CURSO BASICO DE MATLAB

Ing Yamil Armando Cerquera Rojas - **yacerque@gmail.com**
Especialista en Sistemas Universidad Nacional
Docente Universidad Surcolombiana
Neiva - Huila

CONTENIDO

Resumen[1]

Este documento presenta una introducción a Matlab dirigida a estudaintes que no han trabajado nunca con Matlab. Las principales características de Matlab así como la creación de programas son explicadas. El documento trata de introducir al usuario en los distintos temas explicados sin profundizar en ellos.

El documento incluye una lista de referencias que el usuario puede utilizar para ampliar sus conocimientos sobre el uso de este programa.

Introducción

Matlab (MATrix LABoratory) es un entorno interactivo basado en matrices para la realización de cálculo numérico y visualización de resultados. Permite la resolución de problemas sencillos sin escribir un programa y con facilidades de representación grafica de los resultados. Además incorpora un lenguaje de programación que permite implementar programas complejos de modo relativamente simple.

Actualmente el sistema Matlab se usa tanto a nivel académico, dentro de la universidad, como a nivel de investigación e industria para la resolución de complicados problemas científicos o de ingeniería. Es empleado para el desarrollo de cálculo numérico de propósito general pudiendo manipular vectores y matrices tanto reales como complejos con funciones y fórmulas de variadas ramas de la matemática y resolución de problemas con formulación matricial que aparecen en control, estadística y procesado de señales. Matlab aporta, por medio de los toolbox (que no se incorporan en el sistema base sino que se adquieren separadamente), funciones para resolver problemas específicos como por ejemplo procesado de señales, diseño de sistemas de control, identificación de sistemas, simulación de sistemas dinámicos, optimización, redes neuronales, etc.

Entre los toolbox más importantes se encuentran:

- **Curve fitting:** ajustes de modelos y análisis.
- **Data Acquisition:** adquiere y envía datos a un instrumento electrónico conectado al computador. (sólo para Windows)
- **Excel link:** permite usar Matlab con datos leídos directamente desde planillas Excel.
- **Image processing:** permite el procesamiento de imágenes, análisis y desarrollo de algoritmos.
- **Partial differential equation:** soluciona y analiza sistema de ecuaciones diferenciales parciales.
- **Signal Processing:** permite el procesamiento de señales, análisis y desarrollo de algoritmos.

[1] **Escrito por Ing. Yamil Armando Cerquera Rojas para la electiva básica MatLab para ingeniería. Varios ejemplos utilizados proceden de una versión anterior.**

- **Spline:** crea y manipula modelos de aproximación de datos Spline.
- **Statistics:** permite aplicar modelos estadísticos y modelos de probabilidades.
- **Structural Dynamics:** analiza modelos de elementos finitos y lleva a cabo análisis modales de sistemas mecánicos.
- **Wavelet:** analiza, comprime y saca el ruido de señales e imágenes usando técnicas de wavelet.

Matlab, como ya se ha dicho, esta orientado al cálculo numérico, a diferencia de otros software como Mapple y Mathematica que están orientados al cálculo simbólico. Otra característica importante de Matlab es que es orientado al arreglo (vectores y matrices), es decir, las operaciones o funciones matemáticas que son válida para números escalares también lo son para arreglos. Si por ejemplo V es un vector de 5 elementos, entonces cos(V) entregará un vector de 5 elementos con los valores de coseno para cada elemento del vector original.

En las siguientes secciones se dará una introducción a los tópicos de matlab más usados. Para profundizar más en cualquier comando que aquí se muestre se puede ejecutar “help <comando>” en el ambiente matlab.

El programa MATLAB se distingue en si por una serie de características notables para los análisis numéricos, entre las cuales se pueden citar:

- La programación es mucho más sencilla.
- Hay continuidad entre valores enteros, reales y complejos.
- La amplitud de intervalo y la exactitud de los números son mayores.
- Presenta una biblioteca matemática amplia.
- Presenta abundantes herramientas gráficas.
- Incluye funciones de interfaz gráfica con el usuario.
- Presenta capacidad de vincularse con lenguajes de programación clásicos.

El presente documento es una recopilación de información que puede ser útil para aquellas personas interesadas en conocer esta poderosa herramienta de cálculo, simulación y modelado matemático. Vale la pena mencionar en esta introducción todos los elogios del cual es merecedor este singular programa de cálculo matemático por su amplia área de aplicación en el estudio científico.

Introducción al MatLab.

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

MATLAB dispone también en la actualidad de un amplio abanico de programas de apoyos especializados, denominados Toolboxes, que extienden significativamente el número de funciones incorporadas en el programa principal. Estos Toolboxes cubren en la actualidad prácticamente casi todas las áreas principales en el mundo de la ingeniería y la simulación, destacando entre ellos el 'Toolbox' de proceso de imágenes, señal, control robusto, estadística, análisis financiero, matemáticas simbólicas, redes neurales, lógica difusa, identificación de sistemas, simulación de sistemas dinámicos, etc. es un entorno de cálculo técnico, que se ha convertido en estándar de la industria, con capacidades no superadas en computación y visualización numérica.

De forma coherente y sin ningún tipo de fisuras, integra los requisitos claves de un sistema de computación técnico: cálculo numérico, gráficos, herramientas para aplicaciones específicas y capacidad de ejecución en múltiples plataformas. Esta familia de productos proporciona al estudiante un medio de carácter único, para resolver los problemas más complejos y difíciles.

Existe en la actualidad un grupo de programadores de todo el mundo desarrollando un paquete denominado SCILAB Gnu, para trabajo sobre Linux. En la actualidad el paquete se puede instalar bajo Windows también.

Origen

MATLAB nace como una solución a la necesidad de mejores y más poderosas herramientas de cálculo para resolver problemas de cálculo complejos en los que es necesario aprovechas las amplias capacidades de proceso de datos de grandes computadores.

El nombre MATLAB viene de "Matrix Laboratory" (laboratorio matricial). MATLAB fue originalmente escrito para proveer acceso fácil al software matricial desarrollado por los proyectos LINPACK y EISPACK, que juntos representan el estado del arte e software para computación matricial. Hoy MATLAB es usado en una gran variedad de áreas de aplicación incluyendo procesamiento de señales e imágenes, diseño de sistemas de control, ingeniería financiera e investigación médica. La arquitectura abierta facilita usar MATLAB y los productos que lo acompañan para explorar datos y crear herramientas personalizadas que proveen visiones profundas tempranas y ventajas competitivas.

Características de MATLAB

- Cálculo numérico rápido y con alta precisión
- Manejo simbólico
- Graficación y visualización avanzada
- Programación mediante un lenguaje de alto nivel
- Programación estructurada y orientada a objetos
- Soporte básico para diseño de interfaz gráfica
- Extensa biblioteca de funciones

- Aplicaciones especializadas para algunas ramas de ciencias e ingeniería (toolboxes)

Operación

- Simple y eficiente
- Interactivo
- Sistema de ayuda en línea
- Interacción con otros entornos

Iniciación al MatLab

El Lenguaje de Computación Técnica MATLAB es un ambiente de computación técnica integrada que combina computación numérica, gráficos y visualización avanzada y un lenguaje de programación de alto nivel.

Sea cual fuere el objetivo, un algoritmo, análisis, gráficos, informes o simulación, MATLAB lo lleva allí. El lenguaje flexible e interactivo de MATLAB permite a ingenieros y científicos expresar sus ideas técnicas con simplicidad. Los poderosos y amplios métodos de cómputo numérico y graficación permiten la prueba y exploración de ideas alternativas con facilidad, mientras que el ambiente de desarrollo integrado facilita producir resultados prácticos fácilmente.

MATLAB es la fundación numérica y gráfica para todos los productos de The MathWorks. MATLAB combina computación numérica, gráficos 2D y 3D y capacidades de lenguaje en un único ambiente fácil de usar.

Con su amplio rango de herramientas para modelar sistemas de control, análisis, simulación y procesamiento de prototipos, MATLAB es el sistema ideal para desarrollar sistemas avanzados de control. Usted puede modelar su sistema de control usando las cajas de herramientas para el diseño de controles avanzados de MATLAB - Control System, Robust Control, μ-Analysis and Synthesis, Model Predictive Control, QTF Control Design y LMI control. Posteriores análisis y refinamientos pueden ser efectuados estableciendo una simulación interactiva en Simulink, y luego sintonizar automáticamente los parámetros usando el Nonlinear Control Design Blockset. Finalmente, usted puede generar código C para correr en controladores incrustados con Real Time Workshop.

Combinando MATLAB con Signal Processing Toolbox, Wavelet Toolbox y un conjunto de herramientas complementarias - tales como Image Processing, Neural Network, Fuzzy Logic, Statistics y otras – se puede crear un ambiente de análisis personalizado de señales y desarrollo de algoritmos DSP. Para simulación y desarrollo de prototipos usted puede agregar Simulink y el DSP Blockset para modelar y simular sus sistemas DSP, y luego usar Real-Time Workshop para generar código C para su hardware designado.

El Entorno Matlab

MatLab interactúa con el usuario a través de ventanas. Las principales son las siguientes:

- *Ventana de comandos (Command Window)*: en ella se ejecutan las instrucciones.
- *Historial de comandos (Command History)*: historia de las instrucciones introducidas en la ventana de comandos.
- *Directorio actual (Current Directory)*: muestra los ficheros.
- *Espacio de trabajo (Workspace)*: muestra las variables almacenadas en memoria.
- Ventana de figuras: despliega las graficas que el usuario realice.

Las ayudas de MatLab

Manejo de Comandos y Funciones

Cada comando en MATLAB es un archivo con extensión .m, por lo tanto es necesario tener las librerías en que se encuentran los comandos que se desean utilizar.

Aunque la gran mayoría de los comandos utilizados siempre vienen incluidos en las librerías.

MATLAB NO distingue entre mayúsculas y minúsculas en los comandos (a menos que se trabaje en Unix). El resto de estas notas tratará cada comando mas usado en detalle.

help	Documentación en línea. Nos permite ver lo que 'hace una función'
what	Listado de Directorio de archivos con extensión M-, MAT- y MEX.
type	Lista de archivos M-file.
lookfor	Búsqueda desde el teclado a través del comando HELP.
which	Localización de funciones y archivos.
demo	Ejecución de un demo de MatLab.
helpwin	Documentación en forma interactiva con el usuario

» help	despliega temas de ayuda
» help ops	despliega comandos de un tema. Ej. Lista de operadores
» help exp	uso de un comando específico. Ej. Función exponencial

Adicionalmente, presionando el icono Help usted puede entrar al sistema de ayuda de MATLAB organizado por contenido, índice, búsqueda y demostraciones.

Manejo de variables y workspace (espacio de trabajo).

who	Lista de variables actuales. Nos permite saber características de una variable tal como su tamaño, el tipo o si es real o compleja
whos	Lista de variables actuales, forma larga
load	Cargar variables de disco.
save	Salvar variables de workspace a disco.
clear	Limpiar variables y funciones de memoria.
clear a b c	
clc	despeja la ventana de comandos
pwd	muestra cual es el directorio actual
size	Tamaño de la matriz
length	Numero de elementos de un vector
disp	Muestra los valores de una matriz o texto.
exist('c')	chequea si la variable c existe

De igual forma, se puede interrogar al sistema sobre sus características o las características de las variables que se estén manejando

finite(x)	Devuelve 1 si x es finito, y cero en otro caso
isinf(x)	Da 1 si x es infinito o – infinito, y cero en otro caso
isnan(x)	Da 1 si x es indeterminado y cero en otro caso
isieee	Da 1 si la máquina es IEEE y da 0 en otro caso
computer	Devuelve el tipo de computador
version	Devuelve la versión actual de Matlab
why	Devuelve un mensaje sucinto
lasterr	Devuelve el último mensaje de error
pack	Consolida el espacio de trabajo en memoria
diary	Guarda el texto de la sesión actual de Matlab
unix	Ejecuta comandos del sistema operativo Unix
ver	Da información sobre el programa y sus Toolbox
info	Da información acerca de Matlab
subscribe	Da información sobre la suscripción a Matlab
whatsnew	Informa acerca de características nueva de Matlab
hostid	Identifica el número del host servidor

Comandos especiales

COMPUTER Computer type.

COMPUTER returns a string containing the name of the computer on which MATLAB is executing. Possibilities are:

PCWIN	- MS-Windows
MAC2	- Macintosh
SUN4	- Sun SPARC
SOL2	- Solaris 2
HP700	- HP 9000/700
SGI	- Silicon Graphics

SGI64	- Silicon Graphics R8000
IBM_RS	- IBM RS6000
ALPHA	- Dec Alpha
AXP_VMSG	- Alpha VMS G_float
AXP_VMSIEEE	- Alpha VMS IEEE
LNX86	- Linux Intel
VAX_VMSG	- VAX/VMS G_float
VAX_VMSD	- VAX/VMS D_float

[C,MAXSIZE] = COMPUTER returns also integer MAXSIZE which contains the maximum number of elements allowed in a matrix on this version of MATLAB.

```
» str=computer
str =
        PCWIN
» [str maxsize]=computer
str =
        PCWIN
maxsize =
        2147483647
```

date

El comando ***date*** proporciona la fecha en un formato simple:

```
>> date↵
ans =
10-Sep-2007
```

clock fecha hora, vea su uso con help.

El comando ***clock*** da como resultado de su ejecución: el año, mes, día, hora, minuto y segundo, en un formato de vector fila: [año, mes, día, hora, minuto, segundo].

```
»  clock↵
ans =
1.0e+003 *
2.006    0.0120    0.0120    0.0160    0.0050    0.040
```

La misma información puede obtenerse utilizando un formato entero con *fix(clock)*:

FIX Round towards zero.
FIX(X) rounds the elements of X to the nearest integers towards zero.

```
»  fix(clock) ↵
ans =
        2006          12      12      16      05      40
```

Sistema operativo y archivos

cd Cambiar directorio actual de trabajo.
Ejemplo: cd c:\mb\work

dir Muestra el directorio actual.
Por ejemplo, DIR *.m Lista todos los archivos de extensión M en el directorio actual

delete Borra un archivo u objeto gráfico.
Por ejemplo, DELETE *.p Borra todos los archivos de extensión P desde el directorio actual

Es conveniente abrir una hoja en el editor de MATLAB (M-file) para ir guardando lo que se va escribiendo. También es posible ir generando el programa directamente en el command de MATLAB la desventaja es que no se pueden guardar las instrucciones escritas.

Todos los comandos a que se refieren estas notas pueden utilizarse directamente desde la línea o prompt de comandos del MATLAB (>>). Sin embargo la idea es hacer un archivo (con extensión .m) que contenga el programa que vaya realizando (para poder modificarlo, revisarlo, ejecutarlo otra vez) ya que es más ventajoso así.

Los programas no requieren indentación como en los ejemplos que he dejado aquí, sin embargo es recomendable hacerlo por claridad al intentar modificar el programa o revisarlo.

