Práctica 3
Introducción a Matlab III

Objetivo. El objetivo de esta práctica es dar una introducción al uso de Matlab como un lenguaje de programación. Incluyendo programación mediante scripts y funciones.

Introducción.

En las dos prácticas anteriores hemos usado a Matlab como una calculadora que simplemente ejecuta los comandos conforme se van introduciendo manualmente en ventana de comandos. A partir de esta práctica veremos que Matlab también puede utilizarse como un poderoso lenguaje de programación.

Un programa en Matlab es simplemente un archivo que contiene comandos de Matlab que se ejecutarán en una cierta secuencia. Los archivos de comandos de Matlab se denominan archivos-M (ya que tienen extensión .m). Hay dos tipos de programas en Matlab:

- **Scripts**: Son archivos que contienen comandos de Matlab y que operan sobre las variables del espacio de trabajo.
- **Funciones**: Son archivos que contienen comandos de Matlab, pero que operan sobre variables de entrada para producir variables de salida.

Los archivos-M se pueden generar y editar con cualquier archivo de texto, sin embargo, conviene editarlos con el editor integrado en el ambiente de desarrollo de Matlab, ya que éste proporciona ayudas visuales, tales como resaltar con colores distintos los diferentes tipos de texto introducido (comandos, comentarios, variables, etc.)

Control de flujo del programa.

Matlab posee instrucciones orientadas a controlar la secuencia (o flujo) de ejecución de las instrucciones de un programa. Para realizar el control de flujo del programa Matlab tiene que evaluar expresiones lógicas para decidir entre diferentes caminos de ejecución de las instrucciones de un programa.

Expresiones lógicas.

Matlab permite construir expresiones de valor lógico (falso o verdadero) mediante diversas operaciones lógicas y relacionales (de comparación) que producen este tipo de resultados. Estas operaciones se agrupan en la siguiente tabla, en donde también se incluyen operadores aritméticos tratados en las prácticas anteriores para aclarar su precedencia.

tipo	Operador		Descripción	Precedencia
Arimético	( )		Agrupación por paréntesis	1
	.'	'	Transpuesto y conjugado traspuesto	2
	. ^	^	Potencia y potencia matricial	
	+	−	Signo positivo y signo negativo	3
Lógico	~		Negación lógica	
Aritmético	.*	*	Multiplicación y multiplicación matricial	4
	./ ó .\	/ ó \	División elemento a elemento y matricial	
	+	−	Suma y resta	5
Especial	:		Operador dos puntos	6
Relacional	<	<=	Menor que y menor o igual que	7
	>	>=	Mayor que y mayor o igual que	
	==	~=	Igual a y diferente a	
Lógico	&		Element-wise AND	8
	|		Element-wise OR	9
	&&		Short-circuit AND	10
	||		Short-circuit OR	11

- ☞ La **precedencia** define cual operación se realiza primero en una expresión que contiene variaos de estos operadores. Nuevamente, una expresión siempre se evalúa de izquierda a derecha cuando las operaciones involucradas tienen la misma precedencia.
- ☞ Un Valor aritmético es convertido a valor lógico cuando aparece como operador de una operación lógica, de acuerdo a la siguiente equivalencia:

  Valor aritmético cero = valor lógico 0

  Valor aritmético diferente de cero= valor lógico 1

Ejemplo: La expresión relacional ($1 \le x \le 3$) ó ($5 \le x \le 7$) en Matlab se escribe como sigue: x>=1 & x<=3 | x>=5 & x<=7 . Obsérvese que no es necesario el uso de paréntesis debido a que el operador & tiene mayor precedencia que el operador |. A continuación se muestra la evaluación de esta expresión para dos valores de x:

```
>> x=4; x>=1 & x<=3 | x>=5 & x<=7
ans =
     0
>> x=6; x>=1 & x<=3 | x>=5 & x<=7
ans =
     1
```

