Introducción a MATLAB

Segunda Edición

Kermit Sigmon
Department of Mathematics
University of Florida

Traducido del inglés por:
Celestino Montes
Departamento de Matemática Aplicada II
Universidad de Sevilla

Department of Mathematics • University of Florida • Gainesville, FL 32611
sigmon@math.ufl.edu • sigmon@ufpine.bitnet

Department of Mathematics • University of Florida • Gainesville, FL 32611
sigmon@math.ufl.edu • sigmon@ufpine.bitnet

Introducción

MATLAB es un sistema interactivo basado en matrices para cálculos científicos y de ingeniería. Se pueden resolver problemas numéricos relativamente complejos sin escribir un programa en realidad. El nombre MATLAB es una abreviatura para MATrix LABoratory.

El propósito de estas notas es ayudar en la iniciación a MATLAB. La mejor forma de utilizarlas es poner manos a la obra. Se aconseja, en general, trabajar en el ordenador a la vez que se leen las notas, así como a experimentar libremente con ejemplos.

Se puede utilizar la ayuda de la instrucción `help` para una información más detallada. Después de entrar en MATLAB en la forma que se explica en la sección 1, la instrucción `help` mostrará una lista de funciones para las que se puede obtener ayuda mientras se está trabajando; la instrucción `help` *nombre_de_función* nos dará información sobre una función específica. Así, la instrucción `help eig`, nos dará información sobre la función `eig`, que calcula los autovalores de una matriz. Se pueden ver algunas de las capacidades de MATLAB usando la instrucción `demo`.

El alcance y la potencia de MATLAB van más allá de lo que podemos ver en estas notas. En algún momento puede desear una información más detallada. Es el momento de consultar la guía del usuario y la de referencia. La documentación que acompaña a la edición para estudiantes de MATLAB es una fuente excelente. Pueden encontrarse copias de la guía del usuario en las salas de terminales o en las bibliotecas. Consulte a su instructor o en su centro de cálculo acerca de dónde puede encontrarla en su institución.

MATLAB puede conseguirse para los siguientes entornos: Sun/Apollo/VAXstation/HP workstation, VAX, MicroVAX, Gould, compatibles PC y AT, computadores 80386, Apple Macintosh, y varias máquinas en paralelo. Existe una edición para estudiantes relativamente barata publicada por Prentice Hall. La información de estas notas vale generalmente para todos los entornos.

Aunque la introducción está basada en la versión 3.5 de MATLAB, es compatible con la versión 4.0 con las diferencias que se harán notar. Una nueva edición de estas notas basada en la versión 4.0 está en preparación.

El archivo fuente para TEX (y un archivo PostScript `primer.ps`) de la última versión en inglés de estas notas se pueden conseguir vía ftp anónimo en `math.ufl.edu` como el archivo `primer.tex` en el directorio `pub/matlab`. Si no dispone de acceso a ftp, puede obtenerse vía `listserv` enviando un mensaje por correo electrónico a `listserv@math.ufl.edu` con la única línea `send matlab/primer.tex`. También puede obtenerse enviando la petición al autor en `sigmon@math.ufl.edu`. La última versión en español también está disponible allí.

MATLAB está patentado por The MathWorks, Inc., Cochituate Place, 24 Prime Park Way, Natick, MA 01760, (508)653-1415, Fax: (508)653-2997, Email: info@mathworks.com.

Indice

Página

1. Acceso a MATLAB.

Después de entrar a la mayoría de los sistemas, para acceder a MATLAB basta utilizar la instrucción `matlab` y para salir, la instrucción `exit` o `quit`. Por ejemplo si estamos en un PC, salvo que tengamos el programa en un directorio aparte, basta con escribir

```
C> matlab
```

Podemos salir de él con la instrucción:

```
>> quit
```

En sistemas que permiten procesos múltiples, como el UNIX, será conveniente, por razones que se verán en la sección 14, tener activos a la vez MATLAB y el editor local. Si se trabaja en una estación de trabajo con ventanas múltiples será deseable tener activos MATLAB en una ventana y el editor en otra. Los detalles de la instalación local se pueden obtener del centro de cálculo correspondiente, o consultando al instructor.

2. Introducción de matrices.

MATLAB trabaja esencialmente con un solo tipo de objetos: una matriz numérica rectangular con entradas posiblemente complejas; todas las variables representan matrices. A veces, las matrices 1×1 se consideran escalares, y las matrices con una sola fila o columna se consideran como vectores.

Hay varias formas diferentes para introducir una matriz en MATLAB. A saber:

- Introduciendo una lista explícita de elementos,
- Generándola mediante funciones y declaraciones,
- Creándola en un archivo .m (ver secciones 12 y 14),
- Cargándola de un archivo de datos externo (ver Guía del usuario).

Por ejemplo, cualquiera de las declaraciones

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

y

```
A = [
1 2 3
4 5 6
7 8 9 ]
```

crea la matriz 3×3 que se espera y la asigna a una variable A. Inténtelo. Los elementos en una fila de una matriz pueden separarse tanto por comas como por espacios en blanco.

Cuando alguno de los números se escribe en forma exponencial (por ejemplo 2.34e-9), deben evitarse los espacios en blanco. La escritura de una matriz grande debe hacerse preferentemente en un archivo .m, donde es más sencillo corregir errores (ver secciones 12 y 14).

Las funciones internas `rand`, `magic`, y `hilb`, por ejemplo, proporcionan una forma sencilla para crear matrices con las que experimentar. La instrucción `rand(n)`, resp. `rand(m,n)`, creará una matriz $n \times n$, resp. $m \times n$, con entradas aleatoriamente generadas, distribuidas uniformemente entre 0 y 1. `magic(n)` creará una matriz cuadrada

mágica (las filas y las columnas suman la misma cantidad) con entradas enteras; `hilb(n)` creará la matriz de Hilbert de orden n, la reina de las matrices mal condicionadas. m y n, por supuesto, denotan enteros positivos. También se pueden crear matrices utilizando bucles `for` (ver sección 6). Inténtelo.

Las entradas individuales de una matriz o de un vector se pueden obtener poniendo los índices entre paréntesis de la forma usual. Por ejemplo, $A(2,3)$ denota la entrada en la segunda fila y tercera columna de la matriz A y $x(3)$ denota la tercera coordenada del vector x. Inténtelo. Sólo se pueden usar como índices de vectores y de matrices enteros *positivos.*

3. Operaciones con matrices, operaciones a coordenadas.

Disponemos en MATLAB de las siguientes operaciones con matrices:

$+$	adición
$-$	sustracción
$*$	multiplicación
$\hat{}$	potenciación
$'$	traspuesta
$\backslash$	división izquierda
$/$	división derecha

Estas operaciones para matrices se aplican también a escalares (matrices 1×1). Si los tamaños de las matrices son incompatibles para la operación matricial se obtiene un mensaje de error, exceptuando el caso en que uno de los operandos sea un escalar y el otro una matriz (para la adición, sustracción, división y multiplicación). En esta situación se opera el escalar con cada término de la matriz.

