INDICE

INTRODUCCION	**PAG**
PRIMERA PARTE	4
Uso básico de MATLAB	13
1. Generalidades	13
1.1. Manejo de variables	14
1.2. Manejo de expresiones	14
1.3. Manejo de comandos	15
1.4. Manejo de archivos con extensión .m	15
2. Comandos Básicos de Programación	16
2.1. Comando END	16
2.2. Comando IF	16
2.3. Comando WHILE	17
2.4. Comando FOR	19
2.5. Comando PLOT	20
2.6. Funciones de dos variables	22
2.7. Comando DISP	23
2.8. Comando INPUT	24
SEGUNDA PARTE **Comandos Básicos Matemáticos**	26
1. Vectores y matrices	26
1.1. Operaciones matemáticas simples con matrices y vectores	27
1.2. Comandos matemáticos para vectores	28
1.2.1. Comando NORM	28
1.2.2. Comando MIN	29
1.2.3. Comando MAX	31
1.2.4. Comando CROSS	32
1.2.5. Comando LENGTH	32
1.3. Comandos Matemáticos para Matrices	33
1.3.1. Comando NORM	33
1.3.2. Comando MIN	33
1.3.3. Comando MAX	33
1.3.4. Comando SIZE	33
1.3.5. Comando EIG	34
1.3.6. Comando INV	36
1.3.7. Comando DET	37
TERCERA PARTE **Aplicaciones Básicas de MATLAB**	38

1. Modelaje de Sistemas Lineales	38
1.1. Definiendo matrices	39
1.2. Matrices especiales	40
1.3. Aritmética de matrices	40
2. Solución de Sistemas Lineales	42
3. Interpolación Polinomial	46
3.1. Función MATLAB para los Mínimos Cuadrados	50
4. Números Reales y Complejos	51
4.1. Asignación de valores a variables	51
4.2. Operaciones matemáticas simples	51
4.3. Comando matemáticos para números (Complejos y Reales)	53
4.3.1. Comando ABS	53
4.3.2. Comando SQRT	54
4.3.3. Comando ANGLE	54
5. Integrales Definidas	56
5.1. Comando TRAPZ	56
5.2. Creación de vectores decentes	57
6. Gráficas	58
Ejercicios de Aplicación	61
Referencias	68

INTRODUCCION

El nombre de MATLAB proviene de la contracción de los términos **MAT**rix **LAB**oratory y fue inicialmente concebido para proporcionar fácil acceso a las librerías LINPACK y EISPACK, las cuales representan hoy en día dos de las librerías más importantes en computación y cálculo matricial.

MATLAB es un entorno de computación y desarrollo de aplicaciones totalmente integrado orientado para llevar a cabo proyectos en donde se encuentren implicados elevados cálculos matemáticos y la visualización gráfica de los mismos. MATLAB integra análisis numérico, cálculo matricial, proceso de señal y visualización gráfica en un entorno completo donde los problemas y sus soluciones son expresados del mismo modo en que se escribirían tradicionalmente, sin necesidad de hacer uso de la programación tradicional.

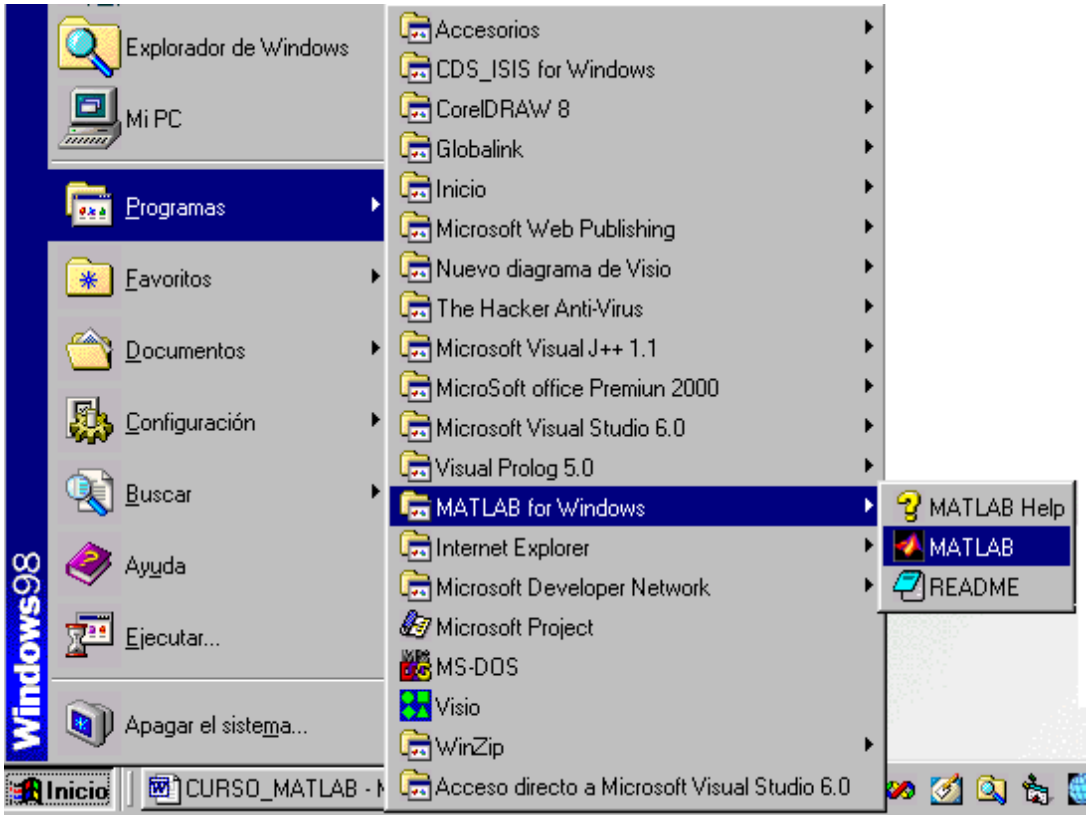
En los medios universitarios MATLAB se ha convertido en una herramienta básica, tanto para los profesionales e investigadores de centros docentes, como una importante herramienta para el dictado de cursos universitarios, tales como sistemas e ingeniería de control, álgebra lineal, proceso digital de imagen, señal, etc. En el mundo industrial MATLAB está siendo utilizado como herramienta de investigación para la resolución de complejos problemas planteados en la realización y aplicación de modelos matemáticos en ingeniería. Los usos más característicos de la herramienta los encontramos en áreas de computación y cálculo numérico tradicional, prototipaje algoritmico, teoría de control automático, estadística, análisis de series temporales para el proceso digital de señal.

MATLAB es la disponibilidad de los toolboxes especializados. Estos son paquetes especializados, orientados a ingenieros, científicos y otros tipos de profesionales técnicos. Entre los más destacados están:

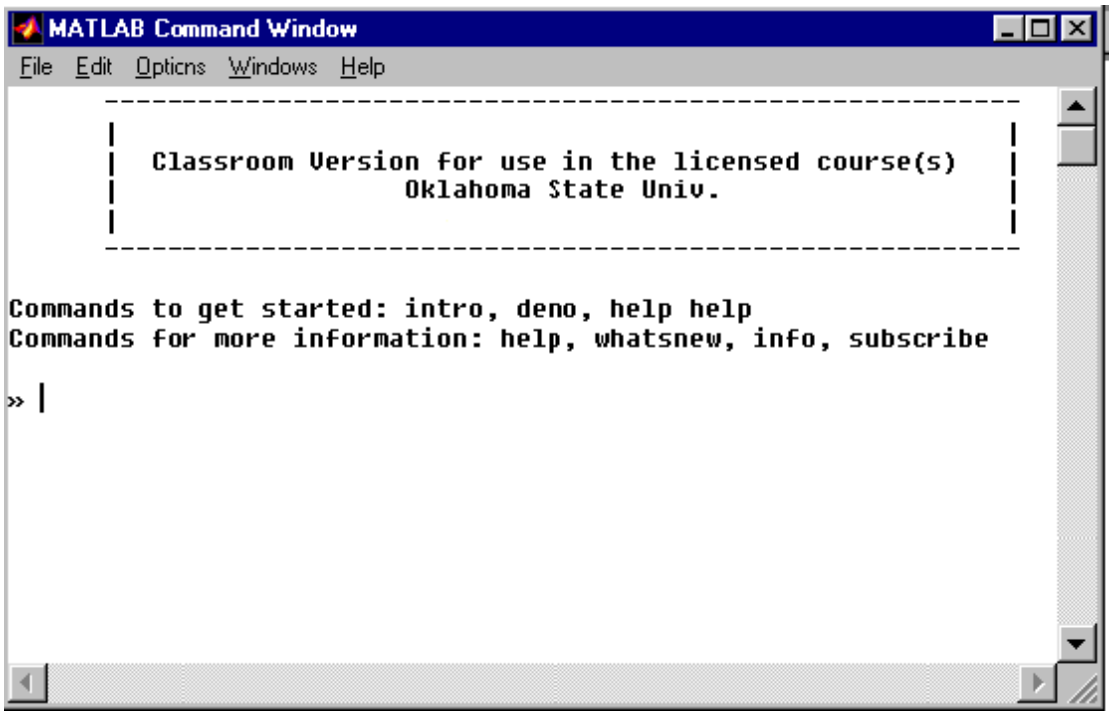
• Procesamiento de Señal • The MATLAB C Math Library	• Diseño de Sistemas de Control • Control Robusto

• Matemáticas Simbólicas • Procesamiento de Imagen • The MATLAB Compiler • Redes Neuronales • Estadística • Splines	• Identificación de Sistemas • Optimización • Simulación • Diseño de Control no Lineal • Lógica Difusa • NAG Fundation Toolbox

Para iniciar MATLAB, seleccionamos el programa MATLAB de un menú del sistema.



y aparece la siguiente ventana de MATLAB, que nos dice que MATLAB está esperando que introduzcamos un comando.



Seguidamente se presentan comandos más usados.

- Para salir de MATLAB, use **quit** o **exit**.
- El comando **clc** despeja la ventana de comandos, y el comando **clf** borra la figura actual y por tanto despeja la ventana de gráficos.
- El comando **clear** no afecta a las ventanas, pero si borra todas las variables de la memoria
- Para ver algunas de las capacidades de MATLAB, usar el comando **demo**, que inicia el MATLAB EXPO, un entorno gráfico de demostración que ilustra algunos tipos de operaciones que se pueden realizar con MATLAB.
- Para abortar un comando en MATLAB, mantener presionada la tecla de control y oprima c (Ctrl. + c). Esto ocasiona un interrupción local dentro del MATLAB.
- Para acceder al menú de ayuda se debe usar el comando **help**.
- El simbolo » denota el *prompt* de MATLAB y no se escribe al entrar instrucciones.
- El ; al final de la instrucción omite el eco o salida a la pantalla.

Exhibición de Números:

Comando MATLAB	**Exhibición**	**Ejemplo**
format short	Por omisión	2.3333
format short e	4 decimales	2.3333e+000
format long	14 decimales	2.33333333333333
format long e	15 decimales	2.333333333333334e+000
format bank	2 decimales	2.33
format hex	exp. hexadecimal	4002aaaaaaaaaab
format +	+, −, espacio	+

Operaciones Aritméticas:

ESCALAR	MATRIZ	VECTOR	DESCRIPCIÓN
+	+	+	Adición
–	–	–	Sustracción
*	*	.*	Multiplicación
/	/	./	División hacia la derecha
\	\	.\	División hacia la izquierda
^	`	.'	Transposición

Operadores Relacionales

OPERADOR	DESCRIPCIÓN
<	**menor que**
< =	**menor o igual que**
>	**mayor que**
> =	**mayor o igual que**
= =	**Igual**
~ =	**no igual**

Operadores Lógicos

OPERADOR	DESCRIPCIÓN
&	Y (and)
|	O (or)
~	NO (not)

Combinaciones:

P	Q	~ P	P |Q	P&Q
falso	Falso	Verdadero	Falso	falso
Falso	verdadero	Verdadero	Verdadero	falso
Verdadero	Falso	Falso	Verdadero	falso
Verdadero	verdadero	Falso	verdadero	verdadero

Caracteres Especiales:

CARACTERES	DESCRIPCIÓN
[ ]	Se utilizan para formar vectores y matrices
( )	Define precedencia en expresiones aritméticas. Encierra argumentos de funciones en forma usual
,	Separador de elementos de una matriz, argumentos de funciones y declaraciones en líneas con declaraciones múltiples
;	Separador de declaraciones, termina renglones de una matriz

Ejemplos Básicos

```
>> 13/3

ans =

4.3333

>> 3\13

ans

4.3333

>> 4^11

ans

4194304

>> 2*pi^3

ans

62.01255336059963

a = [0 1 2 3 4 5 6 7 8 9 10]

a=

0 1 2 3 4 5 6 7 8 9 10

b= a + 3

b =

3 4 5 6 7 8 9 10 11 12 13

t = 0:2:20

t =

0 2 4 6 8 10 12 14 16 18 20

c= a+b

c =

3 5 7 9 11 13 15 17 19 21 23

d = [1; 3; 5]
```

```
d =

1

3

5

d'

ans =

1 3 5

f = [4; 6; 9]

f =

4

6

9

>> d*f

??? Error using ===> *

Inner matrix dimensions must agree.

>> d.*f

ans =

4

18

45

>> d * f'

ans =

4 6 9

12 18 27

20 30 45

>>d.*f
```

??? Error using ==> .*

Matrix dimensions must agree.

