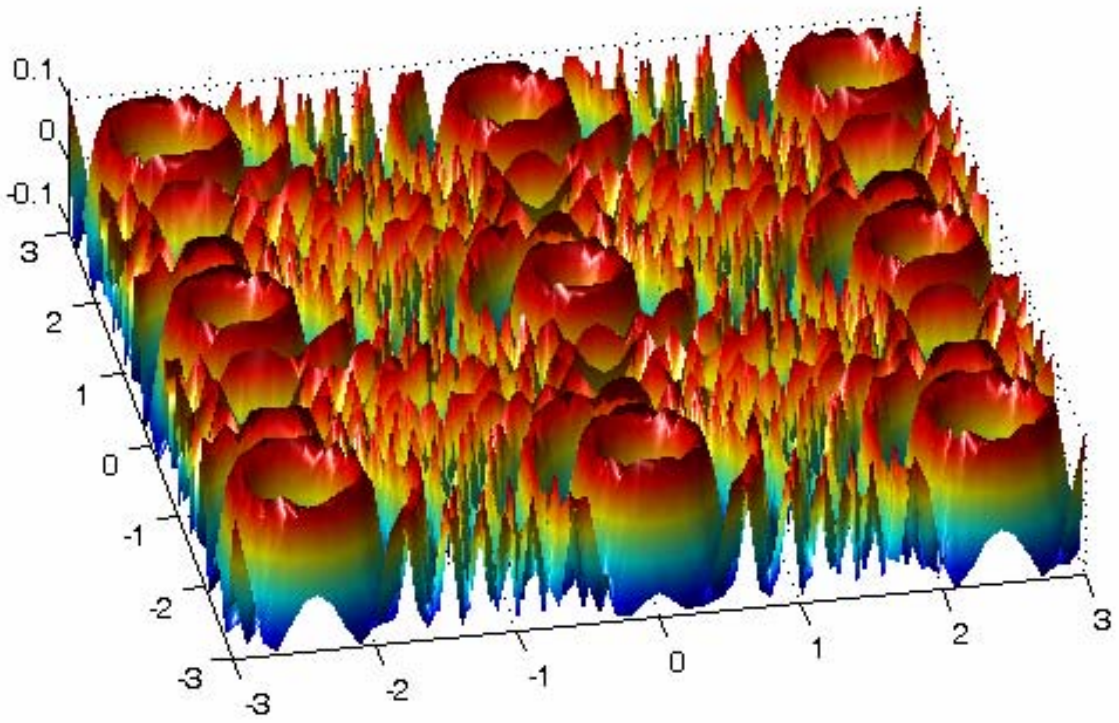
Tutorial de Matlab

“Introducción a Matlab”



J. Undurraga L.

R. Venegas C.

Índice

1. Introducción

1.1. *Que es Matlab?*

Matlab es un lenguaje de alto rendimiento para computación técnica. Este integra computación, visualización, y programación de fácil utilización donde los problemas y soluciones son expresados en una familiar notación matemática. Usos típicos incluyen:

- Matemáticas y computación
- Desarrollo de Algoritmos
- Adquisición de datos
- Modelación, simulación y desarrollo de prototipos
- Análisis, exploración y visualización de datos
- Gráficos Científicos y de Ingeniería
- Desarrollo de aplicaciones, incluyendo un constructor de interfaz gráfica

Matlab es un sistema interactivo cuyo elemento básico de dato son arreglos. Esto permite resolver muchos problemas de computación técnica, especialmente aquellos con formulación matricial y vectorial.

El nombre Matlab viene de *Matriz Laboratory*, y ha sido desarrollado durante varios años con la ayuda de muchos usuarios. En ambientes universitarios es la herramienta estándar para la instrucción de cursos básicos y avanzados de matemáticas, ingeniería y ciencias.

Matlab presenta una familia de aplicaciones específicas llamadas *toolboxes* muy importantes para la mayoría de los usuarios de Matlab, ya que permiten conocer y aplicar tecnología especializada. Los toolboxes con una colección sencilla de funciones de Matlab (M-files) que extienden el ambiente de Matlab para resolver problemas en particular. Las áreas que los toolboxes incluyen son: procesamiento de señales, sistemas de control, redes neuronales, lógica difusa, wavelets, simulación, y muchas otras.

1.2. *Ayuda de Matlab*

Matlab cuenta con un poderoso y completo sistema de ayuda el cual puede ser llamado como se aprecia en la Figura 1-1:

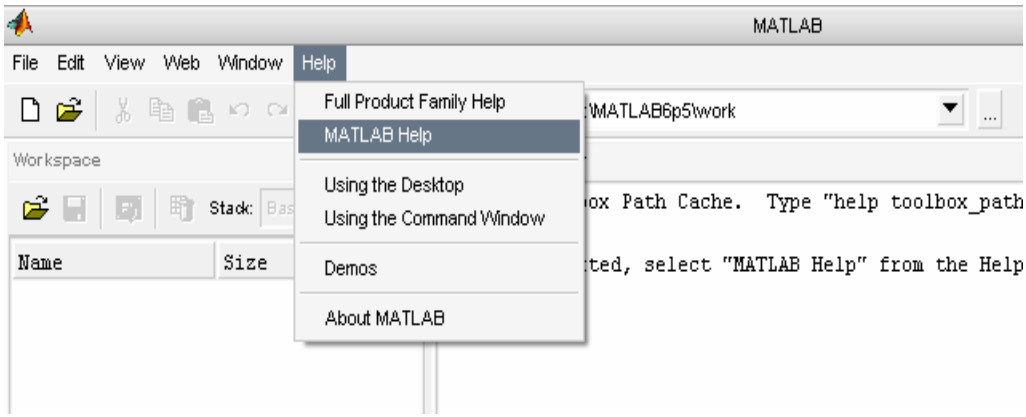


Figura 1-1

Matlab consta de un poderoso sistema de búsqueda y además está dotado con una completa gama de textos instructivos que permiten al usuario tanto de nivel básico, como de nivel avanzado lograr realizar una gran gama de tareas, explicando en detalle la estructura, utilización, implementación y ejemplificación de las diferentes herramientas y funciones que posee.

Además, se puede también recurrir a la ayuda desde la línea de comandos. Por ejemplo:

```
» help
» help fft
» help laplace
```

El comando **helpwin** seguido de un nombre de comando muestra la información correspondiente a ese comando en la ventana **Help Window**, incluyendo también comandos similares sobre los que se ofrece ayuda.

2. Comenzando a utilizar Matlab

2.1. *El Desktop de Matlab*

El escritorio de matlab se puede apreciar en la Figura 2-1:

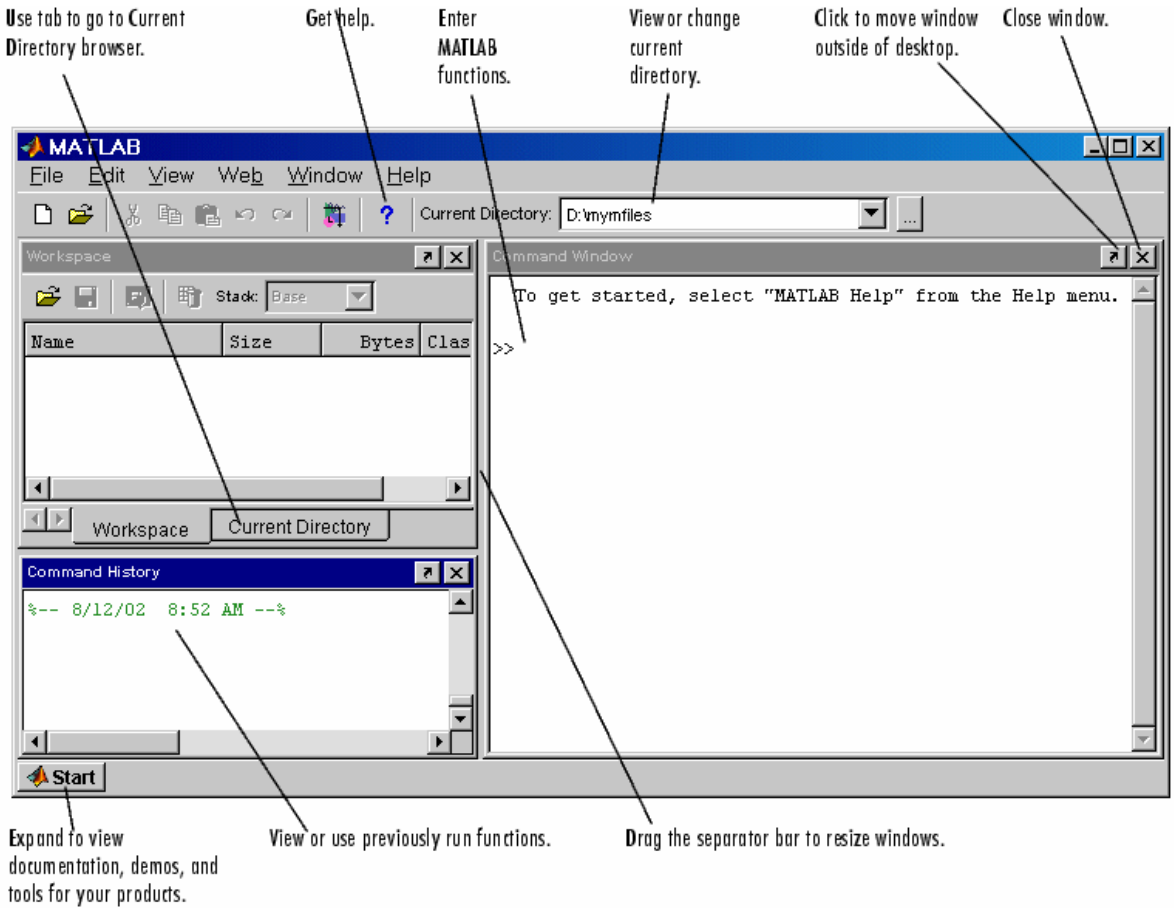


Figura 2-1

El escritorio de matlab posee las siguientes herramientas:

- Command Window
- Command History
- Start Button and Launch Pad
- Help Browser
- Current Directory
- Browser Workspace
- Browser Array
- Editor Editor/Debugger
- Profiler

2.1.1. **Descripción de herramientas de escritorio:**

2.1.1.1. **Command Window (ventana de comandos)**:

Se utiliza para ingresar las variables, ejecutar las funciones y los archivos m con los cuales trabaja matlab. En la Figura 2-2 se puede apreciar un pequeño ejemplo:

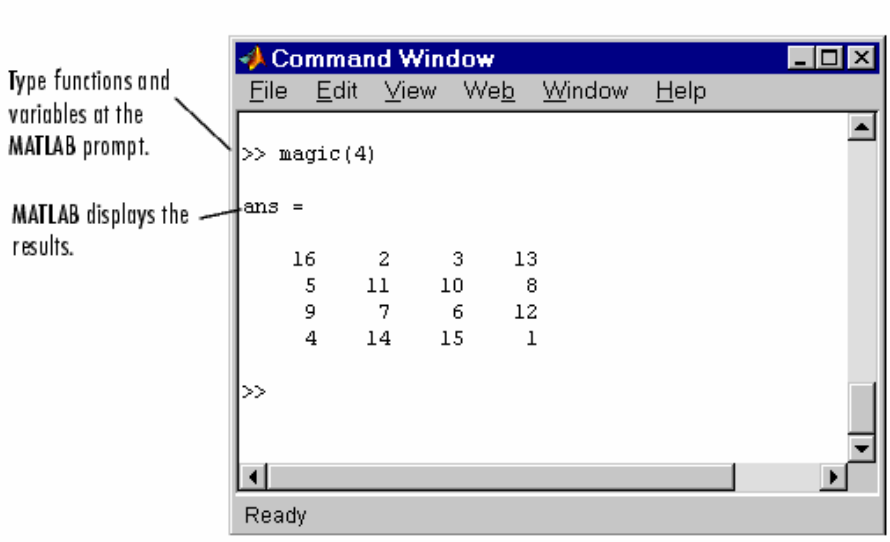


Figura 2-2

2.1.1.2. **Command History (historial de comandos):**

Las declaraciones hechas en la ventana de comandos quedan almacenadas en el historial de comandos. Aquí se pueden observar y ejecutar declaraciones previamente realizadas, como copiar y ejecutar declaraciones seleccionadas. En la Figura 2-3 se puede apreciar la Command History.

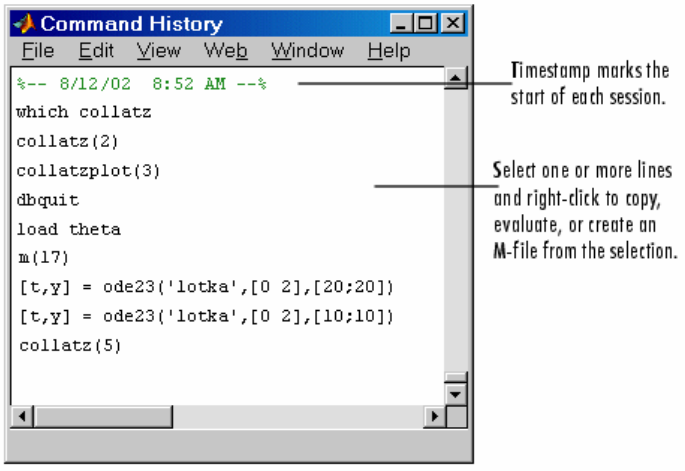


Figura 2-3

El insertar el signo de exclamación y posteriormente el nombre de un programa permite ejecutar un programa directamente, por ejemplo:

```
» !calc
```

Esta línea ejecuta la calculadora de Windows.

2.1.1.3. **Start Button and Launch Pad:**

El botón de inicio y la plataforma de lanzamiento permiten un rápido acceso a las herramientas, demos y documentación al hacer clic en la opción deseada.