La "división matricial" merece un comentario especial. Si A es una matriz invertible y b es una columna, resp. fila, compatible, entonces

$x = A\backslash b$ es la solución de $A * x = b$ y, resp.,
$x = b/A$ es la solución de $x * A = b$.

En la división izquierda, si A es cuadrada, se factoriza utilizando eliminación gaussiana. Con los factores se resuelve $A*x = b$. Si la matriz A no es cuadrada, se factoriza utilizando la ortogonalización de Householder con pivoteo de columnas. Con los factores se resuelve el sistema indeterminado o sobredeterminado en el sentido de los mínimos cuadrados. La división derecha se define a partir de la izquierda por $b/A = (A'\backslash b')'$.

Operaciones a coordenadas. Las operaciones de adición y sustracción operan intrínsecamente a coordenadas pero las otras operaciones matriciales dadas antes no: Son operaciones *matriciales.* Es importante observar que para estas otras operaciones, $*$, $\hat{}$, $\backslash$, y $/$, puede hacerse que operen a coordenadas precediendolas de un punto. Por ejemplo, tanto `[1,2,3,4].*[1,2,3,4]` como `[1,2,3,4].^2` darán `[1,4,9,16]`. Inténtelo. Esto es particularmente útil cuando se utilizan los gráficos de MATLAB.

4. Declaraciones, expresiones y variables; almacenamiento de una sesión.

MATLAB es un lenguaje de *expresiones*; las expresiones que se escriben son interpretadas y evaluadas. Las instrucciones de MATLAB son, normalmente, de la forma

variable = *expresión*, o simplemente
expresión

Las expresiones se componen, usualmente, a partir de operadores, funciones y nombres de variables. La evaluación de una expresión produce una matriz, que se muestra en pantalla, y se asigna a la variable para su posterior uso. Si se omiten la variable y el signo =, se crea una variable llamada `ans` (por answer) a la que se asigna el resultado de la expresión.

Una instrucción termina, normalmente, con el retorno de carro. Si se desea continuar una expresión en la línea siguiente, basta escribir tres (o más) puntos antes del retorno de carro. Si por el contrario, deseamos escribir varias instrucciones en una misma línea, podemos hacerlo separandolas por comas o puntos y comas.

Si el último carácter de una instrucción es un punto y coma el resultado no se mostrará en pantalla, aunque por supuesto se realizará la asignación. Esto es esencial para evitar pérdidas de tiempo al mostrar los resultados intermedios.

MATLAB distingue las letras mayúsculas de las minúsculas en los nombres de instrucciones, funciones y variables. Así, `resolvente` no es lo mismo que `ReSoLvEnTe`.

La instrucción `who` muestra las variables que se encuentran en el espacio de trabajo. Para eliminar una variable de la memoria se utiliza la instrucción `clear` *nombre_variable*. Si se escribe sólo `clear` se borran todas las variables no permanentes.

La variable permanente `eps` (épsilon) da la precisión de la máquina—alrededor de 10^{-16} en la mayoría de ellas. Es útil para determinar la tolerancia en procesos iterativos.

Cualquier tipo de cálculo, gráfico, o impresión puede detenerse sin salir del programa con CTRL-C (CTRL-BREAK en PC).

Almacenamiento de sesiones. Cuando salimos de MATLAB se pierden todas las variables. Para evitarlo se puede utilizar la instrucción `save` antes de salir. Esto hace que las variables se almacenen en el archivo de disco `matlab.mat`. Al acceder de nuevo a MATLAB, se pueden recuperar todas las variables con la instrucción `load`.

5. Funciones para la construcción de matrices.

Las siguientes funciones están disponibles en MATLAB:

eye	matriz identidad
zeros	matriz de ceros
ones	matriz de unos
diag	ver más adelante
triu	parte triangular superior de una matriz
tril	parte triangular inferior de una matriz
rand	matriz generada aleatoriamente
hilb	matriz de Hilbert
magic	matriz mágica
toeplitz	ver `help toeplitz`

Por ejemplo, `zeros(m,n)` produce una matriz nula $m \times n$, y `zeros(n)` produce otra cuadrada de orden n; si A es una matriz, entonces `zeros(A)` produce una matriz de ceros del mismo orden que A.

Si x es un vector, `diag(x)` es la matriz diagonal con x en su diagonal; si A es una matriz cuadrada, `diag(A)` es un vector formado por la diagonal de A. ¿Qué será entonces `diag(diag(A))`? Inténtelo.

Las matrices se pueden construir por bloques. Por ejemplo, si A es 3×3, entonces

```
B = [A, zeros(3,2); zeros(2,3), eye(2)]
```

dará una cierta matriz 5×5. Inténtelo.

6. For, while, if — y relaciones.

Básicamente, las instrucciones para el control de flujo de MATLAB operan como en la mayor parte de los lenguajes usuales.

for. Por ejemplo, las instrucciones

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

o

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

darán como resultado un cierto vector mientras que

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

dará el mismo vector en orden inverso. Pruebe con ellas. Las instrucciones

```
for i = 1:m
   for j = 1:n
      H(i, j) = 1/(i+j-1);
   end
end
H
```

producirán e imprimirán en pantalla la matriz de Hilbert $m \times n$. El punto y coma de la instrucción interior suprime la impresión no deseada de los resultados intermedios mientras que el último `H` muestra el resultado final.

while. La forma general de un bucle `while` es

```
while relación
   instrucciones
end
```

Las instrucciones se repetirán mientras la relación sea cierta. Por ejemplo, dado un número a, las instrucciones siguientes calculan y muestran el menor entero no negativo n tal que $2^n \geq a$:

```
n = 0;
while 2^n < a
   n = n + 1;
end
n
```

if. La forma general de un bucle `if` simple es

```
if relación
   instrucciones
end
```

Las instrucciones se ejecutarán sólo si la relación es cierta. También son posibles las ramificaciones múltiples, como se ilustra con

```
if n < 0
   paridad = 0;
elseif rem(n,2) == 0
   paridad = 2;
else
   paridad = 1;
end
```

Si sólo tenemos dos ramificaciones podemos omitir, desde luego, la porción correspondiente a elseif.