>> d*4

ans =

4

12

20

>>f.*4

ans =

16

24

36

>>a/7

ans =

0.1429

0.4286

0.7143

>> a./7

ans =

0.1429

0.4286

0.7143

>> d^f

??? Error using ==> ^

Matrix dimensions must agree.

>> d.^f

ans =

1

729

1953125

>> d ^2

??? Error using ==> ^

Matrix must be square.

>> d.^2

ans =

1

9

25

>> 3 d

??? Error using ==> ^

Matrix must be square.

>>3.^d

ans =

3

17

243

Para entrar la matriz

- 2
- 4

y lo guardamos en una variable a,

>> a = [1 2; 3 4]

Para redisplay la matriz, simplemente teclee su nombre:

>> a

Primero elevemos al cuadrado la matriz a:

>> a*a

Ahora nosotros probaremos algo un poco más dificultoso. Primero definimos una matriz b:

>> b = [1 2; 0 1]

Entonces nosotros computamos el producto a*b:

>> a*b

Finalmente, efectuamos el producto en el otro orden:

>> b*a

Sabemos que los dos productos son diferentes: porque la multiplicación de matrices es no conmutativa.

Por supuesto, también podemos sumar matrices:

>> a + b

Ahora guardemos el resultado de esta suma para que nosotros podamos usarlo después:

>> s = a + b

Las matrices a veces puede invertirse:

>> inv(s)

Para verificar que esto es correcto, nosotros computamos el producto de s y su inverso:

>> s * inv(s)

El resultado es la unidad, o matriz de identidad. También podemos escribir el cómputo como

>> s/s

también podemos escribir

>> s\s

qué es igual que

>>inv(s) * s

Para ver que estas operaciones son correctas y/o diferentes, nosotros hacemos lo siguiente:

>> a/b

>> a\b

No todas las matrices pueden invertirse, o puede usarse como el denominador en la división de matrices:

>> c = [1 1; 1 1]

>> inv(c);

Una matriz puede invertirse si y sólo si su determinante es diferente de cero:

>> det(a)

>> det (c)

PRIMERA PARTE

Uso básico de MATLAB

Normalmente se requiere de modelos computacionales con el fin de resolver problemas de ingeniería. Muchas veces puede ser útil hacer un programa que utilice matrices, complejos, y otras estructuras matemáticas, pero fácil de escribir y revisar. MATLAB es ideal para esto. Ya que es una herramienta tan útil y poderosa, resolvimos dar una idea general sobre su manejo, con el fin de facilitar su uso.

Esta presentación está organizada de la siguiente forma:

- Generalidades.
- Comandos de programación.
- Comandos matemáticos.
- Programas de ejemplo variados.

Cada uno de los vínculos de estas secciones, contiene una explicación breve y ejemplos pequeños de cada comando. La sección de ejemplos, contiene algunos programas completos, donde se utilizan los comandos tratados.

NOTA:

En todos los programas de ejemplo se utiliza el comando de MATLAB: % el cual se utiliza para añadir un comentario en el programa. Estos comentarios son importantes para que otros puedan entender el contenido con mayor facilidad.

1. Generalidades

Esta es una breve introducción al manejo de variables (escrita para las personas que nunca han usado MATLAB), expresiones y archivos con extensión .m (programas ejecutables por MATLAB), con respecto a su creación y uso.

La idea es tratar de manera general, como es el uso de variables, expresiones y comandos en MATLAB, así como sus características. Adicionalmente dar una introducción al uso de los archivos con extensión .m (programas ejecutables por MATLAB) y como trabajar con ellos.

1.1. Manejo de variables:

En MATLAB como en cualquier otro lenguaje de programación, y/o asistente matemático se utilizan variables. Las variables deben tener un nombre según ciertas reglas. Estas reglas son:

NO pueden comenzar con un número, aunque si pueden tener números (variable1 es un nombre válido).

Las mayúsculas y minúsculas se diferencian en los nombres de variables. (A y a son dos variables diferentes)

Los nombres de variables no pueden contener operadores ni puntos. (No es válido usar /, *, −, +, ...)

Si se trabaja con complejos sólo puede utilizarse un de los nombres i y/o j para variables. Ver complejos.

No es necesario definir el tipo de variable o tamaño (si se usa un vector y despues se expande, no hay problema)

1.2. Manejo de expresiones:

Una expresión en MATLAB, puede ser:

- Una variable o un número. (ej: variable1, x, 3, 22.3)
- Un comando aplicado. (ej: norm(A), sin(2*pi))
- Una expresión matemática. (ej: 2+3*variab1^ 4.5)

Si cualquiera de las anteriores se escribe en la línea de comandos (>>) del MATLAB, él devolverá el nombre de la variable y su valor (en caso de que la expresión tenga nombre, de no tenerlo, MATLAB devolverá ans = resultado). Un punto importante que se debe resaltar es que esto ocurre siempre y cuando la expresión no termine con punto y coma. Al añadir un punto y coma al final de la expresión MATLAB no imprime su valor en la pantalla, aunque si realiza el cálculo. (a=3+2; deja en a el valor de 5, pero no lo muestra).

1.3. Manejo de comandos:

Cada comando en MATLAB es un archivo con extensión .m, por lo tanto es necesario tener las librerías en que se encuentran los comandos que se desean utilizar. Aunque la gran mayoría de los comandos utilizados siempre vienen incluidos en las librerías.

MATLAB NO distingue entre mayúsculas y minúsculas en los comandos (a menos que se trabaje en Unix) . El resto de esta presentación trata cada comando en detalle (los más usados).

1.4. Manejo de archivos con extensión .m:

Todos los comandos a que se refiere esta presentación pueden utilizarse directamente desde la línea de comandos del MATLAB (>>). Sin embargo la idea es hacer un archivo (con extensión .m) que contenga el programa (para poder modificarlo, revisarlo, correrlo otra vez ....) ya que es más ventajoso así. Los programas no requieren indentación como en los ejemplos que he puesto aquí, sin embargo es recomendable hacerlo por claridad al intentar modificar el programa o revisarlo.

Para trabajar estos archivos, es necesario saber:

- Que es: Es un archivo de texto como cualquier otro donde se encuentra el listado del programa. (sólo que su extensión no es txt sino m)

- Como crear uno: Las formas más fáciles son:

- Desde Unix: con el comando !pico archivo.m donde archivo es el nombre del programa.
- Desde Windows: con el NOTEPAD, teniendo la precaución de cambiar el tipo de archivo a Todos los archivos (*.*) antes de grabarlo. (de lo contrario el archivo quedará con nombre archivo.m.txt y el MATLAB no podrá correrlo, la solución es quitar el .txt.).

- Como correrlo para obtener los resultados: Desde la línea de comandos de MATLAB se escribe el nombre del archivo (sin el .m)

NOTAS:

El archivo debe quedar grabado en el mismo directorio que MATLAB para poder correrlo. Y si el archivo fue escrito en Unix la extensión tiene que ser escrita en minúscula (m), y debe escribirse el nombre exactamente igual para correrlo (Unix diferencia entre mayúsculas y minúsculas)

2. Comandos básicos de programación

Para la estructura de programación en MATLAB se requiere conocer por lo menos los siguientes comandos:

2.1. Comando END

Determina hasta cual orden llega el efecto de if, for, y while. (Para ejemplos de su uso ver if, while y for)

2.2. Comando IF

Verifica si se cumple cierta condición, y de acuerdo a si se cumple o no realiza la acción que se desee.

La sintaxis de la orden es:

if (condición), (ordenes 1) [else, (ordenes 2)] end;

Donde las ordenes entre [ ] son opcionales.

(ordenes 1) son las ordenes que se realizarán si (condición) se cumple.

(ordenes 2) son las ordenes que se realizarán si (condición) NO se cumple.

(condición) Puede ser:

a == b (verifica si a es igual a b)

a < b (verifica si a es menor que b)

a > b (verifica si a es mayor que b)

a <= b (verifica si a es menor o igual que b)

a >= b (verifica si a es menor o igual que b)

a ~= b (verifica que a y b sean diferentes)

El siguiente ejemplo ilustra el uso de if:

```
%Ejemplo de uso de if.

n=0;

if n==0,

n % al escribir una expresión sin punto y coma final, MATLAB escribe

% su resultado en pantalla.

else,

n = 1

end;

n = 2;

if n == 0,

n

else,

n =1

end;
```

La salida que se obtiene con el programa anterior es la siguiente:

```
n =

0

n =

1
```

Donde el 0 (cero) proviene de entrar al primer if, y el 1 (uno), de entrar al else del segundo if.

2.3. Comando WHILE

Realiza una parte del programa mientras se cumpla alguna condición.

La sintaxis de la orden es:

while (condición), (ordenes) end;

(ordenes) son las ordenes que se realizarán mientras (condición) se cumpla.

(condición) Puede ser:

a == b (verifica si a es igual a b)

a < b (verifica que si a es menor que b)

a > b (verifica que si a es mayor que b)

a <= b (verifica si a es menor o igual que b)

a >= b (verifica que si a es mayor o igual que b)

a ~= b (verifica que a y b sean diferentes)

El siguiente ejemplo ilustra el uso de while:

```
%Ejemplo de uso de while.

n=0;

while n<=5,

n %Al escribir el nombre de la variable (sin punto y coma) MATLAB % imprime su valor.

n = n + 1; %El punto y coma evita que MATLAB imprima el nuevo valor de n.

end;
```

La salida que se obtiene al correr el programa anterior es:

```
n =

0

n =

1

n =

2

n =

3

n =

4

n =
```

5

2.4. Comando FOR

Muy parecido al While, pero utiliza un contador, es útil si se quiere repetir una parte del programa un número. determinado de veces.

La sintaxis de la orden es:

for (contador), (ordenes) end;

(ordenes) son las ordenes que se realizarán (contador) llega a su valor final.
(contador) Es de la forma:

variable = a [, b] : c

Donde:

- *variable* es el contador en sí.
- *a* es el valor inicial del contador (variable).
- *b* es el segundo valor del contador (opcional, si se omite, b=a+1), su función es determinar el incremento del contador.
- *c* es el valor final del contador (variable).

El siguiente ejemplo ilustra el uso de for:

% Ejemplo de uso de for.

for i=0,0.5:2.5,

i %al escribir el nombre de una variable (sin punto y coma)

%MATLAB muestra su valor.

end;

La salida del programa anterior es la siguiente:

i =

0

i =

0.5

i =

1

i =

1.5

i =

2

i =

2.5

2.5. Comando PLOT

Sirve para obtener resultados gráficos en 2D.

La sintaxis de la orden es:

plot(x, y);
x es el vector que contiene los valores de x.
y es el vector que contiene los valores de y,

tal que el valor de **y** en la posición uno del vector corresponde al primer valor del vector **x**. La gráfica se realiza uniendo una serie de rectas entre los puntos incluidos en los vectores X y Y. Si las curvas quedan muy mal hechas (se notan las rectas) puede ser necesario disminuir el paso de los vectores (y aumentar el número de puntos.

Para claridad, puede ser necesario leer la parte correspondiente a vectores a la orden FOR.