Para trabajar estos archivos, es necesario saber:

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

Como crear uno: Las formas más fáciles son:

Desde Unix: con el comando "!pico archivo.m" donde archivo es el nombre del programa.

Desde Windows: con el NOTEPAD, teniendo la precaución de cambiar el tipo de archivo a "Todos los archivos (*.*)" antes de grabarlo. (De lo contrario el archivo quedará con nombre archivo.m.txt y el MATLAB no podrá ejecutarlo, la solución es quitar el .txt).

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

Para tener en cuenta antes de iniciar

Líneas de comentarios

Ya se ha indicado que para MATLAB el carácter ***tanto por ciento*** (%) indica comienzo de comentario. Cuando aparece en una línea de comandos, el programa supone que todo lo que va desde ese carácter hasta el fin de la línea es un comentario. Más adelante se verá que los comentarios de los ficheros ****.m*** tienen algunas peculiaridades importantes, pues pueden servir para definir ***help***'s personalizados de las funciones que el usuario vaya creando.

% precede a un comentario, las sentencias son ignoradas por el programa.

```
» % Este es un curso básico de MatLab ↵
»
```

; hace que la instrucción previa se ejecute pero no devuelve el resultado.

```
» y=2; ↵
» y ↵
y =
    2
```

Se pueden colocar varias instrucciones en una misma línea separando cada instrucción por un punto y coma:

```
» y=2; x=4; k=1; ↵
```

En el ejemplo anterior se guardan los valores en cada una de las variables sin mostrar las operaciones realizadas. En caso de que necesite ver los valores, se puede separar cada una de las instrucciones con una coma.

```
» y=2, x=4, k=1 ↵
y =
    2
x =
    4
k =
    1
```

Algunos de los símbolos mas utilizados en Matlab se relacionan a continuación:

' Alt 39 **(** Alt 40 **)** Alt 41 **.** Alt-42 **/** Alt 47 **:** Alt 58 **;** Alt 59
[Alt 91 **** Alt 92 **]** Alt 93 **^** Alt 94

Caracteres especiales

[ ] Se utilizan para ingresar valores o formar vectores y matrices

( )	Define precedencia en expresiones aritméticas. Encierra argumentos de funciones en forma usual
,	Separador de elementos de una matriz, argumentos de funciones y declaraciones en líneas con declaraciones múltiples
;	Termina renglones de una matriz, separador de declaraciones
%	Comentario

Medida de tiempos y de esfuerzo de cálculo

MATLAB dispone de funciones que permiten calcular el tiempo empleado en las operaciones matemáticas realizadas. Algunas de estas funciones son las siguientes:

cputime	Devuelve el tiempo de CPU (con precisión de centésimas de segundo) desde que el programa arrancó. Llamando antes y después de realizar una operación y restando los valores devueltos, se puede saber el tiempo de CPU empleado en esa operación. Este tiempo sigue corriendo aunque MATLAB esté inactivo.
etime(t2, t1)	Tiempo transcurrido entre los vectores **t1** y **t2** (¡atención al orden!), obtenidos como respuesta al comando ***clock***.
tic *ops* toc	Imprime el tiempo en segundos requerido por ***ops***. El comando ***tic*** pone el reloj a cero y ***toc*** obtiene el tiempo transcurrido.

Otras funciones permiten calcular el número de operaciones de coma flotante realizadas, como las siguientes:

flops(0)	Inicializa a cero el contador de número de operaciones aritméticas de punto flotante (*flops*)
flops	Devuelve el número de *flops* realizados hasta ese momento Por ejemplo, el siguiente código mide de varias formas el tiempo necesario para resolver un sistema de 500 ecuaciones con 500 incógnitas. Téngase en cuenta que los tiempos pequeños (del orden de las décimas o centésimas de segundo), no se pueden medir con gran precisión.

```
» A=rand(500); b=rand(100,1); x=zeros(500,1);
» tiempo=clock; x=A\b; tiempo=etime(clock, tiempo)
» time=cputime; x=A\b; time=cputime-time
» tic; x=A\b; toc
```

Donde se han puesto varias sentencias en la misma línea para que se ejecuten todas sin tiempos muertos al pulsar ***intro***. Esto es especialmente importante en la línea de comandos en la que se quiere medir los tiempos. Todas las sentencias de cálculos matriciales van seguidas de punto y coma (;) con objeto de evitar la impresión de resultados.

Prompt del MatLab. Iniciando el programa MatLab

Después de ejecutar el programa MatLab desde el sistema operativo empleado, por ejemplo haciendo doble click sobre el icono de MatLab en el escritorio de Windows, aparece el indicador de comandos el cual está listo para recibir instrucciones en lenguaje MatLab. Este indicador es de la siguiente forma: **»**

Al iniciar el uso de MatLab, están disponibles varios comandos de ayuda y demostración. Para ejecutarlos se escribe el comando en la línea de comandos después del símbolo » y se presiona la tecla Enter (↵). Por ejemplo:

» help ↵ Permite obtener una ayuda sobre los diferentes comandos de MatLab.
» demo ↵ Hace una demostración de las diferentes aplicaciones de MatLab.
» intro ↵ El programa responderá con un tour a través de algunas opciones disponibles, observe las líneas de comando, las de comentarios y las respuestas del programa.
» census ↵ Muestra algunas de las capacidades del programa para el ajuste de curvas y predicciones..
» *spline2d* ↵ *Muestra un método de entrada de datos por pantalla para construir una curva.*
» quit ↵ Para cerrar o finalizar el uso de MatLab se usa el comando quit.
» help general ↵ lista de comandos generales

Nota: Todo el trabajo lo puede realizar apoyándose en el editor de texto del MatLab o cualquier otro editor de texto plano, o si la idea es guardar todo lo que se realice a través del prompt del MatLab se puede utilizar la función DIARY. Esta función permite guardar todo lo que se vaya realizando a través del paquete. Su uso es de la siguiente forma.

Lectura y escritura en ficheros externos

MATLAB permite salvar y recuperar datos de diferentes tipos de ficheros, diferenciándolos básicamente por su extensión.

mat Fichero de datos binario. Se genera automáticamente con la instrucción **save** *fichero* y se recupera mediante la instrucción **load**. Hay que indicar que se pueden salvar los datos como caracteres ASCII empleando la opción –*ascii*, en cuyo caso el fichero no toma la extensión *.mat*.
dia Fichero ascii que almacena las instrucciones ejecutadas durante una sesión de trabajo. Se genera automáticamente mediante la instrucción **diary**, pudiendo posteriormente convertirse en un *m-fichero*.
dat Fichero de datos, generados por el usuario. Si son homogéneos (no se mezclan distintos tipos de datos) y mantienen la estructura (mismo numero de datos por linea) pueden recuperarse mediante la instrucción **load**.

diary *nombrefichero*: escribe los comandos entrados en el prompt y las salidas producidas de la sesión actual en el fichero *nombrefichero*. Se debe ejecutar al inicio de la sesión que se desea grabar.

» **diary c:\yamil.txt** ↵ guarda toda la sesión o mejor decirlo, lo que ejecute a partir de esta en el archivo yamil.txt en la raíz del disco duro identificado con la letra C:

» **diary('c:\yamil.txt')** ↵ Entre apóstrofes (Alt-39) se coloca el nombre del archivo y la ubicación donde quedarán guardadas todas las instrucciones y resultados que se vayan escribiendo en el prompt y que se va viendo en la pantalla del MatLab. En el caso del ejemplo, se está deseando guardar en un archivo yamil.txt y se le da la orden para que dicho archivo sea creado en el root o raíz del disco duro C: (C:\)

» **diary y.txt** guarda las instrucciones y sus respectivas salidas en le archivo y.txt en el directorio de trabajo de MatLab. (work por default)

Antes de salir del MatLab es necesario dar la instrucción diary off con el fin de que el archivo sea cerrado y no se pierdan datos.

» **diary off/on** Al utilizar el diary con la opción off después de haber iniciado una sesión, el MatLab deja de guardar las instrucciones a partir de ese momento y al querer volver a grabar entonces lo activa utilizando diary on como se ilustra en el siguiente ejemplo.

En el siguiente ejemplo si el archive y.txt ya existe, MatLab anexa los datos al final del mismo.

```
» diary  y.txt ↵
» y=2; ↵       % se asignó 2 a la variable y
» y ↵
y =
    2
» diary off ↵
» x=4 ↵        % se asignó 4 a la variable x
x =
    4
» diary on ↵
```

Nota: La asignación de la variable x no queda registrada en el archivo y.txt

Para poder acceder al fichero ***filename.txt*** con ***Notepad*** es necesario que ***diary*** esté en ***off***. Si no se incluye el nombre del fichero, se utiliza por defecto un fichero llamado ***diary***.

help nombre_función da información sobre la función y el nombre de otras funciones relacionadas.

» help sin ↵

lookfor texto Informa de las funciones y comandos en los que aparece el texto.

» lookfor sin ↵

Operaciones Básicas con MatLab

Manejo de variables:

En MATLAB como en cualquier otro lenguaje de programación, y/o asistente matemático se utilizan variables. No es necesario declarar los nombres de las variables no sus tipos como en algunos lenguajes. Cualquier variable en MatLab puede adoptar valores reales, complejos y enteros. En Matlab todas las variables son arreglos. Incluso un valor escalar es un arreglo de dimensión o tamaño 1x1. Las variables deben tener un nombre según ciertas reglas. Estas reglas son:

- NO pueden comenzar con un número, aunque si pueden tener números y/o algunos caracteres especiales (variable_1 es un nombre de variable válido).
- Las mayúsculas y minúsculas se diferencian en los nombres de variables por trabajar con C nativo. (A y a son dos variables diferentes)
- Los nombres de variables no pueden contener operadores ni puntos. (No es válido usar /, *, -, +, ...)
- Si se trabaja con complejos sólo puede utilizarse un de los nombres i y/o j para variables. Ver complejos.
- No es necesario definir el tipo de variable o tamaño (si se usa un vector y después se expande, no hay problema)
- ans es la variable por omisión provista por MATLAB
- Matlab realiza la asignación de memoria a variables durante la ejecución.

» x=3↵	x es de tipo real
» x='mensaje'↵	x es de tipo literal (use comillas simples alt-39)
» syms x↵	x es un símbolo
» x=[2 7 4] ↵	x es un vector
» x=2+3i↵	x es de tipo complejo
» x↵	muestre el contenido actual de la variable
» whos x↵	muestre el tipo actual de la variable
» disp(x) ↵	muestre solamente el contenido
» x=input('¿dato?') ↵	ingrese un valor para una variable desde el teclado
» exp(x)/3↵	
» ans↵	la variable ans contiene el último resultado
» y=2*ans↵	y lo puede usar

Si se utiliza como nombre de variable algún nombre de función definido por MatLab, la variable anula la función hasta tanto no sean borradas las variable con el comando **clear** o se salga de matlab.

```
» sin=4 ↵
sin =
     4
» sin(3) ↵
???  Index exceeds matrix dimensions.
» sin(pi) ↵
Warning: Subscript indices must be integer values.
???  Index exceeds matrix dimensions.
```

En el caso del ejemplo anterior, se redefine el nombre de la función seno (**sin**) como variable. Por lo tanto en adelante no se puede usar como función. En el ejemplo marca error al intentar sacar el seno al valor 3 ó pi, porque ha sido redefinido su nombre como variable y a sido anulado como función.

Nombre de variable	Significado	Valor
eps	Epsilon de la máquina	2.2204e-16
pi	π	3.14159…
i y j	Unidad imaginaria	$\sqrt{-1}$
Inf	Infinito	∞
NaN	No es número	
date	Fecha	
flops	Contador de operaciones de pto flotante	
nargin	Numero de argumentos de entradas en una función	
nargout	Numerote argumentos de salida en una función	

Instrucciones de MATLAB

Si omite el nombre de la variable y el signo "=", MATLAB automáticamente crea la variable ***ans*** para guardar el resultado. Todos los nombres de funciones deben estar dados en letras minúsculas.

Expresiones y variables:

Matlab es un lenguaje basado en expresiones formadas por variables, operadores y funciones. Una sentencia es la asignación de la evaluación de una expresión a una variable (variable = expresión). Tras interpretar y evaluar expresión, el resultado se

visualiza en pantalla y se asigna a la variable. Si se omite variable, el resultado es asignado a la variable por defecto ans (answer). Se debe tener en cuenta lo siguiente:

- Una sentencia termina con un retorno de carro.
- Si una sentencia ocupa más de una línea se puede continuar en la siguiente siempre que terminemos la línea con tres o más puntos (...) seguidos de un retorno de carro.
- Es posible escribir varias sentencias en la misma línea siempre que se separen por una coma (,) o un punto y coma (;).
- Si no se quiere que el resultado de una expresión se visualice en pantalla se debe terminar la sentencia correspondiente con punto y coma (;).
- Por defecto, Matlab diferencia las mayúsculas de las minúsculas. Por ejemplo, la variable temp no es igual que la variable Temp. Sin embargo, es posible modificar esta opción de modo que no se produzca distinción entre mayúsculas y minúsculas.

Una *variable* es un nombre que se da a una entidad numérica, que puede ser una cadena, una matriz, un vector o un escalar. El valor de esa variable, e incluso el tipo de entidad numérica que representa, puede cambiar a lo largo de una sesión de MATLAB o a lo largo de la ejecución de un programa. La forma más normal de cambiar el valor de una variable es colocándola a la izquierda del *operador de asignación* (=).

Una expresión de MATLAB puede tener las dos formas siguientes: asignando su resultado a una variable, variable = expresión

Una expresión en MATLAB, puede ser:

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

Si cualquiera de las anteriores se escribe en la línea de comandos o prompt (») del MATLAB, este devolverá el nombre de la variable y su valor (en caso de que la expresión tenga nombre, de no tenerlo, MATLAB devolverá ans = resultado).

Ya que MatLab se basa en el álgebra de matrices, estas pueden estar formadas por un sólo elemento (escalar), por una fila o una columna (vector) o por una serie de filas y columnas (matriz propiamente dicha).

»A=1↵

Define A como un escalar de valor 1. Al definir A, automáticamente MatLab presenta en pantalla su valor si no ha sido colocado el punto y coma (;) al final de la instrucción.

A =

```
        1
```

Para no presentar el valor de la variable creada, debe agregarse punto y coma (**;**) al final del comando o instrucción.