Relaciones. Los operadores relacionales en MATLAB son

$<$	menor que
$>$	mayor que
$<=$	menor o igual que
$>=$	mayor o igual que
$==$	igual
$\sim=$	no igual.

Hagamos notar que se usa "=" en las asignaciones mientras que para las relaciones se usa "==". Las relaciones pueden conectarse o cuantificarse por los operadores lógicos

&	y
|	o
$\sim$	no.

Cuando se aplican a escalares, una relación es realmente el escalar 1 o 0 dependiendo de si la relación es verdadera o falsa: Pruebe con `3 < 5`, `3 > 5`, `3 == 5` y `3 == 3`. Cuando se aplica a matrices del mismo orden, una relación entre ellas da lugar a una matriz de ceros y unos, dando el valor de la relación entre las correspondientes entradas. Pruebe con `a = rand(5), b = triu(a), a == b`.

Cuando se utiliza una relación entre matrices en un bucle `while` o `if`, la relación se entiende verdadera si cada una de las entradas de la matriz de relación es no nula. Por tanto, si se quiere ejecutar *algo* cuando las matrices A y B sean iguales, se puede escribir:

```
if A == B
   algo
end
```

pero si se desea ejecutar la misma instrucción si A y B son distintas, hay que recurrir a:

```
if any(any(A ~= B))
   algo
end
```

o, más simplemente,

```
if A == B else
   algo
end
```

Recalcamos que la aparentemente obvia

```
if A ~= B, algo, end
```

no hará lo que deseamos ya que la instrucción sólo se ejecutará si *todas* las entradas de A son distintas de las de B. Las funciones `any` y `all` pueden utilizarse de forma creativa para reducir relaciones entre matrices a relaciones entre vectores y escalares. Se requieren dos `any`s en el ejemplo anterior ya que `any` es un operador vectorial (ver sección 8).

La instrucción `for` permite usar cualquier matriz en vez de `1:n`. Ver la Guía del usuario para los detalles de cómo esta posibilidad amplía la potencia de la instrucción `for`.

7. Funciones escalares.

Algunas funciones de MATLAB operan esencialmente sobre escalares, aunque lo hacen también sobre matrices (elemento a elemento). Las funciones más comunes entre estas son:

sin	asin	exp	abs	round
cos	acos	log (natural)	sqrt	floor
tan	atan	rem (resto)	sign	ceil

8. Funciones vectoriales.

Otras funciones de MATLAB operan fundamentalmente sobre vectores (fila o columna), aunque también pueden operar sobre matrices $m \times n$ ($m \geq 2$) haciendolo en este caso columna a columna, produciendo, por tanto, un vector fila que contiene el resultado de su aplicación a cada columna. Para conseguir que actúen por filas basta usar la traspuesta; por ejemplo, `mean(A')'`. Veamos algunas de estas funciones:

max	sum	median	any
min	prod	mean	all
sort		std	

Por ejemplo, la entrada máxima de un matriz A se obtiene con `max(max(A))` en vez de `max(A)`. Inténtelo.

9. Funciones matriciales.

Las funciones matriciales más útiles de MATLAB son las siguientes:

eig	autovalores y autovectores
chol	factorización de Cholesky
svd	descomposición en valores singulares
inv	inversa
lu	factorización LU
qr	factorización QR
hess	forma de Hessenberg
schur	descomposición de Schur
rref	forma escalonada reducida por filas
expm	matriz exponencial
sqrtm	matriz raíz cuadrada
poly	polinomio característico
det	determinante
size	tamaño
norm	norma 1, norma 2, norma de Frobenius, norma ∞
cond	número de condición en la norma 2
rank	rango

Las funciones de MATLAB admiten argumentos de salida simples o múltiples. Por ejemplo,

`y = eig(A)`, o simplemente `eig(A)`

genera un vector columna conteniendo los autovalores de A mientras que

```
[U,D] = eig(A)
```

produce una matriz U cuyas columnas son los autovectores de A y una matriz diagonal D con los autovalores de A en su diagonal. Pruebe.

10. Comandos de edición de línea y rellamada.

Es fácil editar la línea de comandos en MATLAB. El cursor se posiciona con las flechas izquierda/derecha mientras que para borrar caracteres pueden usarse las teclas Retroceso o Suprime. También son accesibles otras posibilidades de edición. En un PC pruebe con las teclas Inicio, Fin, y Suprime; en otros sistemas ver `help cedit` o `type cedit`.

Una posibilidad muy útil es usar las flechas arriba/abajo para recuperar los comandos previos. Se puede, por tanto, recuperar una línea de comandos previa, editarla, y ejecutarla revisada. Para pequeñas rutinas, esto es más conveniente que usar un archivo .m lo que requiere moverse entre MATLAB y el editor (ver secciones 12 y 14). Por ejemplo, el número de operaciones (ver sección 15) para el cálculo de la inversa de matrices de varios tamaños podría ser comparado recuperando, editando y ejecutando repetidamente:

```
a = rand(8); flops(0), inv(a); flops
```

Si se desea comparar las gráficas de las funciones $y = \text{sen} mx$ y $y = \cos nx$ en el intervalo $[0, 2\pi]$ para varios m y n, se podría hacer lo mismo con la línea de comandos:

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

11. Submatrices y notación de dos puntos.

Los vectores y submatrices son utilizados a menudo en MATLAB para conseguir efectos de manipulación bastante complejos. La "notación de dos puntos" (que se utiliza para generar vectores y submatrices), y la indexación por vectores son las llaves para una manipulación eficiente de estos objetos. Su uso de forma creativa permite minimizar el número de bucles (que enlentecen a MATLAB) y hacen que las instrucciones sean más simples y legibles. *Debe hacerse un esfuerzo especial para familiarizarse con esta notación.*

La expresión `1:5` (que ya encontramos en los bucles `for`) es realmente un vector fila: el `[1 2 3 4 5]`. Los números no tienen que ser enteros ni el incremento uno. Por ejemplo,

```
0.2:0.2:1.2
```

da como resultado `[0.2 0.4 0.6 0.8 1.0 1.2]`, mientras que con

`5:-1:1` se obtiene el vector `[5 4 3 2 1]`.

Las siguientes instrucciones, por ejemplo, generarán una tabla de senos. Pruebe.

```
x = [0.0:0.1:2.0]';
y = sin(x);
[x y]
```

Hagamos notar que al operar `sin` a coordenadas, produce un vector y a partir de x.

La notación de dos puntos permite acceder a submatrices. Por ejemplo, `A(1:4,3)` es el vector columna con las cuatro primeras entradas de la tercera columna de A.