El siguiente ejemplo ilustra el uso de plot:

% Ejemplo de uso de plot.

for i =1:101,

x(i) = (i−1) /100;

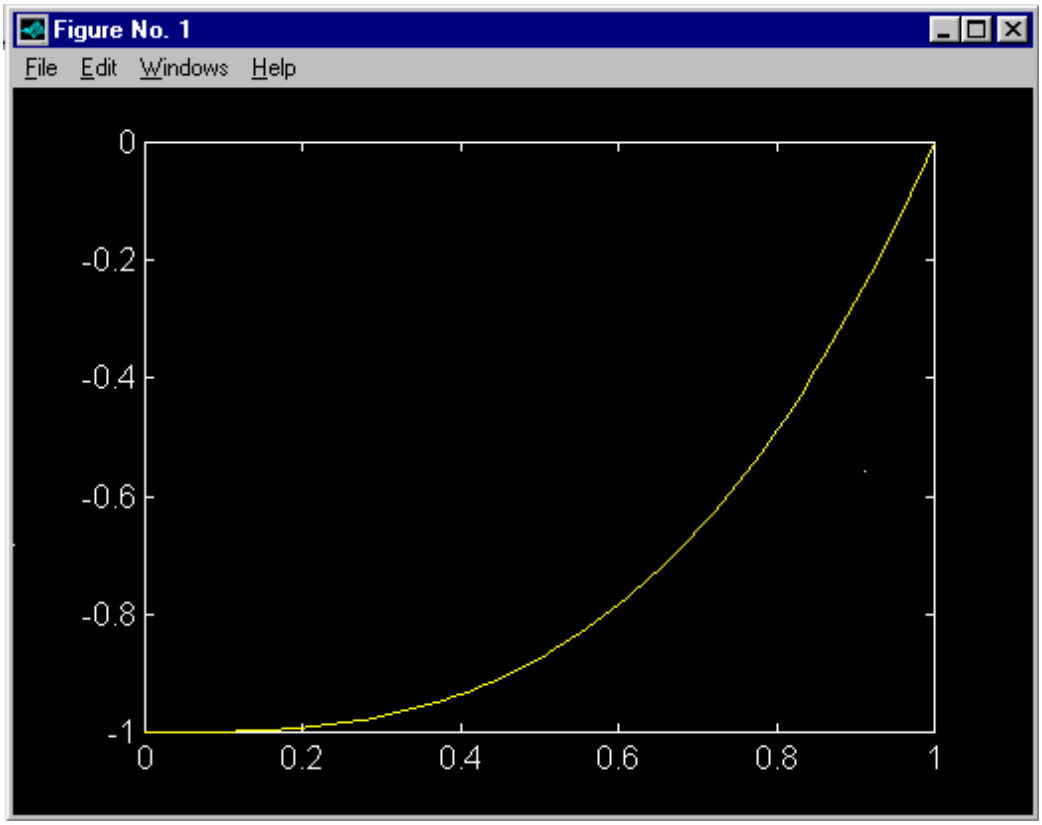
y(i)=x(i) − 1; % Organiza en vectores la función y=x^3 − 1

end;

plot(x, y);

pause; %pausa el computador hasta que se presione una teclaesta orden es necesaria cuando se hace más de una gráfica, para poder ver cada una por separado. Ya que MATLAB las dibuja en la misma ventana siempre. (a menos que se use el comando FIGURE).

Al correr el programa se obtiene la gráfica de la curva y=x3 −1 (para 0<=x<=1). La gráfica aparecerá en una ventana aparte llamada Figure 1, y la recta se verá asi:



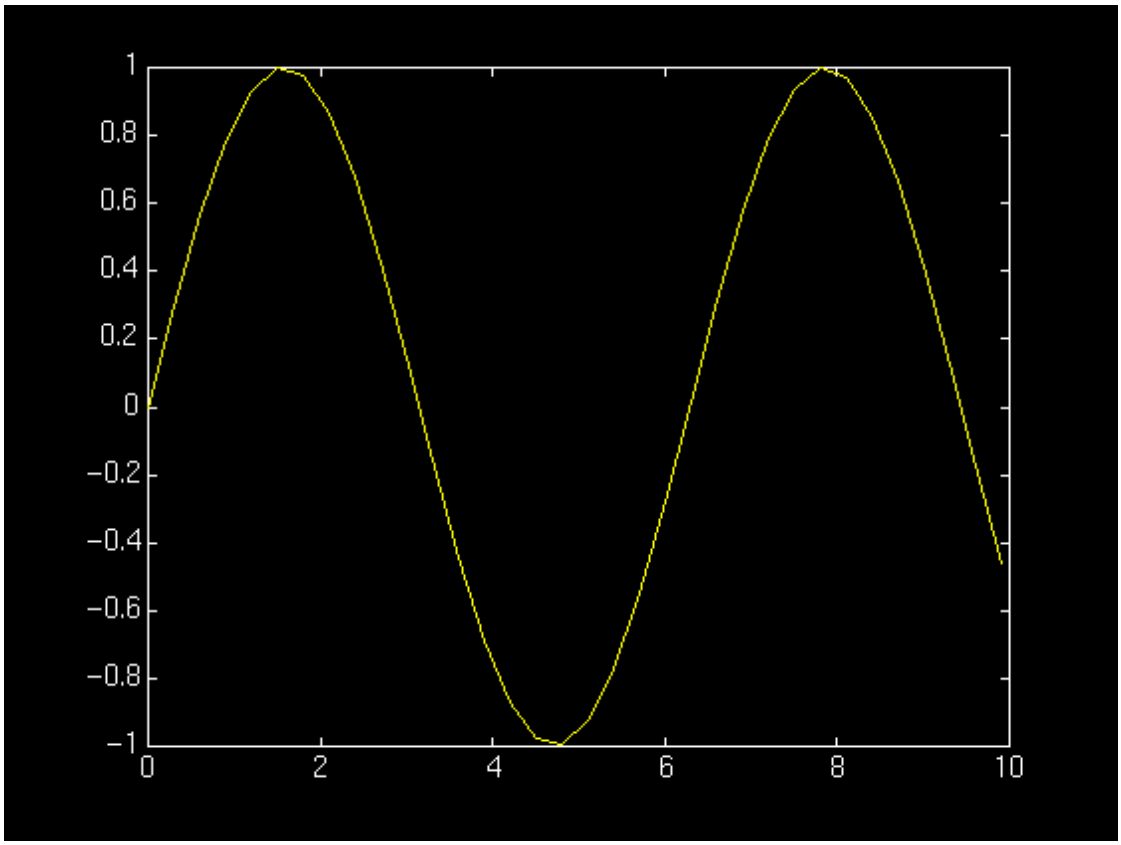
Otro ejemplo, para la gráfica de y = sin(t) en el intervalo t = 0 a t = 10; donde debemos hacer lo siguiente:

>> t = 0:.3:10;

>> y = sin(t);

>> plot(t , y)

Aquí el resultado gráfico:



El comando t = 0: .3 :10; define el vector entre los componentes desde 0 a 10 incrementados de 0.3. La expresión y = sin(t); va a definir los valores de los componentes que son: sin(0), sin(0.3), sin(0.6), etc. Finalmente, plot(t,y) usa el vector de t y los valores de y para la construcción del gráfico.

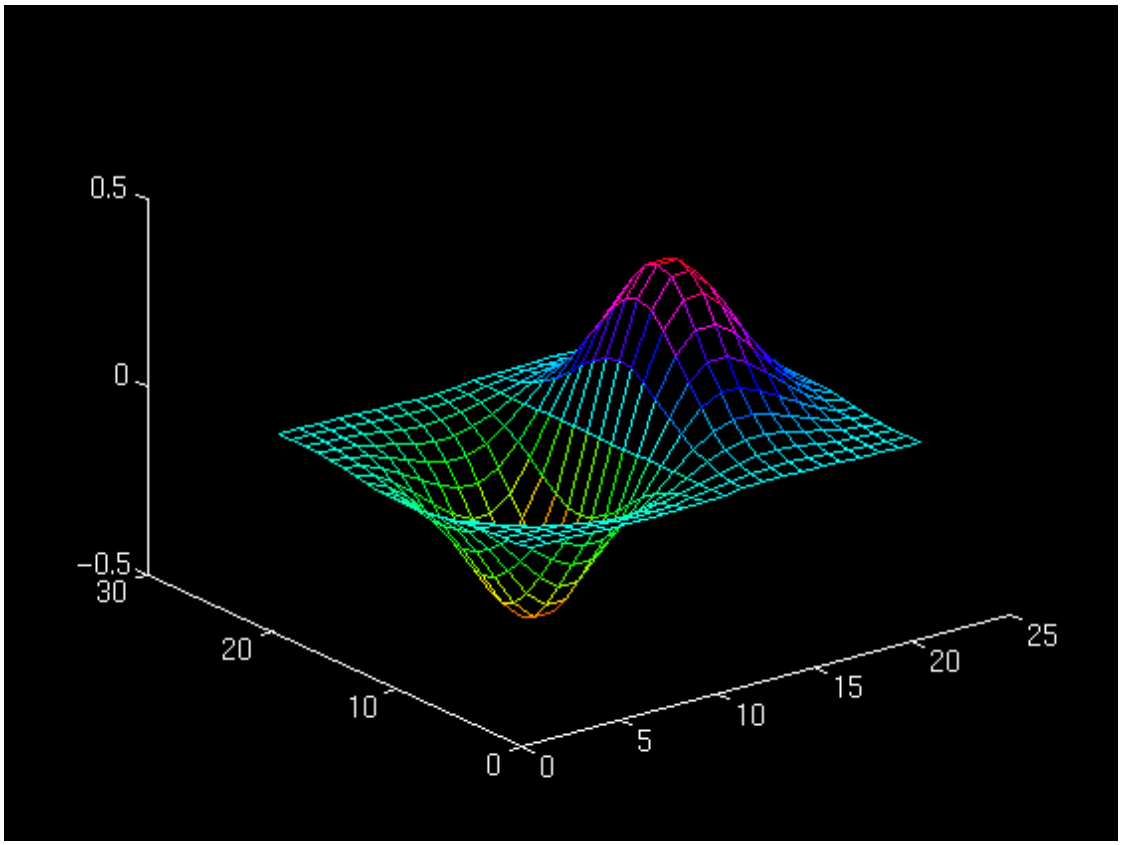
2.6. Funciones de dos variables

Vamos a elaborar la gráfica de la función z(x,y) = x exp(− x^2 − y^2):

>> [x,y] = meshdom(−2:.2:2, −2:.2:2);

>> z = x .* exp(−x.^2 − y.^2);

>> mesh(z)



El primer comando crea una matriz para hacer la entrada de los puntos en el espacio cuadrado de −2 <= x <= 2, −2 <= y <= 2. Los cuadrados pequeños que constituyen la reja son de 0.2 unidades ancho y 0.2 unidades de alto. La segunda orden crea una matriz cuyas entradas son los valores de la función z(x,y) a los puntos de la reja. La tercera orden usa esta información para construir el gráfico.

2.7. Comando DISP

Sirve para escribir texto de salida o vectores. de resultados.

La sintaxis de la orden es:

disp(X);

X Puede ser:

- Un vector.
- Una matriz.
- Una cadena de texto.

El siguiente ejemplo ilustra el uso de disp:

%Ejemplo de uso de disp.

a = [1, 2, 3, 4, 9 11]; % Un vector

disp(a);

a = [1, 2 , 7 ; 6, 3, 4]; % Una matriz

```
disp(a);

a = `Texto se puede escribir así `; % Cadena de texto

disp(a);

disp( ` También se puede escribir así.' );
```

La salida del programa anterior será:

```
1 2 3 4 9 11

1 2 7

6 3 4

Texto se puede escribir así

También se puede escribir así.
```

2.8. Comando INPUT

Se utiliza para que el programa pida valores de variables mientras se ejecuta.

La sintaxis de la orden es:
variable = input (texto);

variable es un nombre válido de variable, en la que se quiere almacenar el valor que se pregunta.
texto puede ser:

- Una variable o,
- Una cadena.

El siguiente ejemplo ilustra el uso de input:

```
%Ejemplo de uso de input.

a = 0; % hace válido el nombre de variable a.

a = input( ` Teclee el valor de a: `);

tex = ` Cual es el nuevo valor de a? `;

a % Al escribir el nombre de una variable (sin punto y coma al final)

% MATLAB muestra su valor.

a = input(tex);

a
```

La salida de este programa será:

Teclee el valor de a: (espera)

a =

xxx % aquí se imprime el valor asignado para *a*.

Cual es el nuevo valor de a? (espera)

a =

yyy

Donde xxx y yyy son valores introducidos por el usuario en el momento de correr el programa.

