Laboratorio 2: Introducción a Matlab II.

Cálculo Numérico 521230/525240

1. Programas en Matlab

Un programa informático es un conjunto de instrucciones (comandos) escritas en un deteminado lenguaje de programación. MATLAB incorpora muchos comandos para la solución de varios problemas matemáticos. En el Laboratorio 1 vimos, por ejemplo, el comando `norm` para el cálculo de normas vectoriales, o el comando `abs` para el cálculo del valor absoluto de números reales.

Algunos detalles importantes acerca de lo programas de MATLAB son:

1. Todos los programas deben ser escritos en el editor de MATLAB , el cual se puede abrir al hacer click en el botón *New Script*, ubicado en la esquina superior izquierda.

2. Todos los programas MATLAB deben ser guardados en archivos con extensión `.m`. Para ello, en la ventana del editor haga click en el ícono *Save* (tercer botón a la izquierda) y luego seleccione el directorio donde guardará su archivo.

3. Un programa debe ser ejecutado desde la ventana de comandos. Al llamar a un programa desde la ventana de comandos, MATLAB buscará el archivo con nombre igual al programa y extensión `.m` en sus directorios de búsqueda.

4. Una vez encontrado el archivo y si lo hemos llamado de la manera correcta, MATLAB ejecutará las instrucciones en él.

5. En un programa, todo lo que siga al símbolo `%` es interpretado como un comentario y los comentarios escritos en las primeras líneas constituyen una ayuda al programa y se mostrarán cuando en la ventana de comandos se escriba
   `help <nombre del programa>`.

Existen dos tipos de programas en MATLAB: ruteros y funciones. Una función permite parámetros de entrada y salida, mientras que un rutero no (más aún, el rutero es una "recopilación" ordenada de comandos que pueden ser ejecutados con una instrucción en la ventana de comandos). La primera línea del programa en una función siempre tiene que ser de la forma

`[salida] = <nombre funcion>(entrada)`.

`salida` es una lista con los nombres de las variables de salida de la función separados por comas, mientras que `entrada` es una lista con los nombres de todos los parámetros de entrada de la función, también separados por comas.

Observaciones:

1. Una diferencia importante entre funciones y ruteros es que las variables que se usen en una función, exceptuando los parámetros de salida, son locales (es decir, no se almacenan en memoria), mientras que aquellas que se usan en un rutero siguen existiendo una vez que se termine la ejecución del programa.

2. Una forma directa de crear una función desde la ventana principal de MATLAB es hacer click en `New>Function`.

3. Un programa tipo rutero puede ser ejecutado desde el editor haciendo click en el botón *Run* (que tiene una flecha verde). En la flecha negra incluida en ese botón aparecer otras opciones de ejecución más avanzadas que nosotros no veremos en este curso.

2. Matlab y sus funciones

La gran utilidad de MATLAB se basa en la gran cantidad de funciones que ya trae incorporadas para poder hacer cálculos matemáticos. En particular, hay una función cuyo propósito es entregarnos información acerca del uso de otras funciones. Esta es la función `help`. Por ejemplo, si ejecutamos

```
help abs
```

obtendremos la salida

```
abs    Absolute value.
    abs(X) is the absolute value of the elements of X. When
    X is complex, abs(X) is the complex modulus (magnitude) of
    the elements of X.
```

Esto nos indica que la función `abs()` nos permite obtener el valor absoluto de los elementos de una matriz, y en caso que sean números complejos nos entrega el módulo. Algunas de las principales funciones de MATLAB son:

Comando	**Función**
abs()	Valor absoluto
cos(), sin(),tan()	Funciones trigonométricas
sqrt()	Raíz cuadrada
log(), log10(), log2()	logaritmo en base natural, 2 y 10, resp.
ones()	matriz con todas sus entradas iguales a 1
zeros()	matriz con todas sus entradas iguales a 0
plot()	permite graficar figuras planas
norm()	permite calcular normas vectoriales

3. Ciclos y condicionales en Matlab

Los ciclos y los condicionales son herramientas fundamentales a la hora de escribir programas en MATLAB.