Después de crear una variable, puede presentarse su valor en pantalla escribiendo la variable después del prompt (**»**).

```
»A↵
A =
        1
```

En Matlab es correcto declaraciones de variables de este tipo

```
» sin=1; ↵
» sin+1 ↵
ans=
        2
```

De esta forma ***sin*** pasa a ser una variable que sobrescribe el valor original que tenía como función seno. Para recuperar el valor original de la función seno, basta ejecutar

```
>> clear sin ↵
```

Cadenas de caracteres

Las variables de cadena son arreglos. Una variable de cadena v definida por:

```
» V='yamil'
V =
        Yamil
```

Equivale a un vector de la siguiente manera

```
» V=['y', 'a', 'm', 'i', 'l']
V =
        Yamil
```

Y la variable v se puede convertir en un vector columna de la siguiente manera v’

Otros ejemplos de manejo de cadenas

» x='Matematica';	Asignación de una cadena (use comillas simples)
» x(4)	Manejo de un carácter de la cadena, use un índice En MATLAB los índices se escriben entre Paréntesis y son numerados desde 1
» t=x(2:5);	manejo de una subcadena, use: (inicio: final)
» n=length(x)	Longitud de la cadena
» c=strcat(x, t)	Concatenación de cadenas

» help strfun Liste las funciones para cadenas

Las funciones para manejo de cadenas son similares a las usadas en el lenguaje C.

Operaciones Matemáticas Básicas

Símbolo	Operación
+	Suma de escalares, vectores o matrices
-	Resta de escalares, vectores o matrices
*	Producto de escalares o de matrices
.*	Producto de escalares o de vectores
\	A\B=inv(A)*B, siendo A y B matrices
.\	A.\B cociente elemental de B entre A (dim(A)=dim(B))
/	Cociente escalar o B/A=B*inv(A), siendo A y B matrices
./	A./B cociente elemental de A entre B (dim(A)=dim(B))
^	Potencia de escalares o potencia escalar de matriz
.^	Potencia elemental de los elementos del primer operando elevados a los correspondientes elementos del segundo operando

Al combinar varias operaciones en una misma instrucción, han de tenerse en cuenta los criterios de prioridad habituales entre ellas, que determinan el orden de evaluación de la expresión.

Las operaciones matemáticas, son operaciones que se pueden realizar digitando solo la operación y dando enter (↵). La maquina resolverá la expresión y asigna el resultado a la variable ans, cuando se trabaja como simple calculadora.

Suma	Resta	Multiplicación	División	División Masiva
» 3 + 4↵	» 3 - 4↵	» 3 * 4↵	» 3 / 4↵	» [6 9] / 3↵
ans =	ans =	ans =	ans =	ans =
7	-7	7	0.7500	2 3

» 3 * (2 + 14.7 – 4 / 6) / 3.5 ↵
ans =
13.7429

» help ops↵ % Lista todos los operadores y caracteres especiales

Otras operaciones fundamentales:

Potencia	Raíz Cuadrada	Raíz Cuadrada	División Derecha
» 2 ^ 4↵	» sqrt(16) ↵	» 16 ^ (1/2) ↵	» 2 \ 4↵
ans =	ans =	ans =	ans =
16	4	4	2

El programa MATLAB permite la ejecución de cálculos de parámetros a través de operaciones matemáticas que realiza en forma secuencial, es decir conforme se vayan introduciendo los comandos o instrucciones y los datos respectivos. Como ejemplo, se considerará el caso en que se quiere calcular el volumen de una esfera de radio r (es importante el tener en cuenta que al definir variables al programa, el mismo diferencia entre letras mayúsculas y letras minúsculas. Para MATLAB la variable r es diferente de la variable R). La formula que permite el cálculo del volumen de una esfera de radio r, es la siguiente:

$$Vol = \frac{4}{3} * \pi * r^3$$

Si el radio de la esfera es 2, la ejecución con MATLAB es la siguiente:

```
» r=2; ↵                    % Definición del radio de la esfera
» vol=(4/3)*pi*r^3; ↵       % Definición de la ecuación para el calculo del volumen
» vol↵
vol =
       33.5103              % Respuesta entregada
```

En ésta ejecución puede observarse que, después de la declaración bien sea de datos o instrucción de ejecución aparece el signo (**;)** el cual permite que una vez pulsada la tecla ENTER, la variable y el valor declarado no aparezcan en la ventana una vez que han sido registrados. De igual forma puede introducirse un comentario después del signo **%** Para obtener el valor del resultado se le solicita al programa escribiendo la correspondiente variable a calcular (vol en éste caso) y pulsando la tecla ENTER.

MATLAB solamente acepta una instrucción por línea, pero en una misma línea puede escribirse dos o más instrucciones si se separan cada una por medio del signo **;** Se debe pulsar la tecla ENTER cada vez que se requiera la ejecución de una línea de instrucción o comando.

Otra forma de ejecutar el cálculo anterior sería:

```
» r=2; ↵                    % Definición del radio de la esfera
» vol=(4/3)*pi*r^3 ↵        % Definición de la ecuación para el calculo del volumen
vol =
       33.5103
```

En esta ejecución puede observarse que después de declarada la expresión para el cálculo de vol, no aparece el signo**;** por lo que al pulsar la tecla ENTER aparece el resultado de la operación.

Otra forma de efectuar la operación anterior con MATLAB será:

```
» r=2;vol=(4/3)*pi*r^3↵    % Cálculo del volumen de una esfera
vol =
       33.5103
```

En este caso se están escribiendo las dos instrucciones en una sola línea de instrucción, separadas por ***;***. Puede observarse que la segunda parte de la línea de instrucción no finaliza con ***;*** por lo que el resultado de la operación aparece una vez pulsada la tecla ENTER.

Precisión utilizada:

En los lenguajes de programación ordinarios, los números se clasifican como sencillo, dobles, reales, enteros y complejos. En MatLab, todas las variables se tratan igualmente con double precisión. No hay distinción entre variables enteras y reales, ni entre variables reales y complejas. La forma como se asigna un valor a una variable depende exclusivamente del usuario. Matlab representa los resultados con exactitud, pero aunque internamente siempre trabaja con cálculos exactos para no arrastrar errores de redondeo, pueden habilitarse diferentes formatos de representación aproximada, que en ocasiones facilitan la interpretación de los resultados. A continuación se citan los comandos que permiten aproximaciones numéricas.

Aproximadamente 16 dígitos significativos en computadoras utilizando aritmética flotante IEEE. El rango aproximado es: 10^-308 a 10^308. El formato de salida para datos en MatLab en format short.

Formatos de salida: Tomando el siguiente ejemplo 4/3↵

a) format short	1.3333
b) format short e	1.3333e+000
c) format long	1.33333333333333
d) format long e	1.33333333333333e00
e) format bank	1.33
f) format hex	3ff5555555555555
g) format rat	4/3

- ✓ format hex Ofrece los resultados en el sistema hexadecimal
- ✓ vpa (‘operaciones’,n) Ofrece el resultado de las operaciones con n dígitos decimales exactos
- ✓ numeric(‘expr’) Ofrece el valor de la expresión de forma numérica aproximada según el formato actual activo.
- ✓ digits(n) Ofrece los resultados con n dígitos exactos.

Una función útil para “embellecer” la salida del Matlab:

»pretty(...) ↵ %Muestra en una forma más legible el resultado de cualquier función (dentro se puede pasarle lo que se quiera solve, limit, diff...).

```
» syms x; ↵
» f=(x^2+3*x)/x ↵
f =
(x^2+3*x)/x
» pretty(f) ↵
                 2
                x  + 3 x
                --------
                   x
```

Constantes incorporadas

pi: Número pi: 3'1415926535897...

```
» pi ↵
ans=
      3.1416
```

i ó j: Unidad imaginaria (raíz cuadrada de-1)

» i ↵ representa al símbolo complejo $\sqrt{-1}$

```
ans =
        0 + 1.0000i
```

MATLAB mantiene una forma especial para los *números muy grandes* (más grandes que los que es capaz de representar), que son considerados como ***infinito***. Por ejemplo, obsérvese cómo responde el programa al ejecutar el siguiente comando:

```
» 2 / 0 ↵
      Inf
```

Es el símbolo ∞

Así pues, para MATLAB el ***infinito*** se representa como ***inf*** ó ***Inf***. MATLAB tiene también una representación especial para los resultados que no están definidos como números. Por ejemplo, ejecútense los siguientes comandos y obsérvense las respuestas obtenidas:

```
» 1 / 0 ↵
» 0 / 0 ↵
      NaN
```

Significa "Not A Number" (valor indeterminado)

En ambos casos la respuesta es ***NaN***, que es la abreviatura de ***Not a Number***. Este tipo de respuesta, así como la de ***Inf***, son enormemente importantes en MATLAB, pues permiten controlar la fiabilidad de los resultados de los cálculos matriciales. Los ***NaN*** se propagan al realizar con ellos cualquier operación aritmética, en el sentido de

que, por ejemplo, cualquier número sumado a un ***NaN*** da otro ***NaN***. MATLAB tiene esto en cuenta. Algo parecido sucede con los ***Inf***.

ans: Variable creada automáticamente para representar el último resultado procesado al que no se le ha asignado previamente ninguna variable.

realmin: El menor número real positivo utilizable

```
» realmin↵                El menor número real en MATLAB
ans =
  2.2251e-308
```

realmax: El mayor número real positivo utilizable

```
» realmax↵                El mayor número real en MATLAB
ans =
  1.7977e+308
```

eps Variable permanente cuyo valor es inicialmente la distancia desde 1.0 al siguiente número en coma flotante más elevado. Se trata de la tolerancia por defecto para operaciones en coma flotante (acuracidad relativa en punto flotante). En máquinas actuales (máquinas IEEE) su valor es 2^(-52)

```
» eps↵        % Epsilon de máquina
ans =
  2.2204e-016
```

```
» exp(1) ↵    % operación de elevar el numero de euler a una potencia (1 en este
              caso)
ans =
        2.7183
```

La forma como se presenta cada uno de las constantes anteriores depende del formato que haya configurado al MatLab.

Funciones Matemáticas en MATLAB:

Al igual que en otros lenguajes de programación, MATLAB presenta numerosas funciones matemáticas, desde las más elementales hasta las de más alto nivel. Estas funciones matemáticas pueden agruparse en tres categorías.

- Funciones Trigonométricas.
- Otras Funciones Elementales.
- Funciones que realizan Tareas.

»help matlab\elfun % Le entrega un listado de funciones que se usan en calculo matemático.

Las funciones matemáticas presentan, para su uso en MATLAB, dos diferencias significativas con respecto a su uso en otros lenguajes de programación (Fortran, C, C++). Primera: Las funciones matemáticas operan con variables complejas sin discriminación alguna. Segunda: Las funciones matemáticas pueden operar con argumentos vectoriales y argumentos matriciales.

Funciones matemáticas elementales que operan de modo escalar

Estas funciones, que comprenden las funciones matemáticas trascendentales y otras funciones básicas, actúan sobre cada elemento de la matriz como si se tratara de un escalar. Se aplican de la misma forma a escalares, vectores y matrices. Algunas de las funciones de este grupo son las siguientes:

La librería MATLAB dispone de una gama muy completa de funciones predefinidas que se corresponden con las funciones matemáticas más utilizadas.

Funciones trigonométricas e hiperbólicas

Función	Inversa	Hiperbólica	Hiperbólica Inversa
sin(Z)	asin(Z)	sinh(Z)	asinh(Z)
cos(Z)	acos(Z)	cosh(Z)	acosh(Z)
tan(Z)	atan(Z) atan2(Z)	tanh(Z)	atanh(Z)
sec(Z)	asec(Z)	sech(Z)	asech(Z)
csc(Z)	acsc(Z)	csch(Z)	acsch(Z)
cot(Z)	acot(Z)	coth(Z)	acoth(Z)

Funciones exponenciales

log(x)	logaritmo natural
log10(x)	logaritmo decimal
exp(x)	función exponencial
sqrt(x)	raíz cuadrada

Funciones específicas de variable numérica

abs(Z)	Módulo o valor absoluto
angle(Z)	Argumento, ángulos de fase
ceil(x)	Redondea los decimales al mayor entero más cercano
ceil(Z)	Aplica la función ceil a real (Z) y a imag(Z)
conj(Z)	Complejo conjugado
fix(x)	Elimina la parte decimal del real x
fix(Z)	Aplica la función fix a real (Z) y a imag(Z)

floor(x)	Redondea los decimales al menor entero más cercano
floor(Z)	Aplica la función floor a real (Z) y a imag(Z)
gcd(x)	máximo común divisor
imag(Z)	Parte imaginaria
lcm(x)	mínimo común múltiplo
mod(x,y)	Similar a *rem* (Ver diferencias con el *Help*)
real(Z)	Parte real
rem(a,b)	Da el resto de la división entre los reales a y b
rem(Z1,Z2)	Resto de la división de los términos de Z1 y Z2
round(x)	El entero más próximo al real x
round(Z)	Aplica la función round a real (Z) y a imag(Z)
sign(x)	Signo del real x (1 si x>0, -1 si x<0)
sign(x)	Función signo. devuelve -1 si <0, 0 si =0 y 1 si >0. Aplicada a un número complejo, devuelve un vector unitario en la misma dirección

```
» sin(pi/6) ↵
ans =
      0.5000
» y=tan(pi/4); ↵
» z=atan2(0,(-1)); ↵
» [y  z] ↵
ans =
      1.0000    3.1416
```

Los anteriores valores corresponden; el primero a *y,* y el segundo a *z*.

Logaritmos:

- ✓ Neperiano:

```
» log(10) ↵
ans =
2.3025850929940459
```

- ✓ Decimal:

```
» log10(20) ↵
ans =
1.3010299956639813
```

- ✓ En base 2:

```
» log2(4) ↵
ans =
    2
```

- ✓ Para otras bases: Hay que aplicar la fórmula:

$$\log_a(b) = \log(b)/\log(a)$$

Por ejemplo, para poder calcular el logaritmo en base 3 de 9 ($\log_3(9)$):

```
» log(9)/log(3) ↵
ans =
2
```

Raíces:

Cuadrada:

```
» sqrt(4) ↵
ans =
     2
```

Pero para las raíces nésimas hay que usar la siguiente propiedad:

$\sqrt[n]{a} = a^{1/n}$

y

$\sqrt[n]{a^m} = a^{m/n}$

Por tanto, para hacer $\sqrt[4]{16}$:

```
» 16^(1/4) ↵
ans =
     2
```

Y para obtener el resultado de $\sqrt[3]{25^2}$:

```
» 25^(2/3) ↵
ans =
     8.5498797333834844
```

Aritmética:

```
»lcm(a,b) ↵          Mínimo común múltiplo (del inglés least common multiple):
»gcm(a,b) ↵          Máximo común divisor (greatest common divisor)
»rem(dividendo,divisor) ↵          Resto de una división entera
```

Redondeo y otras funciones útiles para números:

```
ceil(n) ↵      %Redondea por encima.
floor(n) ↵     %Redondea por debajo.
fix(n) ↵       %Primer entero en dirección a 0.
round(n) ↵     %Redondea al entero más cercano.
abs(n) ↵       %Valor absoluto.
sign(n) ↵      %Signo del número (1, positivo; -1, negativo).
```

Espacio de Trabajo:

El conjunto de variables definidas en una sesión forman el espacio de trabajo (workspace) de la sesión. Para visualizar las variables que se han creado en la sesión actual, pertenecientes al espacio de trabajo actual, puede utilizarse la orden who. Para una información más amplia es preferible utilizar la orden whos. La misma información se obtiene en Matlab accediendo al editor gráfico del espacio de trabajo

Los ejemplos que se han dado se han guardado en variables que están en el espacio de trabajo de MATLAB. Para listar las variables en el espacio de trabajo se utiliza el comando who. Para ver información adicional acerca de estas variables se utiliza el comando whos.

```
» who ↵
Your variables are:
ans      y       z

» whos ↵
  Name      Size          Bytes  Class
  ans       1x2              16  double array
    y       1x1               8  double array
    z       1x1               8  double array

Grand total is 4 elements using 32 bytes
```

Es posible eliminar una variable del espacio de trabajo por medio de la orden clear nombrevariable.