SEGUNDA PARTE

Comandos básicos matemáticos

Lo que hace verdaderamente poderoso al MATLAB es la facilidad para realizar operaciones matemáticas con elementos como: (en cada vínculo se encuentra las ordenes y sintaxis para cada tipo de elemento).

- **Vectores y Matrices**

Los vectores y matrices en MATLAB se trabajan igual en cuanto a asignación, por eso se explican juntos. Pero las operaciones posibles, si son diferentes, y están separadas bajo los encabezados correspondientes.

Asignación:

La asignación de variables en MATLAB es sencilla, y los vectores y matrices no son la excepción. Cuando se desea dar el valor a toda una matriz se puede realizar directamente de la siguiente forma:

```
A = [1 2 3 4 ; 5 6 7 8; 9 0 1 2]; ó
A = [1, 2, 3, 4;5, 6, 7, 8;9, 0, 1, 2];
```

donde la matriz escrita arriba es:

1 2 3 4

5 6 7 8

9 0 1 2

Las filas se separan por punto y coma y las columnas por espacios o comas. De lo anterior se ve fácilmente que un vector fila se asigna así:

```
v = [1 2 3]; ó
v = [1, 2, 3];
```

y un vector columna se asigna así:

```
v = [1; 2; 3];
```

Manejo de subíndices:

Otra forma de asignar valores a una matriz (o un vector) es por medio de los subíndices. *El menor subíndice utilizado por MATLAB es 1*. Y va añadiendo valores a medida que se requieran. Los subíndices se escriben entre paréntesis. Por ejemplo:

A(2, 3) =1; Asigna al elemento en la fila 2, columna 3 el valor de 1.

Si se desea cambiar todo el valor de una fila o una columna, es muy sencillo hacerlo con el operador : así:

A(1 , :) = [4 5 6];

Asigna a la fila 1 el vector [4, 5, 6] (cambia la fila 1 por 4, 5, 6). Así si A era una matriz de 3 x 3 de ceros, ahora queda:

4 5 6

0 0 0

0 0 0

Igualmente a veces se requiere trabajar con vectores que son una columna o una fila de una matriz. Esto se realiza fácilmente guardando este vector en un vector , así:
v = A(:,1);
Asigna al vector v la primera columna (completa) de la matriz A.

1.1. Operaciones matemáticas simples con matrices y vectores:

Esto es algo en lo que MATLAB hace las cosas verdaderamente simples, si se tienen dos matrices (o vector y matriz, o dos vectores), y se quieren: sumar, multiplicar ó restar sólo es necesario anotar esta operación normalmente (como se haría con números). Por ejemplo:

Si se quieren multiplicar dos matrices A y B y almacenar el resultado en C:
C = A * B; (Si se hace entre dos vectores (uno fila y el otro columna) el resultado es el producto punto entre los dos)

Si se quieren sumar ó restar y almacenar el resultado en C:
C = A + B;
ó
C = A − B; (Sin importar que sean matrices o vectores.)

1.2. Comandos matemáticos para vectores:

Los comandos matemáticos más empleados con vectores son:

1.2.1. <u>Comando NORM</u>

Calcula la norma de un vector o matriz.

La sintaxis de la orden es:

Norma = norm(Matriz [, Tipo]);

Los signos [ ] son para decir que Tipo es opcional.

Matriz es la matriz o vector al que se desea calcular la norma.

Tipo es el tipo de norma que se desea calcular. Tipo puede ser una de las siguientes:

Si se omite: calcula la norma 2

en un vector es la magnitud del vector

2: calcula la norma 2

inf: calcula la norma infinito: en un vector es el

máximo valor absoluto, en una matriz es la

suma más grande de las filas.

En Norma se almacena el valor de la norma calculada.

El siguiente ejemplo ilustra el uso de norm: (ver orden de programación DISP)

```
%Ejemplo de uso de norm.
A = [1 2; 3 4]
v = [1 2 3 4]
disp( `Para la matriz:' );
n2 = norm(A)
ni = norm(A, inf)
disp( `Para el vector:' );
n2 =norm(v)
ni = norm(v, inf)
% Al escribir una expresión sin punto y coma al final
% MATLAB muestra su valor en pantalla.
```

Al correr el programa se obtienen como salida los siguientes resultados:

```
A =
1 2
3 4
```

v =

1 2 3 4

Para la matriz:

n2 =

5.4650

ni =

7

Para el vector:

n2 =

5.4772

ni =

4

1.2.2. Comando MIN

Retorna el (los) menor (es) componente (s) de un vector o matriz. Para el caso de los vectores: retorna el menor valor contenido en sus componentes. En el caso de una matriz MIN retorna un vector (fila) que contiene el mínimo elemento que se encontró en cada una de las columnas (la primera componente del vector tiene el menor elemento en la primera columna de la matriz, y así sucesivamente).

La sintaxis de la orden es:

Mínimo = min(matriz x **);**

Matriz es la matriz o vector al que se desea encontrar la (s) mínima (s) componente (s).
En Mínimo se retorna (n) el (los) mínimo (s) valor (es) encontrado (s) en la matriz o vector.

El siguiente ejemplo ilustra el uso de min:

%Ejemplo de uso de min.

A=[1 2; 3 4]

v=[1 2 3 4]

M=min(A)

m=min(v) % MATLAB diferencia entre máyusculas y mínusculas.

% Al escribir una expresión sin punto y coma al final

% MATLAB muestra su valor en pantalla.

Al correr el programa anterior se obtiene como salida lo siguiente:

A =

1 2

3 4

v =

1 2 3 4

M =

1 2

m =

1

1.2.3. Comando MAX

Retorna el (los) mayor (es) componente (s) de un vector o matriz. Para el caso de los vectores: retorna el mayor valor contenido en sus componentes. En el caso de una matriz MAX retorna un vector (fila) que contiene el máximo elemento que se encontró en cada una de las columnas (la primera componente del vector tiene el mayor elemento en la primera columna de la matriz, y así sucesivamente).

La sintaxis de la orden es:

Máximo = max(Matriz);

Matriz es la matriz o vector al que se desea encontrar la (s) máxima (s) componente (s).

En Máximo se retorna (n) el (los) máximo (s) valor (es) encontrado (s) en la matriz o vector.

El siguiente ejemplo ilustra el uso de max:

%Ejemplo de uso de max.

A = [1 2; 3 4]

v = [1 2 3 4]

M = max(A)

m = max(v) % MATLAB diferencia entre mayúsculas y minúsculas.

Al correr el programa anterior se obtiene como salida lo siguiente:

A =

1 2

3 4

v =

1 2 3 4

M =

3 4

m =

4

1.2.4. Comando CROSS

Calcula el producto cruz entre dos vectores.

La sintaxis de la orden es:

Vector1 = cross(Vector2, Vector 3);

Vector2 y Vector3 son los vectores a los que se les quiere aplicar el producto cruz. Tanto Vector2 como Vector3 deben ser vectores tridimensionales.
Vector1 es el vector (tridimensional) resultante del producto cruz de Vector2 y Vector3.

El siguiente ejemplo ilustra el uso de cross:

%Ejemplo de uso de cross.

x = [1 0 0]

y = [0 1 0]

z = cross(x, y)

Al correr el programa se obtiene la siguiente salida:

x =

1 0 0

y =

0 1 0

z =

0 0 1

1.2.5. Comando LENGTH

Determina el número de componentes de un vector. La sintaxis de la orden es:

Longitud = length (Vector);

Vector es el vector que se quiere medir (número de componentes).
Longitud es el número de componentes de Vector.

El siguiente ejemplo ilustra el uso de lenght:

%Ejemplo de uso de length.

x = [1 2 3 4 5 6 7]

l = length(x)

Al correr el programa se obtiene la siguiente salida:

x =

1 2 3 4 5 6 7

l =

7

1.3. Comandos matemáticos para matrices:

Los comandos matemáticos más empleados con matrices son:

1.3.1. Comando NORM

Calcula la norma de un vector o matriz.

1.3.2. Comando MIN

Retorna el (los) menor (es) componente (s) de un vector o matriz.

1.3.3. Comando MAX

Retorna el (los) mayor (es) componente (s) de un vector o matriz.

1.3.4. Comando SIZE

Devuelve el tamaño de la matriz (dimensiones).

La sintaxis de la orden es:

[Filas, Columnas] = size(Matriz);

(Los símbolos [ ] se escriben.)

ó también:

Tamaño = size(Matriz);

Matriz es la matriz a la que se le desea determinar el tamaño (dimensiones).
En Filas se almacena el número de filas.
En Columnas se almacena el número de columnas.
Tamaño es un vector (fila) en cuyas componentes se almacenan el número de filas y de columnas, siempre en ese orden.

El siguiente ejemplo ilustra el uso de size:

%Ejemplo de uso de size.

A= [1 2 3; 4 5 6]

y = size(A)

[f, c] = size(A);

f % Al escribir una expresión sin punto y coma final MATLAB

c % muestra el valor por pantalla

Al correr el programa se obtiene la siguiente salida:

A =

1 2 3

4 5 6

y =

2 3

f =

2

c =

3

1.3.5. Comando EIG

Calcula los valores y vectores propios (ortovalores y ortovectores) de la matriz.

La sintaxis de la orden es:

[Vectores, Diagonal] = eig(Matriz);

(Los símbolos [ ] se escriben.)
ó también:

Valores = eig(Matriz);

Matriz es la matriz (cuadrada) a la que se le desea calcular los valores o vectores propios.
Diagonal es una matriz diagonal que contiene los valores propios de Matriz.
Vectores es una matriz en la que se devuelven los vectores propios (unitarios) donde cada columna de la matriz Vector es un vector propio de matriz; tal que el primer vector corresponde al primer valor propio y así sucesivamente.
Valores es un vector columna que contiene los valores propios de Matriz.

El siguiente ejemplo ilustra el uso de eig:

```
%Ejemplo de uso de eig.

A = [1 2; 3 4]

y = eig(A)

[V, D] = eig(A);

V %Al escribir una expresión sin punto y coma final MATLAB

% muestra el valor por pantalla

D
```

Al correr el programa se obtiene la siguiente salida:

```
A =

1 2

3 4

y =

−0.3723

5.3723

V =

−0.8246 −0.4160

0.5658 −0.9094

D =

−0.3723 0
```

0 5.3723

1.3.6. Comando INV

Sirve para invertir una matriz.

La sintaxis de la orden es:

matriz1 = inv(matriz2);

matriz2 es la matriz que se desea invertir
En matriz1 se almacena la matriz inversa de matriz 2.

El siguiente ejemplo ilustra el uso de inv:

%Ejemplo de uso de inv.

A = [1 2; 3 4]

I = inv(A);

I % Al escribir una expresión sin punto y coma al final

% MATLAB muestra su valor en pantalla.

Al correr el programa se obtiene como salida la matriz que se desea invertir (A), y su inversa (I). La salida se ve así:

A =

1 2

3 4

I =

−2.0000 1.0000

1.5000 −0.5000

1.3.7. Comando DET

Sirve para calcular el determinante de una matriz.

La sintaxis de la orden es:

Valor = det (Matriz)

Matriz es la matriz (cuadrada) a la que se le desea calcular el determinante.

Valor es donde se almacena el valor del determinante.

El siguiente ejemplo ilustra el uso de det:

```
% Ejemplo de uso de det

A = [1 2 7; 4 5 8; 6 −7 10]

d = det(A)
```

Al correr el programa se obtiene la siguiente salida:

```
A =

1 2 7

4 5 8

6 −7 10

d =

−284
```

TERCERA PARTE

APLICACIONES BASICAS DE MATLAB

1. Modelaje de Sistemas Lineales

Ahora consideremos el sistema de ecuaciones lineales

ax + by = p

cx + dy = q

Nosotros podemos escribir esto más sólidamente como

AX = B

donde la matriz A de los coeficiente es:

a b

c d

el vector X de variables desconocidas es

x

y

el vector B en el lado derecho es

p

q

Si A es invertible, X = (1/A)B, o, usando anotación de MATLAB, X = A\B. Se prueba esto resolviendo AX = B.

