MÉTODOS NUMÉRICOS
GUÍA DE LABORATORIO NRO. 1
INTRODUCCIÓN AL MATLAB (Parte 2)

OBJETIVOS:

Conocer la declaración de vectores y matrices en MATLAB.
Realizar operaciones entre vectores y matrices en MATLAB.
Conocer las bifurcaciones y bucles en MATLAB.
Resolver problemas con errores.

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. VECTORES Y MATRICES EN MATLAB

Arreglos en MATLAB:

- Declaración de vector fila: V = [0 1 2 3 4] separa cada elemento por un espacio.
- Declaración de vector columna: V = [0; 1; 2; 3; 4] separa cada columna con ‘;’.
- Declaración de matrices: M = [11 12 13; 21 22 23; 31 32 33] separa los elementos de cada fila con espacio y cada columna con un ‘;’.
- Generar vectores: Para generar un vector, existen múltiples comandos, entre ellos están:
- V=a:div:b; donde a es el punto inicial, b es el punto final y div es el número de pasos entre cada par de elementos cercanos.
- V=linspace(a,b,c); crea un vector de a hasta b de c elementos.

Operaciones básicas entre vectores:

- **Suma – Resta (+/-):** Para sumar o restar 2 o más vectores deben contener la misma cantidad de elementos entre sí para poder realizar la operación. La operación se realiza de igual manera que la adición de números, de la siguiente manera: A + B – C, donde A, B y C son vectores. Esto también puede extenderse a las operaciones con matrices.
- **Transpuesto (‘):** El transpuesto es el que permite convertir un vector fila en un vector columna y viceversa, al igual que sacar la transpuesta de una matriz. La operación se realiza con el carácter especial ‘, Ej. A’.
- **Multiplicación (*):** Para multiplicar dos vectores se deben seguir las mismas condiciones del algebra lineal. Ej. sea A y B dos vectores fila de igual número de elementos, el producto entre ambos debe ser A por B transpuesto, Ej. res = A*B’=escalar.
- **Multiplicación punto a punto (.*):** La multiplicación punto a punto entre vectores da como resultado otro vector en el que sus componentes son el resultado del producto de elemento a elemento entre los vectores

Para definir un vector fila, solo se debe introducir sus coordenadas entre corchetes:

```
>>v=[1 2 3]        % Vector de 3 coordenadas v=
1 2 3

>>w=[4 5 6];
```

El operador ' es el de trasposición (en realidad trasposición y conjugación):

```
>>w'

ans =
4
5
6
```

Si queremos declarar un vector de coordenadas equiespaciadas entre dos dadas, por ejemplo, que la primera valga 0, la última 20 y la distancia entre coordenadas sea 2, basta poner:

```
>>vect1=0:2:20

vect1 =
0 2 4 6 8 10 12 14 16 18 20
```
Equivalentemente, si lo que conocemos del vector es que la primera coordenada vale 0, la última 20 y que tiene 11 en total, escribiremos:
```
>>vect2=linspace(0,20,11)

vect2 =
0 2 4 6 8 10 12 14 16 18 20
```

A las coordenadas de un vector se accede sin más que escribir el nombre del vector y, entre paréntesis, su índice:
```
>>vect2(3)
ans =
4
```

y se pueden extraer subvectores, por ejemplo:
```
>>vect2(2:5)
ans=
2 4 6 8
```

o,
```
>>vect1(:)
ans=
0
2
4
6
8
10
12
14
16
18
20
```

Las matrices se escriben como los vectores, pero separando las filas mediante un punto y coma; así una matriz 3x3:

```
>>M=[1 2 3;4 5 6;7 8 9]

M =
1 2 3
4 5 6
7 8 9
>>M'        % Su traspuesta (su adjunta)
ans =
1 4 7
2 5 8
3 6 9

>>mat=[v;w;0 0 1]      % También es una matriz 3x3

mat =
1 2 3
4 5 6
0 0 1
```

