombrar un script son:

El nombre deberá contener sólo letras, números o guiones bajos.
No deberá comenzar con un carácter diferente a una letra (Por ejemplo: 102metodo.m, es un nombre inválido dado que comienza con un número).

Evite utilizar nombres de funciones nativas de MATLAB o palabras reservadas del lenguaje que podrían ocasionar conflictos.

Para saber cuáles son las palabras reservadas del lenguaje puede teclear iskeyword en la ventana de comandos y MATLAB le devolverá un cell array de strings:

```
>> iskeyword
ans =
'break'
'case'
'catch'
'classdef'
'continue'
'else'
'elseif'
'end'
'for'
'function'
'global'
'if'
'otherwise'
```

'parfor'
'persistent'
'return'
'spmd'
'switch'
'try'
'while'

Además, puede verificar si el nombre de un fichero existe utilizando la función exist:

```
>>  exist('size')
ans =
      5
```

Si devuelve un resultado diferente de cero, entonces ese nombre está siendo utilizado en una de las funciones/scripts incluidas en el path de MATLAB.

Ejecutando scripts

La utilidad de los scripts radica en la posibilidad de almacenar comandos de manera estructurada y poderlos ejecutar posteriormente, para hacerlo puede ir a la opción Run de la interfaz principal de MATLAB y entonces se ejecutarán todas las instrucciones que conforman el script.

Otra forma es ubicarse en la carpeta del script y teclear el nombre del fichero en el Command Window. Claro que si el fichero no se encuentra en el Current Folder este no se ejecutará, exceptuando aquellos que sean agregados al Path de MATLAB.

De los ficheros de funciones…
Para ejecutar los ficheros que contienen definiciones de funciones no se procede como se ha descrito anteriormente, puesto que, normalmente, estos necesitan información extra o argumentos de entrada que deben pasarse utilizando la sintaxis de llamada de funciones.

Modificando el Path de MATLAB

Primero, ¿qué es el path de MATLAB?, en resumen, son directorios o carpetas en los cuales MATLAB busca las funciones, clases y/o ficheros en general que el usuario demanda durante una sesión.
Si teclea el comando path, este imprimirá en pantalla una lista de directorios, ordenados de manera jerárquica, en los cuales MATLAB busca las sentencias introducidas.
Por ejemplo, vamos a suponer que en nuestro directorio actual tenemos un fichero llamado principal.m y que tenemos también una carpeta utils en la cual tenemos algunos códigos necesarios ("codigo1.m y codigo2.m') para que nuestro código principal funcione:

|
• • • • principal.m
• • • • utils
• • • • codigo1.m
• • • • codigo2.m

Una solución evidente (pero muy tosca) es colocar los ficheros codigo1.m y codigo2.m en el mismo directorio, pero claro, eso implicaría tener muchos ficheros en una misma carpeta, lo cual no suele ser buena idea.
Y la otra solución consiste en agregar la carpeta utils al path de MATLAB, lo cual es tan sencillo como ejecutar:

```
>>  path(path,  'utils');
```

Con esto podrá llamar los ficheros codigo1.m y codigo2.m desde principal.m sin necesidad de colocarlos en la misma carpeta.

TAREA 2. FUNCIONES.

Funciones, una introducción

Las funciones son porciones de código que por lo general aceptan argumentos o valores de entrada y devuelven un valor de salida. Una función es una herramienta muy útil en la programación, dado que

permite la reutilización de código para procedimientos que lo requieran, así como una facilidad significativa para mantener el código, lo cual se traduce en una mayor productividad. MATLAB, de hecho, está compuesto por una multitud de funciones agrupadas en *toolboxs*, cada una de ellas pensada para resolver una situación concreta.

Una función debe definirse en un fichero único, es decir, por cada función creada debemos utilizar un archivo *.m, mismo que tendrá el nombre dado a la función.

La estructura básica de una función contiene los siguientes elementos:

- La palabra reservada function
- Los valores de salida
- El nombre de la función
- Los argumentos de entrada
- Cuerpo de la función

Para una mejor comprensión de cada uno de esos elementos, refiérase a las siguientes líneas de código:

```
function res = suma(a,b)
        res = a+b;
end
```

La función anterior llamada *suma,* recibe como argumentos de entrada dos valores numéricos a y b, y devuelve un resultado guardado en res que equivale a la suma aritmética de las variables de entrada. Note que el valor de retorno debe asignarse a la variable indicada en la línea de definición.

Si ejecutamos la función en la ventana de comandos obtenemos algo similar a esto:

```
>> s=suma(3,2)
s =
```

5

Si no hace una asignación el resultado devuelto se guarda en la variable ans.