Hagamos un poco de programación. Sea A sea la matriz

0.8 0.1

0.2 0.9

y sea X el vector de la columna

1

0

Consideramos X para representar (por ejemplo) el estado de la población de una isla. La primera entrada (1) da el fragmento de la población en la mitad oriental de la isla, la segunda entrada (0) dé el fragmento en la otra mitad oriental. El estado de la población en las unidades de tiempo T son dadas después por la regla Y = AX. Esto expresa el hecho de que un individuo en medio de las estancias orientales puestas con probabilidad 0.8 y que se mueve del este con probabilidad 0.2 (notar que 0.8 + 0.2 = 1), y el hecho que un individuo en las estancias orientales puestas con probabilidad 0.9 y oeste de los movimientos esté con probabilidad 0.1. Así, los estados de la población sucesivos pueden ser predicho/computado por la multiplicación de la matriz repetida. Esto puede ser realizado por el programa de MATLAB siguiente:

>> A = [0.8 0.1; 0.2 0.9]

>> x = [1; 0]

>> for i = 1:20, x = a*x, end,

Hasta aquí hemos aprendido a escribir un tipo de loop simple. Ésta es una manera fácil de ordenar a la máquina, en otras palabras, hacer un trabajo muy repetitivo.

1.1. Definiendo Matrices

Si queremos definir la siguiente matriz en MATLAB:

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{pmatrix}$$

entonces escribimos:

»A=[1 2 3 4;5 6 7 8;9 10 11 12;13,14,15,16];

»x=4:−1:1

genera el vector *fila* x=[4,3,2,1]. La instrucción

»C=A(3:4,1:3);

se refiere a la submatriz

$$C = \begin{pmatrix} 9 & 10 & 11 \\ 13 & 14 & 15 \end{pmatrix}$$

de A. También D=A([1,3],3:4) genera

$$D = \begin{pmatrix} 3 & 4 \\ 11 & 12 \end{pmatrix}$$

1.2. Matrices Especiales

En MATLAB podemos generar *matrices especiales* con las siguientes instrucciones:

rand(n,m) − matriz n x m de entradas aleatorias entre 0 y uno.
eye(n) − matriz identidad n x n.
zeros(n,m) − matriz cero de tamaño n x m.
ones(n,m) − matriz n x m con todas las entradas uno.

Combinando estas instrucciones podemos generar matrices bastante complicadas. Por ejemplo, la instrucción

»E=[eye(2),ones(2,3);zeros(2),[1:3;3:−1:1]]

genera la matriz

$$E = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 2 & 3 \\ 0 & 0 & 3 & 2 & 1 \end{pmatrix}$$

La instrucción round(x) redondea x al entero más cercano a x. Podemos combinar funciones en MATLAB. Por ejemplo, round(10*rand(4)) genera una matriz con entradas aleatorias entre 0 y 10.

1.3. Aritmética de Matrices

Considere las siguientes matrices:

$$A = \begin{pmatrix} 4 & 5 \\ 2 & 3 \end{pmatrix} \quad , \quad B = \begin{pmatrix} -1 & 1 & 2 \\ 4 & 3 & 1 \end{pmatrix} \quad , \quad C = \begin{pmatrix} -1 & 2 \\ 2 & 4 \end{pmatrix}$$

Entonces las operaciones A*B (producto matricial de A con B), A+B (suma de A mas B), 3*A (multiplicación escalar de 3 por A) tienen los siguientes resultados:

»A*B

ans =

16 19 13
10 11 7

»A+B

??? Error using ==> +
Matrix dimensions must agree.

»3*A

ans =

12 15
 6 9

Note que MATLAB anuncia que A+B no se puede calcular. Las operaciones A' (transpuesto de A), inv(A) (inversa de A), y A^3 (esto es A*A*A) tienen como resultados:

»A'

ans =

4 2

5 3

»inv(A)

ans =

1.5000 −2.5000

−1.0000 2.0000

»A^3

ans =

174 235
 94 127

Si precedemos las operaciones matriciales *, ^ con el punto ., entonces estas se hacen termino a termino. Por ejemplo A.*C y A.^2 generan:

» A.*C

ans =

```
−4 10
 4 12

» A.^2

ans =

16 25
 4  9
```

2. Solución de Sistemas Lineales

Considere el sistema lineal

$$\begin{cases} x_1 - 2x_2 + 3x_3 = 1 \\ 4x_1 + x_2 - 2x_3 = -1 \\ 2x_1 - x_2 + 4x_3 = 2 \end{cases}$$

Definimos la matriz de coeficientes y el lado derecho por las instrucciones:

```
»A=[1 −2 3; 4 1 −2; 2 −1 4];
»b=[1 −1 2]';
```

Note que la transpuesta en b se usa para hacerlo un vector columna. Vamos a resolver este sistema por tres métodos:

- eliminación Gaussiana
- forma echelon reducida o método de Gauss−Jordan
- método de la inversa

En el método de Gauss−Jordan, luego de obtener la forma echelon de la matriz de coeficientes aumentada, eliminamos también la parte de arriba de la matriz hasta producir una matriz donde las columnas con unos, solo tienen un uno. Esto se conoce como la forma *echelon reducida* (ver texto). Para comparar los tres métodos utilizamos la instrucción flops de MATLAB que estima el número de operaciones de punto flotante entre dos llamadas sucesivas a flops. Una llamada de la forma flops(0) inicializa el contador de operaciones a cero. La sucesión de instrucciones:

```
» flops( 0 )
» x=A\b

x =

−0.0417
 0.4167
 0.6250

» flops
```

lleva a cabo eliminación Gaussiana en el sistema de arriba y produce como resultado:

```
ans =

73
```

entonces, aquí se necesitaron aproximadamente 73 operaciones de punto flotantes (sumas, restas, multiplicaciones ó divisiones) para resolver el sistema con eliminación Gaussiana.

Para el método de Gauss−Jordan tenemos:

```
» flops(0)
» rref([A b])

ans =

1.0000 0 0 −0.0417
0 1.0000 0 0.4167
0 0 1.0000 0.6250

» flops

ans =

483
```

el cual requiere 483 operaciones de punto flotante.

Finalmente el método de la inversa se realiza con la siguiente secuencia de instrucciones:

```
» flops(0)
» x=inv(A)*b

x =

−.0417
 0.4167
 0.6250

» flops

ans =

108
```

el cual toma 108 operaciones.

Vemos pues que la eliminación Gaussiana es el mejor de los tres métodos lo cual es cierto en general.

Usando MATLAB podemos estudiar la relación entre la solubilidad del sistema Ax=b y la no singularidad de la matriz de coeficientes A. En clase vimos que el sistema Ax=b tiene solución única para cualquier lado derecho b si y solo si la matriz A es no singular. ¿Qué sucede si A es singular? ¿Entonces Ax=b no tiene solución? Si A es singular el sistema Ax=b puede tener solución para algunos b's pero de seguro hay al menos un b* para el cual Ax=b* no tiene solución. Vamos a generar una matriz singular con MATLAB:

» A=round(10*rand(6));

» A(:,3)=A(:,1:2)*[4 3]'

A =

2 5 23 9 7 3
0 8 24 8 9 6
7 0 28 5 8 8
7 1 31 1 3 10
9 5 51 7 0 4
4 7 37 4 7 2

(Como usamos la instrucción rand, el resultado de esta y cualquier secuencia de instrucciones que use esta función de MATLAB, no siempre será el mismo). La primera instrucción genera una matriz aleatoria con entradas enteras entre 0 y 10, y con la segunda instrucción remplazamos la tercera columna de A con cuatro veces la primera columna mas tres veces la segunda columna. ¡La matriz resultante es singular! (Explique esto sin calcular el determinante). Generamos ahora un lado derecho arbitrario mediante la instrucción:

» b=round(20*(rand(6,1)−0.5))

b =

10
4
5
3
−9
3

Esto genera una matriz 6x1 aleatoria con entradas enteras entre −10 y 10. Resolvemos el sistema Ax=b calculando la forma echelon reducida de la matriz de coeficientes aumentada [A b]:

» rref([A b])

ans =

1 0 4 0 0 0 0
0 1 3 0 0 0 0
0 0 0 1 0 0 0
0 0 0 0 1 0 0
0 0 0 0 0 1 0
0 0 0 0 0 0 1

Como la última fila es de la forma $\left(0\cdots 0\middle|1\right)$
el sistema es inconsistente, i.e., no tiene solución. ¡Recuerde que A es singular! Esto no quiere decir que Ax=b nunca tenga solución. Si definimos c=A*b, con el b de arriba digamos, el sistema Ax=c tiene solución x=b (¿por qué?). De hecho si calculamos la forma echelon reducida de [A c] tenemos:

» c=A*b;
» rref([A c])

ans =

1 0 4 0 0 0 30
0 1 3 0 0 0 19
0 0 0 1 0 0 3
0 0 0 0 1 0 −9
0 0 0 0 0 1 3
0 0 0 0 0 0 0

el cual denota un sistema consistente dependiente con soluciones:

$$(x_1, x_2, x_3, x_4, x_5, x_6) = (30 - 4x_3, 19 - 3x_3, x_3, 3, -9, 3)$$

donde x3 es arbitrario.

Recordemos que MATLAB posee una gran cantidad de funciones matriciales. De las más comunes que tenemos que repasarlas son:

- min(A), max(A) − dan el mínimo y máximo respectivamente por columnas de A
- sum(A), prod(A) − producen la suma y producto respectivamente por columnas de A
- norm(A,p) − norma p de la matriz A donde p=1,2, ó inf
- eig(A) − vector cuyos componentes son los valores propios de A
- det(A) − el determinante de A

inv(A) − la matriz inversa de A

3. Interpolación Polinomial

La función de interpolación es aquella función que pasa realmente por todos los puntos dados o aquella que mejor ajuste a esos puntos. Cuando se ve gráficamente, la línea pasa por cada uno de los puntos dados.

Para conveniencia seleccionamos un polinomio de grado n para n+1 pares de ordenadas (x,y).

En general el polinomio puede escribirse:

y = a0 + a1x + a2x2 + a3x3 + ... + anxn

Para evaluar la interpolación polinomial de grado `n' para cualquier conjunto de datos, vamos a evaluar los `n+1' coeficientes, a0, a1, a2, a3, ...an.

Dado los `n+1' pares de datos, podemos formas las `n+1' ecuaciones

y1 = a0 + a1x1 + a2x12 + a3x13 + ... + anx1n

y2 = a0 + a1x2 + a2x22 + a3x23 + ... + anx2n

y3 = a0 + a1x3 + a2x32 + a3x33 + ... + anx3n

. . . . . .

. . . . . .

. . . . . .

yn+1 = a0 + a1xn+1 + a2xn+12 + a3xn+13 + ... + anxn+1n

Este es el conjunto soluble de `n+1' ecuaciones desconocidas

Los coeficientes desconocidos son a0, a1,.....an. Si solo existen dos o tres ecuaciones podemos seguir con la Regla de Cramer para hallar estos coeficientes. Pero, en general vamos a recurrir al álgebra de matrices para manipular estas ecuaciones usando un formato de n sustituciones hacia atrás para obtener los coeficientes desconocidos.

Observe que el conjunto de ecuaciones puede expresarse convenientemente por la ecuación matricial:

[y] = [X][a]

donde,

[y] = [X] = [a] =

y1 1 x1 x12.....x1n+1 a1

y2 1 x2 x22.....x2n+1 a2

y3 1 x3 x32.....x3n+1 a3

. . . . . .

. . . . . .

. . . . . .

yn+1 1 xn+1 xn+12... xn+1n+1 an+1

Notamos que el producto de las matrices del lado derecho es una matriz columna de orden `n+1'x 1.

La solución que usa la eliminación de Gauss usamos la división por la izquierda de `MATLAB' o

[a] = [X]\[y]

Usando este método, podemos encontrar los coeficientes de a para el polinomio de grado n que pasa exactamente a través de los `n+1' puntos.

Si tenemos un conjunto grande de datos, cada uno de los datos se aparean incluyendo un error experimental, el polinomio de grado n no es ***una buena opción***.