1. Ciclo `for`: tiene la forma

```
for var = vi:incr:vf
<instrucciones>
end
```

Las instrucciones en el cuerpo de este ciclo se ejecutan tantas veces como los valores tome la variable `var`, la que comienza en `vi` y toma los valores `vi,vi+incr,vi+incr+incr`,...,`vf`. Si `incr` es 1, entonces se puede escribir simplemente

```
for var = vi:vf
<instrucciones>
end
```

Si queremos, por ejemplo, sumar los primeros 10 números enteros positivos con ayuda de un ciclo `for`, podemos hacerlo como sigue:

```
suma = 0;          % iniciamos la suma en 0
for i = 1:10
suma = suma + i;   % sumamos cada numero desde 1 hasta 10
end
```

Si queremos, por ejemplo, guardar en un vector `p` los 10 primeros términos de la progresión geométrica (con primer término igual a 1 y razón igual a 3)

$$1, 3, 9, 27, 81, \ldots$$

podemos hacerlo con ayuda de un ciclo `for` de la siguiente manera:

```
p = zeros(10,1);   % inicializamos el vector
p(1) = 1;          % creamos primer elemento igual a 1
for j = 2:10
p(j) = 3*p(j-1);
end
```

2. Ciclo `while`: tiene la forma

```
while <condicion>
<instrucciones>
end
```

Las instrucciones del cuerpo del ciclo se ejecutan mientras `condicion` sea verdadera. El vector `p` del ejemplo anterior también se puede crear con ayuda de un ciclo `while` de la siguiente manera:

```
p = zeros(10,1);    % inicializamos el vector
p(1) = 1;           % primer elemento es 1
j = 2;              % creamos el contador
while j <= 10
p(j) = 3*p(j-1);    % calculamos el siguiente termino
j = j + 1;          % actualizamos el contador para el siguiente paso
end
```

3. Un condicional en MATLAB tiene la forma general

```
if <condicion 1>
<instrucciones 1>
[elseif <condicion 2>
```

```
<instrucciones 2>
else
<instrucciones 3>]
end
```

Aquí los comandos encerrados entre `[ ]` son opcionales. El programa ejecutará las `instrucciones 1` si es que `condicion 1` es verdadera. Cuando se agrega `elseif <condicion 2>`, el programa verificará si `condicion 2` es verdadera, y en tal caso ejecutará `instrucciones 2`. Cuando `condicion 1` y `condicion 2` no son verdaderas, entonces el programa ejecutará `instrucciones 3`. En ese sentido, `else` puede entenderse como *"...si no ocurre ninguna de las anteriores, entonces..."*.

Ejemplo 1. Queremos evaluar la función

$$f(x) = \begin{cases} \sin(x) & \text{si } x < 0 \\ 1/2 & \text{si } x = 0. \\ \cos(x) & \text{si } x > 0. \end{cases}$$

Para ello, escriba la siguiente función en el editor de MATLAB.

```
function F = mifuncion1(v)
% funcion para evaluar f(x) = sin(x) si x es negativo,
% f(x) = cos(x) si x es positivo, y f(0) = 1/2.

% Inicializamos el vector F con la misma cantidad de elementos de v
F = zeros(length(v),1);

% ciclo para evaluar las componentes de F
for i = 1:length(v)
    % preguntamos si la componente i-esima de v es positiva
    if v(i) > 0
        F(i) = cos(v(i));
    elseif v(i) < 0
        F(i) = sin(v(i));
    else
        F(i) = 1/2;
    end
end

end
```

Guarde esta función. Note que el nombre por defecto MATLAB le da es `mifuncion1.m`. Regresemos a la ventana de comandos de MATLAB para ejecutar esta función. Si queremos, por ejemplo, averiguar el valor de f en $-2, -1, 0, 1, 2$, podemos hacerlo mediante

```
mifuncion1(-2:2)     % evaluar f en [-2 -1 0 1 2]
whos                 % ni F ni v estan en memoria, son variables locales
```

Ejemplo 2. Escribamos ahora un rutero para evaluar la función f anterior en 100 puntos equidistantes del intervalo $[-\pi, \pi]$. Para ello abrimos el editor y escribimos el siguiente rutero:

```
% rutero para evaluar f en 100 puntos
% equidistantes del intervalo [-pi,pi]

x = linspace(-pi,pi,100);   % creamos los 100 valores
Fx = mifuncion1(x);         % evaluamos la funcion

% ahora graficamos
plot(x,Fx)
```

y guardémoslo con el nombre `plotmifuncion1.m`. Si escribimos

```
help plotmifuncion1
```

MATLAB mostrará los primeros comentarios escritos en `plotmifuncion1.m`. Llamemos ahora al rutero para ver la gráfica de f en el intervalo $[-\pi, \pi]$.

```
plotmifuncion1
whos                % x y Fx aparecen en memoria
```