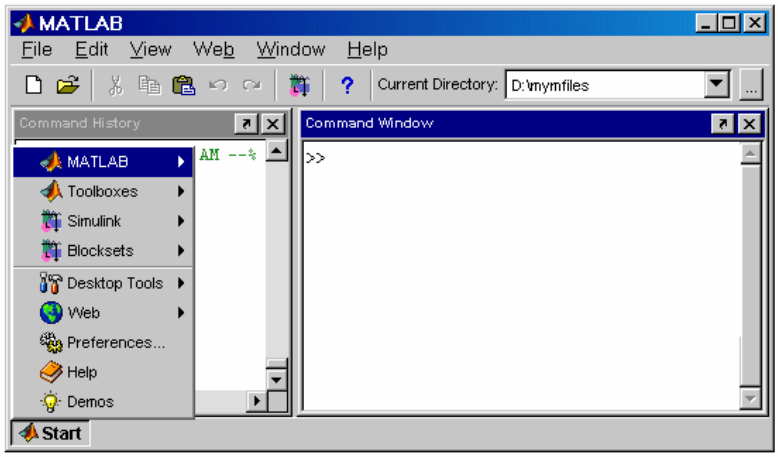


Figura 2-4

La plataforma de lanzamiento provee un acceso similar, pero en forma de árbol, como se aprecia en la Figura 2-5:

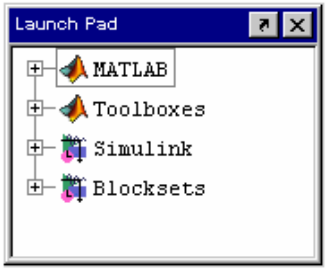


Figura 2-5

2.1.1.4. **Help Browser:**

El buscador de ayuda permite ver y buscar documentación y demos para todos los productos de matlab. Este es un buscador de formato Web integrado a MATLAB, el cual muestra documentos en formato HTML. Para abrir el Buscador de ayuda, hay que hacer clic en el boton **?** de la barra de herramientas, o escribir ***helpbrowser*** en la ventana de comandos.

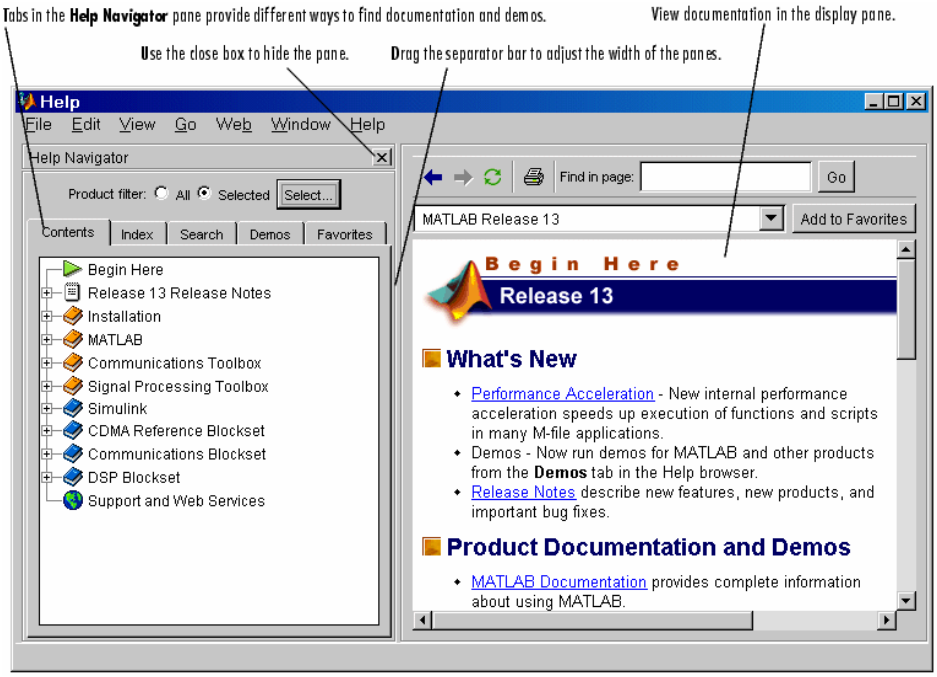


Figura 2-6

2.1.1.5. **Current Directory:**

Los archivos con los cuales opera MATLAB utilizan el Current Directory y el Search Path. Cualquier archivo que se ejecute debe estar en el *current directory* o en el *search path*.

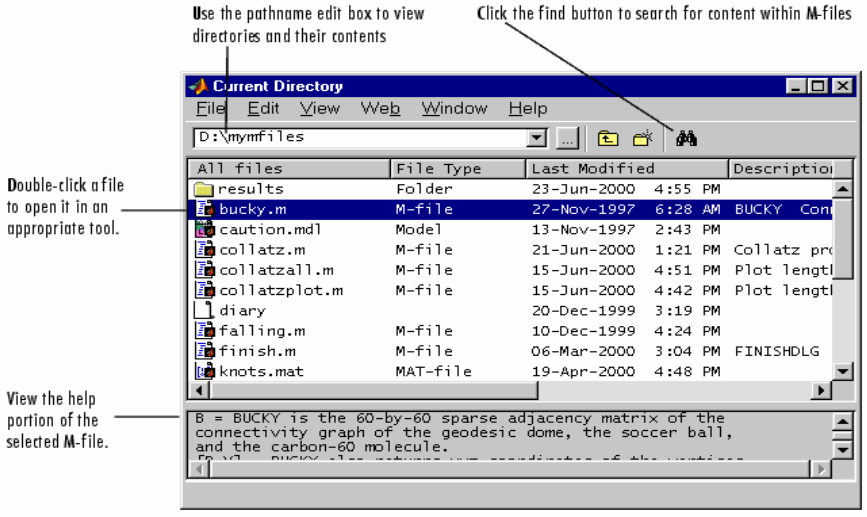


Figura 2-7

2.1.1.6. **Workspace Browser:**

El *worksapace* o espacio de trabajo de MATLAB consiste en un set de variables (arreglos) construidos durante la sesión de trabajo de MATLAB que son almacenados en la memoria. Para ver la información del espacio de trabajo y la información de cada variable se puede utilizar el ***workspace brownser*** (**Figura 2-8**) o utilizar las funciones ***who*** o ***whos.***

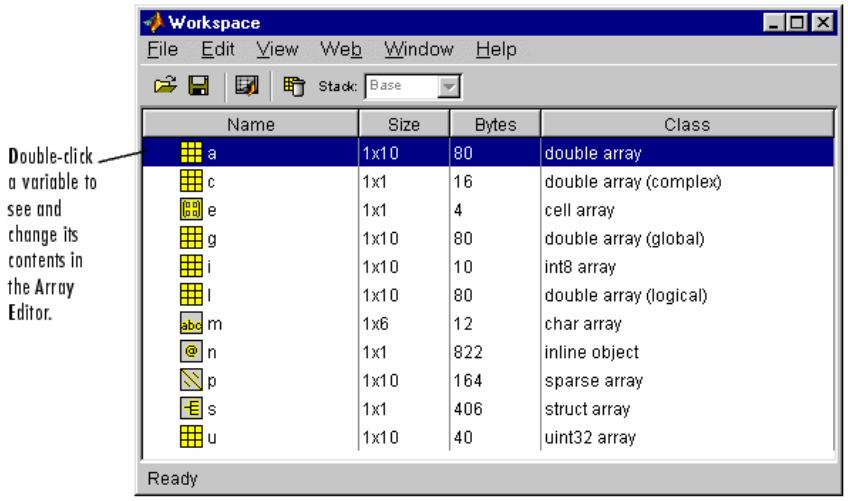


Figura 2-8

2.1.1.7. **Array Editor:**

El editor de arreglos aparece al hacer doble clic en alguna variable del ***workspace brownser***. Este editor puede ser utilizado para observar y editar el contenido de las variables, ver **Figura 2-9**.

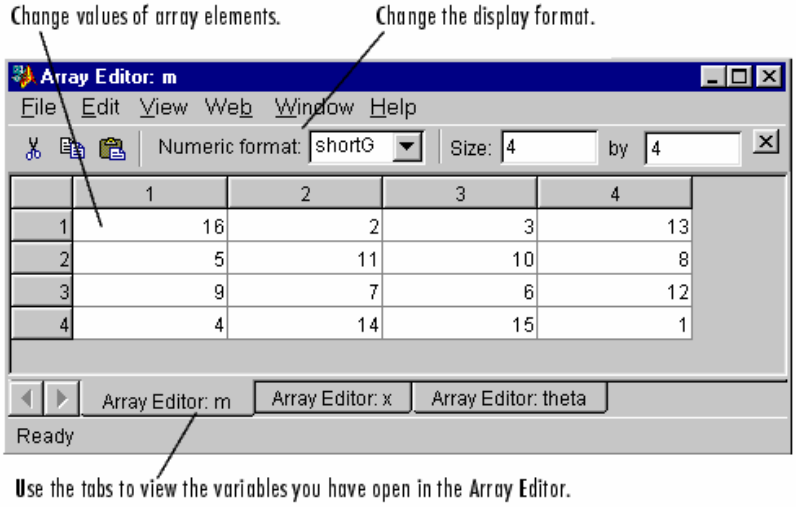


Figura 2-9

2.1.1.8. **Editor/Debugger:**

El editor de matlab es utilizado para crear y construir archivos M. Los cuales son los programas que se escriben para correr las funciones MATLAB, ver **Figura 2-10** .

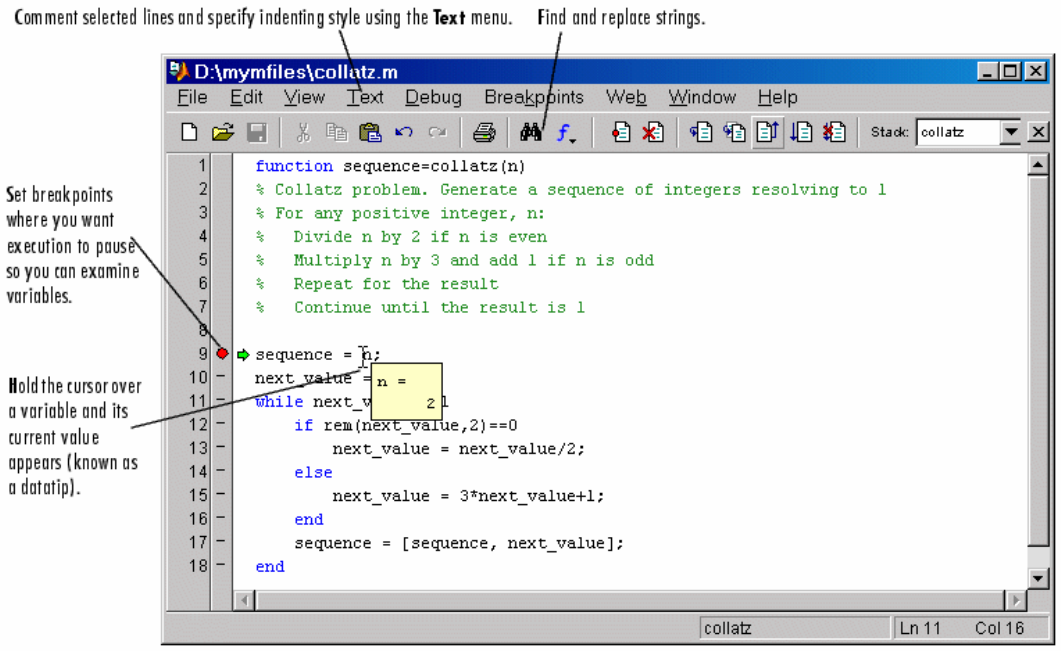


Figura 2-10

2.1.1.9. **Profiler:**

El profiler es una interfaz gráfica que ayuda a mejorar el rendimiento de los archivos M. Para utilizarlo solamente hay que seleccionarlo de la sección View en la barra de herramientas.

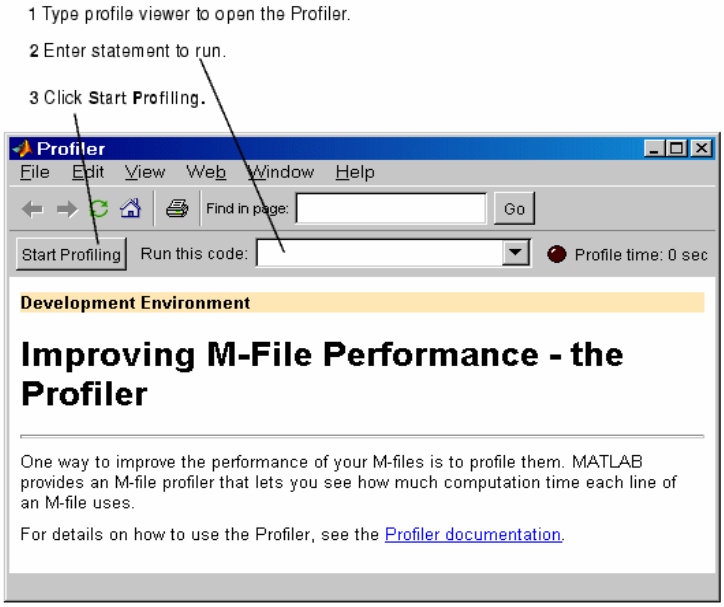


Figura 2-11

3. Manipulación de Matrices

3.1. *Creación de Matrices*

En MATLAB se pueden crear matrices de diferentes formas:

- Entrando una lista explícita de elementos
- Leyendo matrices de archivos externos
- Generar matrices utilizando funciones para construirlas
- Creando matrices con funciones propias en archivos M

Comenzaremos escribiendo en la ventana de comandos lo siguiente:

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

Al hacer esto y presionar ENTER, se creará la siguiente matriz:

```
A =

    16     3     2    13
     5    10    11     8
     9     6     7    12
     4    15    14     1
```

3.2. *Suma, Transpuesta y Diagonal:*

3.2.1. **Suma:**

La función ***sum*** permite sumar los elementos de las columnas de una matriz, obteniéndose de esta forma un vector fila que contiene la suma de los elementos de las columnas. Así al aplicar la función ***sum*** a la matriz A del problema anterior obtenemos:

```
>> sum(A)

ans =

    34    34    34    34
```

Nota: cuando no se especifica la variable de salida, Matlab utiliza la variable de salida ***ans.***

3.2.2. **Transpuesta:**

La transpuesta de una matriz intercambia las filas por as columnas, para hacer esto simplemente se debe poner el símbolo apóstrofe después de la matriz como, por ejemplo:

```
>> A'

ans =

    16     5     9     4
     3    10     6    15
     2    11     7    14
    13     8    12     1
```

3.2.3. **Diagonal:**

Para obtener la diagonal de una matriz, simplemente se debe utilizar la función ***diag*** sobre la matriz como, por ejemplo:

```
>> diag(A)

ans =

    16
    10
     7
     1
```

3.3. ***Subíndice, operador dos puntos***

3.3.1. **Subíndice:**

El elemento de la fila i y la columna j de una matriz puede ser llamado de la forma A(i,j). Por ejemplo, para calcular la suma de los elementos de la cuarta columna de la matriz A se debe realizar la siguiente operación:

```
>> A(1,4) + A(2,4) + A(3,4) + A(4,4)

ans =

    34
```

3.3.2. **Operador dos puntos:**

El operador dos puntos es una de las herramientas más importantes en MATLAB, el cual puede ser utilizado de diferentes formas como, por ejemplo:

```
>> 1:10
```

Este comando crea el siguiente vector fila:

```
ans =

     1     2     3     4     5     6     7     8     9    10
```

Si se desea obtener un espaciamiento no unitario, simplemente se debe indicar de las siguientes formas:

- Ej. 1

```
>> 1:3:12

ans =

     1     4     7    10
```

- Ej. 2

```
>> 0:pi/2:2*pi

ans =

         0    1.5708    3.1416    4.7124    6.2832
```

O en el caso que se desee un decremento, se puede realizar de la siguiente forma:

```
>> 20:-2:2

ans =

    20    18    16    14    12    10     8     6     4     2
```

Al utilizar el operador dos puntos en subíndices de matrices, se puede acceder a porciones de éstas, esto es, si se realiza la operación A(1:k,j) se llaman los primeros k elementos de la columna j. En el caso de que se requiera llamar a todos los elementos de la columna j, basta con utilizar el operador dos puntos de la siguiente forma, A(:,k). Al hacer esto, se estarán llamando todos los elementos de la columna k. Para comprender mejor este punto, veamos el siguiente ejemplo:

```
>> A(1:3,1)

ans =

    16
     5
     9
```

En este ejemplo, se llamaron los tres primeros elementos de la columna 1. Si se desea llamar a todos los elementos de la columna uno, simplemente se debe realizar la siguiente operación:

```
>> A(:,1)

ans =

    16
     5
     9
     4
```

Como otro ejemplo, consideremos que se desea sumar todos los elementos de la última columna de la matriz A, para lograr esto, simplemente se debe realizar la siguiente operación:

```
>> sum(A(:,end))

ans =

    34
```

Donde el operador ***end***, índica que se desea la última fila o columna según sea utilizado.

