Práctica Nº1: MATLAB

1.- Introducción.

El programa MATLAB (el nombre corresponde a la abreviatura Matrix Laboratory) es una potente herramienta de cálculo numérico y visualización gráfica de uso muy difundido entre los científicos para el desarrollo de su tarea de investigación. Tiene la gran ventaja de ser un lenguaje de alto nivel que integra, en un único ambiente software, rutinas de cálculo, visualización y programación. El programa es de fácil uso ya que los problemas se pueden formular usando una notación matemática standard. La representación básica de los datos en MATLAB es en forma matricial.

Algunos de los usos más comunes de MATLAB son, por ejemplo:

Cálculo numérico.

Desarrollo de algoritmos.

Modelado, simulación y desarrollo de prototipos.

Análisis y visualización de datos.

Construcción de gráficas.

MATLAB es un sistema abierto al cual el usuario puede incorporar nuevas funciones para su utilización en aplicaciones particulares. Existen también librerías de funciones MATLAB, denominadas Toolboxes, que permiten resolver problemas específicos en diversas áreas de ciencia e ingeniería. Actualmente existen Toolboxes en áreas tales como Control, Procesamiento de Señales, Identificación, Procesamiento de Imágenes, Redes Neuronales, Wavelets, etc.

En esta primera práctica, esperamos que el alumno se familiarice con los comandos básicos de MATLAB de forma de poder realizar el tratamiento de los datos obtenidos en las prácticas siguientes.

2.- Comandos básicos.

Al iniciar el programa MATLAB se desplegará una ventana desde donde se ejecutan los diferentes comandos.

El programa MATLAB se inicia por defecto en el directorio c:\MATLAB\work. Antes de comenzar a trabajar es conveniente cambiarse al directorio de trabajo. Para verificar en que directorio se está trabajando se utiliza el comando: pwd.

El símbolo » (llamado prompt) indica que el programa está esperando que se le ingrese un comando o una variable.

A continuación veremos algunos comandos básicos de MATLAB. Por más información sobre este tema consultar el help de MATLAB: Starting and Quitting MATLAB. También se pueden listar desde el workspace: >> help MATLAB/general.

Help: Lista la serie de funciones o aplicaciones que abarca MATLAB. Si se desea ver los comandos relacionados con un tópico particular, se debe ejecutar el comando help <comando> o clikear en la barra de herramientas el símbolo ?.

demo Permite acceder a las demostraciones de algunas funciones del MATLAB .

whos Lista las variables en memoria y sus características.

what Lista las M-files existentes en el directorio de trabajo.

close Cierra ventanas de gráficas.

clear Borra las variables en memoria.

clc Limpia la pantalla

clf Limpia las ventanas de gráficos

^C Interrumpe el programa localmente.

3.- Tipos de variables

En MATLAB es posible definir diferentes tipos de variables y utilizando el comando whos es posible visualizar en la ventana de trabajo qué tipo de variables están almacenadas en la memoria de trabajo. A modo de ejemplo:

a) variables numéricas (array): por defecto son de doble precisión. Pueden ser reales o complejas. Ejemplos:

>> a = 25.4591

>> b = 3.5+45*i, donde i = $\sqrt{-1}$

b) variables string (carácter): >> a = 'hola'

c) variables simbólicas: se definen entre ‘, por ejemplo: sym 'x', 'y'

Por más información sobre este tema consultar el help de MATLAB: Data Types in MATLAB.

4.- Funciones básicas.

MATLAB tiene predefinidas un conjunto de funciones matemáticas básicas. En el help hay información detallada de sintáxis para las mismas: >>help MATLAB/elfun.

A continuación veremos algunos ejemplos:

a) Funciones trigronométricas. MATLAB tiene la posibilidad de trabajar con las funciones trigonométricas en radianes o grados:

```
>> a = sin(pi/3) = 0.8660
>> b = sind(60) = 0.8660
>> c = asin(1) = 1.5708
>> d = asind(1) = 90
```

b) Funciones exponenciales:

```
>> a = exp(5) = 148.4132
>> b = log(3) = 1.0986          Logaritmo natural
>> c = log10(100) = 2
```

c) Númeroscomplejos: consideremos el siguiente número complejo: y=5+2.3*i:

```
>> a = real(y) = 5
>> b = imag(y) = 2.3
>> c = abs(y) =  5.5036
>> conj(y)  = 5.0000 - 2.3000i
```

d) Aproximacionesyredondeo: x = [45.23 -23.79 0.3];

```
>>round(x) = [45 -24 0];      Redondea hacia el entero más próximo
>> fix(x) = [ 45 -23 0];      Redondea hacia cero
>> floor(x) = [45 -24 0];     Redondea hacia menos infinito
>> ceil(x) = [46  -23  1];    Redondea hacia más infinito
```

5.- Operaciones con matrices.