Los polinomios de grado mayores que cinco o seis tienen a menudo un comportamiento poco realista aunque la curva polinómica pase por cada punto dado. Como ejemplo, consideremos el siguiente conjunto de puntos:

x y

2 4

3 3

4 5

5 4

6 7

7 5

8 7

9 10

10 9

En este conjunto hay nueve puntos. A manera que el valor de x se incrementa, los valores de y también varían. Sin embargo, podemos observar que la distribución de los puntos no es lineal. Con `MATLAB' vamos a crear dos vectores para estos datos:

x9 = [2:1:10];

y9 = [4 3 5 4 7 5 7 10 9];

Para observar estos puntos trazados, ejecutamos:

plot(x9,y9,'o')

Como existe nueve puntos, es posible construir el polinomio de interpolación de grado 8:

y = a0 + a1x + a2x2 + a3x3 + ... + a8x8

Para encontrar los coeficientes desconocidos, definimos el vector columna y

y = y9'

y la matriz X

X = [ones(1,9);x9;x9.^2;x9.^3;x9.^4;x9.^5;x9.^6;x9.^7;x9.^8]'

Notamos que X es definida usando la transpuesta, la función one( ) y el operador de de array ` .^ `. Con X y y así definidos, estos satisfacen la ecuación

[X][a] = [y]

Resolvemos para los coeficientes de la matriz a indicada anteriormente, entrando el comando

a = X\y

dando resultados en

a = 1.0e+003*

3.8140

−6.6204

4.7831

−1.8859

0.4457

−0.0649

0.0057

−0.0003

0.0000

Note que el noveno coeficiente a(8) parece tener el valor cero, realmente su valor es un número diferente de cero, aparece esto debido a que MATLAB está considerando 4 cifras significativas a la izquierda después del punto decimal. Para observar los coeficientes con cifras más significativas, entrar el comando:

format long

a

y el formato se cambia con 15 dígitos significativos. Claramente vemos que a(8) no es cero, es un número pequeño elevado a la potencia 8.

Ahora que ya contamos con los coeficiente, vamos a generar un número suficiente de puntos para crear una curva continua. Para el eje x, formamos un escala en el rango de

2 <= x <= 10 con incrementos de 0.1.

x = [2:.1:10];

Para y, calculamos el valor en el polinomio de octavo grado para cada uno de los componentes de x:

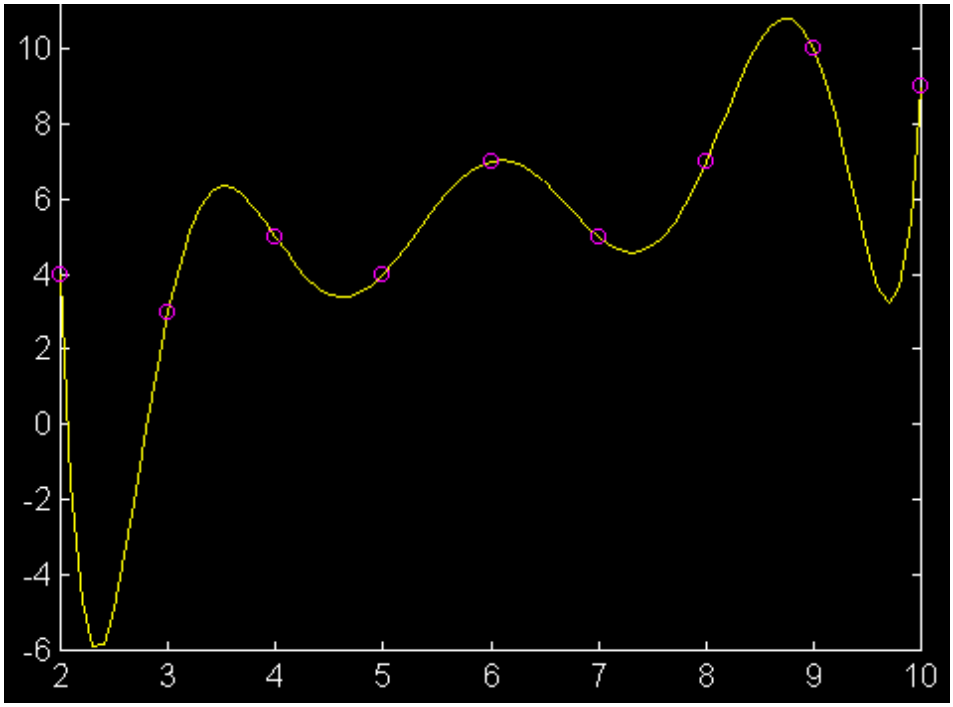
y =a(1)+ a(2).*x + a(3).*x.^2 + a(4).*x.^3 + a(5).*x.^4...

a(6).*x.^5 + a(7).*x.^6 + a(8).*x.^7 + a(9).*x.^8;

Ahora plot(x,y) y los puntos dados (x9,y9).

plot(x,y,x9,y9,'o')

Los resultados polinómicos parecen pasar por cada dato exactamente pero el polinomio no sirve para representar cualquier otro punto en el rango de x.



Por esta razón, no se debe intentar de usar la interpolación polinomial para representar datos experimentales.

3.1. Función MATLAB para los mínimos Cuadrados

Hay en MATLAB una función para que la ecuación encaje con el método de los mínimos cuadrados. El comando es

polyfit(x,y,n)

donde x, y son los vectores con los datos y `n' es el orden del polinomio para el método de los mínimos cuadrados que se desea. El comando `polifit' devuelve un vector cuyos elementos son los coeficientes del polinomio.

Los elementos del vector están en orden inverso, de lo que podemos anticipar que el primer elemento es el coeficiente del orden más alto de x y el último elemento es el coeficiente del orden más bajo (el orden 0 ó x0).

4. Números Reales y Complejos

4.1. Asignación de valores a variables:

Los números complejos se trabajan igual que los reales en lo que se refiere a asignación, a operaciones matemáticas y a comandos. A continuación, unos pocos ejemplos para mostrar como se realiza la asignación:

a = 25.203147;

b = 3;

c = 1 + 2j; ó también:

d = 1.5476 + 2.8*i; (el uso de j ó i es indiferente, desde que se tenga en cuenta la nota importante sobre el uso de las variables i y j)

d = 5.2347;

e = 3j;

4.2. Operaciones matemáticas simples:

Las operaciones simples son las siguientes:

- Suma (operador +)
- Resta (operador −)
- Multiplicación (operador *)
- División (operador /)
- Potenciación (operador ^)

A continuación hay algunos ejemplos para complejos:

a = 3 + 4i;
b = 2 − j;
c = a + b da como resultado:

c =

5.0000 + 3.0000i

d = a ^ b da como resultado:

d =

61.3022 + 15.3369i

Nota importante sobre el uso de las variables i y j:

Puede usarse indistintamente las dos variables incorporadas (i ó j) y MATLAB no pone problema si se usan las dos al tiempo. Pero si se asignan las variables i y/o j en algún lugar del programa, esta variable perderá su valor como raíz de −1. Para clarificar esto es útil un ejemplo:

% Observación para cuando se trabaja con complejos.

i = 8

j = 9

c = 2 + 3*j

La salida del programa anterior es:

i =

8

j =

9

c =

29

Como se puede ver si se intentaba representar un complejo con la variable c, no se logró debido a que se cambiaron las variables i y j. Por lo tanto recomiendo que si se va a trabajar con complejos en un programa: Deje libres las variables i y j (no las utilice en contadores ó en otros propósitos, que no sean representar raiz de −1.

4.3. Comandos matemáticos para números (complejos y reales):

Los comandos matemáticos más empleados con números son:

4.3.1. Comando ABS

Calcula la norma de un complejo, o el valor absoluto de un real.

La sintaxis de la orden es:

Valor = abs(Número);

Valor es la norma del complejo si (Número es complejo) o el valor absoluto de Número (si es real).

Número puede ser un real o un complejo:

Si es Real: calcula el valor absoluto.

Si es Complejo: calcula la norma del complejo.

El siguiente ejemplo ilustra el uso de abs: (ver orden de programación DISP)

```
%Ejemplo de uso de abs.

R = −1.2341

C = 1.5+3j

disp( ` Para un real: ` );

v = abs( R )

disp ( ` Para un complejo: ` );

v = abs( C )
```

Al correr el programa se obtienen como salida los siguientes resultados:

R =

−1.2341

C =

1.5000 + 3.0000i

Para un real:

v =

1.2341

Para un complejo:

v =

3.3541

4.3.2. Comando SQRT

Calcula la raiz cuadrada de un complejo o de un real.

La sintaxis de la orden es:

Valor = sqrt(Número);

En Valor se almacena la raíz cuadrada del número.
Número puede ser un real o un complejo (si es real negativo, el resultado es un complejo)

El siguiente ejemplo ilustra el uso de sqrt:

%Ejemplo de uso de sqrt.

R= − 12.347

raiz = sqrt (R)

C = 2.6 − 7.3j

raiz = sqrt (C)

Al correr el programa se obtienen como salida los siguientes resultados:

R =

12.347

raiz =

0 + 3.5138i

C =

2.6000 − 7.3i

raiz =

2.2748 − 1.6046i

4.3.3. Comando ANGLE

Calcula el ángulo de fase (en radianes) de una matriz (podría querer leer sobre matrices) con elementos complejos. Si la matriz sólo tiene un elemento, calcula el ángulo de fase de ese complejo.

La sintaxis de la orden es:

Valor = angle(Matriz);

Valor es una matriz que almacena el valor del ángulo de fase del complejo (de 0 a 2*pi) que ocupa la misma posición en Matriz (el ángulo de fase del elemento 1,1 lo almacena en la posición 1,1).
Matriz es una matriz (puede tener un solo elemento) cualquiera con componentes complejas (los reales forman parte de los complejos).

El siguiente ejemplo ilustra el uso de angle: (ver orden de programación DISP)

```
%Ejemplo de uso de angle.

C = [1 2i;1+3i 2.3+5i]

c=[1.5+3j]

disp(`Para la matriz:');

v=angle©

disp(`Para un complejo: (matriz de un solo elemento)');

v=angle©
```

Al correr el programa se obtienen como salida los siguientes resultados:

```
C =

2.0000 0 − 2.7000i

3.0000 + 5.0000i 0.7000 − 5.0000i

c =

6.3000 + 7.2300i

Para la matriz:

v =
```

0 −1.5708

1.0304 − 1.4317

Para un complejo: (matriz de un solo elemento)

v =

0.8540

5. Integrales Definidas

Se solucionan numéricamente por medio del comando TRAPZ.

5.1. Comando TRAPZ

Calcula la integral definida entre dos límites de una función (área bajo la curva) representada por uno o dos vectores, como se explica más adelante. El cálculo de la integral se realiza numéricamente, por medio de una aproximación de la función a trapecios (En ningún momento calcula la integral simbólica).

Debido a que el cálculo de la integral es numérico, se deben construir vectores decentes para calcular la integral. Por esta razón, es fundamental aclarar las características de los vectores, con el fin de tener un criterio para decidir como construir el vector de forma apropiada. (para mayor claridad en el ejemplo de esta orden, puede ser necesario leer las secciones sobre FOR y Vectores y matrices).

La sintaxis de la orden es:

Valor = trapz([Vector,] Matriz);

Los símbolos [ ] significan que Vector es opcional.

Matriz puede ser una matriz o un vector. Una matriz si se desea calcular la integral definida para varias funciones en el mismo rango (entre los mismos límites). Un vector si se desea calcular la integral para una sola función (su tamaño tiene relación con el tamaño de Vector, esta relación se muestra en detalle en la explicación de Vector).

Vector es el vector de los valores para los cuales se desea calcular la integral, tal que si Matriz es:

- **Un vector:** Matriz y Vector deben ser de la misma longitud (ya sean vectores fila, o columna). A cada valor almacenado en Vector corresponde el valor almacenado en Matriz (con el mismo subíndice).
- **Una matriz:** Vector debe ser un vector columna y Matriz tiene almacenadas las funciones por columnas (cada columna = una función), Matriz debe tener el mismo número de filas que vector.

Si Vector se omite, es equivalente a introducir un vector con paso 1 (por ejemplo: [0, 1, 2, 3]), note que la integral no depende de los valores que se introducen en Valor, sino de su paso (ya que los valores de la función en cada punto están almacenados en Matriz), en otras palabras la integral sigue siendo la misma (en valor) si la corro hacia un lado y realizo la integral entre el nuevo par de límites.

Valor es donde se almacena el valor de la integral (un real si sólo se calculó para una función, y un vector fila si se calculó para varias).

5.2. Creación de vectores decentes

La integral se realiza aproximando la curva (función) a una serie de rectas, con el fin de aproximar el área bajo la curva a una serie de trapecios contiguos. Por lo tanto la aproximación es buena (decente) si efectivamente la función se comporta como una recta (aproximadamente) en cada sub−intervalo determinado por el paso y número de puntos que se tomen. Una forma empírica de verificar que los vectores están bien construidos es por medio de la orden plot, ya que esta función dibuja los vectores, aproximando la función de la misma forma que trapz. Por lo tanto, si al dibujar la curva con plot, esta se ve suave, los vectores están bien definidos.