4. Acerca de las gráficas de funciones

Ya definimos en la sección anterior la función f. Supongamos que nos interesa además graficar las funciones

$$g(x) = xe^x \qquad \text{y} \qquad h(x) = x^2 - 1.$$

Esto podemos hacerlo en el editor de MATLAB como sigue:

```
function F = funciong(x)
% funcion para evaluar g(x) = x*exp(x).

% evaluamos la funci\'on componente a componente con el operador .*
F = x.*exp(x);

end
```

```
function F = funcionh(x)
% funcion para evaluar g(x) = x^2-1.

% evaluamos la funci\'on componente a componente con el operador .^
F = x.^2-1;

end
```

Después de guardar estas funciones (cada una con el nombre correcto), podemos graficarlas en el intervalo $[0, 5]$, por ejemplo, en una misma figura, de la siguiente manera:

```
x = linspace(0,5,100);
g = funciong(x);
h = funcionh(x);
```

```
plot(x,g)
hold on
plot(x,h)
```

Observación: también podemos definir funciones directamente como *function_handle*. Por ejemplo, las mismas funciones g y h anteriores pueden ser definidas con las instrucciones:

```
g = @(x) x.*exp(x);
h = @(x) x.^2 - 1;
```

La instrucción `@(x)` indica que `x` es asumido como variable de la función, y las operaciones `.*` y `.^` indican que el producto y la potencia, respectivamente, deben ser hechas en cada componente de la entrada `x`. Si luego hacemos en la ventana de comandos

```
x = linspace(0,5,100);
g = @(x) x.*exp(x);
h = @(x) x.^2 - 1;
plot(x,g)
hold on
plot(x,h)
```

la gráfica producida es la misma que en el recuadro anterior.

MATLAB permite incorporar algunas otras opciones para graficar curvas y/o puntos, las cuales tienen que ver con color, estilo de línea y estilo de marcador, como se indica en la siguiente tabla:

Color		Estilo de línea		Estilo de marcador	
y	amarillo			+	cruz
m	magenta			o	círculo
r	rojo	−	sólida	*	asterisco
g	verde	−−	entrecortada	x	equis
b	azul	:	punteada	.	punto
w	blanco	−.	raya-punto	^	triángulo
k	negro			s	cuadrado
				d	diamante

las cuales pueden ser usadas, por ejemplo, como sigue:

```
plot(x,y,'r*')   % dibuja con linea roja continua y con marcadores asterisco
plot(x,y,'b--')  % dibuja con linea azul entrecortada
plot(x,y,'+')    % dibuja puntos no conectados con un cruces
```

5. Ejercicios

1. Considere nuevamente la progesión geométrica

$$1, 3, 9, 27, 81, \ldots$$

y escriba una función que reciba como entrada un número natural n y devuelva el $n-$ésimo término de dicha progresión.

2. Considere la *Sucesión de Fibonacci*:

$$a_1 = 1, a_2 = 1, \qquad a_n = a_{n-1} + a_{n-2}, \quad n = 2, 3, \ldots.$$

Escriba una función que reciba como entrada un número natural n y entregue como salida el $n-$ésimo término de esta sucesión.

3. Escriba una función que reciba como entrada un número natural n y devuelva como salida la matriz de n filas y n columnas dada por

$$M_n = \begin{pmatrix} 1 & 2 & 3 & \cdots & n \\ 2 & 3 & 4 & \cdots & n+1 \\ 3 & 4 & 5 & \cdots & n+2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ n & n+1 & n+2 & \cdots & n+(n-1) \end{pmatrix}.$$

4. Considere la matriz de n filas y n columnas, y el vector columna de n componentes

$$A_n = \begin{pmatrix} n & 1 & 0 & \cdots & 0 \\ 1 & n-1 & 2 & \cdots & 0 \\ 0 & 2 & n-2 & \cdots & \vdots \\ \vdots & \vdots & \ddots & \ddots & n-1 \\ 0 & \cdots & 0 & n-1 & 1 \end{pmatrix}, \qquad b_n = \begin{pmatrix} 1/n \\ 2/(n-1) \\ 3/(n-2) \\ \vdots \\ n/1 \end{pmatrix}.$$

Escriba una función en MATLAB que reciba como entrada un número natural n y realice lo siguiente:

a) Construya la matriz A_n y el vector b_n.

b) Resuelva el sistema de ecuaciones $A_n x = b_n$ usando el comando \.

c) Calcule $\|A_n x - b_n\|_2$.