5.1.- Construcción de matrices.

Veamos ahora como construir vectores y matrices en MATLAB. Esto es fundamental pues los datos deben ser presentados en forma de matrices para que el programa pueda procesarlos.

En el siguiente ejemplo definimos una matriz A de 2 filas y 3 columnas, escribiendo los elementos que constituyen sus filas.

```
>>A = [3 4 5; 6 7 8];
```

El comando size da las dimensiones de la matriz (el primer número corresponde a la cantidad de filas y el segundo a la cantidad de columnas):

```
>> size(A)
```

Si por ejemplo quisiéramos identificar el elemento ubicado en la fila 2 y la columna 1 de la matriz A, ejecutamos:

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

De esta forma podemos seleccionar un elemento dado de una matriz por la fila y la columna a la cual pertenece:

```
>> a = A(2,1);
```

5.2.- Formas de definir vectores

Definamos un vector v que contenga los enteros 3, 4 y 5 en ese orden:

```
>> v = [3 4 5];
```

Otra forma de definir el vector anterior sería:

```
>> v = [3:5];
```

En general se puede definir un vector cuyos elementos se obtienen a partir del anterior más un cierto incremento. Por ejemplo, definamos un vector a de N elementos y cuyo primer elemento sea n_i, el último sea n_f y el incremento sea dn, se cumple:

$$N = (n_f - n_i)/dn + 1; \quad v = [n_i : dn : n_f];$$

Un vector también puede definirse a partir de los elementos de una fila o columna de una matriz dada. En el siguiente ejemplo los elementos del vector v corresponden a la segunda fila de la matriz A, y los elementos del vector w corresponden a la primera columna de la misma matriz:

```
>>A = [3 4 5; 6 7 8;1 2 3];
>> v = A(2,:)
>> w = A(:,1)
```

Ejecutando el comando length podemos obtener la longitud de un vector dado:

```
>> L = length(w);
```

5.3.- Operaciones con matrices y vectores.

<u>Suma yResta.</u>

En este caso las matrices deben tener las mismas dimensiones. Por ejemplo, definamos dos matrices M_1 y M_2, y luego efectuemos la suma entre ellas;

comprobaremos en el siguiente ejemplo que cada elemento de la matriz suma es la suma de los elementos correspondientes de las matrices sumandos (ídem para la resta):

```
>> M1 = [1 2 3; 4 5 6];
>> M2 = [0 1 0;-1 2 1];
>> s = M1 + M2
```

Productoentrematrices.

Sea P la matriz producto de las matrices A y B. En este caso el número de columnas de A debe ser igual al número de filas de B. Veamos el siguiente ejemplo:

```
>> A = [3 4 5;6 7 8];
>> B = [3 2;1 1;0 -3];
>> p = A*B
```

Productodedosmatriceselementoaelemento en el caso que las matrices tengan iguales dimensiones. Se obtiene entonces una matriz donde cada elemento está definido por el producto de los elementos correspondientes en las matrices dadas. Por ejemplo:

```
>> A = [ 3 4 5;6 7 8];
>> B = [1 -1 2; 0 2 4];
>> P = A.*B
```

Notar que en este caso hay que anteponer el punto (.) al símbolo del producto (*) para que realice la operación elemento a elemento.

Todo lo anterior referido al producto entre matrices es válido igualmente para la división entre matrices (siempre que el determinante de la matriz divisor sea no nulo), y también para el producto y división entre vectores (por ser el vector un caso particular de matrices).

Potencia enésimadeuna matriz (sólo posible para matrices cuadradas):

```
>> A^n
```

Potencia enésimadeuna matrizelementoaelemento (en este caso la matriz NO tiene porque ser cuadrada).

```
>> A.^n
```

Productoescalar: dot(x,y)

Productovectorial: cross(x,y)

6.- Lectura y almacenamiento de datos.

6.1.- Almacenamiento de datos.

MATLAB permite varias opciones para almacenar las variables con las cuales se trabaja para su posterior utilización. En todos los casos el comando es save.

Almacenamiento en código ASCII. La sintaxis es:

save <nombre del archivo> <variables> -ascii

Ejemplo:

```
>> a = [0:1:100];
>> save ejemplo1.dat a -ascii
```

Esta forma de almacenar presenta la gran ventaja de que este archivo de datos puede ser leído por cualquier programa de manejo de texto y/o planillas. Por ejemplo: block de notas, word, excel, etc.

La terminación .dat no es obligatoria, pero se suele utilizar para identificar rápidamente el archivo como un archivo de datos en ascii.

El mayor inconveniente que tiene este método es que todas las variables deben tener la misma dimensión para ser almacendas.

Almacenamiento en binario.