El comando ***clear*** tiene varias formas posibles:

clear sin argumentos, ***clear*** elimina todas las variables creadas previamente (excepto las variables globales).
clear A, b borra las variables indicadas.
clear global borra las variables globales.
clear functions borra las funciones.
clear all borra todas las variables, incluyendo las globales, y las funciones.

Usando argumentos iguales:

Cuando se tiene el mismo argumento al evaluar una expresión

```
» 30*pi/180; ↵
» sin(ans)^2+cos(ans)^2 ; ↵
```

O asignando previamente el valor del ángulo a una variable o identificador como se les llama comúnmente.

```
» y=30*pi/180; ↵
» sin(y)^2+cos(y)^2 ; ↵
```

Calcular funciones trigonométricas para diferentes ángulos referenciados con variables.

```
» conf=pi/180; ↵
» alpha=30; ↵
» beta=52; ↵
» gama=76; ↵
» sin(conf*alpha); ↵
» cos(beta+alfa*gama); ↵
```

Números Aleatorios

rand	Devuelve un número decimal aleatorio distribuido uniformemente en el intervalo [0,1]
rand(n)]	Devuelve una matriz de dimensión nxn cuyos elementos son números decimales aleatorios distribuidos uniformemente en el intervalo [0,1
rand(m,n)	Devuelve una matriz de dimensión mxn cuyos elementos son números decimales aleatorios distribuidos.
rand(size(A))	Devuelve una matriz del mismo tamaño que la matriz A y cuyos elementos son números decimales aleatorios distribuidos uniformemente en el intervalo [0,1]
rand(`seed')	Devuelve el valor actual de la semilla generadora de los números aleatorios uniformes-
rand(`seed',n)	Coloca en la cantidad n el valor actual de la semilla generadora de los números aleatorios uniformes
randn	Devuelve un número decimal aleatorio distribuido según una normal de media o y varianza 1
randn(n)	Devuelve una matriz de dimensión nxn cuyos elementos son números decimales aleatorios distribuidos según una normal de media 0 y varianza 1
randn(m,n)	Devuelve una matriz de dimensión mxn cuyos elementos son números decimales aleatorios distribuidos según una normal de media 0 y varianza 1
randn(size(A))	Devuelve una matriz del mismo tamaño que la matriz A y cuyos elementos son números decimales aleatorios distribuidos según una normal de media 0 y varianza 1
randn(`seed´)	Devuelve el valor actual de la semilla generadora de los númeos aleatorios normales
randn(`seed´, n)	Coloca en la cantidad n el valor actual de la semilla generadora de los números aleatorios uniformes.

Operadores relacionales

El lenguaje de programación de MATLAB dispone de los siguientes operadores relacionales:

<	menor que
>	mayor que
<=	menor o igual que
>=	mayor o igual que
==	igual que
~=	distinto que 7

En MATLAB los operadores relacionales pueden aplicarse a escalares, vectores y/o matrices, y eso hace que tengan un significado especial.

Al igual que en C, si una comparación se cumple el resultado es 1 (*true*), mientras que si no se cumple es 0 (*false*). Recíprocamente, cualquier valor distinto de cero es considerado como *true* y el cero equivale a *false*.

```
» a=1;          » a=1;          » a=1;          » a=1;
» b=2;          » b=2;          » b=2;          » b=2;
» a==b          » a~=b          » a>b           » a<b
ans =           ans =           ans =           ans =
     0               1               0               1
```

Operadores lógicos

Los operadores lógicos de MATLAB son los siguientes:

&	and
|	or
~	Negación lógica
xor(x)	Intercambia ceros por unos y viceversa
all(x)	Devuelve 1 sólamente si todos los elementos de x son diferentes de cero. Cierta cuando todos los elementos les vector son unos
any(x)	Devuelve 1 si cualquiera de los elementos de x es diferente de cero, de lo contrario devuelve 0. Cierta si algún elemento del vector es no nulo

Para argumentos matriciales, any y all trabajan por columnas para devolver un vector fila con el resultado para cada columna. Aplicando la función dos veces, any(any(A)), siempre reduce la matriz a una condición escalar.

Saliendo y Guardando el Espacio de Trabajo

Para salir de MATLAB se escribe quit ó exit sobre el prompt. Al terminar una sesión de MATLAB, las variables en el espacio de trabajo se borran. Si deseas guardar tu espacio de trabajo escribes save.

El comando save es el instrumento esencial para guardar datos en ficheros tipo matlab.
Su recíproca es la instrucción load.

save *file var opciones* Almacena todas las variables indicadas en el fichero file de formato matlab binario o ASCII dependiendo de las opciones.

load *file* Recupera todas las variables del fichero file.

Entre las opciones se encuentran

-ascii Salva los valores en formato ASCII de 8 dígitos.
-double Salva los valores en formato ASCII de 16 dígitos.
-tabs Separa los valores por tabuladores (sólo con las opciones anteriores).

Las distintas modalidades de uso de ambos comandos se presentan a continuación.

save Almacena todas las variables del espacio de trabajo en el fichero de formato matlab binario “matlab.mat”.
save X Y Idem, pero almacenando sólo las variables X e Y.
save file Idem, pero usando el fichero “file.mat”
save -ascii Almacena todas las variables del espacio de trabajo en el fichero de formato matlab “matlab”.
save -ascii-double Idem en formato ASCII de 16 dígitos.
save –ascii -tabs Idem en formato ASCII de 8 dígitos con valores delimitados por tabuladores
load Lee todas las variables guardadas con el comando save en el fichero de nombre matlab.mat
load file Lee las variables del fichero binario file.mat
load file.txt Lee el fichero ASCII de nombre file.txt

save matlab.mat: save guarda todas las variables en un archivo llamado matlab.mat. Se puede utilizar save y load con otros nombres de archivos, ó para guardar solo variables seleccionadas

Ejemplo: save **temp X Y Z**

Este ejemplo guarda las los valores de las variables X, Y, Z en el archivo temp.mat. Usando el comando load temp se obtienen nuevamente los valores del archivo temp.mat. load y save también pueden importar y/o exportar datos de archivos ASCII.

» a=5; ↵

```
» b=12; ↵
» c=2; ↵
» save datos ↵      %(Datos es el nombre del fichero donde se guardan las
                    variables.)
```

Para recuperarlos:

```
» load datos↵       %(De nuevo, datos es el nombre del fichero.)
» a↵
a =
      5
» b-c↵
ans =
      10
```

Guardando y leyendo datos en ficheros cualesquiera

Siempre que se quieran leer o escribir datos en un fichero cualquiera (que no tiene porqué ser de formatos ASCII o Matlab), será necesario utilizar en primer lugar el comando fopen para abrirlo. Después se usarán los comandos correspondientes de lectura y escritura (fload, fwrite, fscanf, fprintf, etc) con el fin de realizar las correspondientes operaciones de lectura o escritura en él. Por último, se utiliza el comando fclose para cerrar el fichero.

El fichero que se abre puede ser nuevo o puede existir previamente (con la finalidad de ampliar su contenido o simplemente leerlo.) El comando fopen devuelve un identificador de fichero que consiste en un entero no negativo asignado por el sistema operativo al fichero que se abre. El indicador es realmente una referencia para el manejo del fichero abierto que posteriormente será utilizada para leerlo (comando read), escribir en él correctamente, fopen devuelve -1 como identificador de fichero. Como identificador de fichero genérico suele utilizarse fid. A continuación se muestra la sintaxis de las diferentes instrucciones y algunos ejemplos

fid=fopen*(´file´,'permiso','formato')* Abre el fichero file y le asigna un número de identificación que se gua rda en la variable fid. A partir de su ejecución, cualquier referencia a ese fichero se hará con fid. Si no se ha podido abrir el archivo, fid toma el valor –1.
ok = fclose (*fid*) Cierra el archivo identificado con *fid* o todos (**all**). La variable ok toma el –1 cuando la instrucción no ha sido completada correctamente y 0 en otro caso.

Se pueden dar permisos para leer, escribir, añadir, etc. Asimismo, el formato indica si es el nativo de la máquina, IEEE, Vax, Cray, etc.
fid=fopen('pepe','a+','n') Abre el fichero 'pepe' para leer y añadir (ya debe estar creado), utilizando el formato nativo de la máquina.

[filename,permission,architecture]=fopen(fid) Devuelve el nombre del fichero, el tipo de permiso y el formato numérico de la arquitectura especificada referente al fichero cuyo identificador es fid

n = fprint/fwrite (*fid*,*´formato´*, *a*) Escribe los valores de la variable a en el fichero ASCII o binario respectivamente. Sus opciones son análogas a las instrucciones de lectura.
[a ,n] = fscanf/fread *(fid, ´formato´, m)* Lee datos del archivo ASCII o binario respectivamente, identificado por *fid*. Los *n* datos leídos correctamente se guardan en la variable *a*. La variable *m* indica el número de datos que se desean leer; si *m* es una matriz, se leerán tantos datos como elementos tenga, rellenándose por columnas. Para leerlos todos *m=inf*.

El argumento de formato consiste en una cadena formada por caracteres de escape (precedidos del carácter "\") y por caracteres de conversión según los diferentes formatos (precedidos del carácter "%"). Los posibles caracteres de escape y de conversión son respectivamente:

\n Se ejecuta el paso a nueva línea
\t Se ejecuta un tabulador horizontal
\b se ejecuta un paso hacia atrás de un solo carácter (backspace), borrando el contenido del carácter actual en caso de que exista
\r Se ejecuta un retorno de carro
\f Se ejecuta un salto de página (form feed)
\\ Se ejecuta el carácter backslash
\' Se ejecuta el carácter comilla simple
%d Enteros en el sistema decimal
%o Enteros en el sistema octal
%x Enteros en el sistema hexadecimal
%u Enteros sin signo en el sistema decimal
%f Reales de punto fijo
%e Reales de punto flotante
%g Utiliza d, e o f, seleccionando el de mayor precisión en el mínimo espacio
%c Caracteres individuales
%s Cadena de caracteres

ok = frewind(*fid*) Coloca el puntero al inicio del archivo *fid*.
ok = fseek (fid, n, origen) Coloca el puntero del archivo identificado con *fid* en la posición indicada por la variable *n* (si *n*>0 se avanza el puntero, en caso contrario se retrasa). La variable carácter *origen* indica desde donde se empieza a mover el puntero, tomando los valores: **bof** (inicio del fichero),
cof (posición actual) o **eof** (final del archivo).
ok = ftell (fid) Indica el número de bytes, contados desde el principio del archivo, hasta la posición donde se encuentra el puntero.

Comando DISP

La orden disk exhibe un numero, vector, matriz o cadena en la ventana de comandos sin tener que especificar un nombre de variable, así, puede servir para exhibir mensajes o datos en pantalla. En general sirve para escribir texto de salida o vectores de resultados.

La sintaxis de la orden es: **disp(X);** ó **disp X;**

X Puede ser:

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

El siguiente ejemplo ilustra el uso de disp:

%Ejemplo de uso de disp.

```
» b=3; ↵
» disp(b) ↵
3
» a = [1, 2, 3, 4, 9 11]; ↵          % a es un vector fila de 6 elementos
» disp(a); ↵
1, 2, 3, 4, 9 11
» a = [1, 2 , 7 ; 6, 3, 4]; ↵        % a es una matriz de 2 filas y tres columnas.
» disp(a); ↵
    1     2     7
    6     3     4
» a = 'El texto se puede escribir a través de variables así ';↵% Cadena de texto
disp(a); ↵
El texto se puede escribir a través de variables así

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

También se puede usar el comando Display, funciona de manera similar a Disp. Con la diferencia que display incluye en la respuesta el nombre de la variable.

Comando INPUT

Se utiliza para que el programa pida valores de variables mientras se ejecuta.
La sintaxis de la orden es:

variable = input (texto);

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

- Una variable o,
- Una cadena.

El siguiente ejemplo ilustra el uso de input:

```
a = 0;     % hace válido el nombre de variable a.
a = input( ‘ Teclee el valor de a: ‘);
tex = ‘ Cual es el nuevo valor de a? ‘;
a          % Al escribir el nombre de una variable (sin punto y coma al final
           % MATLAB muestra su valor.
a = input(tex);
a
```

La salida de este programa será:

```
Teclee el valor de a: (espera)
a =
xxx      % aquí se imprime el valor asignado para a.
Cual es el nuevo valor de a? (espera)
a =
yyy
```

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

ÁLGEBRA LINEAL con MATLAB

El álgebra lineal es la herramienta fundamental para el aná lisis de los métodos numéricos y se tiene que las capacidades de operación del programa MATLAB están basadas en las operaciones de vectores y matrices. El fundamento del álgebra lineal está en el planteamiento y solución de ecuaciones lineales. Los comandos de MATLAB
trabajan con ecuaciones lineales en notación de matrices.

Vectores y Matrices: Un vector puede considerarse como un arreglo unidimensional de datos, mientras que una matriz es un arreglo rectangular o bidimensional de datos. En notación matemática, las matrices se encierran entre corchetes [ ] y siguen ciertas reglas matemáticas. Una matriz del orden m por n, indica que la misma está formada por m filas y n columnas. En el caso de vectores, los mismos pueden considerarse como un caso especial de matriz en el cual se pueden presentar dos definiciones: vector fila, constituido por un arreglo de 1 por n (1 fila – n columnas); vector columna constituido por un arreglo de n por 1 (n filas – 1 columna).

ARREGLOS[2] UNIDIMENSIONALES

Vector Fila y vector Columna

La primera forma de interactuar con MatLab es a través de la línea de comandos. Puede ejecutarse un comando si este está escrito después del símbolo **»** y se presiona la tecla Enter. (↵)

MATLAB trabaja esencialmente con matrices numéricas rectangulares. La manera más fácil de entrar matrices pequeñas es enumerando los elementos de ésta de tal manera que:

- Los elementos estén separados por blancos ó comas.
- Los elementos estén cerrados entre corchetes, [ ].
- Deje el final de cada fila con punto y coma (**;**).

VECTOR FILA

OPERADOR DOS PUNTOS (:): Este operador es muy importante en MATLAB y puede usarse de varias formas. Se sugiere al lector que practique mucho sobre los ejemplos contenidos en este apartado, introduciendo todas las modificaciones que se le ocurran y haciendo pruebas abundantes (¡Probar es la mejor forma de aprender!).

Es necesario aclarar, que para ingresar valores a un vector se utilizan las llaves cuadradas “[ ] “ y para hacer referencia bien sea para usar o modificar algún valor a través de los paréntesis “( )”

Para iniciar, defínase un vector **x** con el siguiente comando:

```
» x=1:5 ↵
x =
     1  2  3  4  5
```

Se asignan a la variable x los valores del 1al 5 de forma secuencial de 1 en 1 y no se coloca “**;**” al final de la línea con el fin de ver la asignación realizada a la variable x tal como se ilustra a la izquierda.

