

# Introducción a las Ecuaciones Diferenciales y a la Programación en $M_{\text{ATLAB}}$

Álvaro G. Parra  
*Pontificia Universidad Católica de Chile.*  
*Instituto de Economía*

# Índice

<b>1. Consideraciones Iniciales</b>	<b>3</b>
1.1. Abriendo y Conociendo el Programa . . . . .	3
1.2. Operaciones Básicas . . . . .	4
1.2.1. Las Operaciones Elementales . . . . .	4
1.2.2. Definición de Variables . . . . .	4
1.2.3. Operaciones con Matrices . . . . .	5
1.2.4. Matrices Especiales . . . . .	11
1.3. Funciones Matemáticas . . . . .	13
<b>2. Introducción a los Gráficos</b>	<b>15</b>
2.1. Gráficos de Vectores . . . . .	15
2.2. Gráficos de Funciones . . . . .	17
<b>3. Mapas de Gradientes y Curvas de Solución.</b>	<b>20</b>
3.1. El Default Directory . . . . .	20
3.2. Los M-files . . . . .	21
3.3. Programando un M-file . . . . .	21
3.4. Creando un Mapa de Gradientes . . . . .	23
3.5. Curvas de Solución . . . . .	24
3.6. Curvas de Solución para un Sistema de Ecuaciones Diferenciales . . . . .	26
3.7. Ejercicios . . . . .	27
<b>4. Elementos de Programación</b>	<b>28</b>
4.1. Manipulación Matricial . . . . .	28
4.2. Operadores Lógicos y Relacionales . . . . .	31
4.3. Iterando: Herramientas de Control de Flujo . . . . .	33
4.4. Aplicación 1: Ecuaciones de diferencias . . . . .	34
4.5. Aplicación 2: El Método de Euler . . . . .	35
4.6. Aplicación 3: Programación Dinámica . . . . .	37
<b>5. Otras Herramientas y Aplicaciones</b>	<b>41</b>
5.1. Análisis de Datos . . . . .	41
5.2. Polinomios . . . . .	41
5.3. Formato de Números . . . . .	43
5.4. El Uso de Ayuda . . . . .	44

## Introducción<sup>1</sup>

MATLAB es un ambiente computacional interactivo para el estudio de matrices, el cual puede ser usado para análisis numérico (como lo son las simulaciones de modelos teóricos), cálculos de matrices (como se hace en econometría), hacer gráficos, entre otras variadas aplicaciones. Su nombre viene del inglés MATrix LABoratory donde claramente se hace énfasis en la herramienta central de MATLAB, las matrices. A lo largo de este apunte vamos a aprender varias cosas de gran utilidad, pero hay que hacer notar que nuestra principal meta es la de adquirir las herramientas necesarias para:

- (a) resolver numéricamente ecuaciones y sistemas de ecuaciones diferenciales.
- (b) graficar las soluciones.
- (c) introducir elementos de programación.

También hay que tener en cuenta que éste es un apunte que está pensado para guiar el estudio de MATLAB de manera individual y como tal, requiere mucho trabajo, dedicación y por sobre todo paciencia.

## 1. Consideraciones Iniciales

### 1.1. Abriendo y Conociendo el Programa

Primero que todo, se debe ir al *Menú de Inicio*<sup>2</sup> posteriormente a *Todos los Programas* y por último a *MATLAB 7.5*. Una vez que abra el programa se encontrará con tres pantallas: el *Workspace*, el *Command History* y el *Command Window*. El primero (*Workspace*) es donde se guardan los resultados de las operaciones realizadas en MATLAB. El *Command History* es donde se guardan todos los comandos que se han ejecutado. El más importante de los tres, el *Command Window*, es donde uno se comunica con el programa y pone las operaciones que quiere ejecutar. La mejor forma de aprender MATLAB es a través de ejemplos y ejercicios, así que este manual está construido principalmente en base a éstos.

---

<sup>1</sup>El autor agradece a Gonzalo Edwards por confiarle el desarrollo de este trabajo y a Rodrigo Cerda por facilitar parte del material de su curso Teoría Macroeconómica II. Los errores en este manuscrito son únicamente del autor, el cual puede ser contactado a [agparra@northwestern.edu](mailto:agparra@northwestern.edu).

<sup>2</sup>Como probablemente notó, es un supuesto de este apunte el que usted este usando alguna versión del sistema operativo Windows y que tiene instalado MATLAB.

## 1.2. Operaciones Básicas

### 1.2.1. Las Operaciones Elementales

Para comenzar sume  $2+3$ , para esto escriba  $2+3$  en el *Command Window* y presione la tecla *enter*:

```
>> 2+3
```

```
ans =  
      5
```

Si agrega un *punto y coma* (;) al final de la operación el programa ocultará el resultado (lo que es muy útil cuando se realizan muchos cálculos):

```
>> 2+3;
```

Ejecutar otras operaciones elementales se hace en forma similar. Separando con una *coma* (,) se pueden ingresar varias operaciones al mismo tiempo, por ejemplo:

```
>> 2-3, 2*3, 2^3
```

```
ans =  
     -1  
ans =  
      6  
ans =  
      8
```

### 1.2.2. Definición de Variables

En MATLAB se puede definir variables para después volver a usarlas o ejecutarlas. Considere el siguiente ejemplo:

```
>> manzanas=4      %Numero de Manzanas
```

```
manzanas =  
      4
```

Observe que con el signo de *tanto por ciento* (%) se pueden agregar comentarios (los cuales aparecerán en verde) a las operaciones. Defina las siguientes variables:

```
>> peras=8; kiwis=2;
>> frutas=manzanas+peras+kiwis
```

```
frutas =
      14
```

```
>> costototal=10*manzanas+peras*12+kiwis*18
```

```
costototal =
      172
```

```
>> costo_medio=costototal/frutas
```

```
costo_medio =
      12.2857
```

El comando “who” permite ver qué variables se han definido

```
>> who
```

Your variables are:

```
costo_medio  costototal  frutas  kiwis  manzanas  peras
```

Si lo que se quiere es guardar las variables, se debe ir al Workspace y presionar el ícono de guardar (*save Workspace as.*) Por otro lado, si se requiere limpiar el Workspace se debe escribir el comando “clear”.

### 1.2.3. Operaciones con Matrices

Como se mencionó anteriormente, la herramienta fundamental de MATLAB son las matrices, la forma en que éstas se introducen es la siguiente:

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

lo que produce el siguiente resultado:

```
A =
```

```
1 2 3
4 5 6
7 8 0
```

De ahora en adelante cada vez que uno introduzca **A** en el Command Window **MATLAB** nos reportará la matriz anterior. En general, es muy útil introducir vectores cuyos elementos tengan cierto orden, por ejemplo:

```
>> z=1:4
```

```
z =
    1    2    3    4
```

crea un vector con números del 1 al 4 creciendo en uno a la vez. Para crear vectores en que los números aumenten en una frecuencia distinta a la de uno en uno, por ejemplo en 0.25, escriba:

```
>> w=0:0.25:1 %Valor inicial : Frecuencia : Valor Final
```

```
w =
    0    0.25    0.5    0.75    1
```

El comando también se puede aplicar para secuencias decrecientes:

```
>> p=2:-0.25:1
```

```
p =
    2    1.75    1.5    1.25    1
```

Con las matrices ya definidas se pueden componer nuevas:

```
>> k=[z w]
```

```
k =
    1    2    3    4    0    0.25    0.5    0.75    1
```

Para realizar algunas operaciones con matrices introduzca el siguiente vector

```
>> b = [1 ; 2; 3]
```

```
b=
```

```
1
2
3
```

Éste es un buen momento para mencionar que `MATLAB` es *case sensitive*, es decir, no es lo mismo ingresar en el Command Window “`A`” que “`a`” (haga el intento). Por otro lado, ocupando las flechas del teclado, se pueden reutilizar y modificar los comandos ingresados con anterioridad.