Verificar argumentos de entrada y salida

Cuando se crea una función es recomendable verificar si la cantidad de argumentos de entrada corresponden a los soportados, o bien, si el tipo de dato que se ha introducido es el adecuado para proceder con el resto de la programación; MATLAB proporciona los comandos *nargin* y *nargout* que sirven para *contar* el número de argumentos de entrada y salida respectivamente.

Utilizando como ejemplo la función suma creada con anterioridad, podemos verificar que el número de argumentos sean exactamente dos para poder proceder y en caso contrario enviar al usuario un mensaje de error en la ventana de **comandos**, el código implicado sería similar al siguiente:

```
function res = suma(a,b)
    if nargin==2
        res=a+b;
    else
        error('Introduzca dos argumentos de entrada');
    end
end
```

Si ejecutamos la función pasándole solamente un argumento de entrada nos devolverá un mensaje de error:

```
>> s=suma(7)
Error using suma (line 5)
```

Introduzca dos argumentos de entrada

Sub-funciones

Las sub-funciones son funciones definidas dentro del espacio de otra función principal. Se utilizan como funciones auxiliares con la finalidad de hacer más legible el código y facilitar la depuración de errores. Enseguida se muestra el ejemplo de una sub-función:

```
function r=isfibo(num)
% Determina si un número entero formaparte
% de la sucesión de Fibonacci, devuelve un valor
% de tipo lógico.
    ff=fibonacci(num);
    if any(ff==num)
        r=true;
    else
        r=false;
    end
    function F=fibonacci(n) 12          F(1:2)=1;
        i=3;
        while  1
            F=[F  F(i-1)+F(i-2)];
            if F(end) >= n,break,end;
            i=i+1;
        end
    end
end
```

La función anterior isfibo determina si el entero pasado como

argumento de entrada pertenece a la sucesión de Fibonacci, para ello utiliza como una función auxiliar a la sub-función fibonacci que se encarga de generar la sucesión de Fibonacci en un intervalo dado y guardarlo en un vector de salida. Una sub-función puede ser llamada solamente por la función principal que la contiene.

Argumentos variables

En la introducción a las funciones se ha mencionado que estas por lo general tienen un número específico de argumentos de entrada y salida, no obstante se presentan situaciones en donde los argumentos de entrada o salida de una función no son fijos o bien los argumentos pueden ser demasiados de tal modo que resulte incómodo definirlos en la línea correspondiente. Para solucionar lo anterior MATLAB permite el uso de varargin y varargout como argumentos de entrada y salida respectivamente. Para tener una idea más práctica de lo anterior véase el ejemplo siguiente:

```
function m=max2(varargin)
if nargin==1
        v=varargin{1};
        m=v(1);
        for  i=2:length(v)
                if  v(i)>m
                        m=v(i);
                end
        end
        elseif nargin==2
                a=varargin{1};
                b=varargin{2};
        If a>b
                m=a;
        else
                m=b;
        end
end
```

end

La función anterior max2 emula a la función nativa max, puede recibir como argumento de entrada un vector o bien dos valores escalares. Si observa el código anterior notará que varargin es un cell array que guarda todos los argumentos de entrada pasados a la función, como se verá en el capítulo 3 la manera de acceder a los elementos de un cell array es utilizando la sintaxis: var{k}, donde var es la variable en la que está almacenada el cell array y k es el k-ésimo elemento contenido en el cell array.

Ayuda de una función

Como parte de las buenas prácticas de programación es recomendable incluir comentarios dentro de una función que indiquen el propósito de esta, así como una descripción breve de los argumentos de entrada y salida e incluso un ejemplo concreto de la misma.

Por convención estos comentarios deben colocarse justamente después de la definición de la función y antes de todo el código restante, además de que esto servirá como referencia al resto de usuarios también le permitirá a MATLAB interpretarlo como las líneas de ayuda cuando se le solicite expresamente mediante la función help. Véase el siguiente ejemplo:

```
function [x1,x2]=ecuad(a,b,c)
% Resuelve una ecuación cuadrática de la forma:
%  a*x^2+b*x+c=0
%
% Argumentos de entrada:
%       a     -     Coeficiente cuadrático
%       b     -     Coeficiente lineal
```

```
%       c       -       Coeficiente constante
%
% Argumentos de salida:
%       x1,x2 - Raíces de la ecuación cuadrática
%
%   Ejemplo:
%       >>      [r1,r2]=ecuad(-1,2,1);
%


x1=(1/(2*a))*(-b+sqrt(b^2-4*a*c));
x2=(1/(2*a))*(-b-sqrt(b^2-4*a*c));
end
```