Dos puntos sin más especificación denotan una fila o columna completa:

`A(:,3)` es la tercera columna de A, y `A(1:4,:)` son las cuatro primeras filas.

Se pueden usar como índices vectores enteros arbitrarios:

`A(:,[2 4])` está formada por las columnas segunda y cuarta de A.

Estos índices se pueden usar a ambos lados de una instrucción de asignación:

`A(:,[2 4 5]) = B(:,1:3)` reemplaza las columnas 2, 4 y 5 de A por las tres primeras de B. Se muestra y asigna la matriz A alterada *completa*. Pruebe.

Las columnas 2 y 4 de A pueden multiplicarse por la derecha por una matriz 2×2:

```
A(:,[2,4]) = A(:,[2,4])*[1 2;3 4]
```

De nuevo se muestra y se asigna la matriz completa.

Si denotamos por x un vector con n componentes, ¿cuál es el efecto de la instrucción `x = x(n:-1:1)`? Haga la prueba.

Para comprobar la utilidad de esta notación, comparar estas instrucciones de MATLAB con una rutina de Pascal, FORTRAN, o C que dé los mismos resultados.

12. Archivos .m.

MATLAB puede ejecutar una sucesión de instrucciones almacenadas en archivos de disco. Estos archivos se denominan "archivos .m", debido a que su sufijo debe ser "m". Gran parte del trabajo con MATLAB será el de crear y refinar archivos .m.

Hay dos tipos de archivos .m: *archivos de instrucciones* y *archivos de funciones*.

Archivos de instrucciones. Un archivo de instrucciones consiste en una sucesión de instrucciones normales de MATLAB. Si tuviéramos un archivo denominado `nombre.m`, las instrucciones del archivo pueden ser ejecutadas sin más que escribir la instrucción `nombre`. Las variables en un archivo de instrucciones son globales y, por tanto, cambiarán los valores del espacio de trabajo.

Los archivos de instrucciones son utilizados a menudo para introducir datos en una matriz grande. En un archivo de este tipo es bastante sencillo corregir los errores sin tener que repetir todo el trabajo. Si, por ejemplo, se escribe en el archivo `datos.m`

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

entonces la instrucción de MATLAB `datos` hará que se efectúe la asignación especificada en `datos.m`.

Un archivo .m puede hacer referencia a otros, incluyendo a él mismo.

Archivos de funciones. Los archivos de funciones hacen que MATLAB tenga capacidad de crecimiento. Se pueden crear funciones específicas para un problema concreto, y, a partir de su introducción, tendrán el mismo rango que las demás funciones del sistema. Las variables en las funciones son locales. La versión 4.0 permite declarar una variable como global.

Veremos, en primer lugar, un ejemplo sencillo de archivo de función:

```
function a = ental(m,n)
%ENTAL Matriz entera generada aleatoriamente.
%    ental(m,n) produce una matriz mxn con entradas
%    enteras entre 0 y 9
a = floor(10*rand(m,n));
```

Una versión más general de esta función es la siguiente:

```
function a = ental(m,n,a,b)
%ENTAL Matriz entera generada aleatoriamente.
%    ental(m,n) produce una matriz mxn con entradas
%    enteras entre 0 y 9
%    ental(m,n,a,b) produce las entradas de la matriz entre a y b.
if nargin < 3, a = 0; b = 9; end
a = floor((b-a+1)*rand(m,n))+a;
```

Esto debe escribirse en el archivo `ental.m` (correspondiente al nombre de la función). La primera línea declara el nombre de la función, argumentos de entrada, y argumentos de salida; sin esta línea el archivo sería uno de instrucciones. La instrucción `z = ental(4,5)`, por ejemplo, hará que los números 4 y 5 pasen a las variables m y n en el archivo de función y el resultado se asigna a la variable z. Como las variables en un archivo de función son locales, sus nombres son independientes de los que se encuentren en el espacio de trabajo.

Hagamos notar que el uso de `nargin` ("número de argumentos de entrada") permite asignar un valor por defecto a una variable que se omita—como a y b en el ejemplo.

Una función puede tener también argumentos de salida múltiples. Por ejemplo:

```
function [media, desv] = estad(x)
% ESTAD Media y desviación típica.  Para un vector x,
%    estad(x) da la media y la desviación típica de x.
%    Para una matriz x, estad(x) da dos vectores fila conteniendo,
%    resp., la media y la desviación típica de cada columna.
[m n] = size(x);
if m == 1
   m = n; % caso de un vector fila
end
media = sum(x)/m;
desv = sqrt(sum(x.^2)/m - media.^2)
```

Una vez situado en el archivo de disco `estad.m`, la instrución de MATLAB `[xm, xd] = estad(x)`, por ejemplo, asignará la media y la desviación típica de x a las variables xm y xd, respectivamente. Cuando se dispone de una función con argumento de salida múltiple, se pueden efectuar asignaciones simples. Por ejemplo, `xm = estad(x)` (no son necesarios los corchetes alrededor de xm) asignará la media de x a xm.

El símbolo `%` indica que el resto de la línea es un comentario; MATLAB ignorará el resto de la línea. Las primeras líneas de comentario, que documentan el archivo, son accesibles con la instrucción `help`. Así, para que se muestren en pantalla basta escribir `help estad`. Dicha documentación debe incluirse *siempre* en un archivo de función.

Esta función ilustra algunas de las formas en que MATLAB puede usarse para obtener un código eficiente. Hagamos notar, por ejemplo, que `x.^2` es la matriz de los cuadrados de las entradas de x, que `sum` es una función vectorial (sección 8), que `sqrt` es una función escalar (sección 7), y que la división en `sum(x)/m` opera una matriz con un escalar.