Una vez definida la matriz `A` y el vector `b` se ejecutará algunas operaciones elementales. Comience encontrando la transpuesta de `A` usando el símbolo (`'`). Defina `B=A'`

```
>> B = A'
```

```
B =
```

```
1 4 7
2 5 8
3 6 0
```

La inversa de una matriz se encuentra con el comando “`inv(.)`”:

```
>> C = inv(A)
```

```
C =
```

```
-1.7778  0.8889 -0.1111
 1.5556 -0.7778  0.2222
-0.1111  0.2222 -0.1111
```

Compruebe que `C` sea realmente la inversa; es decir, verifique que  $CA = AC = I$

```
>> C*A, A*C
```

```
ans =
```

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

```
ans =
```

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

Se hace notar que  $AA^{-1}$  y  $A^{-1}A$  se pueden escribir como “A/A” y “A\A” respectivamente:

```
>> A/A, A\A
```

```
ans =
    1 0 0
    0 1 0
    0 0 1
```

```
ans =
    1 0 0
    0 1 0
    0 0 1
```

Por otro lado, el comando “det(·)” entrega el determinante de una matriz cuadrada:

```
>> det(A)
```

```
ans =
    27
```

Para sumar las matrices A y B simplemente escriba:

```
>> A + B
```

```
ans =
    2  6 10
    6 10 14
   10 14  0
```

Otras operaciones con matrices son:

```
>> 5*A, A*B, A*b
```

```
ans =  
    5 10 15  
   20 25 30  
   35 40  0
```

```
ans =  
   14 32 23  
   32 77 68  
   23 68 113
```

```
ans =  
   14  
   32  
   23
```

Suponga que tiene un sistema de ecuaciones de la forma  $Ax = b$ . Si  $A$  es una matriz no singular, entonces el sistema tiene como solución única  $x = A^{-1}b$ :

```
>> x = A\b
```

```
x =  
  -0.3333  
   0.6667  
    0
```

Verifique la solución multiplicando:

```
>> A*x
```

```
ans =  
    1  
    2  
    0
```

Para encontrar el rango de una matriz use el comando “`rank(·)`”:

```
>> rank(A), rank(b)
```

```
ans =  
    3
```

```
ans =
```

```
1
```

Otro comando útil es “`eig(·)`” el cual entrega los valores y vectores propios de una matriz:

```
>> [M,D]=eig(A)
```

```
M =
```

```
0.7471 -0.2998 -0.2763
-0.6582 -0.7075 -0.3884
0.0931 -0.6400 0.8791
```

```
D =
```

```
-0.3884    0    0
0    12.1229    0
0    0    -5.7345
```

La matriz  $M$  es la matriz modal de  $A$ , pues cada columna de  $M$  es un vector propio de  $A$ . Por su parte, la matriz diagonal  $D$  contiene los valores propios asociados a cada vector de  $M$ . Luego, se tiene que cumplir que  $AM = MD$  (verifíquelo).

Se hace notar que los vectores propios han sido normalizados de tal forma de que correspondan a vectores de norma unitaria, es decir, la distancia desde el origen al punto que representa el vector en el plano cartesiano es uno.

Algo que hay que tener en consideración es que como `MATLAB` “piensa” en matrices no es lo mismo  $f(x) = x^2$  que  $x^2$ . En el primer caso se quiere que **cada elemento de la matriz  $x$  se eleve al cuadrado**, mientras que en el segundo se quiere **multiplicar la matriz  $x$  por si misma**. Para distinguir el primer caso del segundo, se agrega un punto delante de la operación. Para notar la diferencia considere el siguiente ejemplo:

```
>> A.^2, A.*A, A^2, A*A
```

```
ans =
```

```
1 4 9
16 25 36
49 64 0
```

```
ans =
```

```
1 4 9
16 25 36
```

```
49 64 0
```

```
ans =  
30 36 15  
66 81 42  
39 54 69
```

```
ans =  
30 36 15  
66 81 42  
39 54 69
```

#### 1.2.4. Matrices Especiales

A continuación se presenta una tabla con matrices especiales y su función:

<b>Matriz<sup>3</sup></b>	<b>Función</b>
<code>zeros(n,m)</code>	Crea una matriz de ceros de orden $n \times m$
<code>ones(n,m)</code>	Crea una matriz de unos de orden $n \times m$
<code>rand(n,m)</code>	Crea una matriz $n \times m$ con elementos aleatorios en el intervalo $[0,1]$
<code>randn(n,m)</code>	Lo mismo pero los elementos distribuyen normal estándar
<code>eye(n,m)</code>	Crea una matriz identidad de orden $n \times m$

```
>> zeros(2), ones(2,3), rand(3,2), randn(3), eye(3,2)
```

```
ans =  
0 0  
0 0
```

```
ans =  
1 1 1  
1 1 1
```

```
ans =  
0.9501 0.4565  
0.2311 0.0185  
0.6068 0.8214
```

---

<sup>3</sup>En todas las matrices generadas por MATLAB se tiene que especificar el número de filas ( $n$ ) y el de columnas ( $m$ ). Si sólo se ingresa un número, el programa asume que la matriz es cuadrada.

```
ans =  
    -0.4326 -1.1465  0.3273  
    -1.6656  1.1909  0.1746  
     0.1253  1.1892 -0.1867
```

```
ans =  
     1  0  
     0  1  
     0  0
```

Es muy importante recalcar que las matrices aleatorias “rand” y “randn” operan a través de semillas o parámetros que generan los números “aleatorios”. Estas semillas se pueden cambiar a discreción pudiendo generar así una misma secuencia de matrices aleatorias cada vez que se necesite. Genere una secuencia de 3 matrices 2 x 3 con la semilla -4,3:

```
>> rand('seed',-4.3) % se introduce la semilla  
>> rand(2,3,3) % se generan 3 matrices 2x3
```

```
ans(:,:,1) =  
     0.7177  0.6240  0.4158  
     0.8488  0.7624  0.7269
```

```
ans(:,:,2) =  
     0.2101  0.0769  0.0196  
     0.8048  0.9159  0.8216
```

```
ans(:,:,3) =  
     0.6377  0.8457  0.8849  
     0.7054  0.9203  0.7292
```

Para generar la misma secuencia se repite el procedimiento:

```
>> rand('seed',-4.3)  
>> rand(2,3,3)
```

```
ans(:,:,1) =  
     0.7177  0.6240  0.4158  
     0.8488  0.7624  0.7269
```

```
ans(:,:,2) =
    0.2101 0.0769 0.0196
    0.8048 0.9159 0.8216
```

```
ans(:,:,3) =
    0.6377 0.8457 0.8849
    0.7054 0.9203 0.7292
```

Si, por el contrario, lo que se necesita es no poder recuperar la secuencia aleatoria, lo usual es poner la semilla en función del reloj del PC. Para esto ocupe el comando `rand('seed',sum(clock))`.

**En Resumen:** *MATLAB puede operar con matrices por medio de operadores y por medio de funciones. Ya se han visto algunos operadores tales como suma, resta o producto, así como también se han visto funciones tales como invertir matrices, sacar determinantes, obtener los valores y vectores propios de una matriz, etc.*

### 1.3. Funciones Matemáticas

A continuación se entrega una lista de funciones matemáticas, las que pueden ser aplicadas a escalares o matrices. Es importante recalcar que las funciones trigonométricas están expresadas en radianes.

Función	Descripción
<code>abs(x)</code>	Valor absoluto o magnitud de un número complejo
<code>acos(x)</code>	Inversa del coseno
<code>angle(x)</code>	Angulo de un número complejo
<code>asin(x)</code>	Inversa del seno
<code>atan(x)</code>	Inversa de la tangente
<code>ceil(x)</code>	Redondea hacia más infinito
<code>conj(x)</code>	Complejo conjugado
<code>cos(x)</code>	Coseno
<code>exp(x)</code>	Exponencial
<code>fix(x)</code>	Redondea hacia cero
<code>floor(x)</code>	Redondea hacia menos infinito
<code>imag(x)</code>	Parte imaginaria de un número complejo
<code>log(x)</code>	Logaritmo natural
<code>log10(x)</code>	Logaritmo decimal
<code>real(x)</code>	Parte real de un número complejo
<code>rem(x,y)</code>	Resto después de la división
<code>round(x)</code>	Redondea hacia el entero más próximo
<code>sign(x)</code>	Devuelve el signo del argumento
<code>sin(x)</code>	Seno
<code>sqrt(x)</code>	Raíz cuadrada
<code>tan(x)</code>	Tangente

Para notar la diferencia entre los cuatro tipos de redondeo de números defina el siguiente vector:

```
>> E=[-1.4 -1.5 -1.6 1.4 1.5 1.6]
```

```
E =  
    -1.4    -1.5    -1.6     1.4     1.5     1.6
```

Redondeo hacia más infinito (hacia arriba):

```
>> ceil(E)
```

```
ans =  
    -1    -1    -1     2     2     2
```

Redondeo hacia cero, es decir, a más infinito si es negativo y menos infinito si es positivo:

```
>> fix(E)
```

```
ans =  
    -1    -1    -1     1     1     1
```

Redondeo hacia menos infinito:

```
>> floor(E)
```

```
ans =  
    -2    -2    -2     1     1     1
```

Redondeo al entero más cercano:

```
>> round(E)
```

```
ans =  
    -1    -2    -2     1     2     2
```

