Universidad Nacional de La Plata
Facultad de Ingeniería
Departamento de Electrotecnia
Cátedra de Control Moderno

Introducción a MATLAB

Fernando D. Bianchi

Año 2001

1. Introducción

Se puede decir que MATLAB es un lenguaje de programación de alto nivel que permite hacer cálculos, visualizar resultados y desarrollar algoritmos utilizando notación matemática. El nombre de MATLAB proviene de *Matrix Laboratory*. Fue creado originalmente como una interfaz para librerías de rutinas de Fortran como *EISPACK* y *LINPACK*, las cuales son considerados como el estado de las artes para resolver problemas de algebra matricial.

Los datos básicos son arreglos, como vectores y matrices, no necesitándose declaración de variables ni solicitar memoria. Esta es la principal característica de MATLAB es un lenguaje de programación, que permite manipular vectores o matrices como simples variable. Hoy día, no se limita solo a resolver problemas numéricos sino que ofrece un gran cantidad de herramientas que permiten vincularse con otros programas, hacer adquisición de datos, control en tiempo real, hacer procesamiento simbólico y mucho más.

MATLAB presenta al usuario un entorno de trabajo, llamado *Command Window* en cual se ingresan los comandos y se obtienen los resultados. En forma similar a otros lenguajes de programación, como C, posee pocas palabra claves, la mayoría de las instrucciones son llamado a rutinas, denominadas aquí *funciones*. Estas funciones se encuentran reunidas en conjuntos o librerías denominadas *toolboxes* de acuerdo a las operaciones que realizan. Algunos de estos *toolboxes* son incluidos dentro de la versión estándar y otros se adquieren por separado, dentro de los más comunes se tiene: *Control Systems*, *Symbolic*, *Signals Processing*, etc.

Para tener un buen manejo de MATLAB se requiere saber tres puntos claves:

- ▷ como se introducen y manipulan las matrices
- ▷ como se utilizan en general las funciones
- ▷ como obtener información de cada función en particular.

Este apunte se centra básicamente en estos tres puntos. Comenzamos dando las nociones básicas del manejo del *Command Window* para luego entrar de lleno con las matrices. Posteriormente, continuaremos con la descripción de algunas de las funciones para luego pasar a los gráficos. En cada sección se presentan varios ejemplos, siendo MATLAB un entorno interactivo se recomienda al lector reproduzca estos ejemplos y también experimente algunas variantes.

2. Nociones generales

Lo primero que se le presenta al usuario al iniciar MATLAB es el denominado *Command Window*, el cual cumple la función de interprete entre el usuario y el núcleo de procesamiento. Esta ventana presenta en el margen izquierdo un prompt » donde deberemos escribir las instrucciones. Cada línea será ejecutada luego que presionemos la tecla ENTER. En casi todos los casos, excepto los comandos gráficos, los resultados aparecerán en el mismo *Command Window*.

Las teclas de los cursores ↑ y ↓ sirve para poner en el prompt de MATLAB las instrucciones ingresadas con anterioridad (similar al doskey del DOS). Si se escribe parte de un comando y luego se presiona alguna de estas teclas aparece en el prompt la primer instrucción que comience con esas letras. Las teclas ← y → se utilizan para desplazarse por la línea de comandos.

2.1. Expresiones y variables

Comencemos por ver la forma general de las expresiones, la cual resultan bastante similar a la notación matemática. Por ejemplo, para una operación de suma utilizaremos,

```
» 2 + 3
ans =
     5
```

También podemos utilizar variables para almacenar valores y realizar operaciones posteriores. Para asignar el valor `3` a una variable de nombre `a` usaremos,

```
» a = 3
a =
     3
```

Esta variable es un sector de memoria al cual le hemos dado el nombre `a` y hemos almacenado el valor `3`. Este sector de memoria es parte del denominado *MATLAB Workspace*, en el cual se almacenan todas las variables que definamos. A diferencia de un lenguaje de programación no es necesario declarar las variables ni gestionar memoria, bastará con asignar valores a las variables para que MATLAB asigne la memoria necesaria en función de las dimensiones y del tipo de variable. En caso que la variable ya esté definida MATLAB sobrescribirá el valor anterior.