La siguiente función, que da el máximo común divisor de dos enteros vía el algoritmo de Euclides, ilustra el uso de un mensaje de error (ver sección siguiente).

```
function a = mcd(a,b)
% MCD Máximo común divisor
%    mcd(a,b) es el máximo común divisor de a y b no nulos a la vez.
a = round(abs(a)); b = round(abs(b));
if a == 0 & b == 0
   error('El mcd no está definido cuando ambos números son nulos')
else
   while b ~= 0
      r = rem(a,b);
      a = b; b = r;
   end
end
```

Algunas posibilidades más avanzadas se ilustran con la siguiente función. Como hicimos notar antes, alguna de las entradas de la función—como *tol* en el ejemplo siguiente, puede hacerse opcional mediante el uso de `nargin` ("número de argumentos de entrada"). La variable `nargout` puede usarse de forma similar. Hagamos notar que el hecho de que una

relación es un número (1 cuando es cierta; 0 cuando es falsa) es usado, y que, cuando `while` o `if` evalúan una relación, "no cero" significa "cierto" y 0 significa "falso". Finalmente, la función de MATLAB `feval` permite tener como variable de entrada una cadena que dé nombre a otra función

```
function [b, pasos] = bisecc(fun, x, tol)
%BISECC Cero de una función de una variable por bisección.
%    bisecc(fun,x) produce un cero de la función.  fun es una cadena
%    conteniendo el nombre de una función real de una variable real;
%    normalmente las funciones están definidas en archivos .m.
%    x es el punto inicial.  El valor producido está cerca de un
%    punto donde la función cambia de signo.  Por ejemplo,
%    bisecc('sin',3) es pi.  Nótense las comillas alrededor de sin.
%
%    Un tercer argumento de entrada opcional fija la tolerancia
%    para la precisión relativa del resultado.  El valor por defecto
%    es eps.  Un argumento de salida opcional produce una matriz
%    con las iteraciones; sus filas son de la forma [c, f(c)].

% Inicialización
if nargin < 3, tol = eps; end
traza = (nargout == 2);
if x ~= 0, dx = x/20; else, dx = 1/20; end
a = x - dx; fa = feval(fun,a);
b = x + dx; fb = feval(fun,b);

% Encontrar un cambio de signo.
while (fa > 0) == (fb > 0)
   dx = 2.0*dx;
   a = x - dx; fa = feval(fun,a);
   if (fa > 0) ~= (fb > 0), break, end
   b = x + dx; fb = feval(fun,b);
end
if traza, pasos = [a fa; b fb]; end

% Bucle Principal
while abs(b - a) > 2.0*tol*max(abs(b),1.0)
   c = a + 0.5*(b - a); fc = feval(fun,c);
   if traza, pasos = [pasos; [c fc]]; end
   if (fb > 0) == (fc > 0)
      b = c; fb = fc;
   else
      a = c; fa = fc;
   end
end
```

Algunas de las funciones de MATLAB son internas mientras que otras se distribuyen como archivos .m. El listado de cualquier archivo .m—de MATLAB o del usuario—puede obtenerse con la instrucción de MATLAB `type` *nombre_de_función*. Pruebe con `type eig`, `type vander`, y `type rank`.

13. Cadenas de texto, mensajes de error, input.

Las cadenas de texto se introducen en MATLAB entre comillas simples. Por ejemplo,

```
s = 'Esto es una prueba'
```

asigna la cadena de texto dada a la variable `s`.

Las cadenas de texto pueden mostrarse con la función `disp`. Por ejemplo:

```
disp('Este mensaje se está mostrando aquí')
```

Los mensajes de error se muestran mejor con la función `error`

```
error('Lo siento, la matriz debe ser simétrica')
```

ya que ésta hace que la ejecución salga del archivo .m.

En un archivo .m el usuario puede ser avisado para introducir datos interactivamente con la función `input`. Si MATLAB se encuentra, por ejemplo, con la instrucción

```
iter = input('Introduzca el número de iteraciones:  ')
```

la cadena entre comillas se muestra y la ejecución se detiene mientras el usuario introduce los datos. Tras pulsar el retorno de carro los datos se asignan a la variable `iter` y continúa la ejecución.

14. Tratamiento de archivos .m.

Mientras se usa MATLAB se necesita usualmente crear o editar un archivo .m y regresar a MATLAB. Sería deseable mantener MATLAB activo mientras se edita un archivo pues, en caso contrario, se perderían todas las variables tras salir.

Esto puede hacerse fácilmente con el signo !. Si, estando en MATLAB, escribe una instrucción del sistema operativo —como las que se usan para editar, imprimir y copiar un archivo— precedida del signo !, se ejecuta la instrucción sin salir de MATLAB. Si por ejemplo, la instrucción del sistema operativo `ed` accede al editor, la instrucción de MATLAB

```
>> !ed nombre.m
```

le permitirá editar el archivo `nombre.m` usando su editor local. Tras dejar el editor, retornará a MATLAB, justo donde se dejó.

Como se hizo notar en la sección 1, en sistemas que permiten procesos múltiples, como los que admiten Unix, podría ser preferible mantener activos a la vez MATLAB y el editor local, manteniendo un proceso en suspenso mientras se trabaja con el otro. Si los procesos pueden mantenerse en ventanas múltiples, como en una estación de trabajo Sun, puede desear mantener activos MATLAB en una ventana y el editor en otra.

Puede consultar a su instructor o en su centro de cálculo local para los detalles de la instalación local.

La versión 4.0 dispone de varias herramientas de rastreo. Ver `help dbtype` y las referencias que se dan allí.

Estando en MATLAB, la instrucción `dir` mostrará los contenidos del directorio activo mientras que `what` mostrará sólo los archivos .m en el directorio. Las instrucciones `delete` y `type` sirven para borrar un archivo de disco e imprimirlo en pantalla, respectivamente, y `chdir` para cambiar el directorio de trabajo. Aunque estas instrucciones reflejan las del sistema operativo, evitan el uso de !.

Los archivos .m deben ser accesibles a MATLAB. En la mayoría de los sistemas o instalaciones en red, los archivos .m personales que se almacenan en un subdirectorio del directorio raíz denominado `matlab` serán accessibles para MATLAB desde cualquier directorio en el que se trabaje. Ver la discusión de MATLABPATH, en la Guía del usuario, para más información.

15. Comparación de la eficiencia de algoritmos: flops y etime.

Dos medidas de la eficiencia de un algoritmo son el número de operaciones realizadas y el tiempo gastado.

La función `flops` es un contador de las operaciones realizadas. La instrucción `flops(0)` (no `flops=0`!) inicializa el contador a 0. Por tanto si usamos `flops(0)` inmediatamente antes de ejecutar un algoritmo, la instrucción `flops` situada justo al final nos dará el número de operaciones que se han efectuado en su ejecución.

La función `clock` da la hora actual aproximada hasta la centésima de segundo (ver `help clock`). Dados dos tiempos `t1` y `t2`, `etime(t2,t1)` proporciona el tiempo transcurrido de `t1` a `t2`. Se puede, por ejemplo, medir el tiempo que requiere la resolución de un sistema de ecuaciones dado $Ax = b$ usando eliminación gaussiana como sigue:

```
tiempo = clock; x = A\b; tiempo = etime(clock,tiempo)
```

Puede desear comparar éste —y la cuenta `flop`— con los valores que se obtienen usando `x = inv(A)*b;`. Inténtelo.