## 2. Introducción a los Gráficos

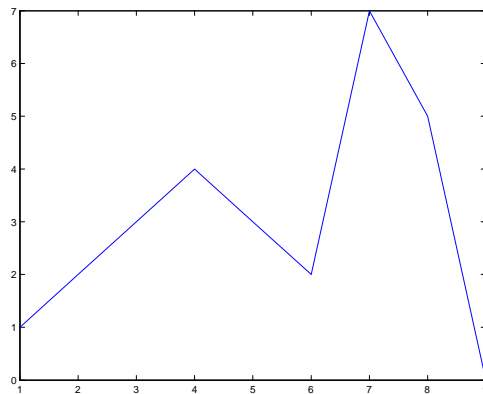
### 2.1. Gráficos de Vectores

Una de las mayores ventajas de `MATLAB` es la facilidad de éste para crear gráficos. Por ejemplo, suponga que tiene algunos datos y quiere graficarlos, basta con introducirlos en forma de vector y luego ejecutar el comando “`plot(·)`”:

```
>>clear, d = [1 2 3 4 3 2 7 5 0], plot(d)
```

```
d =
```

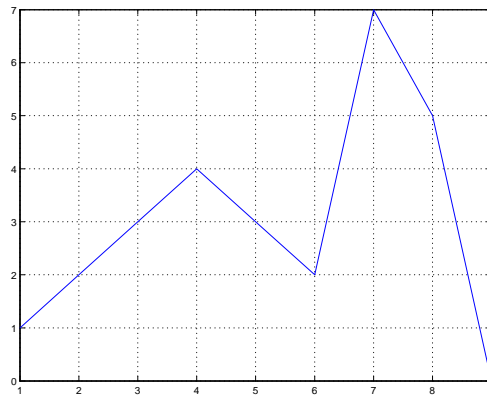
```
1 2 3 4 3 2 7 5 0
```



**Figura 1:** Gráfico de un vector fila.

Para cuadrricular un gráfico se ocupa el comando “`grid`”:

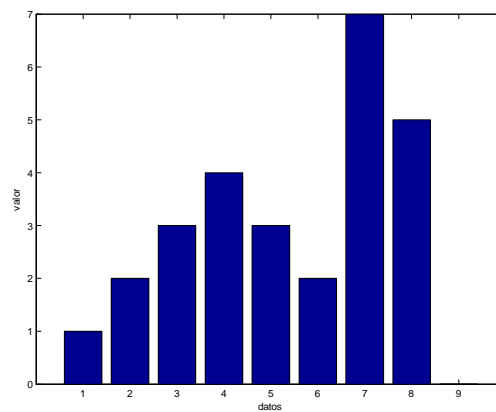
```
>> plot(d); grid
```



**Figura 2:** Gráfico de un vector fila con “rejilla”.

Si se quiere representar los datos en un gráfico de barras basta con ocupar el comando “`bar(·)`” y para nombrar los ejes use los comandos “`xlabel`” e “`ylabel`”, por ejemplo:

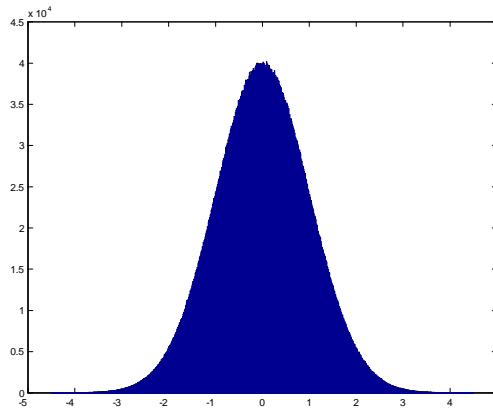
```
>> bar(d); xlabel('datos'); ylabel('valor')
```



**Figura 3:** Gráfico de barras de un vector fila.

También se pueden graficar histogramas. Grafique el histograma de 9.999.999 datos aleatorios provenientes de una distribución normal estándar para el dominio  $[-4,5, 4,5]$  y en intervalos de 0.01:

```
>> y=randn(9999999,1); % Generación de datos aleatorios
>> x=-4.5:0.01:4.5; % Generación del dominio
>> hist(y,x)
```

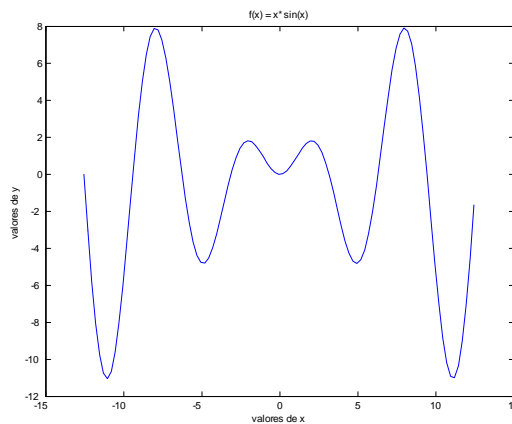


**Figura 4:** Una linda campana de Gauss.

## 2.2. Gráficos de Funciones

Grafique  $x \sin(x)$  en el intervalo  $[-12, 12]$ :

```
>> x=-12:0.1:12;    % Generar el vector dominio
>> f=x.*sin(x);    % Generar el vector recorrido
>> plot(x,f),title('f(x) = x*sin(x)'), xlabel('valores de x'), ylabel('valores de y')
```

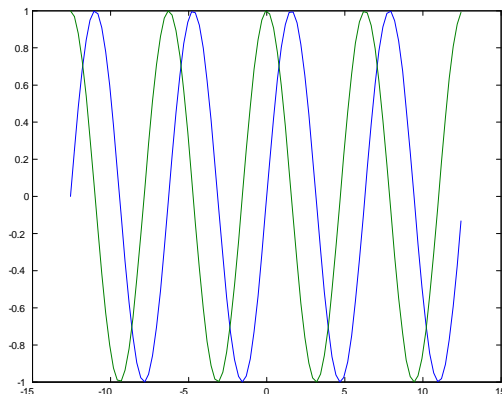


**Figura 5:** Gráfico de  $x \sin(x)$ .

El primer el comando `x=-12:0.1:12` le dice a `MATLAB` que grafique el dominio  $[-12, 12]$  iterando en intervalos de 0.1. El segundo evalúa el dominio en la función  $x \sin(x)$ , recuerde que como lo que se quiere es evaluar cada punto del dominio en la función se tiene que agregar un punto a la operación de multiplicación, es decir `x.*sin(x)`. Por último, `plot(x,f)` significa que queremos graficar los vectores `x` y `f`.

Para graficar más de una función en un gráfico escriba los siguientes comandos:

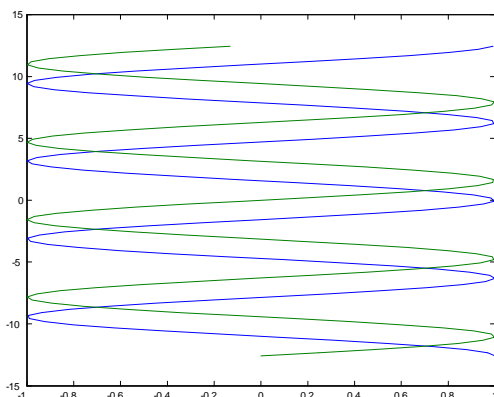
```
>> y=sin(x);
>> z=cos(x);
>> plot(x,y,x,z)
```