El siguiente ejemplo ilustra el uso de trapz:

```
%Ejemplo de uso de trapz.

for i = 1:100,

x(i, 1)=1+ i / 20; % Asigna los valores de x entre 1 y 6 en incrementos

% de 0.05

y(i, 1) = x( i ,1) + 1; % Define la función y = x + 1

z(i, 1) = x( i,1)^2 + 1; % Define la función z=x^2+1

end;
```

Los vectores x, y, z se definieron como vectores columna arriba, con el fin de demostrar el funcionamiento de trapz con varias funciones.

Estos vectores perfectamente hubieran podido ser fila, pero hubiera sido más

difícil armar la matriz. Igualmente se requería construir un x que fuera

columna.

```
A(:, 1)=y;

A(:, 2)=z;

integral = trapz(x, y)

integral = trapz(x, z) % Normalmente se usaría un nombre diferente al de arriba

integral = trapz(x, A) % Normalmente se usaría un nombre diferente al de arriba
```

Al correr el programa se obtiene la siguiente salida:

```
integral =

22.3988
```

integral =

76.5662

integral =

22.3988 76.5662

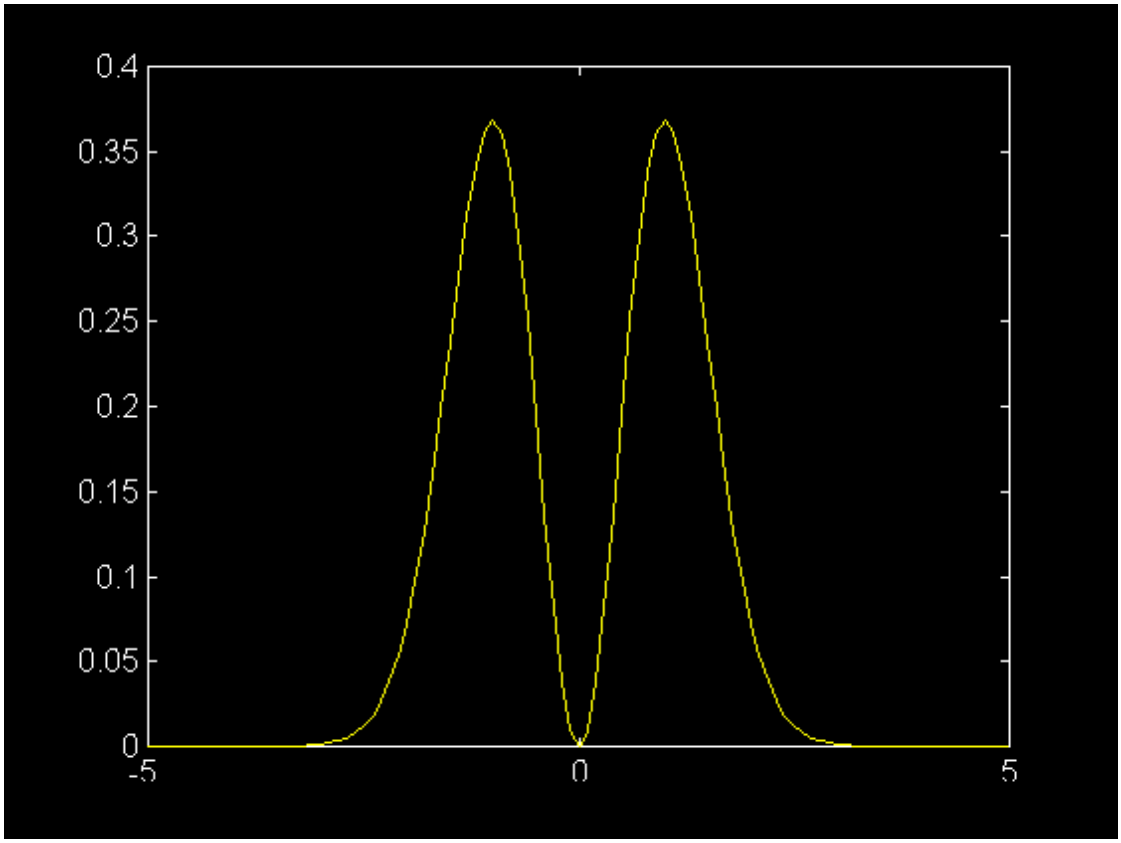
6. Gráficas

MATLAB provee excelentes funciones para gráficas en dos, tres y cuatro dimensiones. Veamos un par de ejemplos sencillos. Suponga que queremos trazar la gráfica de la función

$$f(x) = x^2 e^{-x^2} \quad , \quad x \in [-5,5]$$

Esto lo podemos lograr con las instrucciones:

```
» x=−5:.1:5;
» y=x.^2.*exp(−x.^2);
» plot(x,y)
```



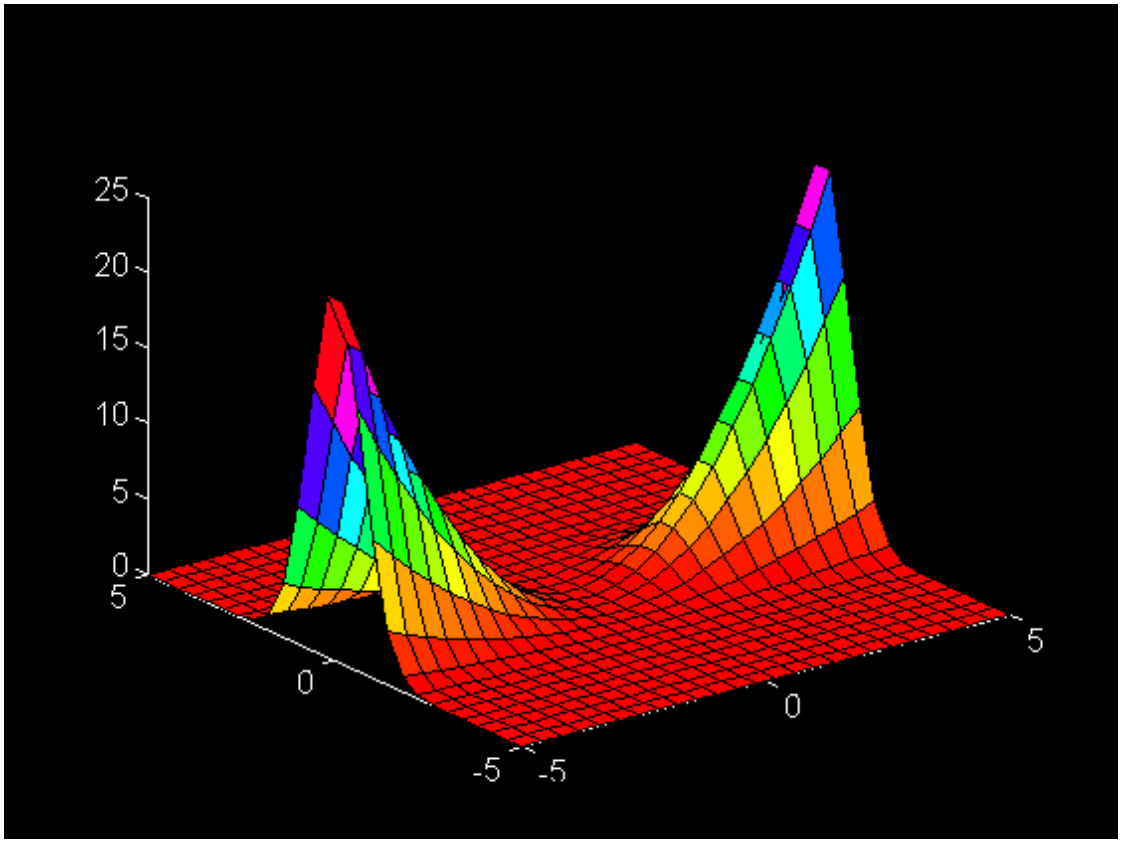
La primera instrucción divide el intervalo [−5,5] en subintervalos de largo 0.1, la segunda instrucción evalúa la función en los puntos de la partición, y finalmente graficamos los resultados con plot. La instrucción plot tiene opciones para cambiar patrones del trazado, poner titulos, etc.

Supongamos ahora que queremos dibujar la superficie:

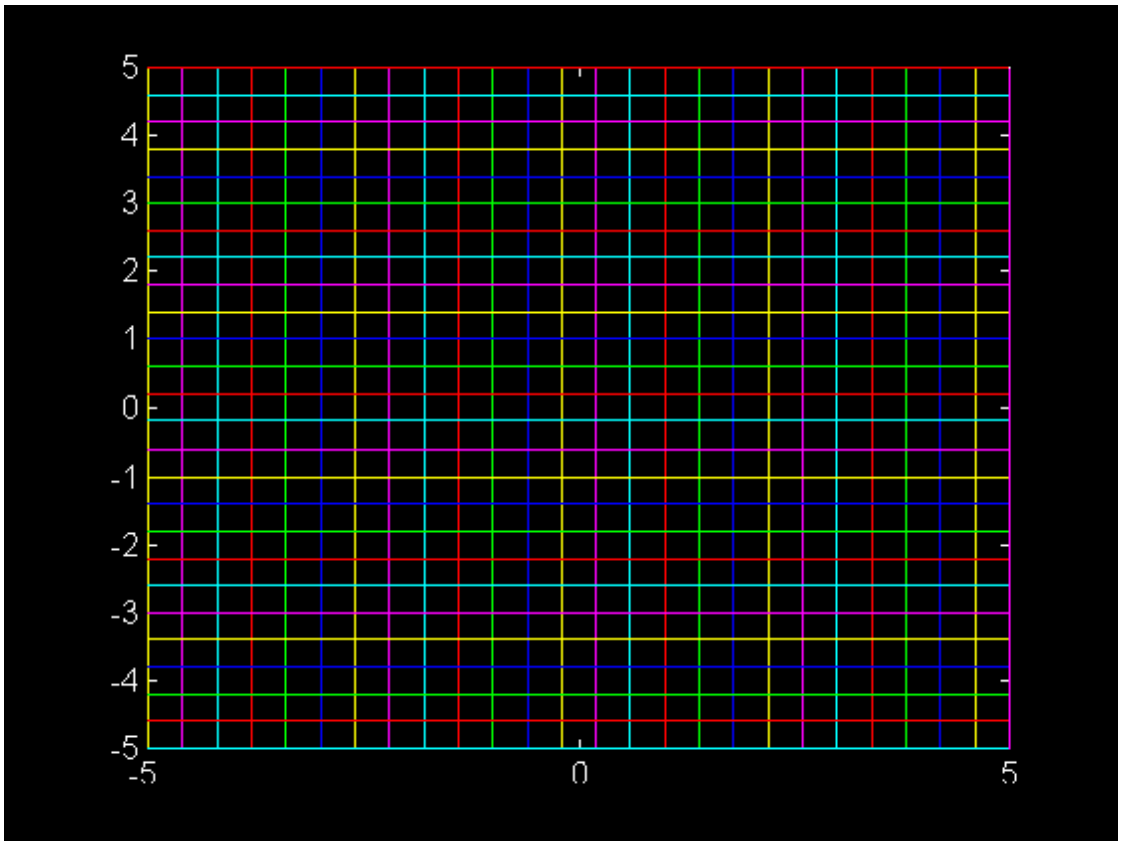
$$f(x,y) = x^2 e^{-y^2} \quad , \quad (x,y) \in [-5,5] \times [-5,5]$$

Esto lo hacemos con la secuencia de instrucciones:

```
» x=−5:.4:5;
» y=x;
» [X,Y]=meshgrid(x,y);
» Z=X.^2.*exp(−Y.^2);
» surf(X,Y,Z)
```



Las primeras dos instrucciones dividen los ejes de x y y en subintervalos de largo 0.4; la tercera instrucción genera una rejilla en el conjunto [−5,5]x[−5,5] con cuadraditos de lados 0.4 como se ilustra en la siguiente figura:



La cuarta instrucción evalúa la función en los puntos de la rejilla, y finalmente trazamos la superficie con surf.

Los temas tratados hasta ahora son suficientes para realizar programas sencillos y útiles. Los comandos disponibles en MATLAB son muchos más, pero los tratados aquí son los más frecuentemente necesitados. En caso de ser necesario emplear otras ordenes, es posible buscar la solución por medio de HELP (todo está en inglés), la cual lista los temas matemáticos que se pueden emplear (separados en librerías llamadas toolbox). HELP [toolbox] lista los comandos en la librería y HELP [comando] explica su uso y sintaxis.