Hagamos notar que, en máquinas que operan a tiempo compartido, `etime` no es una medida fiable de la eficiencia de un algoritmo ya que la velocidad de ejecución depende de lo ocupada que esté la máquina en un momento determinado.

16. Formato de salida.

Aunque todos los cálculos en MATLAB se efectúan en doble precisión, el formato de la salida en pantalla puede ser controlado con las siguientes instrucciones.

`format short`	coma fija con 4 decimales (el defecto)
`format long`	coma fija con 14 decimales
`format short e`	notación científica con 4 decimales
`format long e`	notación científica con 15 decimales

Una vez que se ordena un formato, se mantiene hasta que se ordena un cambio.

La orden `format compact` evitará la mayor parte de las líneas en blanco, con lo que se puede mostrar más información en pantalla. Es independiente de las demás instrucciones de formato.

17. Hardcopy.

La forma más sencilla de obtener una hardcopy[1] es con la instrucción **diary**. La orden

`diary` *nombre_de_archivo*

hace que todo lo que aparezca a continuación en pantalla (excepto los gráficos) sea escrito en el archivo *nombre_de_archivo* (si se omite el nombre se toma por defecto `diary`) hasta que se ordena `diary off`; la instrucción `diary on` hará que se escriba al final del archivo, etc. Al terminar, se puede editar el archivo como se desee e imprimirlo en el sistema local. Todo se puede hacer sin salir de MATLAB usando el signo ! (ver sección 14).

18. Gráficos.

MATLAB puede producir gráficos planos y gráficos de malla de superficies tridimensionales. Para ver algunas de sus posibilidades en la versión 3.5, escriba `plotdemo`.

Gráficos planos. La instrucción `plot` crea gráficos en el plano XY; si x e y son vectores de la misma longitud, la orden `plot(x,y)` accede a la pantalla gráfica y realiza un gráfico plano de los elementos de x contra los elementos de y. Por ejemplo, podemos dibujar la gráfica de la función seno sobre el intervalo $[-4, 4]$ con las instrucciones siguientes:

```
x = -4:.01:4; y = sin(x); plot(x,y)
```

Inténtelo. El vector x es una partición del dominio con paso 0.01 mientras que y es un vector (`sin` es vectorial) con los valores que toma el seno en los nodos de esta partición.

Para volver a la pantalla alfanumérica desde la gráfica, se pulsa cualquier tecla. Por el contrario, para acceder a la pantalla gráfica, se usa la orden `shg` (show graph). Si su máquina soporta ventanas múltiples con una ventana gráfica aparte, puede desear mantener la ventana gráfica expuesta —aunque a un lado— y la ventana alfanumérica activa.

Como un segundo ejemplo, puede dibujar la gráfica de $y = e^{-x^2}$ sobre el intervalo $[-1.5, 1.5]$ como sigue:

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

Hagamos notar que `^` está precedido por un punto para asegurarnos que opera a coordenadas (ver sección 3).

Pueden hacerse también gráficos de curvas definidas paramétricamente. Por ejemplo,

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

La instrucción `grid` hará un cuadriculado en el gráfico actual.

Pueden ponerse títulos, comentarios en los ejes o en cualquier otra parte con los siguientes comandos que tienen una cadena como argumento:

title	título del gráfico
xlabel	comentario en el eje x
ylabel	comentario en el eje y
gtext	texto posicionado interactivamente
text	texto posicionado mediante coordenadas

Por ejemplo, la instrucción

[1] Una copia por impresora o en un archivo de disco.

```
title('La función más bella')
```

proporciona un título al gráfico. El comando `gtext('La mancha')` permite posicionar una cruz en el gráfico con las flechas o el ratón, donde se situará el texto cuando se pulse cualquier tecla.

Por defecto, los ejes se autoescalan. Para evitarlo se usa el comando `axis`. Si $c = [x_{\min}, x_{\max}, y_{\min}, y_{\max}]$ es un vector con 4 elementos, entonces `axis(c)` establece el escalado de ejes a los límites prescritos. `axis`, por sí mismo congela el escalado actual para gráficos subsecuentes; Escribiendo `axis` de nuevo volvemos al autoescalado. El comando `axis('square')` asegura que se use la misma escala en ambos ejes. En la versión 4.0, `axis` ha sido cambiada significativamente; ver `help axis`.

Dos formas de obtener dibujos múltiples se ilustran con

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

y formando una matriz `Y` conteniendo los valores funcionales como columnas

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

Otra forma es con `hold`. El comando `hold` congela la pantalla gráfica actual de forma que los gráficos posteriores se sobreimponen en ella. Escribiendo `hold` de nuevo se libera el "hold." Los comandos `hold on` y `hold off` también están disponibles en la versión 4.0.

Se pueden evitar los tipos de línea y de punto por defecto. Por ejemplo,

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

produce líneas a trazos y de puntos para las dos primeras, mientras que para la tercera se obtiene el símbolo `+` en cada nodo. Los tipos de líneas y de puntos son:

Tipos de línea: sólido (`-`), a trazos (`--`). puntos (`:`), punto y trazo (`-.`)
Tipos de puntos: punto (`.`), más (`+`), estrella (`*`), círculo (`o`), equis (`x`)

Ver `help plot` para los colores de las líneas y puntos.

El comando `subplot` se usa para dividir la pantalla de forma que puedan verse hasta cuatro gráficos a la vez. Ver `help subplot`.

Graphics hardcopy.* La forma más sencilla de obtener una hardcopy de la pantalla gráfica es usar el comando de MATLAB `print`. Este enviará una copia en alta resolución de la pantalla gráfica actual a la impresora, situando el gráfico en la mitad superior de la página.

En la versión 4.0 los comandos `meta` y `gpp` que se describen a continuación han sido absorbidos por el comando `print`. Ver `help print`.

Producir una copia unificada de varios dibujos requiere más esfuerzo. El comando de MATLAB `meta` *nombre_de_archivo* almacena la pantalla gráfica en un archivo denominado *nombre_de_archivo.met* (un "metaarchivo") en el directorio actual. Posteriores `meta` (sin nombre) añadirán las nuevas pantallas gráficas al archivo meta anterior. Este metaarchivo

* Las utilidades descritas en esta subsección no están disponibles en la edición para estudiantes de Matlab. En ella la única forma de obtener una copia por impresora es mediante un volcado de pantalla: Shift-PrtScr.

—que puede contener ahora varios dibujos— puede ser procesado posteriormente con el programa procesador de gráficos (GPP) para obtener una hardcopy de alta resolución, con dos dibujos por página.