3.4. ***Variables:***

MATLAB no requiere ningún tipo de declaración de variables o dimensión. Cuando MATLAB encuentra un nuevo nombre de variable, automáticamente crea la variable y la almacena de la manera más apropiada. Si la variable ya existe, entonces MATLAB cambia su contenido. Por ejemplo:

```
>> Alumnos=15
```

Aquí se crea la variable Alumnos, el cual es una matriz de 1x1 en el cual almacena el valor 15.

Nota: MATLAB es sensible a minúsculas y mayúsculas, es decir la variable A es diferente a la variable a.

En cuanto a los números, MATLAB utiliza la notación decimal convencional, con punte decimal opcional y permitiendo signo positivo o negativo. La notación

científica utiliza la letra ***e*** para indicar el factor escalar de potencia de diez. Los números imaginarios utilizan el sufijo ***i*** o ***j***. Algunos ejemplos son:

```
3              -99            0.0001
9.6397238      1.60210e-20    6.02252e23
1i             -3.14159j      3e5i
```

Todos los números almacenados utilizan el formato ***long*** especificado por el estándar de punto flotante de la IEEE.
Los números de punto flotante tienen una precisión finita de 16 cifras decimales significativos y un rango finito de precisión de 10^{-308} a 10^{+308}.

3.5. ***Operadores:***

Las expresiones utilizan las comunes operaciones aritméticas, que se aprecian en la Tabla 3.1:

+	Suma
-	Resta
*	Multiplicación
/	División
\	División por la izquierda (Utilizado para operaciones Matriciales)
^	Potencia
‘	Conjugado complejo transpuesto
( )	Especifica el orden de evaluación

Tabla 3.1

3.6. ***Funciones:***

MATLAB posee una gran cantidad de funciones elementales estándar, incluyendo **abs**, **sqrt**, **exp**, y **sin**. Al calcular la raíz o el logaritmo de un numero negativo, MATLAB no entrega un error, si no que automáticamente provee la solución compleja. MATLAB también posee muchas funciones avanzadas, como las funciones de Bessel y Gamma.

Para obtener una lista de las funciones elementales, se debe escribir en la línea de comandos lo siguiente:

>>help elfun

Al hacer esto, aparecerá la lista de funciones elementales matemáticas que se muestra en la Tabla 3.2:

Trigonometric.

sin	- Sine.
sinh	- Hyperbolic sine.
asin	- Inverse sine.
asinh	- Inverse hyperbolic sine.
cos	- Cosine.
cosh	- Hyperbolic cosine.
acos	- Inverse cosine.
acosh	- Inverse hyperbolic cosine.
tan	- Tangent.
tanh	- Hyperbolic tangent.
atan	- Inverse tangent.
atan2	- Four quadrant
atanh	- Inverse hyperbolic tangent.
sec	- Secant.
sech	- Hyperbolic secant.
asec	- Inverse secant.
asech	- Inverse hyperbolic secant.
csc	- Cosecant.
csch	- Hyperbolic cosecant.
acsc	- Inverse cosecant.
acsch	- Inverse hyperbolic cosecant.
cot	- Cotangent.
coth	- Hyperbolic cotangent.
acot	- Inverse cotangent.
acoth	- Inverse hyperbolic cotangent.
Exponential.	
exp	- Exponential.
log	- Natural logarithm.
log10	- Common (base 10) logarithm.
log2	- Base 2 logarithm and dissect floating point number.
pow2	- Base 2 power and scale floating point number.
realpow	- Power that will error out on complex result.
reallog	- Natural logarithm of real number.
realsqrt	- Square root of number greater than or equal to zero.
sqrt	- Square root.
nextpow2	- Next higher power of 2.
Complex.	
abs	- Absolute value.
angle	- Phase angle.
complex	- Construct complex data from real and imaginary parts.
conj	- Complex conjugate.
imag	- Complex imaginary part.
real	- Complex real part.
unwrap	- Unwrap phase angle.
isreal	- True for real array.

cplxpair	- Sort numbers into complex conjugate pairs.
Rounding and remainder.	
fix	- Round towards zero.
floor	- Round towards minus infinity.
ceil	- Round towards plus infinity.
round	- Round towards nearest integer.
mod	- Modulus (signed remainder after division).
rem	- Remainder after division.
sign	- Signum.

Tabla 3.2

Para obtener una lista de para funciones avanzadas y matriciales se deben utilizar los siguientes comandos:

>>help specfun
>>help elmat

Al utilizar el comando **help specfun** aparecerá la lista de funciones que se muestra en la Tabla 3.3:

Specialized math functions.	
airy	- Airy functions.
besselj	- Bessel function of the first kind.
bessely	- Bessel function of the second kind.
besselh	- Bessel functions of the third kind (Hankel function).
besseli	- Modified Bessel function of the first kind.
besselk	- Modified Bessel function of the second kind.
beta	- Beta function.
betainc	- Incomplete beta function.
betaln	- Logarithm of beta function.
ellipj	- Jacobi elliptic functions.
ellipke	- Complete elliptic integral.
erf	- Error function.
erfc	- Complementary error function.
erfcx	- Scaled complementary error function.
erfinv	- Inverse error function.
expint	- Exponential integral function.
gamma	- Gamma function.
gammainc	- Incomplete gamma function.
gammaln	- Logarithm of gamma function.
psi	- Psi (polygamma) function.
legendre	- Associated Legendre function.
cross	- Vector cross product.
dot	- Vector dot product.
Number theoretic functions.	

factor	- Prime factors.
isprime	- True for prime numbers.
primes	- Generate list of prime numbers.
gcd	- Greatest common divisor.
lcm	- Least common multiple.
rat	- Rational approximation.
rats	- Rational output.
perms	- All possible permutations.
nchoosek	- All combinations of N elements taken K at a time.
factorial	- Factorial function.
Coordinate transforms.	
cart2sph	- Transform Cartesian to spherical coordinates.
cart2pol	- Transform Cartesian to polar coordinates.
pol2cart	- Transform polar to Cartesian coordinates.
sph2cart	- Transform spherical to Cartesian coordinates.
hsv2rgb	- Convert hue-saturation-value colors to red-green-blue.
rgb2hsv	- Convert red-green-blue colors to hue-saturation-value.

Tabla 3.3

Al utilizar el comando **help elmat** aparecerá la lista de funciones que se muestra en la Tabla 3.4 :

Elementary matrices.	
zeros	- Zeros array.
ones	- Ones array.
eye	- Identity matrix.
repmat	- Replicate and tile array.
rand	- Uniformly distributed random numbers.
randn	- Normally distributed random numbers.
linspace	- Linearly spaced vector.
logspace	- Logarithmically spaced vector.
freqspace	- Frequency spacing for frequency response.
meshgrid	- X and Y arrays for 3-D plots.
:	- Regularly spaced vector and index into matrix.
Basic array information	
size	- Size of array.
length	- Length of vector.
ndims	- Number of dimensions.
numel	- Number of elements.
disp	- Display matrix or text.
isempty	- True for empty array.
isequal	- True if arrays are numerically equal.
isequalwithequalnans	- True if arrays are numerically equal.

isnumeric	- True for numeric arrays.
islogical	- True for logical array.
logical	- Convert numeric values to logical.
Matrix manipulation.	
cat	- Concatenate arrays.
reshape	- Change size.
diag	- Diagonal matrices and diagonals of matrix.
blkdiag	- Block diagonal concatenation.
tril	- Extract lower triangular part.
triu	- Extract upper triangular part.
fliplr	- Flip matrix in left/right direction.
flipud	- Flip matrix in up/down direction.
flipdim	- Flip matrix along specified dimension.
rot90	- Rotate matrix 90 degrees.
:	- Regularly spaced vector and index into matrix.
find	- Find indices of nonzero elements.
end	- Last index.
sub2ind	- Linear index from multiple subscripts.
ind2sub	- Multiple subscripts from linear index.
Multi-dimensional array functions.	
ndgrid	- Generate arrays for N-D functions and interpolation.
permute	- Permute array dimensions.
ipermute	- Inverse permute array dimensions.
shiftdim	- Shift dimensions.
circshift	- Shift array circularly.
squeeze	- Remove singleton dimensions.
Special variables and constants.	
ans	- Most recent answer.
eps	- Floating point relative accuracy.
realmax	- Largest positive floating point number.
realmin	- Smallest positive floating point number.
pi	- 3.1415926535897....
i, j	- Imaginary unit.
inf	- Infinity.
NaN	- Not-a-Number.
isnan	- True for Not-a-Number.
isinf	- True for infinite elements.
isfinite	- True for finite elements.
why	- Succinct answer.
Specialized matrices.	
compan	- Companion matrix.
gallery	- Higham test matrices.
hadamard	- Hadamard matrix.

hankel	- Hankel matrix.
hilb	- Hilbert matrix.
invhilb	- Inverse Hilbert matrix.
magic	- Magic square.
pascal	- Pascal matrix.
rosser	- Classic symmetric eigenvalue test problem
toeplitz	- Toeplitz matrix.
vander	- Vandermonde matrix.
wilkinson	- Wilkinson's eigenvalue test matrix.

Tabla 3.4

Varias funciones entregan resultados constantes bastante útiles, como los que se muestran en la Tabla 3.5

pi	3.14159265...
i	Imaginary unit, $\sqrt{-1}$
j	Same as i
eps	Floating-point relative precision, 2^{-52}
realmin	Smallest floating-point number, 2^{-1022}
realmax	Largest floating-point number, $(2-\varepsilon)2^{1023}$
Inf	Infinity
NaN	Not-a-number

Tabla 3.5

Los nombres de las funciones no son reservados. Es posible sobrescribirlos por una nueva variable, como por ejemplo:

>> eps =1e-6

La función original puede ser restituida por el comando:

>> clear eps

3.7. ***Generación de Matrices:***

MATLAB permite generar las cuatro matrices básicas, las cuales son:

zeros	All zeros
ones	All ones
rand	Uniformly distributed random elements
randn	Normally distributed random elements

Tabla 3.6

Algunos ejemplos son:

```
>> Z=zeros(2,4)

Z =

     0     0     0     0
     0     0     0     0

>> F=5*ones(3,3)

F =

     5     5     5
     5     5     5
     5     5     5

>> N = fix(10*rand(1,10))

N =

     9     2     6     4     8     7     4     0     8     4

>> R = randn(4,4)

R =

   -0.4326   -1.1465    0.3273   -0.5883
   -1.6656    1.1909    0.1746    2.1832
    0.1253    1.1892   -0.1867   -0.1364
    0.2877   -0.0376    0.7258    0.1139
```

3.8. ***Concatenación de Matrices:***

La concatenación de matrices permite crear una gran matriz a partir de pequeñas matrices, por ejemplo, consideremos nuevamente la matriz:

```
A =

    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

Ahora generaremos la matriz B de la forma:

>> B = [A A+32; A+48 A+16]

Esto nos dará como resultado la siguiente matriz:

```
B =

    16     2     3    13    48    34    35    45
     5    11    10     8    37    43    42    40
     9     7     6    12    41    39    38    44
     4    14    15     1    36    46    47    33
    64    50    51    61    32    18    19    29
    53    59    58    56    21    27    26    24
    57    55    54    60    25    23    22    28
    52    62    63    49    20    30    31    17
```

3.9. ***Borrando filas y columnas:***

Se pueden borrar las filas o columnas de una matriz utilizando los paréntesis cuadrados, por ejemplo:

Sea

>> X=A

Entonces para borrar la segunda columna de X, se debe usar:

>>X(:,2)=[]

Así nos da que

```
X =

    16     3    13
     5    10     8
     9     6    12
     4    15     1
```

3.10. ***Algebra Lineal:***

Una matriz es un arreglo bi-dimensional que representa una transformación lineal. Las operaciones matemáticas definidas en matrices son las del tipo del algebra lineal.

3.10.1. **Suma**

Consideremos la matriz A definida como:

```
A =

    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

Entonces la suma de la matriz A y su transpuesta dará una matriz simétrica:

```
>> A+A'

ans =

    32     7    12    17
     7    22    17    22
    12    17    12    27
    17    22    27     2
```

3.10.2. **Multiplicación**

Al multiplicar la matriz transpuesta por la matriz original obtenemos nuevamente una matriz simétrica:

```
>> A'*A

ans =

   378   206   212   360
   206   370   368   212
   212   368   370   206
   360   212   206   378
```

3.10.3. **Determinante**

El determinante de la matriz A es cero, lo que indica que esta es una matriz singular:

```
>> d=det(A)

d =

     0
```

3.10.4. **Operación de reglones fila**

La reducción mediante operaciones de reglones filas de A no es una matriz identidad:

```
>> R=rref(A)

R =

     1     0     0     1
     0     1     0     3
     0     0     1    -3
     0     0     0     0
```

3.10.5. **Inversa**

Al ser la matriz singular, implica que no tienen inversa, por lo que al tratar de computar su inversa ocurrirá lo siguiente:

```
>> X=inv(A)
Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 1.306145e-017.
```

El valor de RCOND (número de condicionamiento, el cual es cercano a 1 cuando la matriz está bien condicionada y cercano a 0 cuando está mal condicionada) que entrega es del orden de ***eps***, la precisión relativa de punto flotante, por lo que la inversa obtenida probablemente no será de utilidad.