Ejemplo: En cambio, la expresión relacional ($x \le 4$) y ($x \le 1$ ó $x \ge 3$) en Matlab se escribe como sigue: x<=4 & (x<=1|x>=3) y en este caso los paréntesis si son necesarios, de lo contrario se estaría evaluando la expresión ($x \le 4$ y $x \le 1$) ó ($x \ge 3$). A continuación se muestra la evaluación de estas dos expresiones (con y sin paréntesis) para diferentes valores de x:

```
>> x=0; x<=4 & (x<=1|x>=3)
ans =
     1
>> x=0; x<=4 & x<=1|x>=3
ans =
     1
>> x=5; x<=4 & (x<=1|x>=3)
ans =
     0
>> x=5; x<=4 & x<=1|x>=3
ans =
     1
```

Las instrucciones de control de flujo forman bloques de instrucciones llamadas estructuras, las diferentes estructuras que se pueden crear en Matlab se resumen en la siguiente tabla.

Estructura de control de flujo	Descripción

`for x = arreglo`
`  comandos`
`end`
	Un ciclo o bucle `for` ejecuta repetidamente los comandos, asignando a la variable x una columna del arreglo en cada iteración, y repite desde la primera hasta la última columna del arreglo.

`while expresión`
`  comandos`
`end`
	Un ciclo o bucle `while` ejecuta los comandos mientras la expresión (lógica) tiene valor verdadero o distinto de cero.

`if expresión`
`  comandos ejecutados si`
`  expresión es verdadera`
`else`
`  comandos ejecutados si`
`  expresión es falsa`
`end`
	Una simple estructura `if-else-end` permite seleccionar entre dos bloques de comandos a ejecutar dependiendo del resultado de una expresión lógica.

`if expresión1`
`  comandos ejecutados si la`
`  expresión1 es verdadera`
`elseif expresión2`
`  comandos ejecutados si la`
`  expresión2 es verdadera`
`elseif expresión3`
`...`
`else`
`  comandos ejecutados si todas`
`  las expresiones1,2,3,...`
`  son falsas`
`end`
	La estructura más general `if-elseif-end`. permite seleccionar entre la ejeccución de diferentes bloques de comandos dependiendo de los resultados de diferentes expresiones lógicas.

`switch switch_expr`
`  case case_expr,`
`  statement,...,statement`
`case{case_expr1,case_expr2,...}`
`  statement, ..., statement`
`  ...`
`otherwise,`
`  statement,...,statement`
`end`
	La estructura `switch-case-otherwise-end`. permite seleccionar entre diferentes bloques de comandos dependiendo de los diferentes posibles resultados de una sola expresión (no necesariamente lógica).

`break`	Termina la ejecución de bucles *for* y *while*.

Programación mediante Scripts.

Los scripts y las funciones son archivos-M y deben tener en cuenta los siguientes puntos:

- Los archivos-M se crean en un editor de texto o en el editor de Matlab.
- El nombre de la función y el nombre del archivo deben de ser idénticos. Por ejemplo la función `linspace` se almacena en un archivo denominado `linspace.m`.
- Las primeras líneas de comentarios hasta la primera línea de comentario en un archivo-M de función son el texto de ayuda que se devuelve cuando se solicita ésta con el comando `help`.
- La primera línea de ayuda, conocida como la línea H1, es la línea que examina la orden `lookfor`.

Como ya se mencionó, los scripts se ejecutan para procesar variables del espacio de trabajo, por lo tanto no tienen propiamente variables de entrada o de salida. En los siguientes ejemplos se ilustra como un script puede trabajar sobre una variable del espacio de trabajo, modificarla y producir un resultado.

Ejemplo: El siguiente script permite invertir el orden de las columnas del array x.

```
%Este script se guarda en el archivo invierte.m
[m,n]=size(x);    %Obtiene n=número de columnas de x
y=zeros(m,n);     %Inicializa vector y del mismo tamaño que x
for i=1:n
     y(:,i)=x(:,n+1-i); %copia elementos de x en y en orden inverso
end
x=y    %reemplaza x por y
```

En la línea de comandos:

```
>> x=[1 2 3 4 5 6];
>> invierte
x =
     6     5     4     3     2     1
```

☞ Como se advirtió desde la introducción en la práctica No. 1, Los ciclos deben evitarse siempre que haya una función intrínseca de Matlab equivalente que permita resolver el problema, ya que al ser Matlab un lenguaje interpretado el tiempo invertido en ejecutar un ciclo crecerá con el número de iteraciones que contiene.