**Figura 6:** Gráfico de  $\sin(x)$ ,  $\cos(x)$ .

Para invertir el orden de los ejes:

```
>> plot(z,x,y,x)
```

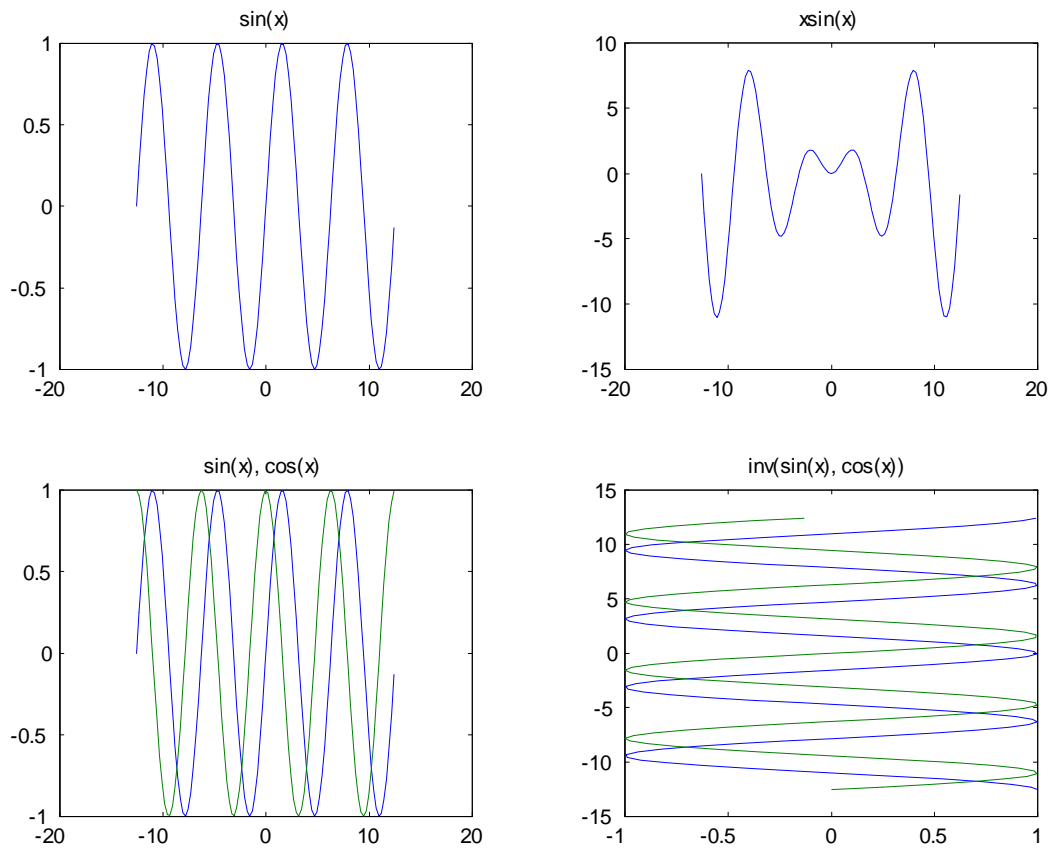


**Figura 7:** Espirales

Con el comando Subplot se pueden generar matrices de gráficos, por ejemplo:

```
>> subplot(2,2,1) % Matriz de gráficos 2x2, gráfico 1
>> plot(x,y), title('sin(x)')
>> subplot(2,2,2) % Matriz de gráficos 2x2, gráfico 2
>> plot(x,f), title('xsin(x)')
>> subplot(2,2,3) % Matriz de gráficos 2x2, gráfico 3
>> plot(x,y,x,z), title('sin(x), cos(x)')
```

```
>> subplot(2,2,4) % Matriz de gráficos 2x2, gráfico 4
>> plot(z,x,y,x), title('inv(sin(x), cos(x))')
```



**Figura 8:** Matriz de gráficos

### 3. Mapas de Gradientes y Curvas de Solución.

#### 3.1. El Default Directory

MATLAB siempre trabaja conectado a una carpeta del PC, a esta carpeta se le llama *default directory* y sirve para abrir, ejecutar y cambiar los documentos con los que se trabaja de una manera rápida e inequívoca. Siga los siguientes pasos:

1. Abra Mi PC.
2. Abra C:
3. Cree una nueva carpeta llamada “matlab” (todo en minúsculas).
4. Descargue un archivo llamado matlab.zip de <http://www.agparra.com/matlab.zip> y guárdelo en la carpeta matlab.
5. Descomprima los archivos de matlab.zip en la carpeta matlab que usted creó.

Ahora, ejecute los siguientes comandos en el Command Window:

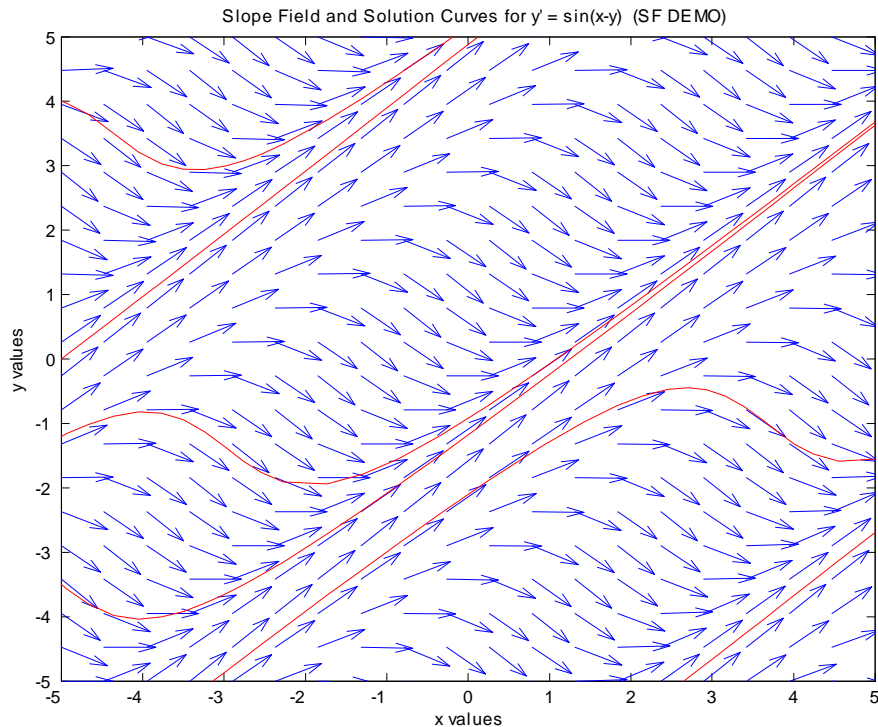
<b>¿Qué queremos hacer?</b>	<b>Comando</b>
Saber cuál es el default directory.	<code>pwd</code>
Conocer qué archivos hay en él.	<code>ls</code>
Cambiar el default directory al que creamos.	<code>cd c:\matlab\</code>
¿Cuál es el default directory?	<code>pwd</code>
¿Qué archivos hay en él?	<code>ls</code>

Donde debiera aparecer lo siguiente:

```
>> ls
. Euler.m demo.m matlab.zip
.. bell.m ejemplo.m sfield.m
```

Si esos son los archivos listados escriba el siguiente comando en el Command Window:

```
>> demo
```



**Figura 9:** Mapa de Gradientes + Curvas de Solución.

En este minuto debió a parecer un gráfico como el de la **Figura 9**. En esta sección aprenderá hacer e interpretar este tipo de gráficos.

### 3.2. Los M-files

En `MATLAB` existen muchos archivos de trabajo, nosotros nos concentraremos en los archivos M o m-files. Estos archivos sirven para crear programas en `MATLAB`. Básicamente lo que aprenderá es programar uno (o varios) m-files para un problema.

Lo primero que necesita saber es cómo crear uno nuevo. En la barra de herramientas de `MATLAB` se encuentra el ítem *File*. Una vez ahí tiene que ir a *New* y luego seleccionar *m-file*.

### 3.3. Programando un M-file

Una vez que ya abrió un m-file nuevo escriba lo siguiente:

```
function dydt = ecuacion1(t,y)
    dydt = sin(t-y);
```

Guárdelo en el directorio que creó (`c:\matlab\`) conservando el nombre que trae (puede notar que el nombre es “ecuacion1”). Lo que hizo fue crear un m-file que

contiene la siguiente función

$$f(t, x) = \sin(t - y)$$

donde  $t$  e  $y$  son las variables. Analizando lo escrito en la primera línea del archivo, el comando “`function`” indica que se va a introducir una función. El término “`dydt`” representa el output de la función, se podría haber puesto cualquier otro nombre pero ponemos `dydt` para simbolizar que el output de esta función será la derivada de  $y$  respecto a  $t$ .<sup>4</sup> El término “`ecuacion1`” es el nombre de la ecuación y por último “`(t,y)`” representa las variables de la función, en resumen, la primera línea es de la siguiente forma:

```
function [Matriz output] = nombre_de_la_función([Matrices input])
```

Por último la segunda línea es la función en sí misma. Para entender lo que hace este m-file<sup>5</sup> realice el siguiente cálculo:

```
>> ecuacion1(1,0)
```

```
ans =  
    0.8415
```

Recuerde que `MATLAB` “piensa” en matrices. Intente lo siguiente:

```
>> a=[1 2 3], b=[0 2 1], ecuacion1(a,b)
```

```
a =  
    1  2  3  
  
b =  
    0  2  1  
  
ans =  
    0.8415    0    0.9093
```

pruebe con la calculadora y asegúrese que cuadren los valores.

**Conclusión:** *Los M-files son los archivos de trabajo de MATLAB, en ellos se pueden introducir funciones y, como veremos más adelante, programas. El archivo `ecuacion1` es el m-file que contiene la función con la que trabajaremos.*

---

<sup>4</sup>Hasta ahora, no hay nada que convierta a  $f(t, y)$  en una ecuación diferencial. Esto lo imponemos cuando queramos resolver la función.

<sup>5</sup>Para los siguientes pasos, tiene que estar seguro de que está trabajando en el default directory que creó y guardó el m-file.