A los elementos de una matriz se accede sin más que escribir el nombre de la matriz y, entre paréntesis, los respectivos índices:

```
>>mat(1,3)        % Elemento en la primera fila y tercera columna de la matriz mat

ans =
3
```

También se puede acceder a una fila o columna completas,

```
>>mat(:,2)        % Segunda columna de mat

ans =
2
5
0
>>mat(2,:)        % Su segunda fila

ans =
4 5 6
```

acceder a la matriz como si fuera una columna,

>>M(2:7) % Los elementos segundo a séptimo de la matriz como columna

ans =
4
7
2
5
8
3

o acceder a cualquiera de sus submatrices

>>mat(2:3,[1 3]) % Submatriz formada por los elementos que están en
% "todas" las filas que hay entre la segunda y la tercera y
% en las columnas primera y tercera

ans =
4 6
0 1

Existen algunas matrices definidas previamente; por ejemplo, la matriz identidad,

>>eye(5) % eye se pronuncia en inglés como I

ans =

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

la matriz nula,
>>zeros(3)

ans =
0 0 0
0 0 0
0 0 0

o la matriz cuyos elementos valen todos 1:

>>ones(4)

ans =

```
     1 1 1 1
     1 1 1 1
     1 1 1 1
     1 1 1 1
```

Se puede conocer el tamaño de una matriz y la longitud de un vector:

>>size(mat) % Dimensiones de la matriz mat (número de filas y de columnas)

ans =
3 3

>>size(v)

ans =
1 3

>>length(v) % Longitud del vector (número de coordenadas)

ans =
3

Existen comandos que permiten crear de forma sencilla matrices. Por ejemplo:

>>diag(v) % Matriz diagonal cuya diagonal es el vector v

ans =
1 0 0
0 2 0
0 0 3

>>diag(diag(M)) % Matriz diagonal con la diagonal de M. La sentencia diag(M) da el vector formado por la diagonal de la matriz M

ans =
1 0 0
0 5 0
0 0 9

>>diag(ones(1,4),1)+diag(ones(1,4),-1) % Matriz tridiagonal 5x5 con 0 en la diagonal principal y 1 en la sub y superdiagonal

ans =

0	1	0	0	0
1	0	1	0	0
0	1	0	1	0
0	0	1	0	1
0	0	0	1	0

>>tril(M) % Matriz formada por la parte triangular inferior de M.

ans =
1 0 0
4 5 0
7 8 9

>>triu(M) % Matriz formada por la parte triangular superior de M.

ans =
1 2 3
0 5 6
0 0 9

Las funciones matemáticas elementales están definidas de forma que se pueden aplicar sobre arrays. El resultado es el array formado por la aplicación de la función a cada elemento del array. Así:

>>log(v)
ans =
0 0.6931 1.0986

>>p=(0:0.1:1)*pi % Vector definido como el producto de un vector por un escalar

p =
Columns 1 through 7
0 0.3142 0.6283 0.9425 1.2566 1.5708 1.8850
Columns 8 through 11
2.1991 2.5133 2.8274 3.1416

```
>>x=sin(p)

x =
Columns 1 through 7
0 0.3090 0.5878 0.8090 0.9511 1.0000 0.9511
Columns 8 through 11
0.8090 0.5878 0.3090 0.0000
```

Las operaciones habituales entre arrays (suma, resta y producto escalar de vectores; suma, resta, producto y potencia de matrices) se representan con los operadores habituales:

```
>>v,w       % Recordamos los valores de v y w

v = 1 2 3
w = 4 5 6

>>z=v*w'  % Producto escalar (producto de matrices 1x3 por 3x1)

z =
32

>>Z=w'*v  % Producto de matrices 3x1 por 1x3 = Matriz 3x3

 Z  =
 4   8 12
 5   10 15
 6   12 18

>>v*w       % Los vectores v y w no se pueden multiplicar

??? Error using ==> *
Inner matrix dimensions must agree.

>>mat       % Recordamos el valor de la matriz mat

mat =
1 2 3
4 5 6
0 0 1

>>mat^2   % Matriz mat elevada al cuadrado
```

```
ans =
9 12 18
24 33 48
0     0       1
```