Ejemplo: El ciclo del ejemplo anterior se puede reemplazar por una sola instrucción:

```
>> x=[1 2 3 4 5 6];
>> x(:,1:6)=x(:,6:-1:1) %Una sola instrucción
x =
     6     5     4     3     2     1
```

Para comparar la eficiencia de los dos métodos usamos los comandos `tic` y `toc` que permiten arrancar y parar la medición de tiempo de ejecución. Para notar mejor la diferencia usamos un vector de 10,000 elementos:

```
>> tic; x=[1:10000];invierte; toc  %%% Usando el ciclo for
Elapsed time is 0.001492 seconds.
>> tic; x=[1:10000];x(:,1:10000)=x(:,10000:-1:1); toc  %%% sin ciclo for
```

```
Elapsed time is 0.000109 seconds.
```

Ejemplo: El siguiente script localiza las coordenadas del punto máximo de la función $f(x)=(x-1)(x-2)(x-3)$ en el intervalo $1 \le x \le 3$:

```
%Este script se guarda en el archivo maxim.m
x=1:0.0001:3;     %intervalo de búsqueda
n=length(x);
ymax=-inf;  %Inicializa valor máximo
for i=1:n
    y=(x(i)-1)*(x(i)-2)*(x(i)-3);   %evalúa la función en el valor x(i)
    if y>ymax
      ymax=y;     %Si encuentra un valor mayor, reemplaza ymax
    end
end
ymax  %Despliega resultado
```

En la línea de comandos:

```
>> maxim
ymax =
    0.3849
```

Ejemplo: El ciclo anterior se puede evitar de la siguiente manera:

```
>> x=1:0.0001:3;     %intervalo de búsqueda
>> y=(x-1).*(x-2).*(x-3);  %Obtiene un vector con los valores de la
función
>> ymax=max(y)
ymax =
    0.3849
```

Programación mediante Funciones.

Las funciones también se guardan en archivos-M y por lo tanto deben respetar las mismas reglas que dichos archivos, sin embargo, a diferencia de los scripts, las funciones poseen las siguientes características:

- Todas las variables dentro de una función son locales a la función y por lo tanto se aíslan del espacio de trabajo de Matlab. Las únicas conexiones entre las variables locales de la función y el espacio de trabajo de Matlab son las variables de entrada y salida. Si una función cambia el valor de cualquier variable de entrada, los cambios aparecen sólo dentro de la función y no afecta a las variables en el espacio de trabajo de Matlab.
- Cuando una función tiene más de una variable de salida, éstas se encierran entre corchetes, por ejemplo `[n,m] = size(A)`.
- El número de variables de entrada pasadas a una función está disponibles dentro de la función en la variable `nargin`. El número de variables de salida solicitadas cuando una función se llama está disponible dentro de la función en la variable `nargout`.
- El comando `echo on` provoca que se visualice la ejecución del archivo-M en el espacio de trabajo de Matlab. El comando `echo off` deshabilita la visualización.
- La función `input` obtiene la entrada del usuario.
- El comando pause causa que se detenga la ejecución y espera un tiempo o espera a que el usuario presione una tecla para continuar con la ejecución

Como un primer ejemplo de función abriremos el archivo `linspace.m` que corresponde a la función intrínseca de Matlab llamada `linspace`, para abrir el archivo podemos usar el comando `type`: (los comentarios en español se han agregado al código original)

```
>> type linspace

function y = linspace(d1, d2, n)
%LINSPACE Linearly spaced vector.
%   LINSPACE(x1, x2) generates a row vector of 100 linearly
%   equally spaced points between x1 and x2.
%
%   LINSPACE(x1, x2, N) generates N points between x1 and x2.
%
%   See also LOGSPACE, :.
%   Copyright (c) 1984-98 by The MathWorks, Inc.
%   $Revision: 5.6 $  $Date: 1997/11/21 23:29:09 $

if nargin == 2     %si no se da ningún valor para n
  n = 100;         %se suministra un valor por default de 100
end
if n~=1            %Si n no es 1 calcula los n valores pedidos
  y = d1:(d2-d1)/(n-1):d2;
else
  y = d2;          %Si n=1 solo hay un valor
end
```

En la línea de comandos:

```
>> x=linspace(0,pi,5)
x =
         0    0.7854    1.5708    2.3562    3.1416
```

Ventajas y desventajas de funciones Vs. scripts.