### 3.4. Creando un Mapa de Gradientes

En esta sección se usará el m-file llamado *sfield*, el cual copié en su carpeta matlab al descomprimir los archivos<sup>6</sup>. La programación de este archivo requiere de conocimientos más profundos de programación de los que se necesita desarrollar para el objetivo de este apunte; pero, como ud. ya tiene el m-file, lo podrá ocupar cada vez que sea necesario. La razón de ser de este archivo es graficar la gradiente de una función en un determinado dominio. Para verlo en práctica, escriba lo siguiente en el Command Window:

```
>>sfield('ecuacion1',[-5 5 -5 5],20);
```

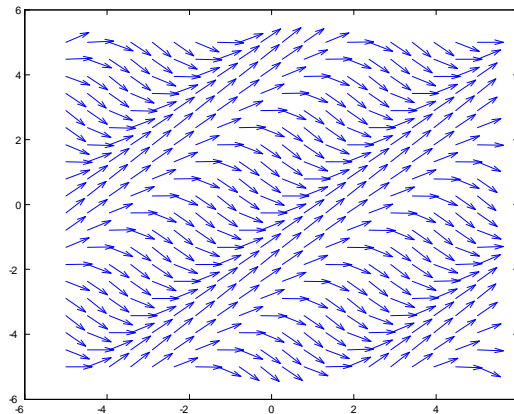


Figura 10: Mapa de Gradientes.

En estos momentos usted debiera tener en frente una gráfica similar a la de la **Figura 10**. Al escribir “`sfield('ecuacion1',[-5 5 -5 5],20)`” le está diciendo a MATLAB que aplique el m-file `sfield` a la función que está en el archivo `ecuacion1` en el dominio  $[-5, 5]^2$  (es decir, se ponen los valores del dominio que se quiere analizar siguiendo el siguiente patrón [min t max t min y max y].) Por último, el número 20 representa en cuántos puntos se quiere dividir cada eje del dominio para evaluar la función, para que nos entregue el vector gradiente en cada uno de esos puntos. En este caso tenemos 20 puntos de de la variable  $t$  y 20 puntos de la variable  $y$ , evaluando la función en todos esos puntos tenemos  $20^2 (= 400)$  gradientes (si no entiende el párrafo anterior cuente las flechas que están en una línea en forma vertical y horizontal, así se dará cuenta que son veinte.) Como ejercicio pruebe cambiar algunos de estos parámetros y ver qué pasa con el mapa de gradientes.

**Conclusión:** *El archivo `sfield.m` es un m-file que mapea las gradientes para una función y dominio dados.*

<sup>6</sup>Este archivo fue desarrollado por J. R. White, UMass-Lowell en Enero de 1998.

### 3.5. Curvas de Solución

MATLAB posee muchos **solvers** de ecuaciones diferenciales, cada uno de ellos representa un método distinto. Nosotros ocuparemos principalmente el solver “**ode23**”, el cual ocupa el método de **Runge-Kutta** de segundo y tercer orden. Alternativamente, el solver “**ode45**” ocupa Runge-Kutta de cuarto y quinto orden. Si lo que tenemos es un dominio muy grande y un computador de baja capacidad, el primer solver nos conviene, ya que hace menos iteraciones para un intervalo dado que el segundo, pero se pierde precisión (algo demasiado importante). Ilustremos lo anterior con un ejemplo. Escriba lo siguiente en el Command Window:

```
>> [t,y]=ode23('ecuacion1',[0 1000],-1.2);
```

Este comando le dice a MATLAB que resuelva la ecuación1 como una ecuación diferencial usando el solver ode23 para el dominio de la variable independiente (tiempo) entre cero y mil, con la condición inicial  $y(0) = -1,2$ . [t,y] significa que guarde la información en las matrices **t** e **y**, formalmente escribimos:

```
[matriz_var_inde,matriz_var_depen]=solver('función',[t0 tf],y(t0));
```

En el Workspace aparecieron las matrices **t** e **y**, las cuales son de orden 19 x 1. Abra la matriz **y**, y revise el último valor correspondiente a  $y(1000)$ , el cual debió ser igual a 998.43. Ahora haga la misma operación pero con el solver ode45:

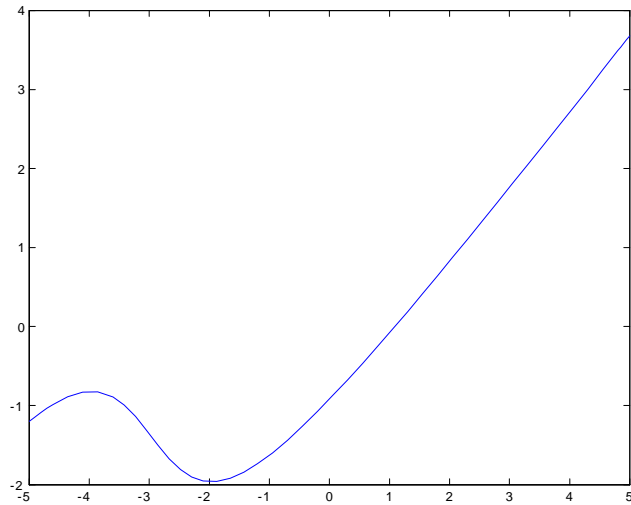
```
>> [t,y]=ode45('ecuacion1',[0 1000],-1.2);
```

Vaya al workspace y note que las matrices ahora son de orden 453 x 1. Es decir, MATLAB hizo más iteraciones para el mismo intervalo. Ahora revise  $y(1000)$  el cual debió ser 593.98, aproximadamente dio una diferencia de 400, que corresponde a un error de aproximación. Corra ahora el solver (cualquiera) para el dominio [-5 5].

```
>> [t,y]=ode45('ecuacion1',[-5 5],-1.2);
```

y escriba lo siguiente:

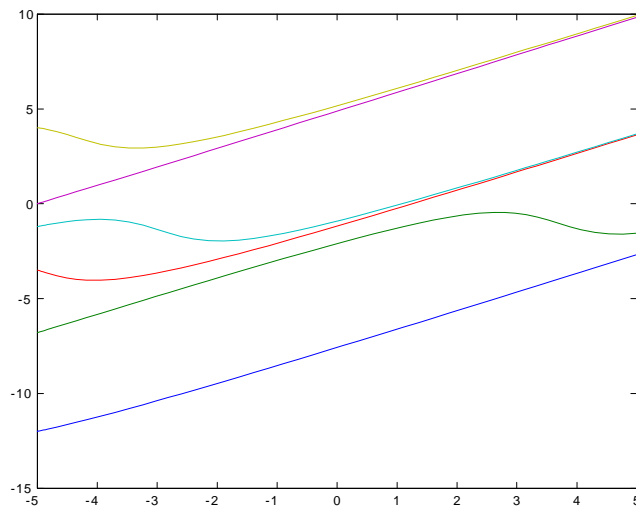
```
>> plot(t,y);
```



**Figura 11:** Curva de solución para  $y(0) = -1,2$ .

Pero recuerde que la mayor virtud de `MATLAB` son las matrices. Escoja ahora un vector de condiciones iniciales:

```
>> v=[-12 -6.8 -3.5 -1.2 0 4]; [t,y]=ode45('ecuacion1',[-5 5], v); plot(t,y);
```



**Figura 12:** Curvas de solución para un vector de condiciones iniciales.

Ya ha aprendido a resolver y dibujar ecuaciones diferenciales. Para hacer un gráfico que mezcle un mapa de gradientes con curvas de solución, hay que crear dos m-file uno con la ecuación diferencial y otro con los comandos de ploteo. Si quiere aprender como se hace, escriba “`open demo`” en el Command Window donde se explica el procedimiento.

### 3.6. Curvas de Solución para un Sistema de Ecuaciones Diferenciales

El procedimiento para resolver y graficar curvas de solución para un **sistema** de ecuaciones diferenciales es básicamente el mismo. Suponga que quiere analizar el siguiente problema:

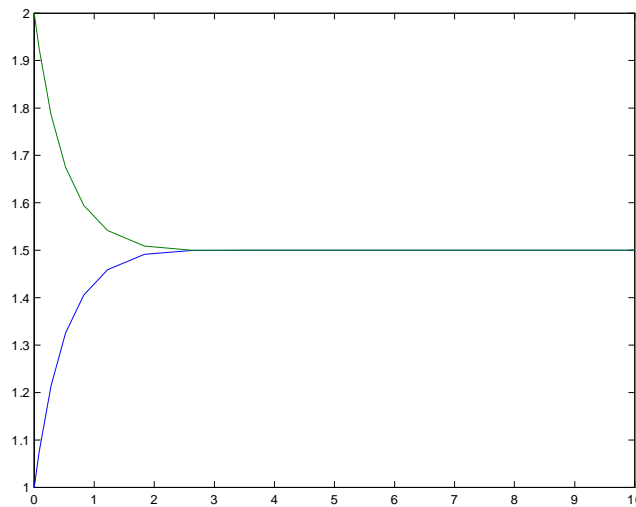
$$\begin{pmatrix} c_t^1 \\ c_t^2 \end{pmatrix} = \begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} c_t^1 \\ c_t^2 \end{pmatrix} \text{ con condiciones iniciales } \begin{pmatrix} c_0^1 = 1 \\ c_0^2 = 2 \end{pmatrix}$$

el m-file se tiene que programar de la siguiente forma:

```
function dc = eqn(t,c)
    dc = zeros(2,1);
    dc(1)=-c(1)+c(2);
    dc(2)=c(1)-c(2);
```

La primera línea es igual a la que ya conocíamos. La segunda le dice a `MATLAB` que el output es una matriz de orden 2 x 1. Por último, las últimas dos líneas representan el sistema escrito en forma de ecuaciones. Luego, para dibujar las curvas de solución, hay que escribir lo siguiente en el Command Window:

```
>> co=[1 2], [t,c]=ode23('eqn',[0 10],co); plot(t,c)
```



**Figura 13:** Sistema de ecuaciones diferenciales.

Donde `co` es el vector de condiciones iniciales y el intervalo `[0 10]` el dominio para la variable independiente.