El programa GPP (graphics post-processor) es un comando de *sistema*, no un comando de MATLAB. Desde luego, en la práctica se invoca desde MATLAB con la ayuda del signo "!" (ver sección 14). Actúa sobre un archivo independiente de dispositivo (metaarchivo) para producir un fichero de salida apropiado para varios dispositivos de harcopy diferentes.

La selección del dispositivo específico se hace con la opción "`/d`". Por ejemplo, los comandos del sistema

```
gpp nombre_de_archivo /dps
gpp nombre_de_archivo /djet
```

producirán archivos `nombre_de_archivo.ps` y `nombre_de_archivo.jet` apropiados para su impresión en, respectivamente, impresoras PostScript y HP LaserJet. Pueden imprimirse con el comando usual para los sistemas locales—por ejemplo, `lpr nombre_de_archivo.ps` en un sistema UNIX. Si se escribe el comando de sistema `gpp` sin argumentos se obtiene una lista de los dispositivos que soporta. En un PC, la mayor parte de este trabajo puede automatizarse editando apropiadamente el archivo `gpp.bat` distribuido con MATLAB.

Gráficos de malla de superficies tridimensionales. Los gráficos de malla de superficies tridimensionales se hacen con la función `mesh`. La instrucción `mesh(z)` crea un gráfico tridimensional en perspectiva de la matriz z. La superficie de malla está definida por las coordenadas z de los puntos sobre un cuadriculado rectangular en el plano XY. Por ejemplo, pruebe con `mesh(eye(10))`.

Para dibujar la gráfica de una función $z = f(x, y)$ sobre un rectángulo, se definen en primer lugar los vectores xx e yy que dan particiones de los lados del rectángulo. Con la función `meshdom` (mesh domain; el nombre es `meshgrid` en la versión 4.0) se crea una matriz x, en la que cada fila es igual a xx, y de igual forma una matriz y, con todas sus columnas iguales a yy, como sigue:

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

Entonces se computa la matriz z, obtenida evaluando f entrada a entrada sobre las matrices x e y, para aplicarle la función `mesh`.

Se puede, por ejemplo, dibujar la gráfica de $z = e^{-x^2-y^2}$ sobre el cuadrado $[-2, 2] \times [-2, 2]$ como sigue (inténtelo):

```
xx = -2:.1:2;
yy = xx;
[x,y] = meshdom(xx,yy);
z = exp(-x.^2 - y.^2);
mesh(z)
```

Se podría, desde luego, cambiar las tres primeras líneas en lo anterior por

```
[x,y] = meshdom(-2:.1:2, -2:.1:2);
```

Para más detalles sobre `mesh`, ver la Guía del usuario.

En la versión 4.0, se han ampliado considerablemente las posibilidades gráficas respecto a las superficies tridimensionales. Consulte la ayuda para `plot3, mesh`, y `surf`.

19. Consulta.

Hay muchas características de MATLAB que no pueden incluirse en estas notas introductorias. Mostraremos a continuación algunas de las funciones disponibles en MATLAB, agrupadas por áreas[1]. Use la instrucción `help` o consulte la Guía del usuario para una información más detallada sobre las funciones.

Hay muchas funciones aparte de estas. Existen, en particular, varias "cajas de herramientas" de funciones para áreas específicas; entre ellas, las de proceso de señales, teoría de control, control robusto, identificación de sistemas, optimización, splines, quimiometría, μ-análisis y síntesis, identificación, y redes neurales[2]. Para explorar siempre se puede usar la instrucción `help`.

General	
help	ayuda
demo	demostraciones
who	muestra variables en memoria
what	muestra archivos .m en el disco
size	número de filas y columnas
length	longitud de un vector
clear	limpia el espacio de trabajo
computer	tipo de computadora
^C	interrupción local
exit	salida de MATLAB
quit	lo mismo que exit

Operadores matriciales		Operadores puntuales	
$+$	suma	$+$	suma
$-$	resta	$-$	resta
$*$	multiplicación	$.*$	multiplicación
/	división derecha	./	división derecha
\	división izquierda	.\	división izquierda
$\hat{}$	potenciación	$.\hat{}$	potenciación
'	conjugada traspuesta	.'	traspuesta

Operadores Lógicos y Relacionales			
$<$	menor que	&	y
$<=$	menor o igual que	|	o
$>$	mayor que	$\sim$	no
$>=$	mayor o igual que		
$==$	igual		
$\sim=$	no igual		

[1] Fuente: MATLAB User's Guide, versión 3.5.

[2] Las cajas de herramientas, que son opcionales, pueden no estar instaladas en su sistema.

Caracteres especiales	
=	instrucción de asignación
[	usado para formar vectores y matrices
]	ver [
(	precedencia aritmética
)	ver (
.	punto decimal
...	la instrucción continúa en la siguiente línea
,	separa índices y argumentos de función
;	acaba filas, suprime la impresión
%	comentarios
:	indexación, generación de vectores
!	ejecuta instrucción del sistema operativo

Valores Especiales	
ans	respuesta cuando no se asigna la expresión
eps	precisión
pi	π
i, j	$\sqrt{-1}$
inf	∞
NaN	No Número (Not-a-Number)
clock	reloj de pared
date	fecha
flops	número de operaciones
nargin	número de argumentos de entrada de una función
nargout	número de argumentos de salida de una función

Archivos de disco	
chdir	cambiar de directorio
delete	borrar archivo
diary	diario de la sesión
dir	directorio de archivos en el disco
load	cargar variables de un archivo
save	guardar variables en un archivo
type	mostrar función o archivo
what	mostrar archivos .m en el disco
fprintf	escribir en un archivo
pack	compactar memoria vía **save**

Matrices Especiales	
compan	compañera
diag	diagonal
eye	identidad
gallery	esotérica
hadamard	Hadamard
hankel	Hankel
hilb	Hilbert
invhilb	inversa de Hilbert
linspace	vectores igualmente espaciados
logspace	vectores logarítmicamente espaciados
magic	mágica cuadrada
meshdom	dominio para puntos de malla
ones	constante
pascal	Pascal
rand	elementos aleatorios
toeplitz	Toeplitz
vander	Vandermonde
zeros	cero

Manipulación de matrices	
rot90	rotación
fliplr	invierte el orden de las columnas
flipud	invierte el orden de las filas
diag	diagonal
tril	parte triangular inferior
triu	parte triangular superior
reshape	reordena una matriz en otra
.'	traspuesta
:	convierte una matriz en una columna simple; `A(:)`

Funciones Lógicas y Relacionales	
any	condiciones lógicas
all	condiciones lógicas
find	encuentra índices de valores lógicos
isnan	detecta NaNs
finite	detecta infinitos
isempty	detecta matrices vacías
isstr	detecta variables de cadena
strcomp	compara variables de cadena