También pueden efectuarse multiplicaciones, divisiones y potencias de arrays, entendiéndolas como elemento a elemento (como, de hecho, se realizan la suma y la resta). El operador utilizado para ellas es el habitual precedido por un punto; es decir:

```
>>v.*w % Vector formado por los productos de las respectivas coordenadas:
% ans(i)=v(i)*w(i)

ans = 4 10 18

>>w./v      % Vector formado por el cociente de cada coordenada de w entre la
% coordenada correspondiente de v: ans(i)=w(i)/v(i)

ans =
4.0000 2.5000 2.0000

>>mat.^2 % Matriz cuyos     elementos son los de mat elevados
% al cuadrado: ans(i,j)=mat(i,j)^2

ans =
1     4       9
16   25   36
0     0       1
```

Finalmente, pueden calcularse determinantes:

```
>>det(mat)

ans =
-3
```

y resolverse sistemas de ecuaciones lineales con el versátil comando \:

```
>>mat\v'

ans =
2.6667
-5.3333
3.000
```

TAREA 2. BIFURCACIONES Y BUCLES.

Sentencia if-elseif-else

La sentencia if se utiliza como bifurcación simple por sí sola, es decir, en aquellas situaciones en las cuales se requiera evaluar solamente una condición, por ejemplo, suponga que tiene dos números a y b y necesita comprobar si son iguales y ejecutar una acción, para ello bastaría con una sentencia if simple:

```
if a==b
    disp('a es igual a b')
end
```

A diferencia del caso anterior hay situaciones que requieren la ejecución de una acción cuando la condición se cumpla y de otra en caso contrario, entonces puede utilizarse una bifurcación doble formada por las sentencias if-else. Retomando el ejemplo para la bifurcación if simple, podríamos modificarlo de tal manera que envíe también un mensaje (ejecute una acción) para cuando la condición no se cumple:

```
if a==b
    disp('a es igual a b')
else
    disp('a es diferente de b')
end
```

Ahora imagine que para los ejemplos anteriores se necesita especificar si a=b, si a > b o bien si a < b, lo cual implicaría tener una sentencia de selección múltiple if-elseif-else que permite escoger entre varias opciones, evaluándose en orden descendente, por ejemplo refiérase a la siguiente estructura:

```
if cond1
    % Instrucciones
elseif cond2
    % Instrucciones
elseif cond3
    % Instrucciones
        .
        .
        .
elseif condN
    % Instrucciones
else
    % Instrucciones
end
```

MATLAB evalúa primeramente la condición 1 contenida en la sentencia if (cond1) y en el caso de no cumplirse evalúa la siguiente condición de forma sucesiva (cond2, cond3, …); finalmente y en el caso de que ninguna de las opciones evaluadas se cumpla, se ejecuta la instrucción contenida en la sentencia else. A continuación se muestra el ejemplo de una bifurcación múltiple para la situación descrita al principio:

```
if a==b
    disp('a es igual que b');
elseif a>b
    disp('a es mayor que b');
elseif a<b
    disp('a es menor que b');
end
```

Sentencia switch

La sentencia switch es una bifurcación múltiple que permite seleccionar entre varias opciones o casos la acción a ejecutar. La sintaxis general es:

```
switch var
case opc1
    % Instrucciones
case opc2
    % Instrucciones
        .
```

```
        .
        .
        otherwise
    % Intrucciones
end
```

Siendo var la variable que servirá como criterio de selección. Después de la palabra reservada case, se coloca el valor de var para el cual se ejecutarán esas instrucciones, y en otherwise se insertan las instrucciones que MATLAB deberá ejecutar por defecto en caso de no cumplirse ninguno de los casos especificados.