La sintaxis es:

save <nombre del archivo> <variables>

Por defecto, MATLAB coloca a estos archivos la terminación .mat. Estos archivos no pueden ser leídos desde programas de procesamiento de texto, pero tienen algunas ventajas que se verán más adelante.

En este caso no es necesario que todas las variables tengan la misma dimensión para ser almacendas.

Ejemplo

```
>> a = [0:1:100];
>> b = sin(a);
>> save ejemplo2 a b
```

Creación de un archivo desde el block de notas.

Los resultados experimentales que se obtendrán en el laboratorio serán procesados utilizando MATLAB. Una opción para manejar estos datos desde el programa es generar desde el block de notas un archivo de datos. Este será un archivo ASCII.

Para ello simplemente se debe abrir el block de notas e ingresar los datos en forma de columnas. Al almacenar tener la precaución de ponerle terminación .dat para identificarlos rápidamente como archivo de datos.

6.2.- Lectura de archivos de datos.

El comando para leer archivos de datos es load. Veremos a continuación como trabajar en los dos formatos vistos anteriormente: ASCII y BINARIO.

Archivosascii.

La sintaxis es:

load < nombre del archivo>

Notemos que si hacemos un whos, la variable que tenemos en la memoria del MATLAB tiene el mismo nombre que el archivo, pero sin la terminación. Si se desea se le puede asignar un nuevo nombre a esta variable: b = datos1;

Otra opción es:

a = load('nombre del archivo')

Archivosbinario.

La sintaxis es:

load < nombre del archivo>

De esta forma la variable contiene los datos almacenados en el archivo.

Notemos que si hacemos un whos, las variables que tenemos en la memoria del MATLAB tiene el mismo nombre con que habían sido almacenadas, lo que representa una gran ventaja si se está realizando la lectura en el marco de un programa puesto que se conoce a priori el nombre de las variables.

7.- Creación de M-files

MATLAB permite ejecutar secuencias de comandos almacenados en un archivo. Estos archivos deben tener la extensión ".m", y por eso se denominan M-files. Existen básicamente dos tipos de M-files: los denominados function-files y script-files.

La forma de editar M-files es usando un editor incorporado a MATLAB, el denominado MATLAB Editor/Debugger, al cual se accede desde la opción 'File' de la barra de menú, en la ventana de comando, cada vez que se abre un M-file (nuevo o ya existente).

Script-files

Un script-file consiste de una sucesión de líneas de comando MATLAB. Por ejemplo, si el archivo tiene el nombre prueba.m, cuando se lo corre desde la pantalla de MATLAB (>>prueba), los comandos del archivo se ejecutan en el orden en que aparecen en el listado. Las variables en el script-file son globales. Esto significa que si hay previamente en memoria alguna variable con el mismo nombre, su valor cambiará cuando se ejecute el programa.

El primer paso es abrir un editor de texto para escribir el programa: desde la ventana de comandos de MATLAB se despliega el menú FILE y se elige la opción new m-file, a continuación se desplegará la ventana del editor de MATLAB donde escribimos el programa.

Una vez terminado de escribir el script, lo salvamos (haciendo click en file y luego haciendo click en guardar como... aparecerá otra ventana donde elegimos un nombre para el un archivo del script (npar.m por ej.) y el directorio donde guardar el archivo.

Es importante recordar que los nombres de los programas en MATLAB deben tener extensión .m

Function-files

Los function-files permiten extender la biblioteca de funciones MATLAB para el uso en aplicaciones específicas. A diferencia de los script-files, las variables en un function-file son locales, por lo que variables del mismo nombre en el espacio de trabajo de MATLAB no son modificadas cuando la función es ejecutada.

La primera línea de un function-file siempre comienza con la palabra function seguida de una expresión donde se declara el nombre de la función y se indica cómo la función debe ser llamada desde el espacio de trabajo de MATLAB, y cuáles son los argumentos

de entrada y salida de la función. El nombre del function-file debe ser de la forma function-name.m.

Las líneas que comienzan con el carácter '%' corresponden a comentarios. Las líneas de comentarios que están inmediatamente debajo de la primera línea pueden utilizarse para aprovechar las facilidades de help online de MATLAB. El texto correspondiente a esas líneas aparecerá en el espacio de trabajo cuando se tipee >> help function-name y por lo tanto las mismas se utilizan generalmente para describir la operación que realiza la función. Es recomendable incluir siempre esta documentación en un function-file.

8.- Ejemplos

E.1) Defina las siguientes matrices a = [3 8.2 -1]; b = [0.5 4 -3.2;4 0.7 10];
c = [-3:0.1:25];

a) Defina una variable que indique las dimensiones de cada una de las matrices anteriores. Para las matrices a y c, defina una nueva variable que contenga el número de elementos.

b) Seleccione el segundo elemento del vector a, y el anteúltimo elemento del vector c.

c) Seleccione la segunda fila de la matriz b y su tercera columna.

d) Almacene en un archivo binario todas las variables definidas anteriormente.

e) Almacene en un archivo ascii la matriz b.

f) Borre las variables de la memoria.

g) Lea el archivo almacenado en el item d) y ejecute el comando whos.

h) Borre las variables de la memoria y lea el archivo almacenado en el item e). Ejecute el comando whos.