### 3.7. Ejercicios

- a) Repita todos los pasos anteriores para el siguiente modelo logístico:

$$y' = 0,1(10 - y)y$$

encontrando un dominio adecuado y condiciones iniciales interesantes. No pase al siguiente capítulo sin que resuelva este ejercicio.

- b) Analice gráficamente el siguiente sistema:

$$\begin{pmatrix} x'_t \\ z'_t \\ v'_t \\ y'_t \\ w'_t \end{pmatrix} = \begin{pmatrix} -1 & 2 & 3 & 4 & 5 \\ 5 & -5 & 4 & 3 & 2 \\ 1 & 3 & -4 & 8 & 2 \\ 1 & 1 & 1 & 1 & 1 \\ 2 & 1 & 3 & 1 & 1 \end{pmatrix} \begin{pmatrix} x_t \\ z_t \\ v_t \\ y_t \\ w_t \end{pmatrix}$$

## 4. Elementos de Programación

En éste capítulo se entregan elementos básicos de programación, más algunas aplicaciones de éstos a las ecuaciones diferenciales y a la programación dinámica.

### 4.1. Manipulación Matricial

Una tercera forma de crear matrices es mediante la función “`linspace`”, con ésta uno pone un valor inicial, otro final y cuántos elementos queremos que haya en este intervalo:

```
>> A=linspace(1,9,6)
```

```
A =  
    1    2.6    4.2    5.8    7.4    9
```

Si se necesita **cambiar o agregar** un elemento de A lo puede hacer con:

```
>> A(1,6)=10
```

```
A =  
    1    2.6    4.2    5.8    7.4   10
```

donde se cambió el sexto elemento de la primera fila por un 10. Dos funciones de mucha utilidad en programación son “`size`” y “`length`”. La primera nos entrega el orden de la matriz, la segunda el máximo entre el número de filas y el número de columnas. Por ejemplo:

```
>> B=[9 8 7 6; 5 4 3 2], C=pi:0.5:2*pi, size(B), [f,c]=size(C), length(B),  
length(C)
```

```

B =
    9 8 7 6
    5 4 3 2

C =
    3.1416    3.6416    4.1416    4.6416    5.1416    5.6416    6.1416

ans =
    2 4

f =
    1

c =
    7

ans =
    4

ans =
    7

```

Se pueden ocupar los comandos anteriores para crear matrices especiales

```
>> ones(size(B))
```

```
ans =
    1 1 1 1
    1 1 1 1
```

El símbolo *dos puntos* (:) representa todas las filas o todas las columnas de una matriz:

```
>> B(:,3)% Todos los elementos de la tercera columna
```

```
ans =
    7
    3
```

```
>> B(2,:) % Todos los elementos de la segunda fila
```

```
ans =  
    5    4    3    2
```

y si sólo se ponen los dos puntos se reagrupa la matriz en un vector columna:

```
>> B(:)
```

```
ans =  
    9  
    5  
    8  
    4  
    7  
    3  
    6  
    2
```

Observe que todos los operadores definidos se pueden entremezclar, por ejemplo:

```
>> D = [1 2 3 4 5; 6 7 8 0 1], D(2,1:2:5)
```

```
D =  
    1    2    3    4    5  
    6    7    8    0    1  
ans =  
    6    8    1
```

entrega los elementos de la segunda fila cuyas columnas pertenecen al vector `1:2:5` `= [1 3 5]`. Si se requieren todos los elementos de una columna (o fila), en vez de usar un vector que contenga los números de todas las columnas se usa el símbolo de dos puntos (`:`), por ejemplo:

```
>> D(:,3:5)
```

```
ans =  
    3    4    5  
    8    0    1
```

Por último, la función “`max(·)`” entrega el mayor valor por columna:

```
>> A=[1 2 3; 4 5 6; 7 8 0], max(A)
```

```
A =
    1  2  3
    4  5  6
    7  8  0
```

```
ans =
    7  8  6
```

Pero, si se ocupa de la siguiente forma “`max(k,x)`”, donde  $k$  es un escalar, reemplaza todos los valores menores de  $k$  por  $k$ :

```
>> max(5,A)
```

```
ans =
    5  5  5
    5  5  6
    7  8  5
```

## 4.2. Operadores Lógicos y Relacionales

Estos operadores sirven para evaluar el valor de verdad de dos proposiciones lógicas y entregan valores de 1 si la condición es verdadera y 0 si es falsa, por eso son llamados operadores relacionales:

Operador	Descripción
<	Menor que
<=	Menor o igual a
>	Mayor que
>=	Mayor o igual que
==	Igual a
~=	No igual a

por otro lado son operadores lógicos:

Operador	Descripción
&	y
	o
~	no

Para algunos ejemplos cree las siguientes matrices:

```
>> clear, A=1:9,B=9-A
```

```
A =
  1  2  3  4  5  6  7  8  9
B =
  8  7  6  5  4  3  2  1  0
```

Encuentre los valores de A mayores a 4:

```
>> C=A>4
```

```
C =
  0  0  0  0  1  1  1  1  1
```

Niege C:

```
>> F=~C
```

```
F =
  1  1  1  1  0  0  0  0  0
```

encuentre los valores de A iguales a B:

```
>> D=A==B
```

```
D =
  0  0  0  0  0  0  0  0  0
```

encuentre donde  $A > 2$  y reste el resultado a B:

```
>> E=A>2, B-E
```

```
E =
  0  0  1  1  1  1  1  1  1
ans =
  8  7  5  4  3  2  1  0 -1
```

y para juntar operadores ocupe "&":

```
>> G=E&(A<6)
```

```
G =
  0  0  1  1  1  0  0  0  0
```

### 4.3. Iterando: Herramientas de Control de Flujo

MATLAB NOS entrega herramientas para poder generar flujo de números, es decir, iteraciones. Estas herramientas pueden ser relacionadas con los operadores lógicos anteriores y generar flujos de números que se comporten “inteligentemente”, es decir, de la manera que uno quiera. La siguiente tabla presenta los principales controles de flujo:

Estructura de Control de Flujo	Descripción
<b>for</b> x= una matriz órdenes <b>end</b>	En cada iteración asigna x a la columna i-ésima de la matriz y ejecuta órdenes
<b>while</b> expresión órdenes <b>end</b>	While ejecuta órdenes mientras los elementos de la expresión sean verdaderos
<b>if</b> expresión órdenes si la expresión es verdadera <b>else</b> órdenes si la expresión es falsa <b>end</b>	Esta estructura evalúa un tipo de orden si la expresión es verdadera y otro tipo si es falsa. Si se quiere poner más de dos tipos de órdenes se pone <b>elseif</b> en vez de <b>else</b> , hasta el último condicionante, que se pone <b>else</b> .
<b>break</b>	Termina la ejecución del los comandos while y for

Considere el siguiente ejemplo para la estructura “for”. Escriba:

```
>> for m=1:5      %crea valores del 1 al 5
A(m,1)=m^2      %crea la matriz
end
```

Lo que creó una secuencia de 5 matrices, cada una conformada por la anterior más un nuevo elemento, el cual es sí mismo al cuadrado.