Control de Flujo	
if	ejecuta instrucciones condicionalmente
elseif	usado con **if**
else	usado con **if**
end	termina **if, for, while**
for	repite instrucciones un número de veces
while	repite instrucciones mientras una sentencia lógica sea verdadera
break	sale de los bucles **for** y **while**
return	salida desde funciones
pause	pausa hasta que se pulse una tecla

Programación y archivos .m	
input	obtiene números desde el teclado
keyboard	llamada al teclado como si fuera un archivo .m
error	muestra mensaje de error
function	define función
eval	evalúa un texto en variables
feval	evalúa función dada por una cadena
echo	permite mostrar las instrucciones en pantalla
exist	comprueba si las variables existen
casesen	sensibilidad a las mayúsculas
global	define variables globales
startup	archivo de inicialización
getenv	accede a una variable de entorno
menu	genera un menú
etime	tiempo gastado

Texto y Cadenas	
abs	convierte cadena en valores ASCII
eval	evalúa texto como instrucciones
num2str	convierte números en cadenas
int2str	convierte enteros en cadenas
setstr	indicador de cadenas
sprintf	convierte números en cadenas
isstr	detecta variables de cadena
strcomp	compara variables de cadena
hex2num	convierte cadenas hexadecimales en números

Ventana alfanumérica	
clc	limpia pantalla
home	mueve cursor al comienzo
format	establece el formato de salida
disp	muestra matriz o texto
fprintf	imprime número formateado
echo	permite la muestra de las instrucciones

Gráficos	
plot	gráfico lineal en el plano XY
loglog	gráfico logarítmico en el plano XY
semilogx	gráfico semilogarítmico
semilogy	gráfico semilogarítmico
polar	gráfico polar
mesh	superficie de malla tridimensional
contour	plano de contornos
meshdom	dominio para gráficos de superficie
bar	gráficos de barras
stairs	gráficos de escaleras
errorbar	añade barras de errores

Anotación Gráfica	
title	título
xlabel	anotación en el eje x
ylabel	anotación en el eje y
grid	dibuja cuadriculado
text	posiciona un texto arbitrariamente
gtext	posiciona un texto con el ratón
ginput	input gráfico

Control de la Ventana Gráfica	
axis	escalado manual de ejes
hold	mantiene gráfico en pantalla
shg	muestra la pantalla gráfica
clg	limpia la pantalla gráfica
subplot	divide la pantalla gráfica

Impresión de gráficos	
print	envía gráfico a impresora
prtsc	volcado de pantalla
meta	archivo de gráficos

Funciones elementales	
abs	módulo complejo
angle	argumento complejo
sqrt	raíz cuadrada
real	parte real
imag	parte imaginaria
conj	conjugado complejo
round	redondeo al entero más cercano
fix	redondeo hacia cero
floor	redondeo hacia $-\infty$
ceil	redondeo hacia ∞
sign	función signo
rem	resto
exp	exponencial base ϵ
log	logaritmo natural
log10	logaritmo base 10

Funciones Trigonométricas	
sin	seno
cos	coseno
tan	tangente
asin	arcoseno
acos	arcocoseno
atan	arcotangente
atan2	arcotangente de x/y
sinh	seno hiperbólico
cosh	coseno hiperbólico
tanh	tangente hiperbólica
asinh	arcoseno hiperbólico
acosh	arcocoseno hiperbólico
atanh	arcotangente hiperbólica

Funciones Especiales	
bessel	función de Bessel
gamma	función Gamma
rat	aproximación racional
erf	función de error
inverf	inversa de la función de error
ellipk	integral completa elíptica de primera especie
ellipj	integral elíptica de Jacobi

Descomposiciones y Factorizaciones	
balance	forma equilibrada
backsub	sustitución regresiva
cdf2rdf	convierte diagonales complejas en diagonales reales
chol	factorización de Cholesky
eig	autovalores y autovectores
hess	forma de Hessenberg
inv	inversa
lu	factores de la eliminación gaussiana
nnls	mínimos cuadrados con restricciones
null	base ortonormal del núcleo
orth	base ortonormal de la imagen
pinv	pseudoinversa
qr	factorización QR
qz	algoritmo QZ
rref	forma escalonada reducida por filas
schur	descomposición de Schur
svd	descomposición en valores singulares

Condicionamiento de matrices	
cond	número de condición en la norma 2
norm	norma 1, norma 2, norma de Frobenius, norma ∞
rank	rango
rcond	estimación de la condición (inverso)

Funciones matriciales elementales	
expm	matriz exponencial
logm	matriz logaritmo
sqrtm	matriz raíz cuadrada
funm	función arbitraria de matriz
poly	polinomio característico
det	determinante
trace	traza
kron	producto tensorial de Kronecker

Polinomios	
poly	polinomio característico
roots	raíces de polinomios—método de la matriz compañera
roots1	raíces de polinomios—método de Laguerre
polyval	evaluación de polinomios
polyvalm	evaluación polinomio matricial
conv	multiplicación
deconv	división
residue	desarrollo en fracciones parciales
polyfit	ajuste por un polinomio

Análisis de datos por columnas	
max	valor máximo
min	valor mínimo
mean	valor medio
median	mediana
std	desviación típica
sort	ordenación
sum	suma de elementos
prod	producto de elementos
cumsum	suma acumulativa de elementos
cumprod	producto acumulativo de elementos
diff	derivadas aproximadas
hist	histogramas
corrcoef	coeficientes de correlación
cov	matriz de covarianza
cplxpair	reordena en pares complejos

Proceso de Señales	
abs	módulo complejo
angle	argumento complejo
conv	convolución
corrcoef	coeficientes de correlación
cov	covarianza
deconv	deconvolución
fft	transformada rápida de Fourier
fft2	FFT 2-dimensional
ifft	FFT inversa
ifft2	FFT inversa 2-dimensinal
fftshift	cambia las dos mitades de un vector

Integración Numérica	
quad	función de integración numérica
quad8	función de integración numérica

Solución de ecuaciones diferenciales	
ode23	método de Runge-Kutta de orden 2/3
ode45	método de Runge-Kutta-Fehlberg de orden 4/5

Ecuaciones no lineales y Optimización	
fmin	mínimo de una función de una variable
fmins	mínimo de una función de varias variables
fsolve	solución de un sistema de ecuaciones no lineales (ceros de una función de varias variables)
fzero	cero de una función de una variable

Interpolación	
spline	spline cúbico
table1	genera tablas 1-D
table2	genera tablas 2-D