Ejercicios de Aplicación:

Seguir con las descripciones dadas en MATLAB y explicar los resultados obtenidos. Esto le servirá para recordar todo lo aprendido. Si existe comandos que no recuerda por favor usar el help.

a = magic(4)

b=rand(4)

a'

3*a

a+(−a)

b = max(a)

max(b)

```
[m, i] = max(b)

min(a)

b = 2*ones(a)

a*b (aquí se presenta un error Cuál es?)

a

who

whos

clear

clc

A = magic(5)

b = ones(5,1)

A*b

v = ones(1,5)

v*A

a.*b (Hay un punto allí)

x = 5

x^2

a*a

a^2

a.^2 (otro punto)

a

triu(a)

tril(a)

diag(a)

diag(diag(a))

c=rand(4,5)
```

size©

[m,n] = size©

m

d=.5−c

sin(d)

exp(d)

log(d)

abs(d)

clear

clc

f=[−.5 .1 .5]

round(f)

fix(f)

ceil(f)

floor(f)

sum(f)

prod(f)

%Relaciones y Operaciones Lógicas

a=[1 0 1 0]

b=[1 1 0 0]

a==b

a<=b

~a

a&b

a & ~a

a | b

```
a | ~a

a

any(a)

c=zeros(1,4)

d=ones(1,4)

any( c )

all(a)

all(d)

e=[a',b',c',d']

any(e)

all(e)

any(all(e))

r = input(` Uso de la notación de dos puntos `)

clear

clc

x=−2:1

length(x)

−2:.5:1

−2:.2:1

a=magic(5)

a(2,3)

disp(` Ahora veremos como usamos los dos puntos para seleccionar una columna de a')

a(2,:)

a(:,3)

a

a(2:4,:)
```

```
a(:,3:5)

a(2:4,3:5)

a(1:2:5,:)

a(:,[1 2 5])

a([2 5],[2 4 5])

b=rand(5)

b([1 2],:)=a([1 2],:)

a(:,[1 2])=b(:,[3 5])

a(:,[1 5])=a(:,[5 1])

a=a(:,5:−1:1)

v=[0 1 0 1 1]

a(:,v)

a(v,:)

input(`Cómo estamos en programación con MATLAB???')

[m,n]=size(a);

[k,l]=size(b);

if m~=k | n~=l,

r='ERROR para ser usado: cuando las matrices no son del mismo tamaño';

return,

end

c=zeros(m,n);

for i=1:m,

for j=1:n,

c(i,j)=a(i,j)+b(i,j);

end

end
```

```
% c=mult(a,b). Este es el producto de dos matrices

% de las matrices a y b.

% function c=a*b.

[m,n]=size(a);

[k,l]=size(b);

if n~=k,

c='ERROR para indicar: matrices que no son compatibles para ser multiplicados',

return,

end,

c=zeros(m,l);

for i=1:m,

for j=1:l,

for p=1:n,

c(i,j)=c(i,j)+a(i,p)*b(p,j);

end

end

end

l=0;

m=2;

while m<=n

l=l+1;

m=2*m;

end

%Recursividad

if n==0, y=1;

else y=2*twoexp(n−1);
```

```
end

pi

x = []; for i =1:n, x=[x,i^2], end

x = [];

for i = 1:n

x = [x,i^2]

end

clc

for i = 1:m

for j = 1:n

H(i,j) = 1/(i+j−1);

end

end

H

n = 0;

while 2^n <= a

n = n + 1;

end

n

if n < 0

parity = 0;

elseif rem(n,2) == 0

parity = 2;

else

parity = 1

end
```

```
x = [0.0:0.1:2.0]';

y = sin(x);

[x y]

m=2; n=3; x=0:.01:2*pi; y=sin(m*x); z=cos(n*x); plot(x,y,x,z)

%Archivos m???

y = floor(10*rand(m,n));

if nargin < 3, a=0; b=9; end

y = floor((b−a+1)*rand(m,n))+a;

[m n] = size(x);

if m == 1

m = n; % handle case of a row vector

end

mean = sum(x)/m;

stdev = sqrt(sum(x.^2)/m −mean.^2);

t = clock; x = A\b; time = etime(clock,t)

%Parece que queremos dibujar

x = −4:.01:4; y = sin(x); plot(x,y)

x = −1.5:.01:1.5; y = exp(−x.^2); plot(x,y)

t=0:.001:2*pi; x=cos(3*t); y=sin(2*t); plot(x,y)

x=0:.01:2*pi;y1=sin(x);y2=sin(2*x);y3=sin(4*x);plot(x,y1,y2,y3)

x=0:.01:2*pi; Y=[sin(x)', sin(2*x)', sin(4*x)']; plot(x,Y)

x=0:.01:2*pi; y1=sin(x); y2=sin(2*x); y3=sin(4*x);

plot(x,y1,'`,x,y2,':',x,y3,'+')

xx = −2:.1:2;

yy = xx;

[x,y] = meshgrid(xx,yy);
```

```
z = exp(−x.^2 − y.^2);

mesh(z)

[x,y] = meshgrid(xx,yy);

[x,y] = meshgrid(−2:.1:2,−2:.1:2);

clc

clear

input(`Nuevamente vamos a iniciar');

c = 5.66

c = [5.66]

x = [ 3.5, 33.22, 24.5 ]

x(1) = 2, x(2) = 4

x(3) = −1.

y(1,1)=0, y(1,2) = y(1,3) = 2,

y(1,4) = 3, y(2,1) = 5, y(2,2) = −3,

y(2,3) = 6, y(2,4) = 4

y = [ 0 2 2 3 ; 5 −3 6 4 ]

a = [ sin(pi/2) sqrt(2) 3+4 6/3 exp(2) ]

a = [ 1.0000 1.4142 7.0000 2.0000 7.3891 ]

x1 = [ x 5 8 ] creates the result

x1 = [ 2 4 −1 5 8 ]

x = [ 2 4 −1 0 8 ]

c = [ 4 5 6 3 ]

z = [ y;c ]

e = [3 5 10 0; 0 0 ...

0 3; 3 9 9 8 ]

t = [4 24 9]
```

```
q = [t 0 t]

x = [ 3 6 ]

y = [d;x]

z = [x;d]

clear

r = [ c; x,5]

v = [ c(2,1); b ]

%más?

a = 1:8

b = 0.0 : .2 : 1.0

y = x(:,1)

yy = x(:,2)

z = x(1,:)

c = [ −1 0 0

1 1 0

1 −1 0

0 0 2 ]

d1 = c(:,2:3)

d2 = c(3:4,1:2)

% crear la matriz

g = [ 0.6 1.5 2.3 −0.5

8.2 0.5 −0.1 −2.0

5.7 8.2 9.0 1.5

0.5 0.5 2.4 0.5

1.2 −2.3 −4.5 0.5 ]

a = g(:,2)
```

```
b = g(4,:)

c = [10:15]

d = [4:9;1:6]

e = [−5,5]

f= [1.0:−.2:0.0]

t1 = g(4:5,1:3)

t2 = g(1:2:5,:)

% vamos a usar nuevamente plot(x,y)

% semilogx(x,y) plots log(x) vs y

%semilogy(x,y) plots x vs log(y)

% loglog(x,y) plots log(x) vs log(y)

%title(`text'), xlabel(`text'), ylabel(`text'),

%text(x,y,'text'), text(x,y,'text','sc')

%polar(theta,r)

%bar(x), stairs(x), bar(x,y), stairs(x,y)

x=0:.1:2;

y=exp(x);

plot(x,y)

clc

clear

clf

x1 = 0 : pi/4 : pi

y1 = sin(x1)

plot(x1,y1)

clear

x1 = 0 : .05*pi : pi ;
```

```
y1 =sin(x1);
plot(x1,y1)
angle = 0:.1*pi:3*pi;
radius = exp(angle/20);
polar(angle,radius),...
title(`Un ejemplo de Plot Polar'),...
grid
% Las barras?
x1=0:.05*pi:pi;
y1=sin(x1);
plot(x1,y1)
hold
y2=cos(x1);
plot(x1,y2)
a = 1 : .1 : 3;
b = 10*exp(−a);
plot(x,y,a,b)
yy=[y;exp(1.2*x)];
plot(x,yy)
plot(x,y,x1,y1,'o')
plot(x,y,'r',r,s,'g')
%max(x), min (x), mean(x)
%median(x), sum(x), prod(x)
%std(x), sort(x), hist(x), hist(x,n),
hist(x(:,2))
% Aplicar el concepto de archivo.m usando plot exp(−x/10)sin(x)
```

```
x = [ 0:.2:10 ];

y = exp(−x/10) .* sin(x);

plot(x,y),...

title(`GRAFICO DE LA FUNCION EXPONENCIAL SENO'),...

xlabel(`x'),...

ylabel(`y'),...

text(.6,.7,'y = exp(−x/10)*sin(x)','sc')

clear

clc

clf

x = pi/2; y = sin(x)

z = 0; w = exp(4*z)/5

w = (exp(4*x))/5

A=[1 2 3; 3 3 3; 5 3 1]

B=[2 −3 4;2 −2 2; 0 4 0]

C = A + B

C = B + A

x= [3 5 7]

y = [4; −1; −3]

z = x + y

a = [ 1 2; 3 4];

b = [ 8 7; 6 5];

c = a*b

X = [2 3 ; 4 −1 ; 0 7];

Y = [5 −6 7 2 ; 1 2 3 6];

a .* b
```

a ./ b

a .\ b

a .^ b

G = [1 3 5; 2 4 6];

H = [−4 0 3; 1 9 8];

G .* H

G'

G1 = G(1,:)

G2 = G(2,:)

G1 * G2'

G1' * G2

a=eye(4)

eye(3,2)

eye(2,3)

Para que practiques ¡!!!!!!

Los primeros cuatro términos de la serie de fourier para la ola cuadrada

de amplitud 5 y de periodo 2 es

y = (20/)[sinx + (1/3)sin3x + (1/5)sin5x + (1/7)sin7x)]

Calcule esta serie, término por término, y trace los resultados de su suma parcial para cada uno

%Ecuaciones lineales ya!!!!!

C = [1 −4 3 2; 3 1 −2 1; 2 1 1 −1; 2 −1 3 1]

C_inverse = inv(C);

C_inverse * C

%Resolver el sistema:

x1 − 4x2 + 3x3 = −7

3x1 + x2 − 2x3 = 14

2x1 + x2 + x3 = 5

%Ubícate y sigue

a = [1 −4 3; 3 1 −2; 2 1 1];

b = [−7; 14; 5];

x = inv(a)*b

% resultado ?

%Probar para el sistema:

x1 − 4x2 + 3x3 = −7

13x2 − 11x3 = 35

(34/13)x3 = (68/13)

a = [1 −4 3; 3 1 −2; 2 1 1];

b = [−7; 14; 5];

x1 = a\b

% [x][A] = [B]

%Resolver

x1 − 4x2 + 3x3 = −7

3x1 + x2 − 2x3 = 14

2x1 + x2 + x3 = 5

% Resolver el sistema y tomar el

%tiempo de solución por los tres métodos

r + s + t + w = 4

2r − s + w = 2

3r + s − t − w = 2

r − 2s − 3t + w = −3

% Resolver y tomar el tiempo de solución

%comparando los tres métodos estudiados

$2x1 + x2 - 4x3 + 6x4 + 3x5 - 2x6 = 16$

$-x1 + 2x2 + 3x3 + 5x4 - 2x5 = -7$

$x1 - 2x2 - 5x3 + 3x4 + 2x5 + x6 = 1$

$4x1 + 3x2 - 2x3 + 2x4 + x6 = -1$

$3x1 + x2 - x3 + 4x4 + 3x5 + 6x6 = -11$

$5x1 + 2x2 - 2x3 + 3x4 + x5 + x6 = 5$

Referencias

- MATLAB User's Guide, The MathWorks, Inc., Massachusetts, 1997.
- The MATLAB Handbook, E. Part−Enander, A. Sjoberg, B. Melin, and P. Isaksson, Addison−Wesley, New York, 1996.
- Solución de Problemas de Ingeniería con MATLAB, Delores M. Etter Prentice Hall

México 1998.

68

Tutorial de MATLAB

Red de Información Científica Hospital III Juliaca EsSalud Pág 2 de 53

HACER DOBLE

CLIC AQUÍ