Para entender el comando “while” calcule cuántas iteraciones son necesarias para llegar a 0.0000000001 dividiendo números por la mitad partiendo de 1:

```
>> i=0;n=1;      % i es el número de pasos y uno el número a dividir
>> while n>0.0000000001      % le pedimos que itere mientras n sea mayor
al valor crítico
n=n/2;          %que en cada paso divida a n por la mitad
i=i+1;          % que en cada paso sume 1 a i
end
```

¿Cuántos pasos son?

```
>> i
```

```
i =  
    34
```

Por último, ejemplifiquemos el comando “if”:

```
>> peras=6; costo=6*peras
```

```
costo =  
    36
```

```
>> if peras>5  
costo=0.6*costo  
else  
costo=costo  
end
```

```
costo =  
    21.6
```

#### 4.4. Aplicación 1: Ecuaciones de diferencias

En esta sección se usarán las herramientas desarrolladas para programar y analizar ecuaciones de diferencias. Considere la siguiente ecuación:

$$x_{(k+2)} = 3^k - kx_{(k+1)} - k^2x_{(k)}, \quad x_{(0)} = 2, \quad x_{(1)} = 4$$

¿Cuáles serán los valores de  $x_{(2)}, x_{(3)}, x_{(4)}, \dots, x_{(7)}$ ? Parta introduciendo las condiciones iniciales:

```
>> X=[2 4]
```

```
X =  
    2    4
```

Defina  $n$  como la variable de tiempo:

```
>> n=1
```

```
n =  
    1
```

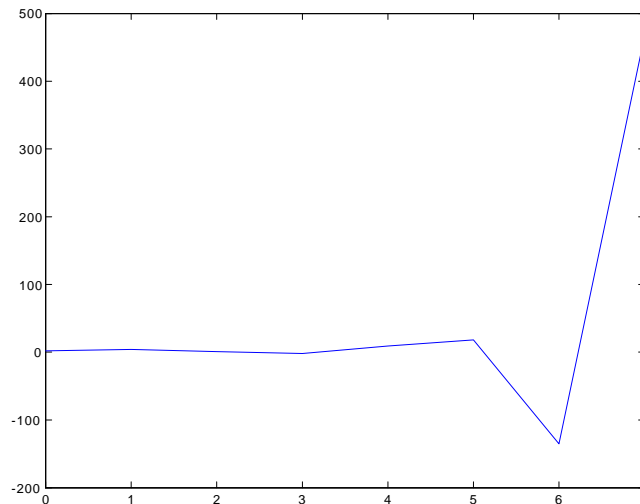
Note que la ecuación de diferencias empieza en tiempo 0 pero, como veremos más adelante,  $n$  no sólo va a servir como dominio sino que también va a buscar elementos de la matriz  $X$ . Procederemos iterando la ecuación:

```
>> while n<7  
X(n+2)=3^(n-1)-(n-1)*X(n+1)-((n-1)^2)*X(n);  
n=n+1;  
end
```

Como se puede apreciar, lo que se hizo fue agregar términos a la matriz de soluciones  $X$  en función de los términos ya existentes, tal como prescribe la ecuación. Restando 1, se ajustaron las variables que dependían del tiempo y que no corresponden a la matriz  $X$ . Pues, como se mencionó anteriormente, el tiempo empieza de cero y  $n$  comienza en uno.

Para graficar el sistema ejecute el siguiente comando:

```
>> m=0:7; plot(m,X) % redefine el dominio para que cuadre y graficamos
```



**Figura 14:** Sistema de ecuaciones en diferencias.

#### 4.5. Aplicación 2: El Método de Euler

En este apartado programará un m-file para poder resolver numéricamente, a través del método de Euler, cualquier sistema de ecuaciones diferenciales de la forma:

$$x'(t) = f(x(t), t) \text{ con condición inicial } x(0) = x_0.$$

Recuerde que el método de Euler aproxima las variables del siguiente modo:

$$x_{(t+\delta)} = x_{(t)} + x'_{(t)}\delta$$

Para comenzar, abra un m-file nuevo y escriba los siguientes comandos:<sup>7</sup>

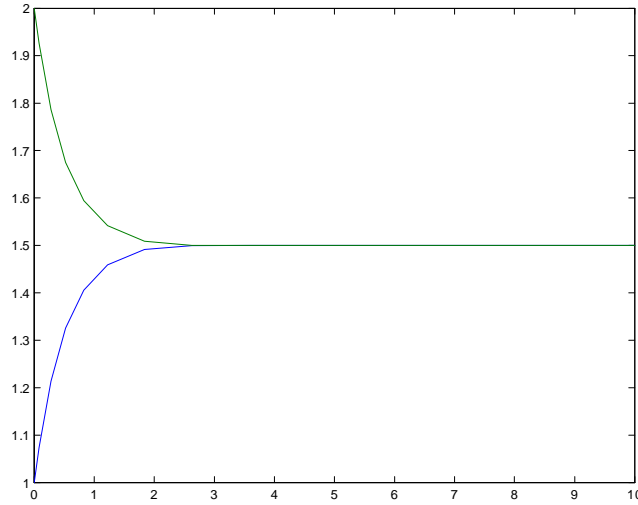
```
%EULER.M Este es el método de Euler para resolver numéricamente una ecuación
diferencial
% Los inputs son:
%   -fxy = es la variable que se usa para representar al m-file que contiene
la ecuación que se desea resolver
%   -xo,xf = son valores inicial y final de la variable independiente, son
escalares
%   -yo = las condiciones iniciales del sistema en xo, es un vector
%   -N = el número de pasos en que se va a resolver el problema
% Los outputs son:
%   -X = un vector que contiene los valores de la variable independiente
%   -Y = una matriz que contiene las variables dependientes
function [X,Y] = euler1(fxy,xo,xf,yo,N)
% para computar nuestro delta se necesita el número de pasos
if N < 2 N = 2; end % dice que no se ejecuta el programa si N<=2
h = (xf-xo)/N; % Calcula el delta
X = zeros(N+1,1); % crea un vector de ceros para ir reemplazando cada
celda en cada iteración
M = length(yo); % número de ecuaciones (columnas de la matriz Y )
Y = zeros(M,N+1); % Crea la matriz de ceros de la variable dependiente
para reemplazar los resultados en cada iteración
% reemplazar en la matriz de ceros las condiciones iniciales
X(1) = xo; Y(:,1) = yo';
% y empieza a iterar
for i = 1:N % lo que sigue es la definición de Euler
k1 = h*feval(fxy,X(i),Y(:,i)); % evalúa la función en el punto i y lo
multiplica por el delta
Y(:,i+1) = Y(:,i) + k1; % incrementa la variable dependiente en
el valor anterior y lo guarda en la fila i de la matriz de ceros
X(i+1) = X(i) + h; % incrementa la variable independiente y lo guarda
en la fila i matriz de ceros
end
% fin del programa
```

Guarde el archivo con el nombre de Euler. Finalmente, para ejecutar el programa tiene que escribir:

---

<sup>7</sup>En este apartado, se ha copiado el m-file realizado por el autor. Por esa razón, las explicaciones vienen después del signo tanto por ciento.

```
>>yo=[1 2];[S,F]=euler('eqn',0,10,yo,30); plot(S,F)
```



**Figura 15:** Método de Euler.

Como se puede apreciar, el resultado es el mismo que en la **Figura 13**. La ventaja del programa Euler es que es posible controlar el número de iteraciones realizadas para calcular el resultado y, por lo tanto, controlar la precisión del resultado.

#### 4.6. Aplicación 3: Programación Dinámica

Considere el siguiente problema:<sup>8</sup>

$$\begin{aligned}
 v_0(k_0) &= \max \sum_{t=0}^T \beta^t u(c_t) \quad \text{s.a.} \\
 k_1 &= f(k_0) - c_0 - \delta k_0 \\
 k_2 &= f(k_1) - c_1 - \delta k_1 \\
 &\vdots \\
 k_T &= f(k_{T-1}) - c_{T-1} - \delta k_{T-1} \\
 k_0 &> 0
 \end{aligned}$$

el que, bajo ciertas condiciones de regularidad, puede ser planteado a través de la ecuación de Bellman:

$$Tv(k_i) = \max_{k'_j} u(f(k_i) - k'_j - \delta k_i) + \beta v(k'_j)$$

donde  $k_i$  es el stock de capital actual y  $k'_j$  es el stock futuro. Para resolver el problema numéricamente, se discretiza la variable de estado (el capital). Para

<sup>8</sup>Este apartado está basado en los apuntes de programación dinámica del curso Teoría Macroeconómica II, dictado por el profesor Rodrigo Cerda.

esto, se calcula el máximo capital posible en la economía y se particiona en  $n$  intervalos iguales, cumpliéndose que:

$$\begin{aligned} k_{(0,t)} &= a \\ k_{(1,t)} &= a + a \\ k_{(2,t)} &= a + a + a \\ &\vdots \\ k_{(n,t)} &= na = k^{\text{máx}} \end{aligned}$$

para conformar la matriz en la Tabla 1.

Se itera la matriz en la Tabla 1 hasta que se cumpla que  $v_t(k_{(i,t)}) = v_{t+1}(k_{(j,t+1)})$  para todo  $i, j$ . Es decir, se itera la matriz hasta llegar a estado estacionario. Posteriormente, se escoje para cada fila el stock de capital que genere la mayor utilidad. Intuitivamente, se construye la mejor respuesta del individuo para cada nivel de stock de capital que posee.