```
» x = [1  10  2  4]; ↵
```

Se asigna a la variable x, 4 valores distintos y que no son secuenciales.

```
» y = 5:-1:1; ↵
x =
     5  4  3  2  1
```

Se asignan a la variable *y,* los valores del 5 al 1 de forma secuencial inversa de 1 en 1

```
» a=[1 3,2 5] ↵
a =
     1    3    2    5
```

Se pueden separar usando mezclas de espacios y comas.

```
» y=[2, 5, 4, ...
```

Para continuar en la siguiente línea se **...**, escriba la

[2] Arreglo es la palabra que se usará en adelante para referirse a cualquier composición de números, es decir un vector, una matriz o composiciones de números de mayor dimensión.

7, -3] ↵ continuación de la línea siguiente y así sucesivamente

En cierta forma se podría decir que el operador (:) representa un *rango*: en este caso, los números enteros entre el 1 y el 5. Por defecto el incremento es 1, pero este operador puede también utilizarse con otros valores enteros y reales, positivos o negativos. En este caso el incremento va entre el valor inferior y el superior, en las formas que se muestran a continuación.

» x=0:0.25:1↵ Se asignan a la variable x, los valores del 0 al 1 con incrementos de 0.25

```
x =
    0    0.2500    0.5000    0.7500    1.0000
```

```
» m=3.12; ↵
» a=0:m:10↵
a =
    0    3.12    6.24    9.36
» w = [1  2  3  4  5] ↵
w =
    1  2  3  4  5
```

Posiciones de un vector

» w=[1 6 2 8 12 13]; ↵

w	1	6	2	8	12	13
	1	2	3	4	5	6

Se puede hacer referencia a un valor o una posición del vector w, simplemente indicando entre paréntesis la posición que ocupa dicho valor dentro del vector. Por ejemplo si se quisiera trabajar con el valor 8 que ocupa la posición 4 del vector w se le indica al Matlab así: w(4).

```
»w(4) ↵
ans=
    8
```

Se puede hacer referencia la posición a traves de un valor numérico o a traves de una variable a la cual se le haya definido un valor previasmente. Ejemplo

```
» x=4; ↵
» w(x) ↵
ans=
    8
```

Se puede hacer referencia a un rango de valores del vector indicando las pociones de dicho rango. Por ejemplo si se quisiera referenciar los valores 6 2 y 8 simplemente se hace referencia a las posiciones 2 a la 4 así:

```
» w(2:4) ↵
ans =
       6  2  8
```

igual que cuando se referencia una siola poisicon a traves de una variable, ahora se puede referenciar varios valores haciendo uso de valores asignados a una variable.

```
» x=2:4;
» w(x) ↵
ans =
       6  2  8
» m=[2  5  4  8  7  3  6  9  2  5  4  2] ↵
m =
   2    5    4    8    7    3    6    9    2    5    4    2
» m(2:2:6) ↵          % Lista los valores de las posiciones 2 4 y 6
ans =
   5    8    3
```

Se puede modificar un valor del vector simplemente cambiando su valor haciendo referencia a la posición del valor a cambiar así:

```
» w(3)=15↵
w =
1  6   15   8   12   13
```

Note que a pesar de haber cambiado la posición 3 cuyo valor era 2 por el valor 15, al no haber dado punto y coma (;) al final, Matlab devuelve todos los valores de w.

Si se asigna un valor en una posición que no existe y está adelante de la última posición de una vector determinado, el Matlab asigna ceros (0) a las posiciones faltantes.

```
» w(10)=7↵
w =
       1  6  15  8  12  13  0  0  0  7
```

w	1	6	2	8	12	13	0	0	0	7
	1	2	3	4	5	6	7	8	9	10

En el ejemplo anterior asignó ceros (0) a las posiciones 7,8 y 9.

Un índice puede ser un vector. Si x y v son vectores, entonces x(v) es [x(v(1)), x(v(2)), ...,x(v(n))]. Para matrices, los índices de vectores permiten acceso a submatrices contiguas y no-contiguas.

Se pueden definir o *redefinir* variables, por ejemplo:

```
»A=[1  2  3] ↵
 A=
        1   2   3
```

El ejemplo anterior define o redefine A como un vector de tres elementos, A(1)=1, A(2)=2 y A(3)=3. Estos elementos deben separase con espacios en blanco o por comas (,) si se desea tener un vector fila.

Definir un vector con los 6 primeros números primos

```
» primos=[2 3 5 7 11 13]; ↵
```

También se puede usar la coma, en lugar de separar con espacios

```
» primos=[2,3,5,7,11,13]; ↵
```

Quedaría guardado de la siguiente manera:

primos	2	3	5	7	11	13	Valores
	1	2	3	4	5	6	Posición

Los elementos de un arreglo se identifican por el index: El index representa la posición que ocupa cada valor dentro del vector

En el siguiente ejemplo se indica al matlab que muestre el valor que ocupa la primera posición dentro del vector primos. En el caso del ejemplo el valor 2.

```
» primos(1) ↵
ans =
    2
```

En el siguiente ejemplo se indica al matlab que muestre el valor que ocupa la segunda posición dentro del vector primos. En el caso del ejemplo el valor 3.

```
» primos(2) ↵
ans =
    3
```

Se puede hacer referencia a datos de un vector a través de valores que sirven como index, y están guardados en una variable.

```
» x=[8  7  9  5  6];
» p=[2  4  1];          vector para direccionar al vector  x
» t=x(p)                t  contiene los elementos de posición 2, 4 y.1 del vector x
t =
   7    5    8          Valores de las posiciones 2 4 y 1.

» a=1:5;
a =
     1     2     3     4     5
» a(:)=0
a =
     0     0     0     0     0
```

En el comando anterior se indica a MatLab que vuelva cero todas las posiciones del vector a.

```
» a(:)=[]
a =
     []
```

En el comando anterior se indica a MatLab que deje vacío todas las posiciones del vector a. Es decir se vuelve un vector vacío.

Comando Length

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

Longitud = length (Vector);

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

El siguiente ejemplo ilustra el uso de lenght:

```
x = [1  2  3  4  5  6  7 ]
l = length(x)
```

Al ejecutar el programa se obtiene la siguiente salida:

```
x =
     1   2   3   4   5   6   7
l =
     7
```

Se puede entrar al arreglo elemento por elemento

» clear primos el comando clear elimina de la memoria la variable primos

» primos(3)=5;primos(4)=7;primos(5)=11;primos(6)=13; ↵

primos	0	0	5	7	11	13
	1	2	3	4	5	6

Como no se definieron valores para las posiciones 1 y 2, MatLab le asigna a las posiciones faltantes automáticamente valores cero (0).

Comandos lógicos en vectores

isempty(x) chequea si un vector x está vacío
Ejemplo

```
» x=[ ] ↵
x =
    [ ]
» isempty(x)
ans =
     1

» x=[1   3   4   6] ↵
x =
     1          3          4          6
» isempty(x) ↵
ans =
     0
```

any(x) determina si el vector contiene algún valor no cero

Ejemplo:

```
» x=[1   3   4   6] ↵
x =
     1          3          4          6
» any(x) ↵
ans =
     1
» x=[ ] ↵
x =
    [ ]
» any(x) ↵
ans =
     0
» x=[0   0] ↵
x =
     0          0
```

```
» any(x) ↵
ans =
     0
```

Una función interesante y muy útil es la función find. Su sintaxis es find (condición) y encuentra los índices de la matriz donde se cumple la condición.

t=find(x) obtiene índices de elementos del vector no ceros

Ejemplo:

```
» x=[1   0   2   5   0   7]; ↵
» t=find(x) ↵
t =
     1         3         4         6
```

Lo anterior indica que el la posición 1,3,4 y 6 del vector x, existen valores diferentes de cero (0).

t=find(x>3) obtiene los índices de cada elemento > 3

```
» x=[1   0   2   5   0   7]; ↵
» t=find(x>3) ↵
t =
     4         6
```

Lo anterior indica que el la posición 4 y 6 del vector x, existen valores mayores a 3.

Ahora suponga:

B=find(Condicion)

```
» A =  [2  4  5  7  4]; ↵
» B = find (A/2 = =  fix (A/2) ) ↵ produce un vector con los índices de los elementos
                                    que son pares, es decir:
B =
        [ 1  2  5 ]
```

Lo anterior indica que en la posición 1 2 y 5 del vector A, existen valores pares.

```
» x = [0 5 3 7] ↵
» y = [0 2 8 7] ↵
» m = (x > y) & (x > 4) ↵
m  =
        [0  1  0  0]
» n = x | y ↵
n  =
```

```
[0  1  1  1]
```

Está claro que, como el resultado de estas operaciones es un vector con elementos del tipo 0-1, éste se puede usar para extraer los elementos deseados. Por ejemplo:

```
» x((x >y) & (x > 4)) ↵
ans =
     5
```

Unión de vectores

Se pueden unir vectores de la siguiente manera

```
» x=1:10; ↵
» y=sin(x); ↵
» [x,y] ↵
ans =
Columns 1 through 6
     1     2     3     4     5     6
Columns 7 through 12
     7     8     9    10   0.84147   0.9093
Columns 13 through 18
0.14112  -0.7568  -0.95892  -0.27942   0.65699  0.98936
Columns 19 through 20
0.41212 -0.54402
```

El resultado es un vector de 20 valores donde los 10 primeros valores corresponden al vector x y los 10 últimos valores corresponden al vector y.

ó se puede unir también de la siguiente manera:

```
» [x y] ↵
ans =
Columns 1 through 6
     1     2     3     4     5     6
Columns 7 through 12
     7     8     9    10   0.84147   0.9093
Columns 13 through 18
0.14112  -0.7568  -0.95892  -0.27942   0.65699  0.98936
Columns 19 through 20
0.41212  -0.54402
```

Se puede generar vectores con rangos de valores de otros vectores. En el siguiente ejemplo se genera el vector z a partir del vector x y y. El vector z finalmente queda con los 10 primeros valores del vector x y los 5 valores primeros del vector y. Es decir el vector z queda con 15 valores.

```
» x=1:100; ↵
» y=sin(x); ↵
» z=[x(1:10),y(1:5)] ↵
z =
Columns 1 through 6
1       2          3          4          5          6
Columns 7 through 12
7       8          9         10  0.84147   0.9093
Columns 13 through 15
0.14112   -0.7568   -0.95892
```

También es posible generar uniones usando funciones aplicables a vectores.

```
» k=[zeros(1,6) ones(1,6)]; ↵
» k ↵
k =
      0    0    0    0    0    0    1    1    1    1    1    1
```

Operaciones con vectores

MATLAB puede operar con matrices por medio de *operadores* y por medio de *funciones*.

Los operadores matriciales de MATLAB son los siguientes:

+	adición o suma
–	sustracción o resta
*****	multiplicación
'	transpuesta
^	potenciación
****	división-izquierda
/	división-derecha
.*	producto elemento a elemento
./ y .	división elemento a elemento
.^	elevar a una potencia elemento a elemento

Estos operadores se aplican también a las variables o valores escalares, aunque con algunas diferencias. Todos estos operadores son coherentes con las correspondientes operaciones matriciales: no se puede por ejemplo sumar matrices que no sean del mismo tamaño. Si los operadores no se usan de modo correcto se obtiene un mensaje de error.

Multiplicación de un vector por un escalar:

```
>> x=[2   5   8   6   2];b=3; ↵
>> c=x*b↵
c =
6   15   24   18   6
```

Suma y resta entre dos arreglos o vectores.

```
» primos=[2,3,5,7,11,13]; ↵
» natural=[1 2 3 4 5 6]; ↵
» primos+natural; ↵
» primos-natural; ↵
```

Arreglo de elementos igualmente espaciados (equiespaciados)

Primeros seis números impares:
» impar=1:2:11; ↵

Primeros seis números pares:
» par=2: 2: 12; ↵

Primeros seis números naturales:
» natural=1: 6; Se omite el uno (1) como incremento.

Incrementos fraccionales y negativos
» impar-inverso=11:-2:1;
» medio=0:0.5: 10; ↵

Multiplicación punto a punto (elemento por elemento) entre dos vectores: realiza la multiplicación término a término de los elementos de dos vectores de igual dimensión.

```
» natural.*prime↵
```

```
» a=[2  3  5];b=[4  3  6];
»c=a.*b↵
c =
       8   9   30
```

División:

```
» natural. / prime
```

Elevar a una potencia los elementos de un arreglo

```
» natural.^2
» a=[2 4 6 5];p=3; ↵          % Eleva los elementos del vector a la potencia p
» c=a.^p↵
c =
       8  64  216  125
```

Muchas funciones construidas en Matlab se pueden aplicar a un arreglo usando el nombre del arreglo como un argumento

```
» angulo=[0:10:90]; ↵
» angulo=pi*angulo/180; ↵
» sin(angulo) ↵
```

VECTOR COLUMNA

Es un arreglo de datos almacenados en una variable tipo matriz de orden m por 1

```
» A=[2;3;5;7;11;13]; ↵
```

Otra forma de ingresar los valores

```
» A=[2↵
3↵
5↵
7↵
11↵
13]; ↵
```

Transponiendo Vectores: Vector transpuesto (')

Si se trata de asignar un rango de datos en forma de columna, se puede asignar el rango inicialmente a un vector fila y luego transponerlo en un vector columna o al contrario, si se tiene un vector columna se puede asignar dichos valores a un vector fila.

```
» At=A';
» At'
```

Ejemplo: Vector Columna:

```
» a=[2;1;2] ↵                » b=[1;2;3] ↵
```

```
a =     2                    b =     1
        1                            2
        2                            3
```

Se transpone a un Vector Fila, tanto a como b de la siguiente manera:

```
» a' ↵                       » b' ↵
ans =  2   1   2             ans =  1   2   3
```

Una forma de transponer un vector fila a un vector columna es de la siguiente manera

```
» a=1:10 ↵
a =
    1    2    3    4    5    6    7    8    9   10
» b=a(:) ↵
b =
    1
    2
    3
    4
    5
    6
    7
    8
    9
   10
```

Producto escalar entre vectores

Multiplicación punto y multiplicación vectorial

```
» a.*b↵                      » a * b' ↵
ans =  2                     ans =  2   4   6
       2                            1   2   3
       6                            2   4   6
```

Multiplicando un escalar por un vector o viceversa.

```
» a*3↵                       » b.*3↵
ans =  6                     ans =  3
       3                            6
       6                            9
```

Dividiendo un Vector por un escalar.