Enseguida se muestran dos ejemplos correspondientes a la sentencia de selección switch:

```
X=input('Inserte 0 o 1: ');
switch X
case 0
    disp('Insertó cero')
case 1
    disp('Insertó uno')
otherwise
    warning('Valor incorrecto, verifique')
end

letra=input('Inserte una letra: ','s');
switch letra
case {'a','e','i','o','u'}
    disp('Es una vocal');
otherwise
    disp('Es una consonante');
end
```

Bucle for

La sintaxis general de un bucle for se muestra enseguida:

```
for i=inicio:incremento:fin
    % Instrucciones...
end
```

El valor inicio es a partir del cual se ejecutará el ciclo, el incremento es

la cantidad que varía en cada paso de ejecución, y el valor de final establece el último valor que tomará el ciclo.
El siguiente código muestra un ciclo for muy básico, el cual simplemente muestra en consola el valor actual adquirido por la variable.

```
for  i=1:10
      fprintf('Valor  actual:  %g  \n',i)
end
```

Cuando no se especifica el incremento, como el caso anterior, MATLAB asume que es unitario.
Es posible utilizar ciclos for anidados, por ejemplo para cuando se requiere recorrer una matriz en sus dos dimensiones y ejecutar operaciones elemento por elemento. Véase el siguiente ejemplo:

```
A=round(rand(5)*10);
for  i=1:5
      for  j=1:5
            disp(A(i,j));
      end
end
```

Bucle while

El bucle while se utiliza, por lo general, cuando no se tiene un rango definido sobre el cual se realice la ejecución del ciclo o bien cuando la terminación del mismo viene dada por una condición. La sintaxis más común es:

```
while cond
% Instrucciones
      % ...
      % ...
      % ...
end
```

Donde cond es la condición que determina la finalización de ejecución. Enseguida se muestra un ejemplo muy básico que muestra en pantalla el valor de una variable utilizada como referencia:

```
k=1;
while k<10
```

```
    disp(k);
    k=k+1;
end
```

Lo anterior muestra en consola el valor de k mientras esta sea menor a 10, es decir muestra todos los valores enteros en el intervalo [1, 9], es importante notar que la variable k debe incrementarse en cada ciclo para que en un momento determinado la condición de finalización se cumpla, de lo contrario se convertiría en un bucle infinito.

TAREA 3. PROBLEMAS DE ERRORES

Resolver los siguientes problemas en la ventana de comandos de Matlab:

a) Hallar el épsilon de la máquina utilizando el algoritmo dado en clases y comparar con el épsilon de la máquina definido en Matlab.
b) La aproximación de una raíz cuadrada por el método babilónico implica realizar n iteraciones mediante la siguiente expresión:

$$r_n(x) = \frac{1}{2}\left(\frac{x}{r_{n-1}} + r_{n-1}\right)$$

Donde x es el número del cual se calcula la raíz cuadrada. A continuación se muestra el código implementado en MATLAB utilizando un bucle while.

Nota.- El bucle termina cuando la diferencia entre el valor actual y el anterior es inferior a la tolerancia numérica (eps) soportada por MATLAB y por ende pasan a considerarse como valores iguales.

c) Mostrar pi y exp(1) de Matlab en todos los formatos posibles.
d) De las medidas de tiempo de un recorrido efectuadas por diferentes alumnos: 3,01 s; 3,11 s; 3,20 s; 3,15 s, hallar los errores absolutos y relativos.
e) Completa la siguiente tabla:

Número exacto	Aproximación décimas	Error absoluto	Error relativo
11/3	3,7		
5/11	0,5		
3,24	3,2		
2,888888….	2,9		
7/13	0,5		
4/3	1,3		
2,93333…	2,9		
4,66666	4,7		
13/6	2,2		
4,11111…	4,1		
15,2377945	15,2		