3.10.6. **Valores Propios**

Los valores propios de la matriz A son:

```
>> e=eig(A)

e =

   34.0000
    8.9443
   -8.9443
    0.0000
```

Uno de los valores propios es cero, lo cual es consecuencia de la singularidad de la matriz A. El mayor valor propio es 34, esto se debe a que un vector propio esta compuesto de unos, como se puede apreciar:

```
>> v=ones(4,1)

v =

     1
     1
     1
     1

>> A*v

ans =

    34
    34
    34
    34
```

La matriz a puede ser normalizada al dividirla por su máximo valor propio:

```
>> P=A/34

P =

    0.4706    0.0588    0.0882    0.3824
    0.1471    0.3235    0.2941    0.2353
    0.2647    0.2059    0.1765    0.3529
    0.1176    0.4118    0.4412    0.0294
```

Este tipo de matrices representa la transición de probabilidades en los procesos de Markov.

3.10.7. **Potencia**

Repetidas potencias de la matriz P representan pasos del proceso de Markov. Por ejemplo, la quinta potencia:

```
>> P^5

ans =

    0.2511    0.2491    0.2492    0.2506
    0.2495    0.2504    0.2502    0.2499
    0.2501    0.2498    0.2496    0.2505
    0.2494    0.2508    0.2509    0.2489
```

De este resultado se puede apreciar que si la potencia k tiende a infinito, los elementos de la matriz P^k tienden a ¼.

3.10.8. **Polinomio característico**

Los coeficientes del polinomio característico de la matriz A pueden ser obtenidos de la siguiente manera:

```
>> poly(A)

ans =

  1.0e+003 *

    0.0010   -0.0340   -0.0800    2.7200   -0.0000
```

Esto nos indica que el polinomio característico de $\det(A-\lambda I)=0$ es:

$$\lambda^4 - 34\cdot\lambda^3 - 80\cdot\lambda^2 + 2720\cdot\lambda$$

El término constante es cero porque la matriz es singular.

4. **Arreglos**

La diferencia entre una matriz y un arreglo es que en esta última las operaciones son realizadas sobre cada elemento independiente, de tal manera que la suma y la resta son iguales que en matrices, pero las operaciones multiplicativas son diferentes. Esto nos permite realizar operaciones elemento por elemento. MATLAB utiliza el punto, o el punto decimal para las operaciones multiplicativas en los arreglos.

En la Tabla 4.1 se presenta una lista que muestra los operadores

+	Suma
-	Resta
.*	Multiplicación Elemento por elemento
./	División Elemento por elemento
.\	División por la izquierda Elemento por elemento
.^	Potencia Elemento por elemento
.‘	Transpuesta no conjugado Elemento por elemento

Tabla 4.1

4.1. *Operaciones Multiplicativas*

4.1.1. **Multiplicación**

Consideremos nuevamente la matriz A

```
A =

    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

Si se multiplica la matriz A como arreglo por ella misma se obtiene:

```
>> A.*A

ans =

   256     4     9   169
    25   121   100    64
    81    49    36   144
    16   196   225     1
```

Se puede apreciar que cada elemento resultante es el cuadrado de cada elemento de la matriz A.

4.1.2. **División**

Para dividir cada elemento de la matriz A por los elementos de ella misma se debe realizar la siguiente operación:

```
>> A./A

ans =

     1     1     1     1
     1     1     1     1
     1     1     1     1
     1     1     1     1
```

4.1.3. **Potencia**

Para elevar cada elemento de la matriz A a la potencia deseada, simplemente se debe realizar la siguiente operación:

```
>> A.^3

ans =

        4096           8          27        2197
         125        1331        1000         512
         729         343         216        1728
          64        2744        3375           1
```

Se puede apreciar que cada elemento de la matriz A esta elevado al cubo.

4.2. ***Construcción de Tablas***

Las operaciones de Arreglos son útiles para la creación de tablas. Suponga el vector columna n de la forma:

```
>> n=(1:9)'

n =

     1
     2
     3
     4
     5
     6
     7
     8
     9
```

Entonces la operación >> tabla = [n n.^2 2.^n] creará la tabla que contiene los cuadrados de n y las potencias de 2.

```
>> tabla=[n n.^2 2.^n]

tabla =

     1     1     2
     2     4     4
     3     9     8
     4    16    16
     5    25    32
     6    36    64
     7    49   128
     8    64   256
     9    81   512
```

Las funciones matemáticas elementales también operan elemento por elemento. Por ejemplo:

```
>> x=(1:0.1:2)';
>> logs=[x log10(x)]

logs =

    1.0000         0
    1.1000    0.0414
    1.2000    0.0792
    1.3000    0.1139
    1.4000    0.1461
    1.5000    0.1761
    1.6000    0.2041
    1.7000    0.2304
    1.8000    0.2553
    1.9000    0.2788
    2.0000    0.3010
```

En este ejemplo se genera una tabla que contiene los valores de x y su respectivo logaritmo en base 10.

4.3. *Datos multivariable*

MATLAB permite realizar análisis estadístico multivariable. Cada columna de un conjunto de datos representa una variable y cada fila una observación. El elemento (i,j) es la i-esima observación de la j-esima variable.

Como ejemplo, consideremos el conjunto de tres variables:

- Frecuencia cardiaca
- Peso
- Horas de ejercicio por semana

Para 5 horas de observación el arreglo resultante podría ser de la forma:

```
D =
        72          134          3.2
        81          201          3.5
        69          156          7.1
        82          148          2.4
        75          170          1.2
```

La primera fila contiene la frecuencia cardiaca, el peso y las horas de ejercicio por semana del primer paciente, respectivamente. La segunda fila contiene la frecuencia cardiaca, el peso y las horas de ejercicio por semana del segundo paciente, y así sucesivamente.

Ahora, si se desea obtener la media y la desviación estándar de cada columna, se puede hacer mediante los siguientes comandos:

```
>> mu=mean(D)

mu =

   75.8000  161.8000    3.4800

>> sigma=std(D)

sigma =

    5.6303   25.4990    2.2107
```

5. Gráficos

La Tabla 5.1 muestra los pasos necesarios para la construcción de un gráfico básico:

Paso	Código típico

1. Preparar los datos	`x = 0:0.2:12;`
	`y1 = bessel(1,x);`
	`y2 = bessel(2,x);`
	`y3 = bessel(3,x);`

2. Seleccionar una ventana y la posición del gráfico dentro de la ventana	`figure(1)` `subplot(2,2,1)`
3. Llamar a la función elemental de gráfico	`h = plot(x,y1,x,y2,x,y3);`
4. Seleccionar las características de la línea y del marcador	`set(h,'LineWidth',2,{'LineStyle'},{'--';':';'-.'})` `set(h,{'Color'},{'r';'g';'b'})`
5. Seleccionar los ejes de los límites y la grilla	`axis([0 12 -0.5 1])` `grid on`

6. Anotar la rotulación, leyenda y texto del gráfico	`xlabel('Time')`
	`ylabel('Amplitude')`
	`legend(h,'First','Second','Third')`
	`title('Bessel Functions')`
	`[y,ix] = min(y1);`
	`text(x(ix),y,'First Min \rightarrow',...`
	`'HorizontalAlignment','right')`

7. Exportar el gráfico	`print -depsc -tiff -r200 myplot`

Tabla 5.1

5.1. *Gráficos Básicos*

MATLAB provee una variedad de funciones para mostrar los datos de un vector en forma de gráfico de línea. La Tabla 5.2 resume las funciones que producen gráficos de línea básicos:

Función	Descripción
plot	Gráfico 2D con ambos ejes en escala lineal
plot3	Gráfico 3D con ambos ejes en escala lineal
loglog	Gráfico con ambos ejes en escala logarítmica
semilogx	Gráfico con eje x en escala logarítmica y eje y en escala lineal
semilogy	Gráfico con eje y en escala logarítmica y eje x en escala lineal
plotyy	Gráfico con dos marcadores en eje y (izquierda y derecha del gráfico)

Tabla 5.2

5.2. *Creación de gráficos de línea*

La función ***plot*** tiene diferentes formas, dependiendo del argumento de entrada. Por ejemplo, si **y** es un vector, plot(y) produce un gráfico lineal de los elementos de y versus el índice del elemento y.
Si se especifica dos vectores como argumento, plot(x,y) produce un gráfico y versus x.

Ejemplo: Para graficar la función coseno en el intervalo $[0, 2\pi]$ a incrementos de $\pi/100$ se deben implementar el siguiente código:

```
>> t = 0:pi/100:2*pi;
y = cos(t);
plot(t,y)
grid on
```

En la Figura 5-1 se puede apreciar el resultado:

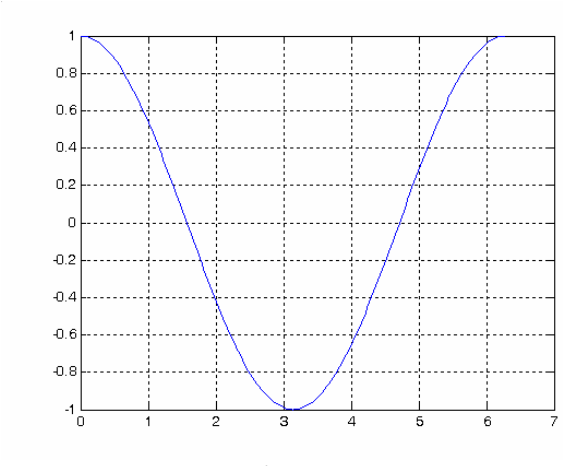


Figura 5-1

También es posible realizar múltiples gráficos usando pares de vectores x-y. MATLAB automáticamente asigna colores para permitir la discriminación entre los datos.

Por ejemplo:

```
>> y2=cos(t-0.25);
>> y3=cos(t-0.5);
>> plot(t,y,t,y2,t,y3)
```

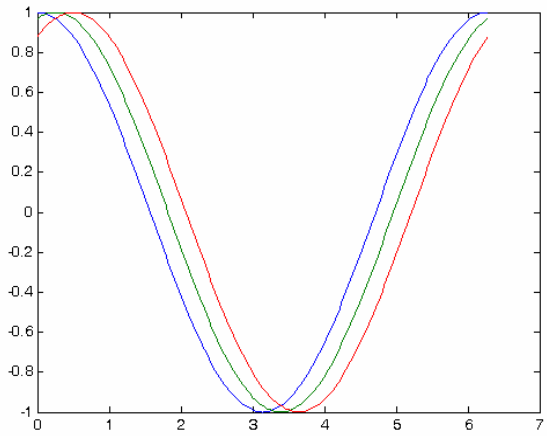


Figura 5-2

5.2.1. **Especificación del estilo de línea**

Es posible especificar diferentes tipos de línea para cada conjunto de datos utilizando el string identificador en la función ***plot***.

Por ejemplo:

```
>> t = 0:pi/100:2*pi;
>> y = cos(t);
>> y2=cos(t-0.25);
>> y3=cos(t-0.5);
>> plot(t,y,'-',t,y2,'--',t,y3,':')
```

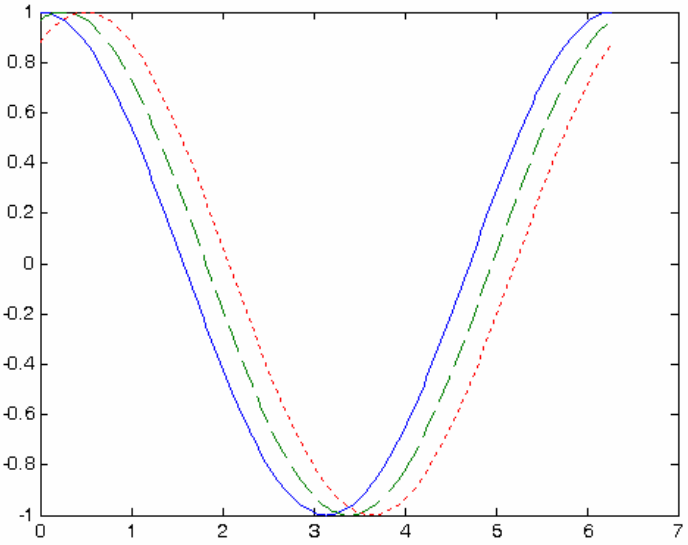


Figura 5-3

5.2.2. **Color, estilo de línea y marcador**

Las funciones básicas de gráfico aceptan como argumento caracteres tipo string que especifican varios tipos de línea, marcadores y colores para cada vector graficado. En forma general es de la siguiente forma:

```
plot(x,y,'linestyle_marker_color')
```

linestyle_marker_color es un string construido de la forma:

- Estilo de línea (por ejemplo, guiones, puntos, etc.)
- Tipo de marcador (por ejemplo, x, *, o, etc.)
- Especificación de color predefinido (c, m, y, k, r, g, b, w)

Por ejemplo:

```
plot(x,y,':squarey')
```

Aquí el gráfico resultante es una línea punteada, con marcadores cuadrados en cada dato, ambos de color amarillo.

Obs:
Si se especifica un marcador, pero no un estilo de línea, MATLAB solamente grafica los marcadores.

La especificación puede consistir de una o ninguna especificación parcial, por ejemplo, la especificación:

'go--'

define una línea de guiones, con marcadores circulares de color verde.

También es posible especificar el tamaño del marcador y, para marcadores que son formas cerradas, es posible especificar separadamente el color del borde y el interior.
Por ejemplo:

```
>>  x = -pi:pi/10:pi;
y = exp(-x).*cos(3*x);
plot(x,y,'--rs','LineWidth',2,...
                'MarkerEdgeColor','k',...
                'MarkerFaceColor','g',...
                'MarkerSize',10)
```

Esta línea de comando produce la Figura 5-4:

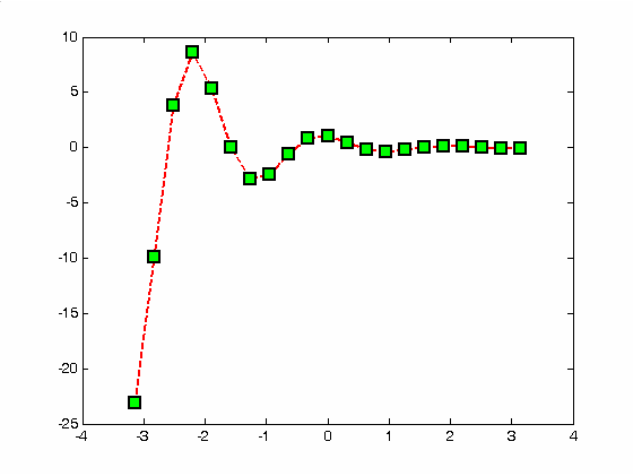


Figura 5-4

5.2.3. **Añadiendo gráficos**

Es posible añadir gráficos nuevos a gráficos existentes utilizando la función ***hold on.*** Al usar esta función, MATLAB no elimina el gráfico existente.
Por ejemplo:

```
>> semilogx(1:100,'+')
hold on
plot(1:3:300,1:100,'--')
hold off
```

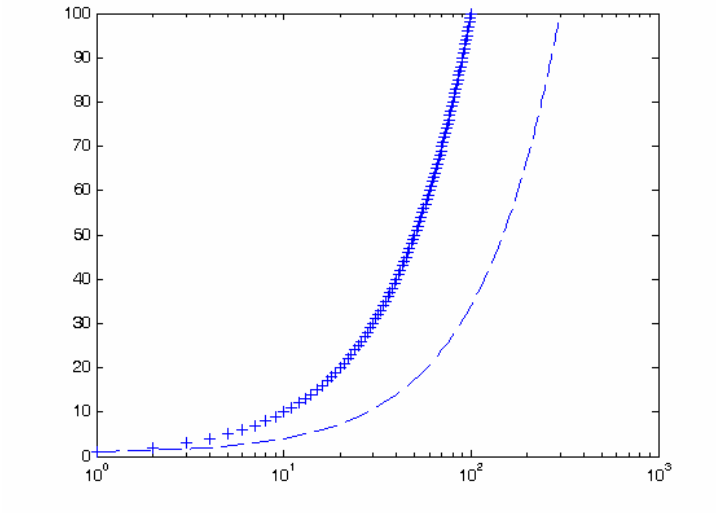


Figura 5-5

En la Figura 5-5 se puede apreciar el resultado de la función ***hold on,*** donde cabe mencionar que si bien, el eje x se acomoda a los nuevos datos, la escala logarítmica se conserva.

5.2.4. **Graficando líneas de matrices de Datos**

La función ***plot*** pude ser utilizada con una matriz como argumento. MATLAB grafica una línea por cada columna de la matriz. El eje x es rotulado con el índice del vector fila, 1: m, donde m es el número de filas de la matriz.

Por ejemplo, considere la matriz A dada por:

```
>> A=sin([(0:pi/16:pi)' (0:pi/16:pi)'+pi/4 (0:pi/16:pi)'+pi/2 (0:pi/16:pi)'+3*pi/4]);
```

La matriz A es de 17x4. Al graficarla obtenemos:

```
>> plot(A)
```

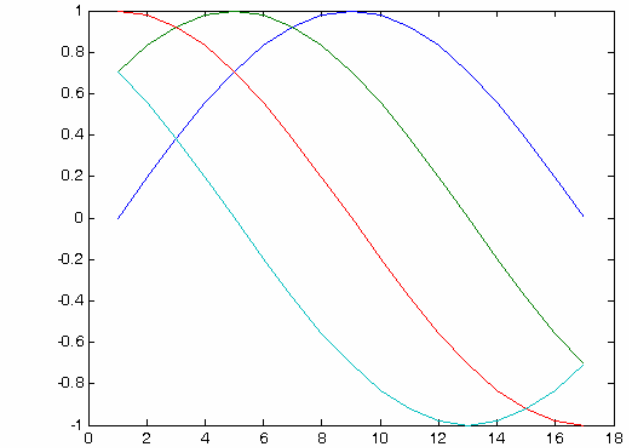


Figura 5-6

También es posible graficar matrices con pares como argumentos, por ejemplo:

```
>> y=1:length(A);
>> plot(y,A,y,A+pi/8,y,A+pi/6)
```

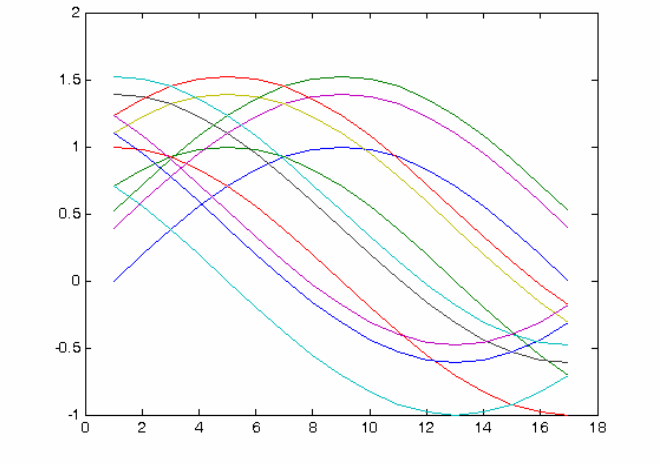


Figura 5-7

5.2.5. **Gráfico de Números complejos**

Cuando los argumentos a graficar son complejos, MATLAB ignora la parte imaginaria excepto cuando el argumento es un único argumento complejo. Para este caso especial el comando es equivalente a graficar la parte real versus la parte imaginaria, es decir, si Z es un vector o matriz complejo, la función ***plot(Z)*** es equivalente a ***plot(real(Z),imag(Z)).***

Por ejemplo:

```
>> plot(eig(randn(20,20)),'o','MarkerSize',6)
```

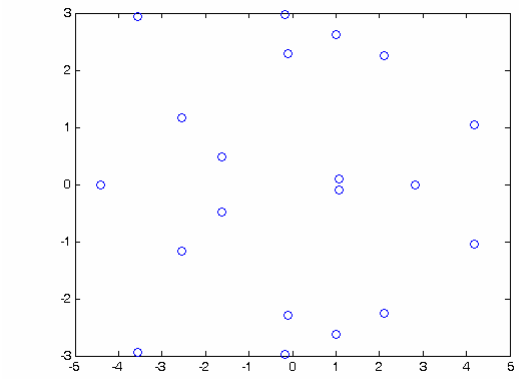


Figura 5-8

5.2.6. **Límites de los ejes**

Es posible especificar manualmente los límites de los ejes utilizando la función ***axis***, tal como se muestra:

```
axis([xmin,xmax,ymin,ymax])
```

En el siguiente ejemplo queda más clara su utilización:

```
>> x=0:pi/16:2*pi;
>> plot(x,sin(x));
>> axis([0 pi -1 1])
```

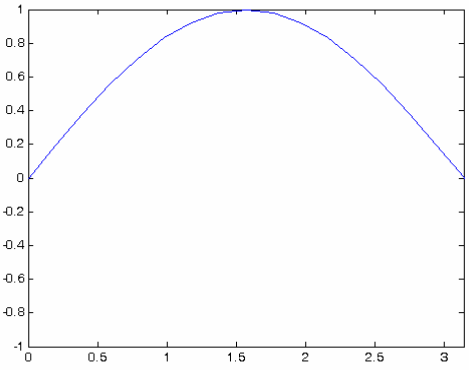


Figura 5-9

5.2.7. **Ventanas de Figuras y múltiples gráficos por figuras**

Al momento de realizar un gráfico, éste aparece en una ventana llamada ***figure.*** Para crear ventanas de figuras se debe utilizar la función ***figure*** de la siguiente manera:

```
>> figure(n)
```

Donde **n** es un entero.

Es posible crear varios gráficos dentro de una ventana ***figure.*** Esto es posible de realizar mediante la función ***subplot***. Esta función es llamada de la forma:

```
subplot(m,n,i)
```

El argumento **m,n** divide la ventana de figura en una matriz de **m** por **n** sub-gráficos, y el argumento **i** selecciona el i-esimo gráfico. Para una mejor comprensión observar los siguientes ejemplos:

```
>> t = 0:pi/20:2*pi;
[x,y] = meshgrid(t);
subplot(2,2,1)
plot(sin(t),cos(t))
axis equal
subplot(2,2,2)
z = sin(x)+cos(y);
plot(t,z)
axis([0 2*pi -2 2])
subplot(2,2,3)
z = sin(x).*cos(y);
plot(t,z)
axis([0 2*pi -1 1])
subplot(2,2,4)
z = (sin(x).^2)-(cos(y).^2);
plot(t,z)
axis([0 2*pi -1 1])
```

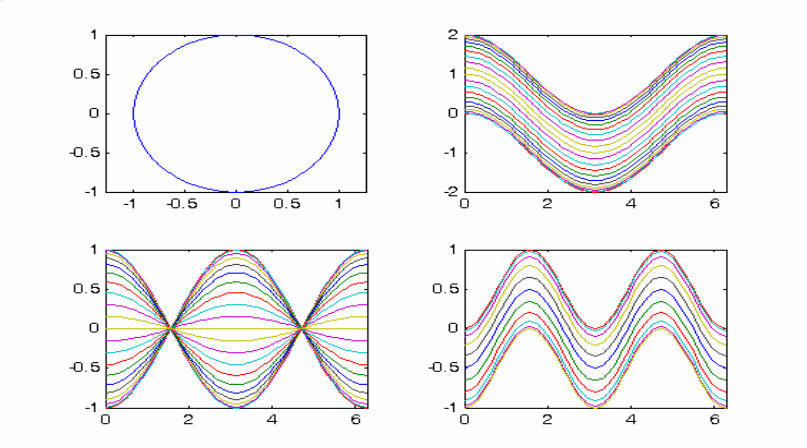


Figura 5-10

5.3. *Creación de gráficos especializados*

5.3.1. **Gráficos de barra**

Los gráficos de barra y área son utilizados para mostrar vectores o matrices de datos. Estos tipos de gráficos son útiles para ver resultados sobre un período de tiempo, comparar los datos de distintos conjuntos de datos, y mostrar como los elementos individuales contribuyen a aumentar cierta cantidad. Los gráficos de barra son útiles para mostrar datos discretos, mientras que los de área son más útiles para mostrar datos continuos.

Función	Descripción
bar	Muestra las columnas de una matriz de **m x n** como **m** grupos de **n** barras verticales
barh	Muestra las columnas de una matriz de **m x n** como **m** grupos de **n** barras horizontales
bar3	Muestra las columnas de una matriz de **m x n** como **m** grupos de **n** barras verticales 3D
bar3h	Muestra las columnas de una matriz de **m x n** como **m** grupos de **n** barras horizontales 3D
area	Muestra vectores de datos como gráficos de área acumulada

Tabla 5.3

5.3.1.1. **Bar**

Por defecto un gráfico de barra representa cada elemento de una matriz como una barra. Las barras en gráficos 2D creadas por la función ***bar*** son distribuidas a lo largo del eje x donde cada elemento en una columna es dibujado en una localización diferente. Los elementos de una fila son agrupados alrededor de una misma localización en el eje x.

Por ejemplo, considere la matriz

```
Y =

     5     2     1
     8     7     3
     9     8     6
     5     5     5
     4     3     2
```

Al aplicar la función bar se obtiene:

```
>> bar(Y)
```

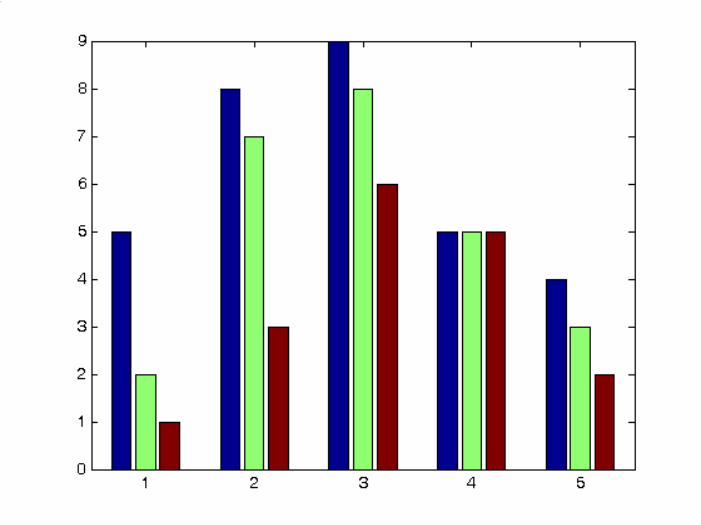


Figura 5-11

5.3.1.2. **Bar3**

La función bar3 es una simple manera de dibujar cada elemento como un bloque 3D por separado, con los elementos de cada columna distribuidos a lo largo del eje del eje y. Barras que representan elementos en la primera columna de la matriz son centrados en 1 a lo largo del eje x.

Por ejemplo:

```
>> bar3(Y)
```

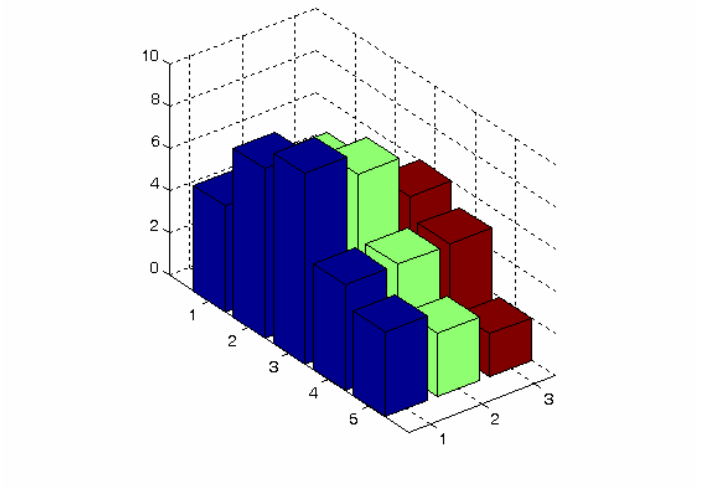


Figura 5-12

También es posible realizar gráficos de barra 3D agrupados. Para hacer esto simplemente hay que indicarlo de la siguiente manera:

```
>> bar3(Y,'group')
```

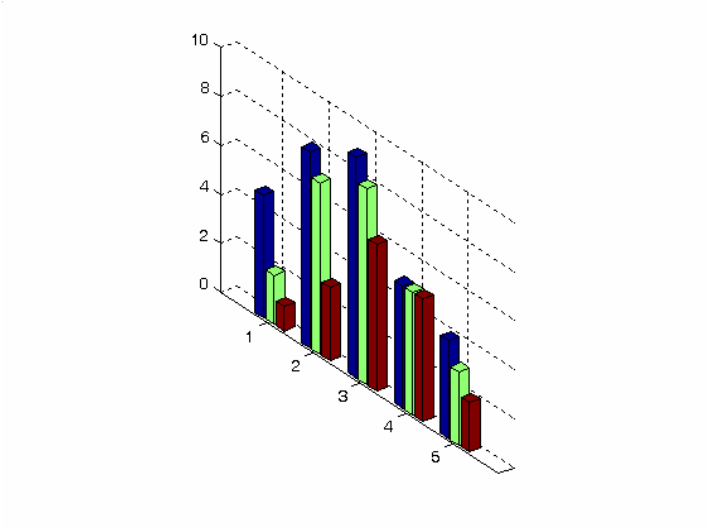


Figura 5-13

También es posible apilar las barras para pode observar cual es la contribución de cada elemento. Para hacer esto se debe hacer lo siguiente:

```
>> bar(Y,'stack')
>> grid on
```

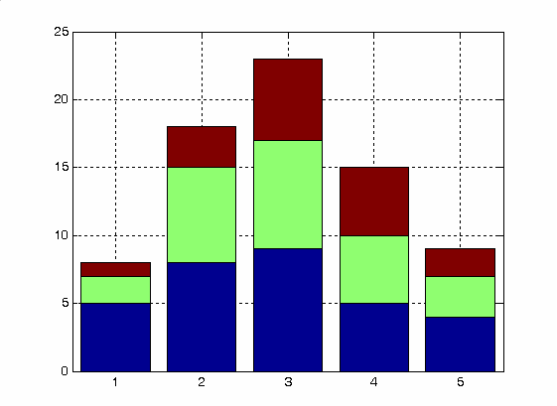


Figura 5-14

También es posible graficar barras con argumento en los ejes x e y. Por ejemplo:

```
>> dias=1:30;
ruido=60*rand(1,30)+45;
bar(dias,ruido)
xlabel('Dias monitoreados')
ylabel('Nivel de Ruido')
>> grid on
```

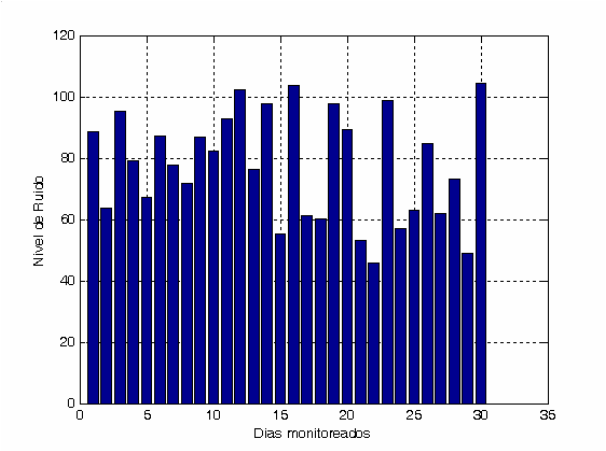


Figura 5-15

5.3.2. **Gráficos de área**

La función ***area*** muestra curvas generadas a partir de los datos de un vector o de las columnas de una matriz. La función ***area*** dibuja los valores de cada columna de la matriz como curvas separadas y llena el área entre las curvas y el eje x.

Graficar áreas es útil para ver como los elementos en un vector o matriz contribuyen a la suma de todos los elementos en un particular valor del eje x. Por defecto, la función ***area*** acumula todos los valores de cada fila en una matriz y crea una curva con esos valores.

Por ejemplo, considere la siguiente matriz

```
Y =

     5     1     2
     8     3     7
     9     6     8
     5     5     5
     4     2     3

>> area(Y)
```

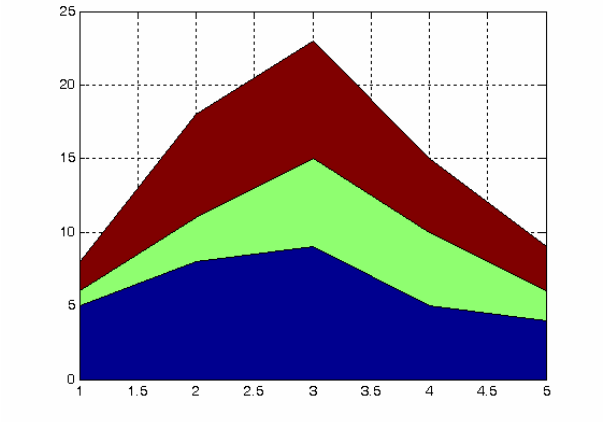


Figura 5-16

5.3.3. **Histogramas**

La función ***hist*** muestra la distribución de los elementos en Y como un histograma dividido en partes iguales entre los valores mínimos y máximos en Y. Si Y es un vector y es el único argumento el histograma se divide en 10 partes.

Por ejemplo:

```
>> y=randn(10000,1);
>> hist(y)
```

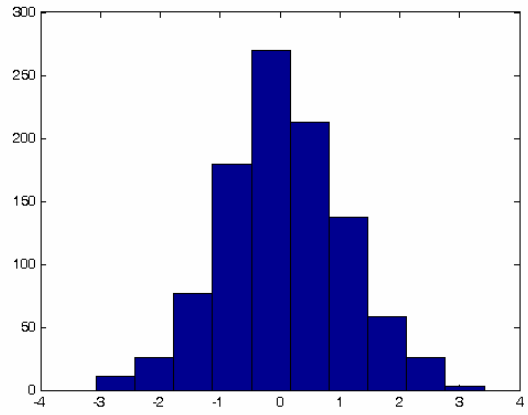


Figura 5-17

Cuando Y es una matriz, la función ***hist*** crea un conjunto de porciones por cada columna, mostrando cada conjunto con colores diferentes.

Por ejemplo:

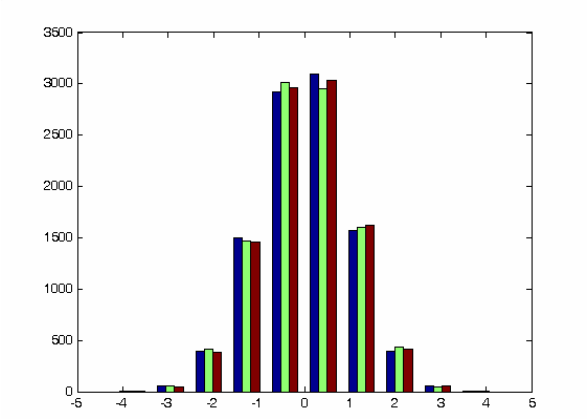


Figura 5-18

Si se desea dividir el histograma en más partes, se debe indicar como se muestra a continuación:

```
>> ynormal=randn(10000,1);
>> yuniforme=rand(10000,1);
>> subplot(1,2,1)
>> hist(ynormal,30)
>> title('Distribucion Normal')
>> subplot(1,2,2)
>> hist(yuniforme,30)
>> title('Distribucion uniforme')
```

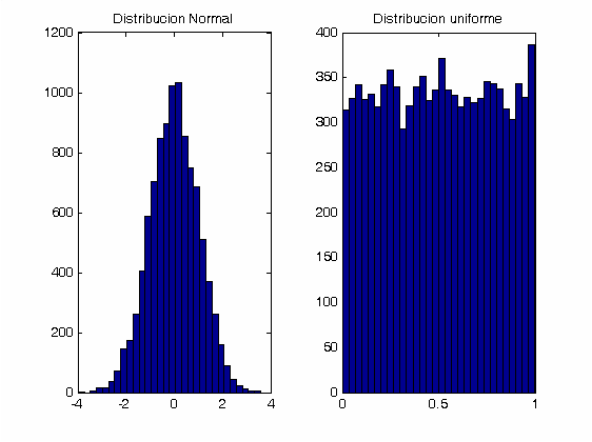


Figura 5-19

En este caso ambos histogramas fueron divididos en 30 partes.

5.3.4. **Gráficos Polares**

La función ***polar*** permite crear gráficos polares a partir de las coordenadas del ángulo y del radio de la forma ***polar(theta,radio)***.

Por ejemplo:

```
>> t = 0:.01:2*pi;
polar(t,sin(2*t).*cos(2*t),'--r')
```

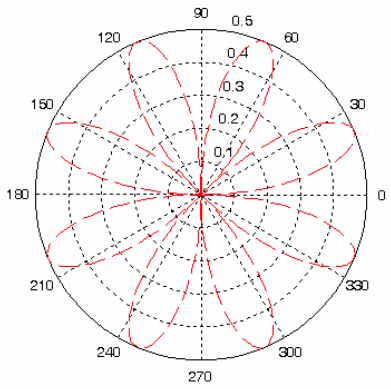


Figura 5-20

5.4. *Gráfico de datos discretos*

5.4.1. **Stem**

La función ***stem*** muestra la secuencia discreta bidimensional. Por ejemplo, a función $y = e^{-\alpha t} \cdot \cos(\omega \cdot t)$:

```
>> alpha = .02; beta = .5; t = 0:4:200;
y = exp(-alpha*t).*sin(beta*t);
>> stem(t,y)
>> xlabel('Tiempo')
>> ylabel('Amplitud')
```

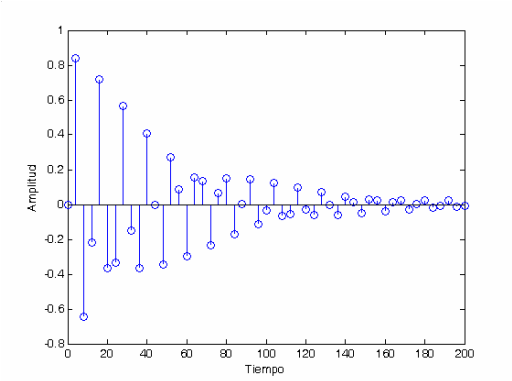


Figura 5-21

5.4.2. **Stem3**

La función ***stem3*** permite realizar gráficos discretos en 3D a partir del plano xy. Por ejemplo:

```
>> t = (0:255)/256*2*pi;
x = cos(t);
y = sin(t);
z=abs(fft(sin(t.^3)));
stem3(x,y,z)
xlabel('eje x')
ylabel('eje y')
zlabel('Magnitud Espectral')
```

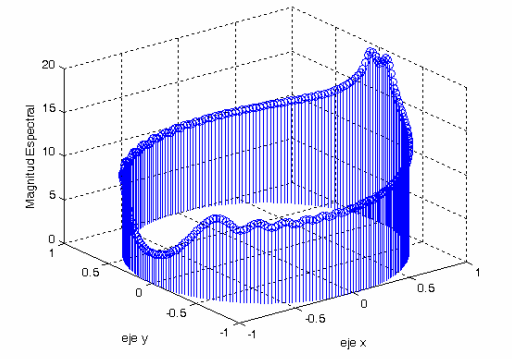


Figura 5-22

5.4.3. **Stairstep**

La función ***stairs*** permite graficar los datos de forma escalonada, como se puede apreciar en el siguiente ejemplo:

```
>> alpha = 0.01;
beta = 0.5;
t = 0:10;
f = exp(-alpha*t).*sin(beta*t);
>> stairs(t,f)
```

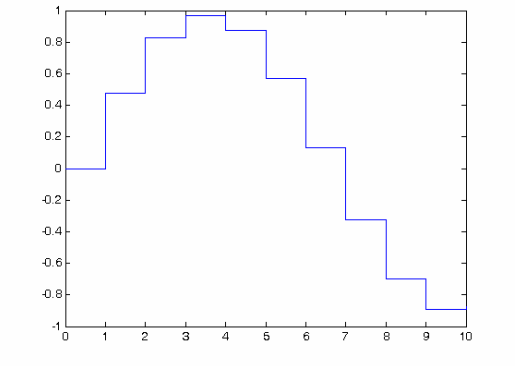


Figura 5-23

5.5. ***Gráficos vectoriales de velocidad y dirección***

Varias funciones de matlab muestran datos que corresponden a vectores de velocidad y dirección.

Algunas de estas funciones se aprecian en la Tabla 5.4:

Función	Descripción
compass	Muestra vectores que comienzan en el origen de un plano en coordenadas polares.
feather	Muestra vectores que se extienden desde el origen de puntos igualmente espaciados a lo largo de la línea horizontal
quiver	Muestra vectores 2D especificados por las componentes (u,v)
quiver3	Muestra vectores 3D especificados por las componentes (u,v,w)

Tabla 5.4

5.5.1. **Compass**

La función ***compass*** muestra vectores que emanan desde el origen de un gráfico. La función toma las coordenadas Caretesianas y las dibuja en una grilla circular.

Ejemplo:

Este gráfico ***compass*** muestra la dirección y la velocidad del viento durante un periodo de 12 horas:

```
>> direccion = [45 90 90 45 360 335 360 270 335 270 335 335];
velocidad= [6 6 8 6 3 9 6 8 9 10 14 12];
direccionRad = direccion * pi/180;
[x,y] = pol2cart(direccionRad,velocidad);
compass(x,y)
>> Info = {'Velocidad y direccion del viento',
        'Universidad Perez Rosales',
        '3 de Enero 2005',
        '14 de Enero 2005'};
text(-28,15,Info)
```

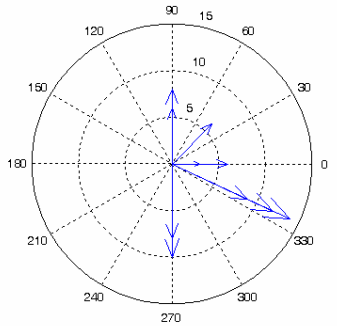


Figura 5-24

Observación: la función ***pol2cart*** transforma las coordenadas polares a cartesianas.

5.5.2. Feather

La función ***feather*** muestra vectores que emanan desde una línea recta paralela al eje x. Por ejemplo, crear vectores de magnitud unitaria que vayan desde los 90° a los 0°.

Nota: Antes de crear el gráfico, se deben transformar los datos a coordenadas cartesianas.