```
» a / 3↵
ans =  0.6667
       0.3333
       0.6667
```

```
» a./3↵
ans =  0.6667
       0.3333
       0.6667
```

```
» a.^b↵
ans =  2
       1
       8
```

```
» a.^2 ↵
ans =  4
       1
       4
```

```
» 2.^a↵
ans =  4
       2
       4
```

Producto vectorial entre vectores

DOT producto entre vectores.

C = DOT(A,B) retorna el producto escalar de los vectores A y B.. A y B deben ser vectores de la misma longitud. Cuando A y B son ambos vectores columna o vectores fila, DOT(A,B) es lo mismo que A'*B ó A*B'

```
» a=[1 4 2] ↵
a =
    1    4    2
» dot(a,a) ↵          Corresponde a (1*1 + 4*4 + 2*2)
ans =
   21
```

```
» a=[1 2 3] ↵
a =
    1    2    3
» b=[4 5 6] ↵
b =
    4    5    6
» dot(a,b) ↵          Corresponde a (1*4 + 2*5 + 3*6)
ans =
   32
» cross(a,b) ↵
ans =
   -3    6   -3
```

CROSS

Calcula el producto cruz entre dos vectores.

La sintaxis de la orden es:

Vector1 = cross(Vector2, Vector 3);

Vector2 y Vector3 son los vectores a los que se les quiere aplicar el producto cruz.

Tanto Vector2 como Vector3 deben ser vectores tridimensionales.

Vector1 es el vector (tridimensional) resultante del producto cruz de Vector2 y Vector3.

El siguiente ejemplo ilustra el uso de cross:

%Ejemplo de uso de cross.

```
x = [1 0 0] ↵
y = [0 1 0] ↵
z = cross(x, y) ↵
```

Al correr el programa se obtiene la siguiente salida:

```
x =
     1     0     0
y =
     0     1     0
z =
     0     0     1
```

Norma de un vector

Norm

```
» format short
» x=0:4
x =
    0    1    2    3    4
» sqrt(1+4+9+16)      Longitud euclidiana
ans =
   5.4772
»sqrt(sum(x.^2));     La misma longitud euclidiana haciendo uso de la función sum.
» norm(x)
ans =
   5.4772
```

GRÀFICOS SIMPLES

```
» x=0:pi/90:pi; ↵
» y=sin(x); ↵
» plot(x,y) ↵
» grid↵
» xlabel('x, radianes') ↵
» ylabel('sen(x)') ↵
» title('Figura1. señal seno') ↵
```

Algunas de las funciones que actúan sobre vectores

Las siguientes funciones actúan sobre vectores (no sobre matrices ni sobre escalares)

Máximo y Mínimo: ***max***(x) encuentra el máximo valor en el vector x y ***min***(x) encuentra el valor mínimo. El argumento x puede ser un vector fila o columna o una matriz. Si x es una matriz, la respuesta es un vector fila que contiene el máximo o el mínimo de cada columna de la matriz. La misma regla se cumple para las funciones ***sort*** y ***sum***

t=**max(x)** El mayor valor del vector x

```
» x=1:10; ↵
» max(x)
ans =
   10
```

[xm,im]=max(x) máximo elemento de un vector. Devuelve el valor máximo xm y la posición que ocupa im

```
» x=[3 1 6 4 9 6 -1] ↵
x =
    3    1    6    4    9    6   -1
» [a b]=max(x) ↵
```

En **a** devuelve el máximo valor y en **b** devuelve la posición que ocupa ese máximo valor dentro del vector x, caso del ejemplo.

```
a =
    9
b =
    5
```

t=**min(x)** Mínimo elemento de un vector. Devuelve el valor mínimo y la posición que ocupa

t=**sum(x)** Suma de los elementos de un vector

```
» x=1:10 ↵
x =
    1    2    3    4    5    6    7    8    9   10
```

```
» sum(x) ↵          Proceso para sumar (1+2+3+4+5+6+7+8+9+10)
ans =
   55
```

t=**diag(x)** t es un vector con la diagonal de la matriz x

t=**cumsum(x)** Suma acumulada

```
» x=[3 1 6 4 9 6 -1] ↵
x =
    3    1    6    4    9    6   -1
» cumsum(x) ↵
ans =
    3    4   10   14   23   29   28

» A=[1 2 3 ; 4 5 6] ↵
A =
    1    2    3
    4    5    6
» cumsum(A) ↵       Por defecto acumula columna por columna
ans =
    1    2    3
    5    7    9
» cumsum(A,1) ↵     Es el mismo valor por defecto
ans =
    1    2    3
    5    7    9
» cumsum(A,2) ↵     Acumula fila por fila
ans =
    1    3    6
    4    9   15
» a=magic(4) ↵
a =
   16    2    3   13
    5   11   10    8
    9    7    6   12
    4   14   15    1
» cumsum(a,2) ↵
ans =
   16   18   21   34
    5   16   26   34
    9   16   22   34
    4   18   33   34
```

cumprod(x) devuelve el vector producto acumulativo de los elementos de un vector

```
» x=1:4; ↵
x =
```

```
    1     2     3     4
» cumprod(x) ↵
ans =
    1     2     6    24
» x=[3 1 6 -1 2] ↵
x =
    3     1     6    -1     2
» cumprod(x) ↵
ans =
    3     3    18   -18   -36
» A=[1 2 3 ; 4 5 6]
A =
    1     2     3
    4     5     6
» cumprod(A) ↵      Realiza la multiplicación acumulativa columna por columna
ans =
    1     2     3
    4    10    18
» A=[1 2 3;4 5 6]
A =
    1     2     3
    4     5     6
» cumprod(A,2) ↵
ans =
    1     2     6
    4    20   120
```

t=**mean(x)** Media aritmética

```
» x=1:4 ↵
x =
    1     2     3     4
» mean(x) ↵
ans =
   2.5000
```

t=**median(x)** Mediana

t=**std(x)** Desviación estándar

$$s=\left(\frac{1}{n-1}\sum_{i=1}^{n}(x_i-\bar{x})^2\right)^{1/2} \quad (1) \qquad s=\left(\frac{1}{n}\sum_{i=1}^{n}(x_i-\bar{x})^2\right)^{1/2} \quad (2)$$

n es el numero de elementos del vector

Para vectores, **s=std(x)** retorna la desviación Standard usando la formula 1. Para matrices s=std(x) es un vector fila conteniendo la desviacion standard de cada columna.

s=std(x,flag) para flag =0 es lo mismo que usar solo std(x), y para flag=1 o valor diferente a cero (0) el cálculo se hace con la ecuación 2.

```
» x=[1 5 9;7 15 22] ↵
x =
    1    5    9
    7   15   22
» std(x) ↵
ans =
   4.2426   7.0711   9.1924
» std(x,2) ↵
ans =
   3.0000   5.0000   6.5000
» std(x,1) ↵
ans =
   3.0000   5.0000   6.5000
» std(x,0) ↵
ans =
   4.2426   7.0711   9.1924
```

t=**trapz(x)** Integración numérica trapezoidal.

```
» x=[3 1 6 -1 2]; ↵
» trapz(x) ↵
ans =
   8.5000
```

t=**sort(x)** Ordenamiento ascendente

```
» x=[3 1 6 -1 2]; ↵
x =
    3    1    6   -1    2
» sort(x) ↵
ans =
   -1    1    2    3    6
» a=magic(3) ↵
a =
    8    1    6
    3    5    7
    4    9    2
» sort(a) ↵
ans =
    3    1    2
    4    5    6
    8    9    7
```

t=**dsort(x)** Ordenamiento descendente

```
» dsort(x) ↵
ans =
    6
    3
    2
   -1
    1

» a=magic(3) ↵
a =
    8    1    6
    3    5    7
    4    9    2
» dsort(a) ↵
ans =
    9
    8
    7
    6
    5
    4
    3
    2
```

t=**sortrows(a)** ordena las filas de una matriz **a** en orden ascendente por el primer elemento de la fila

```
» a=magic(3) ↵
a =
    8    1    6
    3    5    7
    4    9    2
» sortrows(a) ↵
ans =
    3    5    7
    4    9    2
    8    1    6
```

t=**sortrows(a,n)** ordena las filas de una matriz en orden ascendente por la columna **n** de la matriz **a**

```
» a=magic(3) ↵
a =
    8    1    6
    3    5    7
    4    9    2
» sortrows(a,3) ↵
ans =
    4    9    2
```

```
    8    1    6
    3    5    7
```

prod(x) producto de los elementos de un vector

```
» x=1:4; ↵
x =
    1    2    3    4
» prod(x) ↵
ans =
   24
```

[y,i]=sort(x) Ordenación de menor a mayor de los elementos de un vector x. Devuelve el vector ordenado y, y un vector **i** con las posiciones iniciales en x de los elementos en el vector ordenado y.

```
» x=[1 5 2 -2 -1 0] ↵
x =
    1    5    2   -2   -1    0
» [y,i]=sort(x) ↵
y =
   -2   -1    0    1    2    5
i =
    4    5    6    1    3    2
```

En el ejemplo anterior el **-2** ocupaba la posición **4** originalmente, el valor 5 ocupaba la posición 2 originalmente y así sucesivamente.

linspace Equiespacea un vector Linealmente.

linspace(x1, x2) Genera u vector fila de 100 valores igualmente espaciados entre x1 y x2.

linspace(x1, x2, N) Genera N puntos entre x1 y x2.

```
» d=linspace(-pi,pi,10) % entrega vector fila con 10 valores linealmente distribuidos
                          entre –pi y +pi
d =
-3.1416 -2.4435 -1.7453 -1.0472 -0.3491 0.3491 1.0472 1.7453 2.4435 3.1416
```

logspace(a, b, n) Genera un vector con espaciado logarítmico desde 10^a hasta 10^b de longitud n.

```
» d=logspace(0,3,8)
d =
1.0e+003 *
0.0010 0.0027 0.0072 0.0193 0.0518 0.1389 0.3728 1.0000
```

En realidad estas funciones *se pueden aplicar también a matrices*, pero en ese caso se aplican por separado a cada columna de la matriz, dando como valor de retorno un vector resultado de aplicar la función a cada columna de la matriz considerada como vector. Si estas funciones se quieren aplicar a las filas de la matriz basta aplicar dichas funciones a la matriz transpuesta.

MATRICES

Para definir una matriz *no hace falta establecer de antemano su tamaño* (de hecho, se puede definir un tamaño y cambiarlo posteriormente). MATLAB determina el número de filas y de columnas en función del número de elementos que se proporcionan (o se utilizan). ***Las matrices se definen por filas***; los elementos de una misma fila están separados por ***espacios blancos*** o ***comas***, mientras que las filas están separadas por pulsaciones ***enter*** o por caracteres ***punto y coma*** (;).

Modos de introducir datos a matrices

1. Por teclado
2. Generadas por funciones
3. Creadas por los m-files
4. Cargadas desde un fichero de datos externo con el comando load.

Arreglos bidimensionales, se ingresan fila por fila separadas con punto y coma (**;**), y cada fila tiene el mismo tratamiento de un vector fila, es decir separados por coma (**,**) o por espacio como se indicó anteriormente.

» A=[1 2 3**;**4 5 6**;**7 8 9]; ↵

También es posible, ingresar fila por fila separados por un enter entre ellas.

» A=[1 2 3↵
4 5 6↵
7 8 9]; ↵

Note que el corchete (**]**) termina la entrada de datos a la matriz y vuelve al prompt, en las dos primeras líneas o filas de la matriz el MatLab espera la entradas de mas valores.

A partir de este momento la matriz **A** está disponible para hacer cualquier tipo de operación con ella (además de valores numéricos, en la definición de una matriz o vector se pueden utilizar expresiones y funciones matemáticas).

Los elementos de la matriz pueden ser, en general, expresiones:

```
» x = [1   sqrt (23)   2  3*[a(1, 1)   3] ] ↵

» A↵
A =
          1    2    3
          4    5    6
          7    8    9
```

Se hace referencia a un valor almacenado en la matriz, dando como referencia la fila y la columna donde se encuentra dicho valor así:

```
» A(1,1) ↵     Referencia el valor contenido en la matriz A en la fila 1 columna 1
ans=
        1
» size(A) ↵
ans =
    3    3
» B=[9 8 7;6 5 4;3 2 1] ↵
B =
          9    8    7
          6    5    4
          3    2    1
» [A B] ↵                    %yuxtaposicion
ans =
          1    2    3    9    8    7
          4    5    6    6    5    4
          7    8    9    3    2    1
» size(ans) ↵          Indica el numerote fila y columnas que tiene ans.
ans =
          3    6
» ndims(ans) ↵       Indica el numero de dimensiones de la matriz an.s
ans =
    2
» [A;B] ↵                    %B debajo de A
ans =
          1    2    3
          4    5    6
          7    8    9
          9    8    7
          6    5    4
          3    2    1
» size(ans) ↵
ans =
          6    3
```

También se puede generar matrices de la siguiente manera

```
» Lista=[1,2]; ↵
» Lista=[Lista;3,4] ↵
ans =
1 2
3 4

» B = [ [1 2 3]' [2 4 7]' [3 5 8]'] ↵
B =
     1   2   3
     2   4   5
     3   7   8
```

Extraer una submatriz de una matriz

Extraer la primera fila de
» A: A(1,:) ↵

Extraer la segunda columna de
» A: A(:,2) ↵

Mostrar filas de 1 a 2 y columnas de 1 a 2:
» A(1:2,1:2); ↵

»A([m, n],[p, q]) ↵ Devuelve la submatriz de A formada por la intersección de las filas n-ésima y m-ésima y las columnas p-ésima y q-ésima.