Podemos teclear help ecuad en la ventana de comandos y verificar lo que MATLAB nos devuelve como ayuda de la función:

```
>> help ecuad
Resuelve una ecuación cuadrática de la forma: 3 a*x^2+b*x+c=0
Argumentos de entrada:
a       -       Coeficiente cuadrático
b       -       Coeficiente lineal
c       -       Coeficiente constante
Argumentos de salida:
x1,x2   - Raíces de la ecuación cuadrática

Ejemplo:
```

```
>>  [r1,r2]=ecuad(-1,2,1);
```

Es común agregar a la ayuda de una función algunas referencias hacia otras funciones similares, para ello en los comentarios debe agregar una línea que comience con las palabras “SEE ALSO” (Ver también), seguidas de las funciones similares separadas por comas, véase el ejemplo a continuación:

```
function [x1,x2]=ecuad(a,b,c)
% Resuelve una ecuación cuadrática de la forma:
%  a*x^2+b*x+c=0
%
% Argumentos de entrada:
%      a      -      Coeficiente cuadrático
%      b      -      Coeficiente lineal
%      c      -      Coeficiente constante
%
% Argumentos de salida:
%      x1,x2 - Raíces de la ecuación cuadrática
%
%   Ejemplo:
%      >>      [r1,r2]=ecuad(-1,2,1);
%
%   SEE   ALSO   roots,solve,fzero
%
x1=(1/(2*a))*(-b+sqrt(b^2-4*a*c));
```

*x2=(1/(2*a))*(-b-sqrt(b^2-4*a*c));*

end

>> help ecuad

*Resuelve una ecuación cuadrática de la forma: 25 a*x^2+b*x+c=0*

Argumentos de entrada:

a - Coeficiente cuadrático

b - Coeficiente lineal

c - Coeficiente constante

Argumentos de salida:

x1,x2 - Raíces de la ecuación cuadrática

Ejemplo:

>> [r1,r2]=ecuad(-1,2,1);

SEE ALSO roots,solve,fzero

TAREA 3. OPERACIONES.

a) Explicar la función round(), las ventajas y desventajas.

b) Suponer que trabaja de cajero en un banco de lunes a viernes, al mes cuatro semanas, realiza 200 transacciones de cobros de cheques al día. Los pagos de los cheques tiene dos decimales. Sin embargo solo puede pagar decimos de centavos y no unidades de centavos. Se desea saber cuantos centavos no le paga a los clientes en un mes y en un año?.

   Realizar un archivo .m, donde se realicen estos cálculos y muestre

el resultado. Sugerencia, utilizar las funciones randi() y round().

c) Definir las siguientes matrices:

$A = \begin{pmatrix} 2 & 6 \\ 3 & 9 \end{pmatrix}$

$B = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$

C= $\begin{pmatrix} 5 & 5 \\ 5 & 3 \end{pmatrix}$

Crear la siguiente matriz (que tiene sobre la diagonal las matrices A, B, C) sin introducir elemento a elemento:

$$G = \begin{pmatrix} 2 & 6 & 0 & 0 & 0 & 0 \\ 3 & 9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 0 & 0 \\ 0 & 0 & 3 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 5 & 5 \\ 0 & 0 & 0 & 0 & 5 & 3 \end{pmatrix}$$

Realizar sobre G las siguientes operaciones, guardando todos los resultados en variables distintas:

(i) Borrar la última fila y la última columna de G.

(ii) Extraer la primera submatriz 4 x 4 de G.

(iii) Extraer la submatriz {1; 3; 6} x {2; 5} de G.

(iv) Reemplazar G(5; 5) por 4.

d) Hacer dos funciones de resolución de sistemas de ecuaciones lineales, parámetros de entrada la matriz A y el vector b, salida el vector X.

(i) El comando inv calcula la matriz inversa de una matriz regular. Por lo tanto, el sistema de ecuaciones lineales Ax =b puede resolverse simplemente mediante

>> inv(A)*b

(ii) Sin embargo, hay una forma de hacer que MATLAB calcule la solución de Ax =b utilizando el método de Gauss (reducción del sistema mediante operaciones elementales de fila). Este método es preferible al anterior ya que el cálculo de la inversa involucra más operaciones y es más sensible a errores numéricos. Se utiliza la llamada división matricial izquierda \.

```
>> A\b
```

Probar los dos métodos con el sistema siguiente:

$$\begin{cases} 2x - y + 3z = 4 \\ x + 4y + z = 2 \\ 6x + 10y + 3z = 0 \end{cases}$$

Analizar los resultados.
Medir los tiempos de cada función utilizando tic y toc de Matlab, y dar conclusiones.