```
>> theta = 90:-10:0;theta = 90:-10:0;
r = ones(size(theta));
>> [u,v] = pol2cart(theta*pi/180,r);
feather(u,v)
axis equal
```

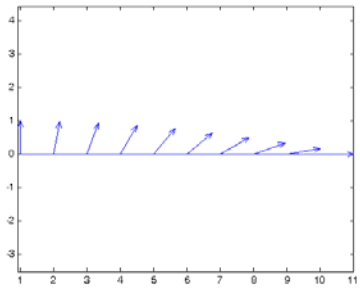


Figura 5-25

Si el argumento es un número complejo **z**, la función ***feather*** interpreta la parte real de **z** como la componente **x** del vector y la parte imaginaria como la componente **y** del vector.

Por ejemplo:

```
>> t = 0:0.5:10;
s = 0.05+i;
Z = exp(-s*t);
feather(Z)
>>
```

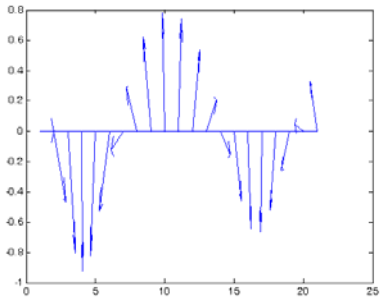


Figura 5-26

5.5.3. Quiver

La función ***quiver*** muestra vectores en un punto dado en un espacio bidimensional. Estos vectores son definidos por las componentes **x** e **y**.
La función ***quiver*** es útil para utilizarla en conjunto con otros gráficos. Por ejemplo, crear 10 contornos de la función ***peaks*** con la función ***contour*.**

```
>> n = -2.0:.2:2.0;
[X,Y,Z] = peaks(n);
contour(X,Y,Z,10)
```

Luego, obtener el gradiente de la función ***peaks*** y trazar los vectores mediante la función ***quiver***:

```
>> [U,V] = gradient(Z,.2);
>> hold on
>> quiver(X,Y,U,V)
```

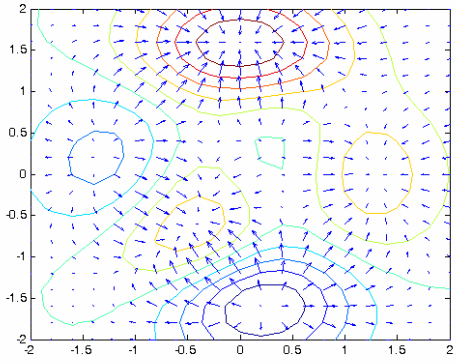


Figura 5-27

5.5.4. **Quiver3**

La función ***quiver3*** muestra los vectores (u,v,w) en las posiciones (x,y,z). Por ejemplo, se puede observar la trayectoria de una partícula como función del tiempo.

$$z(t) = v_z \cdot t + \frac{1}{2} \cdot a \cdot t^2$$

Primero se asignan los valores de los parámetros y luego se obtiene la altura z:

```
>> vz=10;
>> a=-32;
>> t = 0:.1:1;
z = vz*t + 1/2*a*t.^2;
```

Luego se calculan las posiciones en las direcciones **x** e **y**:

```
>> vx = 2;
x = vx*t;
vy = 3;
y = vy*t;
```

A continuación se calculan las velocidades en los respectivos ejes:

```
>> u=gradient(x);
>> v=gradient(y);
>> w=gradient(z);
```

Por último se obtiene el gráfico de la trayectoria del proyectil utilizando la función ***quiever3***.

```
>> scale = 0;
quiver3(x,y,z,u,v,w,scale)
view([70 18])
```

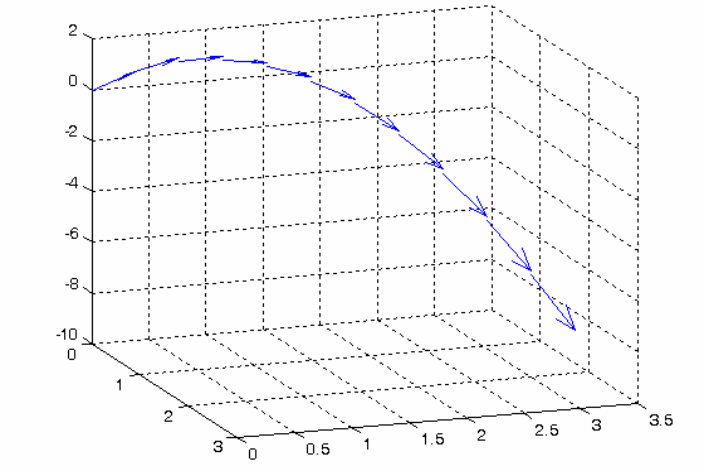


Figura 5-28

5.6. *Gráficos de contornos*

5.6.1. Contour y Contour3

La función ***contour*** y ***contour3*** muestran los contornos en 2D y 3D, respectivamente. Ellas requieren solo una matriz como argumento de entrada, que es interpretada como la altura respecto al plano. Para especificar en número de contornos a visualizar se debe especificar un segundo argumento.

Ejemplo:

```
>> [X,Y,Z] = peaks;
contour(X,Y,Z,20)
```

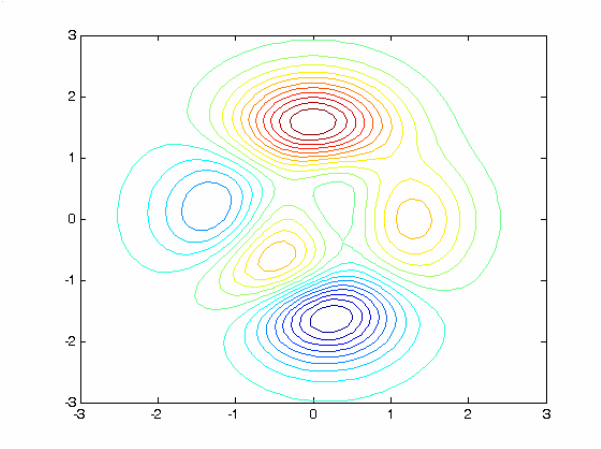


Figura 5-29

O en tres dimensiones

```
>> [X,Y,Z] = peaks;
contour3(X,Y,Z,20)
```

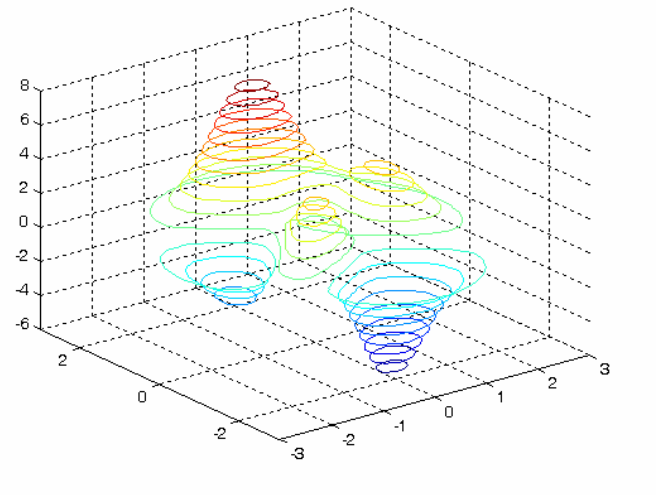


Figura 5-30

Es posible rotular las curvas de contorno de la siguiente forma:

```
>> [X,Y,Z] = peaks;
>> [C,h]=contour(X,Y,Z,10);
>> clabel(C,h)
```

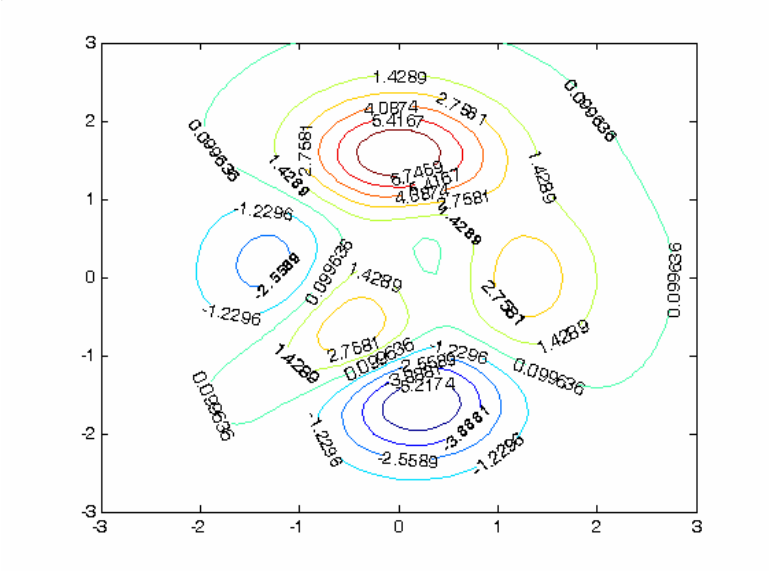


Figura 5-31

5.7. *Gráficos 3D*

5.7.1. **Plot3**

La función ***plot3*** permite graficar los vectores (x,y,z) de forma similar a la función ***plot***. Por ejemplo:

```
>> t = 0:pi/50:10*pi;
plot3(sin(t),cos(t),t)
axis square; grid on
```

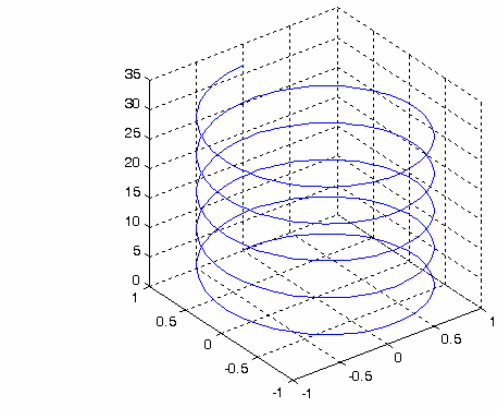


Figura 5-32

Si los argumentos de la función ***plot3*** son matrices del mismo tamaño, MATLAB gráfica las líneas obtenidas de las columnas X, Y y Z. Por ejemplo:

```
>> [X,Y] = meshgrid([-2:0.1:2]);
Z = X.*exp(-X.^2-Y.^2);
plot3(X,Y,Z)
grid on
```

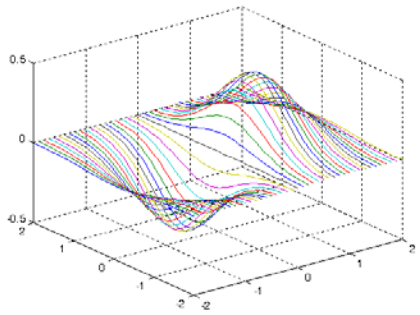


Figura 5-33

5.7.2. **Mesh y Surf**

Las funciones ***mesh*** y ***surf*** crean una superficie 3D de una Matriz de datos. Si Z es una matriz para la cual los elementos Z(i,j) definen la altura de la superficie sobre la grilla (i,j) entonces ***mesh(Z)*** y ***surf(Z)*** generan superficies coloreadas en 3D. Por ejemplo:

```
>> [X,Y] = meshgrid([-2:0.1:2]);
>> Z = X.*exp(-X.^2-Y.^2);
>> subplot(1,2,1)
>> mesh(Z)
>> title('Funcion Mesh')
>> subplot(1,2,2)
>> surf(Z)
>> title('Funcion Surf')
```

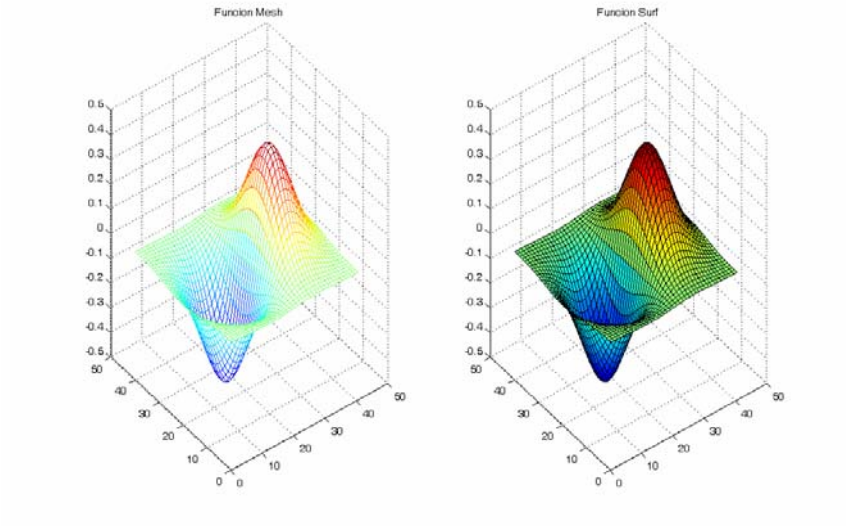


Figura 5-34

También es posible crear imágenes con texturas como se aprecia en el siguiente ejemplo:

```
>> surf(X,Y,Z,'FaceColor','interp',...
    'EdgeColor','none',...
    'FaceLighting','phong')
daspect([5 5 1])
axis tight
view(-50,30)
camlight left
```

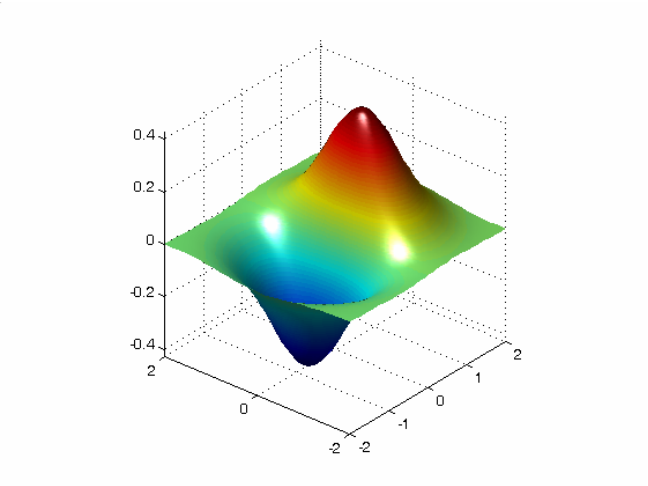


Figura 5-35

5.7.3. **Visualización de variables de dos dimensiones**

El primer paso para visualizar funciones de dos variables, $z = f(x, y)$ es generar las matrices X, Y, las cuales consisten en filas y columnas repetidas, respectivamente. Luego se utilizan estas matrices para evaluar y graficar la función deseada.

La función ***meshgrid*** transforma el dominio especificado por los vectores **x** e **y**, en las matrices X e Y. Cuando se utilizan estas matrices para evaluar funciones de dos variables, las filas de X son copias del vector **x** y las columnas de Y son copias del vector **y**.

Para ilustrar la utilización de la función ***meshgrid*** considere la función $z(x, y) = \sin(x^2 + y^2)/(\sqrt{x^2 + y^2} + eps)$. Para evaluar esta función entre -10 y 10 en ambos ejes, se debe dar el siguiente argumento a la función ***meshgrid***:

```
>> [X,Y] = meshgrid(-10:.5:10);
R = sqrt(X.^2 + Y.^2) + eps;
>> Z = sin(R)./R;
mesh(X,Y,Z)
```

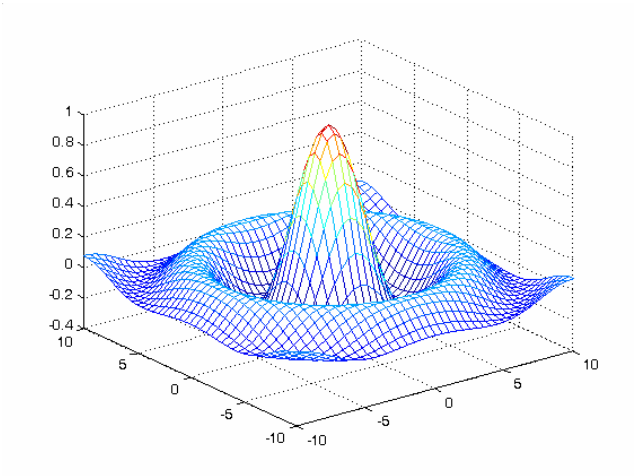


Figura 5-36

5.8. *Animaciones*

Es posible grabar cualquier secuencia de gráficos y luego reproducirlas como una película. Para lograr esto se requieren 2 pasos:

- Utilizar la función ***getframe*** para generar cada cuadro de la película
- Utilizar la función ***movie*** para reproducir la película el número de veces especificados

Como ejemplo, considere el siguiente código:

```
for t=1:50;
    [X,Y]=meshgrid([-3:.1:3]);
    surf(X,Y,exp(-t/6)*sin(t*(X.^2+Y.^2)),'FaceColor','interp',...
                            'EdgeColor','none',...
                            'FaceLighting','phong')
    daspect([5 5 1])
    axis tight
    view(-50+3*t,30)
    camlight left
    mov(t)=getframe;
end
movie(mov,10)
```

También es posible crear archivos de extensión **avi**. La forma de hacerlo es la siguiente:

1. Creación de objeto **avi**: para crear el objeto **avi** se debe utilizar la función objetoavi=AVIFILE(nombre_del_archivo)

2. Seleccionar las propiedades en caso de que se no se deseen utilizar los valores por defecto:

   objetoavi=AVIFILE(nombre_del_archivo,'nombre_propiedad',valor,'nombre_prop',valor).

   Las propiedades a configurar son:

Propiedad	Valores
FPS	Por defecto 15
COMPRESSION	'Indeo3', 'Indeo5', 'Cinepak', 'MSVC', 'RLE' o 'None'
QUALITY	Entre 0-100. Por defecto 75
KEYFRAME	Por defecto 2 key frame por segundo
COLORMAP	Matriz de tres columnas.Debe ser menor a 256 (menor a 236 para compresión Indeo)
NAME	Menor a 64 caracteres, por defecto es el nombre del archivo

Tabla 5.5

3. Cerrar el objeto **avi** utilizando la función ***close***.

El siguiente ejemplo ilustra la creación de un archivo **avi** configurado por defecto:

```
mov = avifile('ejemplo.avi')
for k=1:100;
    plot3(rand(1,1),rand(1,1),rand(1,1),'r-o');
    axis([-1 1 -1 1 -1 1]);
    grid on
    M=getframe;
    mov = addframe(mov,M);
end;
mov = close(mov);
```

El siguiente ejemplo ilustra la creación de un archivo **avi** con la configuración de sus propiedades:

```
mov = avifile('ejemplo.avi','FPS',20,'QUALITY',100)
for k=1:100;
    plot3(rand(1,1),rand(1,1),rand(1,1),'r-o');
    axis([-1 1 -1 1 -1 1]);
    grid on
    M=getframe;
    mov = addframe(mov,M);
end;
mov = close(mov);
```

6. Objetos simbólicos

Un objeto simbólico es una estructura de datos que almacena una representación **string** de un símbolo. Este permite representar variables, matrices y expresiones de forma simbólica.

Para declarar una variable como simbólica se utiliza la función ***sym***, o en el caso de varias variables ***syms***.

El siguiente ejemplo permite observar la diferencia entre una variable estándar de MATLAB y una variable simbólica.

```
>> sqrt(2)

ans =

    1.4142

>> sqrt(sym(2))

ans =

2^(1/2)
```

Además es posible escribir expresiones y realizar diferentes operaciones, como se ilustra en el siguiente ejemplo:

```
>> rho = sym('(1 + sqrt(5))/2')

rho =

(1 + sqrt(5))/2

>> f = rho^2 - rho - 1

f =

(1/2+1/2*5^(1/2))^2-3/2-1/2*5^(1/2)

>> simplify(f)
```

Como segundo ejemplo, se puede generar el polinomio simbólico $f(x) = a \cdot x^2 + b \cdot x + c$:

```
>> syms a b c x
>> f=a*x^2+b*x+c

f =

a*x^2+b*x+c
```

En el caso que se desee convertir una expresión numérica a simbólica se debe especificar como se explica en el siguiente ejemplo:

```
t =

    0.1000

>> sym(t,'r')

ans =

1/10
```

También es posible convertir matrices a una forma simbólica, como es aprecia en el siguiente ejemplo:

```
A =

    1.0000    0.5000    0.3333
    0.5000    0.3333    0.2500
    0.3333    0.2500    0.2000

>> A=sym(A)

A =

[          1,        1/2, 3333/10000]
[        1/2, 3333/10000,        1/4]
[ 3333/10000,        1/4,        1/5]
```

6.1. *Construcción de variables reales y complejas*

El comando ***sym*** las propiedades matemáticas que posee el objeto simbólico utilizando la opción 'real', como es explica en el siguiente ejemplo:

```
>> x = sym('x','real'); y = sym('y','real');
```

O más eficientemente

```
>> syms x y real
z = x + i*y

z =

x+i*y
```

La definición de una símbolo como real permite que la expresión
```
>> f = x^2 + y^2
```

sea estrictamente no negativa. Por lo tanto, z es una variable compleja "formal" y puede ser manipulada como tal.
De esta manera, los comandos

```
>> conj(x), conj(z), expand(z*conj(z))
```

retornan los conjugados complejos de las variables:

```
ans =

x


ans =

x-i*y


ans =

x^2+y^2
```

6.2. ***Creación de funciones abstractas y sustitución de variables***

Si se desea crear una función abstracta de la forma $f(x)$, se debe escribir:

```
>> f = sym('f(x)')
```

Entonces f actúa como $f(x)$, por lo que puede ser manipulada como una variable simbólica. Por ejemplo, para construir el diferencial de $f(x)$ se debe escribir:

```
>> df = (subs(f,'x','x+h') - f)/'h'

df =

(f(x+h)-f(x))/h
```

O también

```
>> syms x h
df = (subs(f,x,x+h)-f)/h

df =

(f(x+h)-f(x))/h
```

Observación: la función ***subs*** sustituye el argumento indicado en la función abstracta.

Si se desean sustituciones múltiples, sebe ser realizado como se expresa en el siguiente ejemplo:

```
>> y=x^2+r^3

y =

x^2+r^3

>> subs(y,{x,r},{3,5})

ans =

   134
```

6.3. *Creación de funciones matemáticas simbólicas*

Existen dos formas de crear funciones simbólicas:

- Usando expresiones simbólicas
- Creando un archivo M

6.3.1. Usando expresiones simbólicas

La secuencia de comandos

```
>> syms x y z
r = sqrt(x^2 + y^2 + z^2)
t = atan(y/x)
f = sin(x*y)/(x*y)
```

Genera las expresiones simbólicas r, t y f. Se pueden utilizar las diferentes funciones matemáticas simbólicas, tales como ***diff***, ***int***, ***subs*** entre otras.

6.3.2. Creando un archivo M

Los archivos M permiten un uso mas general de funciones. Por ejemplo, suponga que desea crear la función simbólica $\sin(x)/x$, este puede ser hecho de la siguiente forma:

```
function z = sinc(x)
if isequal(x,sym(0))
   z = 1;
else
   z = sin(x)/x;
end
```

Observación: Las funciones simbólicas deben ser guardadas en la carpeta de archivos simbólicos, la cual se encuentra en **…MATLAB\toolbox\symbolics.**

6.4. *Cálculo*

6.4.1. **Diferenciación**

Para derivar variables simbólicas se puede usar la función, como se aprecia en el siguiente ejemplo:

```
>> syms a x
f = sin(a*x)

f =

sin(a*x)

>> diff(f)

ans =

cos(a*x)*a
```

Si se desea derivar con respecto a otra variable se debe especificar tal como se aprecia en el siguiente ejemplo:

```
>> diff(f,a)

ans =

cos(a*x)*x
```

Si se desea obtener la derivada n-esima se debe especificar tal como se aprecia en el siguiente ejemplo:

```
>> diff(f,x,2)

ans =

-sin(a*x)*a^2
```

6.4.2. **Jacobiano**

Para calcular la matriz de Jacobi se debe utilizar la función ***jacobian*** , La cual es representada matemáticamente como:

$$J = \frac{\partial(x, y, z)}{\partial(r, \lambda, \varphi)}$$

Por ejemplo:

```
>> syms r l f
x = r*cos(l)*cos(f); y = r*cos(l)*sin(f); z = r*sin(l);
J = jacobian([x; y; z], [r l f])

J =

[    cos(l)*cos(f), -r*sin(l)*cos(f), -r*cos(l)*sin(f)]
[    cos(l)*sin(f), -r*sin(l)*sin(f),  r*cos(l)*cos(f)]
[          sin(l),         r*cos(l),                0]
```

Para encontrar el determinante simplificado de la matriz obtenida, se puede utilizar la función ***det*** y ***simple***, como se aprecia a continuación:

```
>> simple(det(J))
```

Que entrega como resultado

```
ans =

-cos(l)*r^2
```

6.4.3. **Límites**

Es posible calcular el límite de una expresión simbólica utilizando la función ***limit***, la cual requiere como argumento la función simbólica, la variable a evaluar y el valor de ésta variable. Por ejemplo:

```
>> syms h n x
limit( (cos(x+h) - cos(x))/h,h,0 )

ans =

-sin(x)
```

6.4.4. **Integración**

Es posible realizar la integración de variables simbólicas mediante la función ***int***, como se aprecia en el siguiente ejemplo:

```
>> syms x a b
>> y=a*sin(b*x)

y =

a*sin(b*x)

>> int(y)

ans =

-a/b*cos(b*x)
```

Si se desea integrar respecto a otra variable debe especificarse como se expresa en el siguiente ejemplo:

```
>> int(y,a)

ans =

1/2*a^2*sin(b*x)
```

En el caso que se desee obtener la integral evaluada dentro de un intervalo, se debe expresar como se aprecia en el siguiente ejemplo:

```
>> int(y,x,0,2*pi)

ans =

-2*a*(cos(pi*b)^2-1)/b
```

6.4.5. **Sumatoria**

Para realizar una sumatoria simbólica se debe utilizar la función ***symsum*** tal como se puede apreciar en el siguiente ejemplo:

```
>> symsum(x^k,k,0,inf)

ans =

-1/(x-1)
```

6.4.6. **Series de Taylor**

Para encontrar la serie de Taylor de una función simbólica se debe utilizar la función ***taylor***, la cual encuentra la serie:

$$f(x) = \sum_{n=0}^{\infty} (x-a)^n \cdot \frac{f^{(n)}(a)}{n!}$$

Por ejemplo:

```
>> syms x
g = exp(x*sin(x))
t = taylor(g,12,2);
```

Donde el número 12 indica que se obtendrán 12 términos diferentes de cero, y el valor 2 indica que se aproximará en torno al valor 2.

6.5. *Gráfico de funciones simbólicas*

Para graficar funciones simbólicas, se pueden utilizar las mismas funciones utilizadas para graficar vectores y matrices, con la salvedad de que estas funciones deben comenzar con la sigla ez, por ejemplo, la función ***ezplot***:

```
>> syms x
g = exp(x*sin(x));
t = taylor(g,12,2);
>> ezplot(t)
```

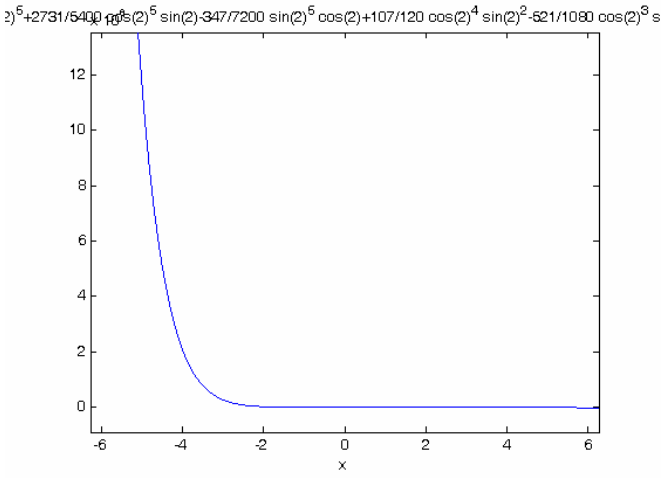


Figura 6-1

O la función ***ezplot3***:

```
>> syms x y z t
>> x=sin(t);
>> y=cos(t);
>> z=t^2;
>> ezplot3(x,y,z,[0,6*pi])
```

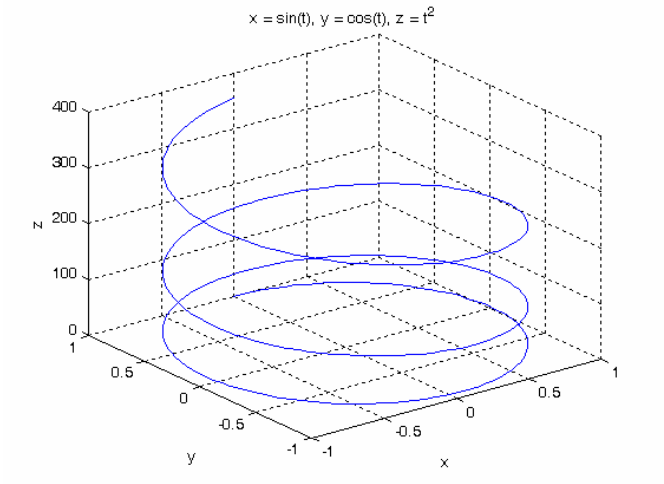


Figura 6-2

7. Bibliografía

- Documentación MATLAB 6.5