Cada elemento de la matriz en la Tabla 1 es de la forma  $u + \beta v$ , donde  $u$  es la utilidad actual,  $\beta$  el factor de descuento y  $v$  la utilidad futura. La utilidad de hoy y el factor de descuento son valores conocidos; luego,  $v$  es el único elemento a calcular. Dado lo anterior, se puede separar la matriz en la Tabla 1 en dos matrices; la matriz  $U$  para la utilidad actual y la matriz  $V$  para las iteraciones del valor futuro de la utilidad descontada. Para poner el método en práctica se definen las siguientes formas funcionales:

$$u(c_t) = \left( \frac{c_t}{1 - \sigma} \right)^{1 - \sigma} \quad \text{y} \quad f(k) = Ak^\alpha.$$

De donde se deduce que  $k^{\text{máx}} = \left( \frac{\delta}{A} \right)^{\frac{1}{\alpha - 1}}$ .

El m-file a programar es el siguiente:

```
% Valores de los parámetros (son inventados)
A=5;% Coeficiente de tecnología
alpha=0.3;% Productividad del capital
delta=0.5;% Depreciación
beta=0.9;% Factor de descuento
kmax=(delta/(A))^(1/(alpha-1))
sigma=3;
% Se discretiza la variable estado (capital) en 100 partes
k=(kmax/100:kmax/100:kmax)'; % k es de orden 100 x 1 pues el vector está
traspuesto
% Se crea un vector columna de unos
ek=ones(size(k,1),1); % ek es de orden 100 x 1
% Crea una matriz con cada discretización repetida por columna
a=ek*k' % a es de orden 100 x 100
b=k*ek' % Crea una matriz con cada discretización repetida por fila
```

```

% Crea la matriz de consumo
c=(max(0.000000001,(A*(a).^alpha+(1-delta)*(a)-(b)))) % Se acota por abajo
ya que la iteración se indefine si C=0
% Crea la matriz de utilidad
U=(c).^(1-sigma)/(1-sigma);
% Crea la matriz V
V=zeros(size(k,1),1);
% Y la matriz V en t+1
Vn=V

clf;
hold on;
i=0; % Variable que cuenta las iteraciones realizadas
crit=1; % Inicialización del valor critico
% Y se empieza a iterar
while crit>0.000000001; % Condición para terminar
plot(k,V); % Grafica
Vn=(max(U+beta*V*ek'))'; % El cómo tiene que iterar
crit=max(abs(Vn-V)); % El cómo se comporta el valor crítico
V=Vn % El como almacena los resultados
i=i+1; % Número de iteraciones
end
% La matriz de mejor respuesta
[V policy]=max(U+beta*V*ek');
hold off
% Grafica
kn=k(policy);
figure; plot(k,kn,k,k)

```

Tabla 1: Matriz de Estados

$\mathbf{k}_{(0,t+1)}$	$\mathbf{k}_{(0,t+1)}$	$\dots$	$\mathbf{k}_{(n,t+1)}$
$\mathbf{k}_{(0,t)}$	$u(f(\mathbf{k}_{(0,t)}) - \mathbf{k}_{(0,t+1)} - \delta \mathbf{k}_{(0,t)}) + \beta v(\mathbf{k}_{(0,t+1)})$	$\dots$	$u(f(\mathbf{k}_{(0,t)}) - \mathbf{k}_{(n,t+1)} - \delta \mathbf{k}_{(0,t)}) + \beta v(\mathbf{k}_{(n,t+1)})$
$\mathbf{k}_{(1,t)}$	$u(f(\mathbf{k}_{(1,t)}) - \mathbf{k}_{(0,t+1)} - \delta \mathbf{k}_{(1,t)}) + \beta v(\mathbf{k}_{(0,t+1)})$	$\dots$	$u(f(\mathbf{k}_{(1,t)}) - \mathbf{k}_{(n,t+1)} - \delta \mathbf{k}_{(1,t)}) + \beta v(\mathbf{k}_{(n,t+1)})$
$\vdots$	$\ddots$	$\ddots$	$\vdots$
$\mathbf{k}_{(n,t)}$	$u(f(\mathbf{k}_{(n,t)}) - \mathbf{k}_{(0,t+1)} - \delta \mathbf{k}_{(n,t)}) + \beta v(\mathbf{k}_{(0,t+1)})$	$\dots$	$u(f(\mathbf{k}_{(n,t)}) - \mathbf{k}_{(n,t+1)} - \delta \mathbf{k}_{(n,t)}) + \beta v(\mathbf{k}_{(n,t+1)})$

## 5. Otras Herramientas y Aplicaciones

En este apartado se presentan herramientas y aplicaciones que no son tan necesarias para el estudio de ecuaciones diferenciales pero si son útiles en otros ámbitos.

### 5.1. Análisis de Datos

MATLAB nos entrega herramientas para poder hacer análisis estadístico sobre un conjunto de datos. Estos datos se almacenan (al igual que en Excel) en forma de matrices con los datos orientados por columnas, es decir, de la misma forma en que se ocupa en econometría. Para importar una base de datos basta con ir a *File* y presionar sobre el ítem *Import Data*, éste guardara los datos en matrices.

Si los datos están representados en una matriz  $x$ , las operaciones estadísticas que se incluyen son:

<b>Función Estadística</b>	<b>Descripción</b>
<code>corrcoef(x)</code>	Coefficientes de correlación.
<code>cov(x)</code>	Entrega la matriz de covarianzas.
<code>cumprod(x)</code>	Entrega el producto acumulativo por columnas.
<code>diff(x)</code>	Calcula las diferencias entre elementos.
<code>cumsum(x)</code>	Entrega la suma acumulativa de columnas.
<code>hist(x)</code>	Grafica el histograma de barras.
<code>mean(x)</code>	Entrega el promedio por columnas.
<code>median(x)</code>	Entrega la mediana de cada columna.
<code>prod(x)</code>	Entrega el producto de elementos en columnas.
<code>sort(x)</code>	Ordena los números en orden ascendente.
<code>std(x)</code>	Entrega la desviación estándar.
<code>sum(x)</code>	Suma los elementos de cada columna.

### 5.2. Polinomios

Sea el polinomio de la siguiente forma:

$$a_n x^{n-1} + a_{n-1} x^{n-2} + \dots + a_3 x^2 + a_2 x + a_1 = 0$$

Entonces se puede representar en `MATLAB` a través del siguiente vector:

$$p = [a_n \ a_{n-1} \ \dots \ a_3 \ a_2 \ a_1]$$

y realizar las siguientes operaciones:

<b>Función</b>	<b>Descripción</b>
<code>roots(p)</code>	Entrega las raíces del polinomio.
<code>poly(r)</code>	Dado un set de raíces nos entrega el polinomio.
<code>conv(p,d)</code>	multiplica los polinomios p y d.
<code>deconv(p,d)</code>	Divide el polinomio p por el d.
<code>polyder(p)</code>	Calcula la derivada del polinomio p.
<code>polyval(p,x)</code>	Evalúa el polinomio p en los puntos x.

Analice el siguiente polinomio  $x^2 - x - 6$ :

```
>> p=[1 -1 -6]
```

```
p =  
    1    -1   -6
```

Encuentre las raíces:

```
>> r=roots(p)
```

```
r =  
     3  
    -2
```

Ahora, con las raíces, encuentre el polinomio:

```
>> poly(r)
```

```
ans =  
     1    -1   -6
```

Multiplique  $p$  por  $x - 2$ :

```
>> d=[1 -2], c=conv(p,d)
```

```
d =  
    1  -2
```

```
c =  
    1  -3  -4  12
```

Y divida el polinomio  $c$  por  $d$ :

```
>> deconv(c,d)
```

```
ans =  
    1  -1  -6
```

Por último, evalúe el polinomio en los siguientes puntos:

```
>> x=[-2 -1 0 1 2 3], polyval(p,x)
```

```
x =  
   -2   -1    0    1    2    3
```

```
ans =  
    0   -4   -6   -6   -4    0
```

Todas estas herramientas pueden ser usadas para el análisis funcional de ecuaciones diferenciales y de diferencias.

### 5.3. Formato de Números

A veces se requiere trabajar con muchos decimales y otras con números enteros. Para hacer este tipo de cambios hay que ejecutar las siguientes órdenes:

Orden	Descripción
<code>format long</code>	16 dígitos.
<code>format short e</code>	5 dígitos más exponente.
<code>format long e</code>	16 dígitos más exponente.
<code>format hex</code>	Hexadecimal.
<code>format bank</code>	2 dígitos decimales.
<code>format +</code>	Números enteros.
<code>format rat</code>	Aproximación racional.

## 5.4. El Uso de Ayuda

MATLAB trae incorporados buenos *demos* y ayudas para ejecutarlas. Se puede proceder de dos maneras: primero escribiendo **help<nombre\_tema\_en\_inglés>** o bien simplemente escribiendo **help**.