```
» A=[1 2 3 4;  4 5 6 7;  7 8 9 10;  8 9 10 11] ↵
A =
    1    2    3    4
    4    5    6    7
    7    8    9   10
    8    9   10   11
» A([1 3],[2 4]) ↵
ans =
    2    4
    8   10
```

Se puede hacer referencia a datos de una matriz a través de posiciones guardadas como datos en variables:

```
» A=[4  7  3;
     5  7  8;
     6  0  9]; ↵

» p=[1 3]; ↵          vector para direccionar las filas de la matriz  a
» q=[2 3]; ↵          vector para direccionar las columnas de la matriz a
```

» t=A(p, q) ↵ t contiene las filas 1 y 3, columnas 2 y 3 de a
t =
7 3
0 9

»A(a:p:b,c:q:d) ↵ %Devuelve la submatriz de A formada por las filas que hay entre la a-ésima y la b-ésima tomándolas de p en p, y por las columnas que hay entre la c-ésima y la d-ésima tomándolas de q en q.

»A(:)↵ % Devuelve un vector columna cuyos elementos son las columnas de A situadas por orden

Como ejemplo se genera a partir de la función magic la siguiente matriz:
» A=magic(6) ↵
» A =

35	1	6	26	19	24
3	32	7	21	23	25
31	9	2	22	27	20
8	28	33	17	10	15
30	5	34	12	14	16
4	36	29	13	18	11

Si se quiere hacer referencia al elemento de la fila 2, columna 3, se escribe:
» A(2,3) ↵
ans =
7

El siguiente comando extrae los 4 primeros elementos de la sexta fila:
» A(6, 1:4) ↵
ans =
4 36 29 13

Los dos puntos aislados representan "todos los elementos". Por ejemplo, el siguiente comando extrae todos los elementos de la 3ª fila:
» A(3, :) ↵
ans =
31 9 2 22 27 20

Para acceder a la última fila o columna puede utilizarse la palabra end, en lugar del número correspondiente. Por ejemplo, para extraer la sexta fila (la última) de la matriz:
» A(end, :) ↵
ans =
4 36 29 13 18 11

El siguiente comando extrae todos los elementos de las filas 3, 4 y 5:
» A(3:5,:) ↵

```
ans =
        31  9   2  22  27  20
         8  28 33  17  10  15
        30  5 34  12  14  16
```

Se pueden extraer conjuntos disjuntos de filas utilizando corchetes [ ]. Por ejemplo, el siguiente comando extrae las filas 1, 2 y 5:

```
» A([1 2 5],:) ↵
ans =
        35   1  6  26  19  24
         3  32   7  21  23  25
        30  5 34  12  14  16
```

En los ejemplos anteriores se han extraído renglones y no columnas por motivos del espacio ocupado por el resultado en la hoja de papel. Es evidente que todo lo que se dice para filas vale para columnas y viceversa basta cambiar el orden de los índices.

El operador dos puntos (:) puede utilizarse en ambos lados del operador (=). Por ejemplo, a continuación define una matriz identidad B de tamaño 6x6 y se van a reemplazar filas de B por filas de A. Obsérvese que la siguiente secuencia de comandos sustituye las filas 2, 4 y 5 de B por las filas 1, 2 y 3 de A,

También se puede hacer referencia a valores de filas o columnas a través de variables tal como se ilustra en el siguiente ejemplo:

```
» a=[1 3 5]; ↵
» b=[1 3]; ↵
» c=magic(5) ↵
c =
   17   24    1    8   15
   23    5    7   14   16
    4    6   13   20   22
   10   12   19   21    3
   11   18   25    2    9
» c(a,b) ↵
ans =
   17    1
    4   13
   11   25
```

Cambio del orden de una matriz: reshape

Redimensiona una matriz. Los elementos de la matriz original son colocados en el número de filas y columnas descrito para la matriz modificada. El orden de colocación es de arriba hacia abajo y de izquierda a derecha. El numero de elementos debe ser el mismo en las dos matrices.

Sintaxis: Matriz_modificada = reshape(Matriz_original, filas, columnas)

Ejemplo:

```
» A = [1 4 7 10; 2 5 8 11; 3 6 9 12] ↵
A =
     1   4   7   10
     2   5   8   11
     3   6   9   12

» B = reshape(A,2,6) ↵
B =
     1  3  5  7   9  11
     2  4  6  8  10  12
```

Note que el orden de la matriz A es de 3*4 es decir 12 elementos que debe ser igual al número de elementos del nuevo orden. 2*6 de B.

Si se da un orden diferente se producirá el siguiente mensaje de error.

```
» B=reshape(A,2,5) ↵
??? Error using ==> reshape
To RESHAPE the number of elements must not change.
```

También se puede cambiar el orden de una matriz de la siguiente manera:
B = A (:) produce B = [1 2 3 4 5 6 7 8 0 0 0 0 3 4 2 1]

Se deja en la variable B los elementos de la matriz pero en forma de vector.

Operaciones elementales con matrices

MATLAB puede operar con matrices por medio de ***operadores*** y por medio de ***funciones***. Se han visto ya los operadores *suma* (+), *producto* (*) y *traspuesta* ('), así como la función *invertir* ***inv( )***.

Los operadores matriciales de MATLAB son los siguientes:

+	Adición o suma
–	Sustracción o resta
*	Multiplicación
'	Transpuesta
^	Potenciación
\	División-izquierda
/	División-derecha
.*	Producto elemento a elemento

./ y .\ División elemento a elemento
.^ Elevar a una potencia elemento a elemento

Estos operadores se aplican también a las variables o valores escalares, aunque con algunas diferencias[3]. Todos estos operadores son coherentes con las correspondientes operaciones matriciales: no se puede por ejemplo sumar matrices que no sean del mismo tamaño.

Si los operadores no se usan de modo correcto se obtiene un mensaje de error. Los operadores anteriores se pueden aplicar también de modo ***mixto***, es decir con un operando escalar y otro matricial. En este caso la operación con el escalar se aplica a cada uno de los elementos de la matriz.

Considérese el siguiente ejemplo:

```
» A=[1 2; 3 4]
A =
     1   2
     3   4

» A*2
ans =
     2   4
     6   8
» A-4
ans =
    -3  -2
    -1   0
```

Genere las siguientes matrices

```
» A = [1 2 3;4 5 6;7 8 9]; ↵   %generación de matriz de 3x3 elementos
» B = [9 8 7;6 5 4;3 2 1]; ↵
```

Se puede sumar estas matrices

```
» S = A+B ↵                    %suma de elementos
```

Resultado, suma elemento a elemento de las matrices

```
S =
   10   10   10
   10   10   10
   10   10   10
```

[3] En términos de C++ se podría decir que son operadores *sobrecargados*, es decir, con varios significados distintos, dependiendo del contexto, es decir, de sus operandos.

Restar las matrices
» S1 = A-B↵ % resta de matrices

Resultado
S1 =
-8 -6 -4
-2 0 2
4 6 8

Multiplicar matrices
» Z = A*B↵ % multiplicación de matrices

Resultado
Z =
30 24 18
84 69 54
138 114 90

Multiplicación de una matriz cuadrada por si misma
» A^2↵
ans =
30 36 42
66 81 96
102 126 150

Multiplicar elemento a elemento
» Z1 = A.*B ↵ % (multiplicación elemento a elemento)

Resultado
Z1 =
9 16 21
24 25 24
21 16 9

Obtención de la inversa
» AA=inv(A); ↵

Si ahora hace lo siguiente:

» AA*A↵

Los operadores de división requieren una cierta explicación adicional. Considérese el siguiente sistema de ecuaciones lineales

Ax = b

En donde x y b son vectores columna, y A una matriz cuadrada invertible. La resolución de este sistema de ecuaciones se puede escribir en las 2 formas siguientes (¡Atención a la 2ª forma, basada en la barra invertida (\), que puede resultar un poco extraña!):

» x = inv(A)*b ↵
» x = A\b↵

Así pues, el operador división-izquierda por una matriz (barra invertida \) equivale a premultiplicar por la inversa de esa matriz.

Ejemplo: Resolver el siguiente sistemas simultaneo de ecuaciones lineales

$$2x + 3y + z + k = 2$$
$$x + 4y + 2z + 9k = 3$$
$$3x - y + z - k = 0$$
$$5x + 2y - 3z + k = 10$$

Se escribe las matrices

» A = [2 3 1 1;1 4 2 9;3 -1 1 -1;5 2 -3 1]; ↵
» C = [2 3 0 10]; ↵
» X = inv(A)*C'; ↵

O ingresando el vector en forma de columna de la siguiente manera:

» A = [2 3 1 1;1 4 2 9;3 -1 1 -1;5 2 -3 1]; ↵
» C = [2; 3; 0; 10]; ↵
» X = inv(A)*C; ↵

Pruebe ahora con X = A\C'↵

La función *linsolve* es la forma más eficiente de que dispone MATLAB para resolver sistemas de ecuaciones lineales. A diferencia del operador barra invertida \, esta función no trata de averiguar las características de la matriz que permitan hacer una resolución más eficiente: se fía de lo que le dice el usuario. Si éste se equivoca, se obtendrá un resultado incorrecto sin ningún mensaje de error.

Las formas generales de la función *linsolve* para resolver Ax=C son las siguientes:

» x = linsolve(A,C') ↵
» x = linsolve(A,C',opts) ↵

Obviamente, si b es una matriz de segundos miembros, x será una matriz de soluciones con el mismo nº de columnas. La primera forma de esta función utiliza la factorización LU con pivotamiento parcial si la matriz A es cuadrada, y la

factorización QR también con pivotamiento por columnas si no lo es. La función *linsolve* da un *warning* si la matriz A es cuadrada y está mas condicionada, o si es rectangular y de rango deficiente. Estos warnings se suprimen si se recoge un segundo valor de retorno r, que representa el inverso de la condición numérica si A es cuadrada o el rango si no lo es:

```
» [x,r] = linsolve(A,b) ↵
```

El argumento opcional opts representa una estructura por medio de la cual el programador proporciona información sobre las características de la matriz.. Los campos de esta estructura se pueden poner a true o a false, y son los siguientes: LT (triangular inferior), UT (triangular superior),

UHESS (forma de Hessenberg superior), SYM (simétrica), POSDEF (definida positiva), RECT (rectangular general) y TRANSA (se desea resolver T = A x b, en lugar de = Ax b).

Aplicando funciones a matrices:

Se define una nueva matriz

```
» A = [1 2 3;4 5 6;7 8 9]; ↵
» C=[10 11;12 13;14 15]; ↵
» A*C ↵
ans =
    76    82
   184   199
   292   316
```

Cálculo del logaritmo en base diez de los elementos de un arreglo

```
» L=log10(A); ↵
L =
         0    0.3010    0.4771
    0.6021    0.6990    0.7782
    0.8451    0.9031    0.9542
```

Otra posibilidad para crear matrices es por medio de sentencias y funciones ya implementadas en Matlab. Las funciones más comunes para construir matrices son eye, zeros, ones, diag, triu, tril, rand, hilb, magic, vander, toeplitz.

Crear una matriz con el mismo número de filas y columnas que A, con todos los elementos iguales a uno.

```
» [m,n]=size(A);ones(m,n); ↵
```

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

También es posible:

```
» ones(size(A)); ↵
```

Se puede recuperar A de la siguiente manera:

```
» (10*ones(size(A))).^L↵
ans =
    1.0000    2.0000    3.0000
    4.0000    5.0000    6.0000
    7.0000    8.0000    9.0000
```

Las matrices se pueden construir por medio de bloques. Por ejemplo, si se ha definido previamente en el sistema una matriz A 3×3, se puede construir una matriz C 6×6 del siguiente modo:

```
» A=ones(3,3) ↵
A =
     1     1     1
     1     1     1
     1     1     1

» C = [A zeros(3,3); ones(3,6)] ↵
C =
     1     1     1     0     0     0
     1     1     1     0     0     0
     1     1     1     0     0     0
     1     1     1     1     1     1
     1     1     1     1     1     1
     1     1     1     1     1     1
```

Crear una matriz con el mismo número de filas y columnas que A: con todos los elementos iguales a cero.

```
» [m,n]=size(A);zeros(m,n); ↵
```

Crear una tabla de arreglos unidimensionales,

```
» angulo=0:10:90; ↵
» seno=sin(pi*angulo/180); ↵
» [angulo' seno'] ↵
```

```
ans =
         0         0
   10.0000    0.1736
   20.0000    0.3420
   30.0000    0.5000
   40.0000    0.6428
   50.0000    0.7660
   60.0000    0.8660
   70.0000    0.9397
   80.0000    0.9848
   90.0000    1.0000
```

Un caso especialmente interesante es el de crear una nueva matriz a partir de otras matrices definidas previamente. Por ejemplo, vamos a ejecutar las siguientes líneas de comandos y a observar los resultados obtenidos:

```
» A=magic(3) ↵
» B=diag(diag(A)) ↵
» C=[A, eye(3); zeros(3), B] ↵
» X = [A A';ones(size(A)) A.^2] ↵
```

Manipulación de los elementos de una matriz.

```
» B=eye(size(A)); ↵
» B([2 4 5],:)=A(1:3,:) (
B =
     1     0     0     0     0     0
    35     1     6    26    19    24
     0     0     1     0     0     0
     3    32     7    21    23    25
     3    19     2    22    27    20
     0     0     0     0     0     1
```

Se pueden realizar operaciones aún más complicadas, tales como la siguiente:

```
» B=eye(size(A)); ↵
» B(1:2,:)=[0 1; 1 0]*B(1:2,:) ↵
```

Como nuevo ejemplo, se va a ver la forma de invertir el orden de los elementos de un vector:

```
» x=rand(1,5) ↵
x =
    0.9103    0.7622    0.2625    0.0475    0.7361
» x=x(5:-1:1) ↵
x =
    0.7361    0.0475    0.2625    0.7622    0.9103
```

Obsérvese que por haber utilizado paréntesis –en vez de corchetes– los valores generados por el operador (:) afectan a los índices del vector y no al valor de sus elementos.

Para invertir el orden de las columnas de una matriz se puede hacer lo siguiente:

```
» A=magic(3) ↵
A =
     8  1  6
     3  5  7
     4  9  2

» A(:,3:-1:1) ↵
ans =
     6  1  8
     7  5  3
     2  9  4
```

Aunque hubiera sido más fácil utilizar la función fliplr(A), que es específica para ello. Finalmente, hay que decir que A(:) representa un vector columna con las columnas de A una detrás de otra.

Matriz nula o vacía A[ ]

Para MATLAB una matriz definida sin ningún elemento entre los corchetes es una matriz que existe, pero que está vacía, o lo que es lo mismo que tiene dimensión cero. Por ejemplo:

```
» A=magic(3) ↵
A =
     8  1  6
     3  5  7
     4  9  2

» B=[ ]↵
B =
     [ ]

» exist(B) ↵
ans =
     [ ]

» isempty(B) ↵
ans =
     1
```

```
» A(:,3)=[ ] ↵          % elimina todos los elementos de la columna 3
A =
        8   1
        3   5
        4   9
```

```
»A(2,:)=[ ] ↵          % Elimina la fila 2 de la matriz A
»A(:,[1,2])=[ ] ↵      % Elimina todos los elementos de la columna 1 y 2 de la matriz A
```

Operaciones lógicas en matrices

La diferencia con C está en que cuando los operadores relacionales de MATLAB se aplican a dos matrices o vectores del mismo tamaño, la comparación se realiza elemento a elemento, y el resultado es otra matriz de unos y ceros del mismo tamaño, que recoge el resultado de cada comparación entre elementos. Considérese el siguiente ejemplo como ilustración de lo que se acaba de decir:

```
» A=[1 2;0 3]; B=[4 2;1 5]; ↵
» A==B↵
ans =
        0 1
        0 0
» A~=B↵
ans =
        1 0
        1 1
```

```
» c=a==b↵              compare igualdad entre matrices (de igual tamaño)
                       El resultado es una matriz binaria (ceros y unos)
»c=a~=b↵               compare si dos matrices no son iguales
                       El resultado es una matriz binaria (ceros y unos)
»c=a>3↵                compare si cada elemento de a es mayor a  3
                       El resultado es una matriz binaria (ceros y unos)
```