Los scripts aún se consideran ventajosos para definir arreglos de una gran cantidad de valores, especialmente cuando estos datos son resultados experimentales, se pueden capturar de uno en uno y guardarlos en un archivo-M como valores de una arreglo matricial en un script.

Sin embargo, normalmente se utilizan funciones y no scripts para hacer programas en Matlab, ya que al usar variables locales las funciones protegen las variables del espacio de trabajo y evitan alterar variables en forma accidental. En adelante se preferirá programar solamente mediante funciones.

Ejemplo: el ejemplo del script que invierte las columnas de un arreglo matricial se puede escribir como una función de la siguiente manera:

```
function A=finv(B)
% FINV
%  A=finv(B) produce una matriz A cuyas columnas son las mismas
%  que la matriz B, pero en orden invertido

[m,n]=size(B);        %Obtiene n=número de columnas de x
A(:,1:n)=B(:,n:-1:1); %Invierte el orden
```

En la línea de comandos:

```
>> help finv
  FINV
   A=finv(B) produce una matriz A cuyas columnas son las mismas
   que la matriz B, pero en orden invertido
>> x=[1 2 3 4 5 6];
>> y=finv(x)
y =
     6     5     4     3     2     1
```

Ejemplo: La siguiente función realiza la gráfica de un tren de pulsos rectangulares de amplitud 1 y periodo 1, es decir, la función periódica cuyo definición en el periodo fundamental $0 \le x \le 1$ es: $f(x) = \begin{cases} 1 & si\ 0 \le x < 0.5 \\ 0 & si\ 0.5 \le x < 1 \end{cases}$ y calcula el error de su aproximación por ciclo de su serie de Fourier $f_s(x) = 0.5 + \frac{2}{\pi}\left[\sin(\omega x) + \frac{1}{3}\sin(3\omega x) + ... + \frac{1}{N}\sin(N\omega x)\right]$ donde $\omega = 2\pi$ es la frecuencia fundamental de $f(x)$ y donde N es el máximo número de armónico deseado y debe ser impar.

```
function error=fourier_aprox(n)
% fourier_aprox
%  error=fourier_aprox(N) calcula el error de la aproximación por serie
%  Fourier usando N armónicos (N impar) y n ciclos.
%  del tren de pulsos periódico de amplitud 1 y periodo 1
%  que vale 1 el primer medio periodo y 0 el segundo medio periodo
x=0:1e-5:n;    %Graficará n ciclos de f(x)
w=2*pi;  %frecuencia fundamental
f=sin(w*x)>0; %Genera el tren de pulsos
clf; %limpia figura actual
plot(x,f,'k-','linewidth',3); grid on; %Grafica el tren de pulsos
axis([0 n -0.2 1.2]); hold on
N=0; %Inicializa N
while (N/2)==round(N/2)
   N=input('Cuantos armónicos? (número impar) N=');
end

%% A continuación evalúa la Serie de Fourier
fs=0.5;  %Inicializa sumatoria
for k=1:2:N
    fs=fs+(2/pi/k)*sin(k*w*x);
end
plot(x,fs,'r-','linewidth',2);
legend('exacto','aproximado');
title('Aproximación por Serie de Fourier','fontSize',12);
e=f-fs;  %Vector de diferencias entre f y fs
error=sqrt(e*e'); %raiz cuadrada de suma de cuadrados de elementos de e.
error= error/n;
disp('el error cuadrático medio por ciclo es');
```

Al ejecutar desde la línea de comandos como sigue, se obtiene la gráfica de la figura 3.1.

```
>> fourier_aprox(2)
Cuantos armónicos? (número impar) N=5
el error cuadrático medio por cada ciclo es
ans =
   28.9287
```