E.2) Defina las siguientes matrices: a = [-3 4 5;2 0 0;2 0 -1]; b = 3*I(3), siendo I(3) la matriz identidad de 3x3; c = [1 3 -2+4*i;0-2*i 1 3;9 0 1];

Calcular:

a) s = a+b, r = a-c; p = a*c; d = a^2; e = a^(1/2) g = exp(a)

b) Producto y cociente elemento a elemento de las matrices a y c

c) Transpuesta de a

d) Matriz conjugada de c

e) Traza de a

f) Determinante de a

g) Inverso de c

h) Almacene las variables generadas en un archivo binario.

9.- Gráficas

MATLAB tiene un excelente manejo de gráficos. Aquí veremos sólo algunos de los comandos básicos para la generación de gráficas en 2 dimensiones. Para aquellos estudiantes que tengan interés, recomendamos hacer un help graphics, help graph2d y help graph3d.

Lista de algunos comandos:

plot Crea el gráfico

hold Permite realizar la superposición de 2 o más gráficos en una misma pantalla. Es un comando on/off.

figure Genera una nueva pantalla de gráficos.

Con las nuevas versiones de MATLAB, todo el etiquetado de los ejes, texto, título, etc. de las gráficas se puede controlar desde la propia pantalla de gráficos.

Esto se verá con más detalle en los ejercicios.

Ejemplos:

E3) Defina un vector t (cuyo primer elemento es 0; el último es 512; y el paso es 0.1) y un vector x = 3t^2+2t-3 y un vector z = exp(t)

a) Grafique x en función de t. Coloque nombre a los ejes, cambie los símbolos, color, coloque título, grilla, etc.

b) Idem para z en función de t, pero en otra ventana de gráfica

c) Grafique x(t) y z(t) en la misma gráfica.

d) Grafique en la misma ventana, pero en distintas gráficas x(t) y z(t)

f) Investigue el comando ginput.

10.- Cálculo simbólico

MATLAB tiene una librería que permite resolver analíticamente una serie de problemas intersantes (consultar help symbolic por más información). Por ejemplo permite derivar, integrar, resolver sistemas de ecuaciones, graficar funciones, calcular límites, expansiones de Taylor, etc. Octave por otra parte aún no cuenta con esta funcionalidad de cálculo simbólico.

Veamos algunos ejemplos:

Derivación:

```
>> a = diff('x^3-3*x^2+x-2')
a = 3*x^2-6*x+1
```

En este caso a será una variable simbólica. Si se quiere transformar en numérica se debe utilizar el comando subs. Por ejemplo si se desea que x=3:

```
>> c = subs(a,'x',3)=10
```

Integración:

```
>> a = int('x^2-2*x')
a =1/3*x^3-x^2
```

Para poder utilizar esta función, la integral debe tener solución analítica.

Sistemadeecuaciones

```
[x y] = solve('3*x+2*y = 4,5*x+1*y = 3')
x = 2/7
y =11/7
```

Graficas

Para graficar analíticamente una función se utiliza el comando ezplot:

```
>> figure(1),ezplot('x^2 - 2*x + 1',[-2 2])
```

11.- Bifurcaciones y bucles

Se van a introducir aquí los primeros conceptos de programación. MATLAB posee un lenguaje de programación que –como cualquier otro lenguaje– dispone de sentencias para realizar bifurcaciones y bucles. Las bifurcaciones permiten realizar una u otra operación según se cumpla o no una determinada condición.

Los bucles permiten repetir las mismas o análogas operaciones sobre datos distintos. Se utiliza la palabra end para indicar que finaliza el bucle.

11.1.- Sentencia if

En su forma más simple, la sentencia if se escribe en la forma siguiente:

```
if condicion
      sentencias
end
```

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
            bloque2
                  elseif condicion3
                        bloque3
      else % opción por defecto para cuando no se cumplan las
      condiciones 1,2,3
      bloque4
end
```

donde la opción por defecto else puede ser omitida: si no está presente no se hace nada en caso de que no se cumpla ninguna de las condiciones que se han chequeado.

11.2.- 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 el índice de uno en uno.

```
for i = 1:n
      sentencias
end
```

En el siguiente ejemplo se presenta una estructura correspondiente a dos bucles anidados. La variable j es la que varía más rápidamente (por cada valor de i, j toma todos sus posibles valores):

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

12.- Bibliografía.

Help MATLAB

Toolbox MATLAB y Octave

http://www.mathworks.com/

https://www.gnu.org/software/octave/