```
» A=magic(3) ↵
A =
        8   1   6
        3   5   7
        4   9   2
» m=find(A>4) ↵        Devuelve las posiciones que ocupan los valores que
                       cumplen la condición dada.
m =
        1
        5
        6
        7
```

```
        8
```

Ahora se van a sustituir los elementos que cumplen la condición anterior por valores de 10.

Obsérvese cómo se hace y qué resultado se obtiene:

```
» A(m)=10*ones(size(m)) ↵
A =
        10   1  10
        3   10  10
        4   10   2
```

Donde ha sido necesario convertir el 10 en un vector del mismo tamaño que **m**. Para chequear si hay algún elemento de un determinado valor –por ejemplo 3– puede hacerse lo siguiente:

```
» any(A==3) ↵
ans =
        1 0 0
» any(ans) ↵
ans =
        1
```

Mientras que para comprobar que todos los elementos de **A** son mayores que cero:

```
» all(all(A)) ↵
ans =
        1
```

En este caso no ha hecho falta utilizar el operador relacional porque cualquier elemento distinto de cero equivale a ***true***.

La función ***isequal(A, B)*** devuelve *uno* si las matrices son idénticas y *cero* si no lo son.

Funciones que actúan sobre matrices

Las siguientes funciones exigen que el/los argumento/s sean matrices. En este grupo aparecen algunas de las funciones más útiles y potentes de MATLAB. Se clasificarán en varios subgrupos:

v = poly(A)	devuelve un vector v con los coeficientes del polinomio característico de la matriz cuadrada A
t = trace(A)	devuelve la traza t (suma de los elementos de la diagonal) de una matriz cuadrada A

[m,n] = size(A)	devuelve el número de filas m y de columnas n de una matriz rectangular A
n = size(A)	devuelve el tamaño de una matriz cuadrada A

Otras formas de definir matrices

Existen en MATLAB varias funciones orientadas a definir con gran facilidad matrices de tipos particulares. Algunas de estas funciones son las siguientes:

eye(n)	Forma la matriz unidad de orden (nxn)
zeros(m,n)	Forma una matriz de *ceros con orden* (mxn)
zeros(n)	Ídem de tamaño (nxn)
ones(n)	Forma una matriz de *unos, cuyo orden es* (nxn)
ones(m,n)	Ídem de orden (mxn)
linspace(x1,x2,n)	Genera un vector con n valores igualmente espaciados entre x1 y x2
logspace(a1,a2,n)	Genera un vector con n valores espaciados logarítmicamente entre10^a1 y 10^a2. Si a2 es pi, los puntos se generan entre 10^a1 y pi
rand(n)	Forma una matriz de números aleatorios entre 0 y 1, con distribución uniforme, de tamaño (nxn)
rand(m,n)	Ídem de tamaño (mxn)
randn(n)	Forma una matriz de números aleatorios de tamaño (nxn), con distribución normal, de valor medio 0 y varianza 1.
magic(n)	Crea una matriz (nxn) con la propiedad de que todas los renglones y columnas suman lo mismo
hilb(5)	Crea una matriz de Hilbert de tamaño (5x5). La matriz de Hilbert es una matriz cuyos elementos (i,j) responden a la expresión (1/(i+j-1)). Esta es una matriz especialmente difícil de manejar
invhilb(5)	Crea directamente la inversa de la matriz de Hilbert
kron(x,y)	Produce una matriz con todos los productos de los elementos del vector **x** por los elementos del vector **y**. Equivalente a x'*y, donde **x** e **y** son vectores fila
compan(pol)	Construye una matriz cuyo polinomio característico tiene como coeficientes los elementos del vector **pol** (ordenados de mayor grado a menor)
vander(v)	Construye la matriz de Vandermonde a partir del vector **v** (las columnas son las potencias de los elementos de dicho vector)

Formación de una matriz a partir de otras

Con MATLAB es posible crear una matriz a partir de matrices previas ya definidas, esto se puede hacer a través de lo siguiente:

- recibiendo alguna de sus propiedades (como por ejemplo el tamaño),
- por composición de varias submatrices más pequeñas,
- modificándola de alguna forma.

[m,n] = size(A)	Devuelve el número de filas y de columnas de la matriz A. Si la matriz es cuadrada basta recoger el primer valor de retorno
n = length(x)	Calcula el número de elementos de un vector x
zeros(size(A))	Forma una matriz de *ceros* del mismo tamaño que una matriz A previamente creada
ones(size(A))	Idem con *unos*
A=diag(x)	Forma una matriz diagonal A cuyos elementos diagonales son los elementos de un vector ya existente x
x=diag(A)	Forma un vector x a partir de los elementos de la diagonal de una matriz ya existente A
diag(diag(A))	Crea una matriz diagonal a partir de la diagonal de la matriz A
triu(A)	Forma una matriz triangular superior a partir de una matriz A (no tiene porqué ser cuadrada)
tril(A)	Ídem con una matriz triangular inferior
E = rref(A)	Reducción a forma de escalón (mediante la eliminación de Gauss con pivotamiento por columnas) de una matriz rectangular **A**
U = chol(A)	Descomposición de Cholesky de matriz simétrica y positivodefinida. Sólo se utiliza la diagonal y la parte triangular superior de **A**. El resultado es una matriz triangular superior tal que A =U'*U
c = rcond(A)	Devuelve una estimación del recíproco de la condición numérica de la matriz **A** basada en la norma sub-1. Si el resultado es próximo a 1 la matriz **A** está bien condicionada; si es próximo a 0 no lo está.
rot90(A,k)	Gira k*90 grados la matriz rectangular A en sentido antihorario. k es un entero que puede ser negativo. Si se omite, se supone k=1
flipud(A)	halla la matriz simétrica de A respecto de un eje horizontal
fliplr(A)	halla la matriz simétrica de A respecto de un eje vertical
reshape(A,m,n)	Cambia el tamaño de la matriz A devolviendo una matriz de tamaño *mxn* cuyas columnas se obtienen a partir de un vector formado por las columnas de A puestas una a continuación de otra. Si la matriz A tiene menos de *mxn* elementos se produce un error.

Otras funciones que actúan sobre vectores y matrices

Las siguientes funciones pueden actuar sobre vectores y matrices, y sirven para chequear ciertas condiciones:

exist('var')	Comprueba si el nombre var existe como variable, función, directorio, fichero,
isnan(A)	Chequea si hay valores *NaN* en A, devolviendo una matriz de

	unos y ceros del mismo tamaño que A.
isinf(A)	Chequea si hay valores *Inf* en A, devolviendo una matriz de unos y ceros del mismo tamaño que A.
isfinite(A)	Chequea si los valores de A son finitos.
isempty(A)	Chequea si un vector o matriz está vacío o tiene tamaño nulo.
ischar()	Chequea si una variable es una cadena de caracteres (*string*).
isstr()	chequea si una variable es una cadena de caracteres (*string*)
isglobal()	Chequea si una variable es global.
Issparse()	Chequea si una matriz es dispersa (*sparse*, es decir, con un gran número de elementos cero).
expm(A)	Función exponencial matricial por defecto $= B = e^A = I + A + \frac{A^2}{2!} + \frac{A^3}{3!} + ...$
expm1(A)	Función exponencial matricial en M- fichero
expm2(A)	Función exponencial matricial vía series de Taylos
expm3(A)	Función exponencial matricial vía autovalores
logm(A) = B	Función logarítmica matricial, de manera que A = e^B
sqrtm(A)=B	Función raíz cuadrada matricial, de manera que B*B = A
max(x)	Máximo de los números los elementos de x
min(x)	Mínimo de los números los elementos de x
funm(A, 'función')	Aplica la función a la matriz

A continuación se presentan algunos ejemplos de uso de estas funciones en combinación con otras vistas previamente. Se define un vector x con un *NaN*, que se elimina en la forma:

```
» x=[1   2   3   4   0/0   6] ↵
Warning: Divide by zero
x =
       1   2   3   4   NaN   6
» i=find(isnan(x)) ↵
i =
       5
» x=x(find(~isnan(x))) ↵
x =
       1   2   3   4   6
```

Otras posibles formas de eliminarlo serían las siguientes:

```
» x=x(~isnan(x)) ↵
» x(isnan(x))=[ ] ↵
```

La siguiente sentencia elimina las filas de una matriz que contienen algún ***NaN***:

```
» A(any(isnan(A)'), :)=[ ] ↵
```

Cálculo del rango, normas y condición numérica:

Existen varias formas de realizar estos cálculos, con distintos niveles de esfuerzo de cálculo y de precisión en el resultado.

El rango se calcula implícitamente (sin que el usuario lo pida) al ejecutar las funciones ***rref(A)***, ***orth(A)***, ***null(A)*** y ***pinv(A)***. Con ***rref(A)*** el rango se calcula como el número de filas diferentes de cero; con ***orth(A)*** y ***null(A)*** –basadas ambas en la descomposición QR– el rango es el número de columnas del resultado (o **n** menos el número de columnas del resultado). Con ***pinv(A)*** se utiliza la descomposición de valor singular, que es el método más fiable y más caro en tiempo de *cpu*. La función ***rank(A)*** está basada en ***pinv(A)***.

Normas de matrices:

norm(A)	Norma sub-2, es decir, máximo valor singular de **A**, ***max(svd(A))***.
norm(A,2)	Lo mismo que ***norm(A)***
norm(A,1)	Norma sub-1 de **A**, máxima suma de valores absolutos por columnas, es decir: ***max(sum(abs((A))))***
norm(A,inf)	Norma sub-∞ de **A**, máxima suma de valores absolutos por filas, es decir: ***max(sum(abs((A'))))***

Normas de vectores:

norm(x,p) norma sub-p, es decir ***sum(abs(x)^p)^(1/p)***.
norm(x) norma euclídea; equivale al módulo o ***norm(x,2)***.
norm(x,inf) norma sub-∞ , es decir ***max(abs(x))***.
norm(x,1) norma sub-1, es decir ***sum(abs(x))***.

NÚMEROS COMPLEJOS

En muchos cálculos matriciales los datos y/o los resultados no son reales sino ***complejos***. MATLAB trabaja sin ninguna dificultad con números complejos. Para ver como se representan por defecto los números complejos, ejecútense los siguientes comandos:

```
» clear i  j ↵  % se borra posibles valores de i y j
» i^2↵
ans=
        -1
» j^2↵
ans=
        -1
» a=sqrt(-4) ↵
a =
```

```
        0 + 2.0000i
» 3 + 4j↵
ans =
        3.0000 + 4.0000i
```

En la entrada de datos de MATLAB se pueden utilizar de manera similar la **i** y la **j** para representar el *número imaginario unidad* (en la salida, sin embargo, puede verse que siempre aparece la **i**). Si la **i** o la **j** no están definidas como variables, puede intercalarse el signo (*).

Esto no es posible en el caso de que sí estén definidas, porque entonces se utiliza el valor de la variable. En general, cuando se está trabajando con números complejos, conviene no utilizar la **i** como variable ordinaria, pues puede dar lugar a errores y confusiones. Por ejemplo,

Obsérvense los siguientes resultados:

```
» i=3↵
i =
        3
» 2+4i↵
ans =
        2.0000 + 4.0000i
» 2+3*i↵
ans =
        11
» 2+3*j↵
ans =
        2.0000 + 3.0000j
```

Cuando **i** y **j** son variables utilizadas para otras finalidades, como *unidad imaginaria* puede utilizarse también la función ***sqrt(-1)***, o una variable a la que se haya asignado el resultado de esta función.

Todas las operaciones matemáticas incluyen la aritmética compleja en el sentido usual

```
» 1+i +5-6i↵                % suma de dos números complejos
ans =
        6.0000 - 5.0000i

» (5+3i)*(5-3i), (1+2i)/(3-4i) ↵
ans =
        13
ans =
        -0.20
```

```
» conj(3e-3+2e-4i) ↵        % conjugado
ans =
        0.0030 - 0.0002i
» abs(3+4i), angle(2i) ↵    % modulo y argumento
ans =
        5
ans =
        1.5708
```

Definir un número complejo

```
z1=3+4*i;
z1_bar=conj(z1);
z2=4+3*i;
z1+z2;
z1*z2;
z1/z2;
z1^2;
```

El signo * no es necesario con la i....

```
z1=3+4i;
```

Si se usa la i como una constante, entonces,

```
i=2;a=3+4*i;
b=3+4i;
```

Dibujar un número complejo.

```
a=real(z1);
b=imag(z1);
plot(a,b,'*');xlabel('Real');ylabel('Imaginario')
title('z1=3+4i')
grid
```

Interpretación geométrica (un número complejo se puede representar como un vector bidimensional).

La asignación de *valores complejos* a vectores y matrices desde teclado puede hacerse de las dos formas que se muestran en el ejemplo siguiente (conviene hacer antes ***clear i***, para que **i** no esté definida como variable.

```
» A = [1+2i 2+3i; -1+i 2-3i] ↵
A =
        1.0000 + 2.0000i 2.0000 + 3.0000i
        -1.0000 + 1.0000i 2.0000 - 3.0000i
» A = [1 2; -1 2] + [2 3; 1 -3]*i↵
A =
        1.0000 + 2.0000i 2.0000 + 3.0000i
```

```
    -1.0000 + 1.0000i 2.0000 - 3.0000i
```

Puede verse que es posible definir las partes reales e imaginarias por separado. En este caso sí es necesario utilizar el operador (*), según se muestra en el ejemplo anterior.

MATLAB dispone también de la función ***complex***, que crea un número complejo a partir de dos argumentos que representan la parte real e imaginaria, como en el ejemplo siguiente:

```
» complex(1,2) ↵
ans =
    1.0000 + 2.0000i
```

Es importante advertir que el *operador de matriz transpuesta* ('), aplicado a matrices complejas, produce la matriz conjugada y transpuesta. Existe una función que permite hallar simplemente la matriz conjugada (***conj()***) y el operador punto y apóstrofo que calcula simplemente la matriz transpuesta (**.'**).

REPRESENTACIONES TRIGONOMETRICAS Y EXPONENCIALES

```
mag=sqrt(z1*z1_bar);
argumento=atan(b/a);
```

La magnitud y la fase se pueden encontrar con:

```
mag=abs(z1);
argumento=angle(z1);
```

Teniendo magnitud y fase, se representa el numero z1 en forma, trigonométrica...

```
z1=mag*(cos(argumento)+i*sin(argumento))
```

Usando la formula de Euler:

```
z1=mag*exp(i*argumento)
```

Funciones de variables complejas,

```
sin(z1);
cos(z1);
sin(z1)^2+cos(z1)^2;
```

Arreglo de números complejos

```
Z=[0,z1,z1+z2];
plot(Z)
text(real(z1),imag(z1),'z1')
text(real(z1+z2),imag(z1+z2),'z2')
xlabel('real')
ylabel('imag')
title('Figura2. vector Z')
grid
sumZ=z1+z2;
plot(real(Z),imag(Z),real(sumZ),imag(sumZ))
Z';   No es la transpuesta del arreglo, sino la transpuesta conjugada compleja...

Z.';
```