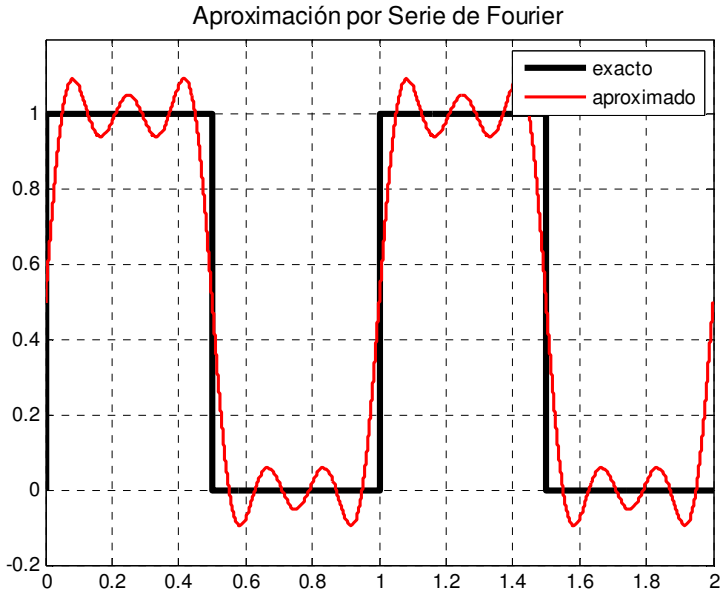

Figura 3.1. Aproximación del tren de pulsos obtenida al ejecutar `fourier_aprox(2)` con N=5.

Ejercicios:

1. Usando el hecho de que en Matlab una desigualdad como $2 \le x \le 3$ tiene un valor lógico, genera sin usar ciclos la gráfica de la figura 3.2

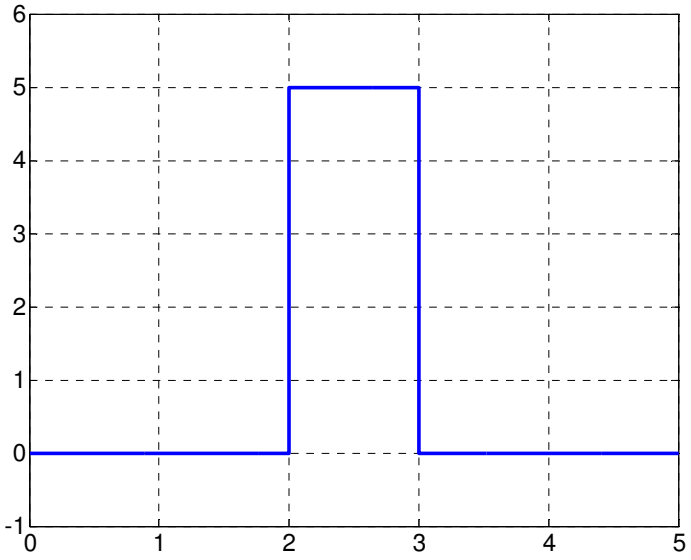

Figura 3.2

2. Modificar el código de la función `fourier_aprox(n)` para que sólo grafique un ciclo del tren de pulsos y de la aproximación por serie de Fourier incrementando el número de armónicos N desde 1 hasta infinito (solo números impares). Esperando en cada valor de N a que el usuario presione una tecla para continuar.

Desarrollo de la Práctica.

1. Probar todos los ejemplos propuestos por el profesor conforme los va explicando.
2. Realizar todos los ejercicios propuestos.
3. Contestar el cuestionario de evaluación de la práctica.

Reportar:

1. Investigar u obtener el desarrollo en serie de Fourier para la función triangular de periodo 1, dada en el periodo fundamental $0 \le x \le 1$ por: $f(x) = \begin{cases} 1-2x & si\ 0 \le x < 0.5 \\ 2x-1 & si\ 0.5 \le x < 1 \end{cases}$ y modificar la función `fourier_aprox(N)` para que grafique un ciclo de esta función y su aproximación en serie de Fourier con N armónicos.
2. Escribir la función `[m,b]=interpola(x1,y1,x2,y2)` para calcular la pendiente y la ordenada al origen de la recta que une los puntos de coordenadas (x1,y1) y (x2,y2) y grafique los dos puntos y la recta.