Nota: En versiones anteriores a la 5.0 todas las variables se almacenaban en un tipo de dato correspondiente al **double** del lenguaje C, es decir, un número de punto flotante con 16 decimales significativos en un rango entre $\pm 1{,}7^{-308}$ y $\pm 1{,}7^{308}$. En la versión 5.0 y superiores se han agregado otros tipos de datos, pero por defecto cuando se asigna un valor a una variable este se almacena en una del tipo double. Veremos que esto se modifica cuando trabajemos con cadena de caracteres.

Cualquier variable a la cual le hemos asignado un valor estará disponible para ser utilizada mientras que no la borremos explícitamente. Para ver el valor de la variable `a`, que habíamos definido antes, bastará con que escribamos su nombre en *Command Window*

```
» a
a =
     3
```

También podemos utilizar las variables para realizar operaciones matemáticas, por ejemplo

```
» b = 5
b =
     5

» c = a + b
c =
     8
```

donde hemos asignado el valor 5 a la variable `b`, luego hemos realizado una operación de suma entre esta y la variable `a` y el resultado lo hemos asignado a la variable `c`. Si el resultado de la operación anterior no se lo asignamos a la variable `c`, MATLAB asigna el resultado a una variable llamada `ans` (de *answer*, respuesta),

```
» a + b
ans =
     8
```

Esta variable puede ser utilizada para otra operación, pero no es recomendable pues su valor será modificado cada vez que no asignemos el resultado a una variable. Por ejemplo si hacemos

```
» c = a + ans
c =
    11
```

y luego realizamos otra operación,

```
» a + 4
ans =
     7
```

al realizar nuevamente,

```
» c = a + ans
c =
    10
```

El resultado es diferente pues `ans` cambió de valor entre ambas operaciones.

Respecto a los nombres de las variables, se deben seguir las siguientes reglas:

- ▷ Hay diferencia entre mayúsculas y minúsculas, p. ej. `A` es diferente de `a`.
- ▷ No debe haber signos de puntuación ni espacios en blanco, p. ej. `a.b`, `a-b` no son nombre válidos.
- ▷ El nombre puede tener hasta un máximo de 19 caracteres, p. ej. `es_un_nombre_valido_pero`, los últimos 5 no son tenidos en cuenta.
- ▷ Los nombre siempre deben comenzar con un letra, p. ej. `5x2` no es válido pero `x2` sí.
- ▷ MATLAB cuenta con ciertas variables especiales, las cuales se listan en la Tabla 1. Ya que se espera que estas variables tengan ciertos valores, es recomendable no utilizar estos nombres para evitar confusiones.

Variable	**Descripción**
`ans`	Almacena el último resultado en caso de no haberse asignado a otra variable.
`i` y `j`	$\sqrt{-1}$
`eps`	Mínima representación.
`pi`	El número π.
`realmin`	Mínimo número real positivo utilizable.
`realmax`	Máximo número real positivo utilizable.
`inf`	Indica infinito, $1/0$.
`NaN`	Corresponde a Not_a_Number, $0/0$.

Tabla 1: Variables especiales de MATLAB

Las variables pueden eliminarse del *MATLAB Workspace* utilizando el comando **clean**, por ejemplo

```
» clean a b
```

De esta manera, MATLAB libera la memoria que había asignado a las variables `a` y `b`. Este comando debe utilizarse con cuidado, pues si no hay una lista de variables a continuación de `clean` eliminaremos todas las variables disponibles en el *Workspace*, sin ninguna advertencia previa.

2.2. Números complejos

En MATLAB los valores que se asigna a las variables son en general complejos. La expresión de un número complejo es igual a la notación matemática, por ejemplo para asignar del número $1 + 2j$ a la variable `a` haremos:

```
» a = 1 + 2j
a =
   1.0000 + 2.0000i
```

Como vemos la parte imaginaria va seguida de la letra `i` o `j`. En cambio, si la parte imaginaria es el resultado de una operación no podemos usar la expresión anterior. Por ejemplo si hacemos,

```
» a = 1 + (4*5+2)i
??? a = 1 + (4*5+2)i
                   |
Missing operator, comma, or semi-colon.
```

MATLAB emite un mensaje de error pues espera el signo `*`. En estos casos deberemos multiplicar el resultado de la operación por $\sqrt{-1}$. Para ello utilizaremos las variables especiales `i` o `j`,

```
» a = 1 + (4*5+2)*i
a =
   1.0000 +22.0000i
```

o la función **sqrt** que calcula la raíz cuadrada,

```
» a = 1 + (4*5+2)*sqrt(-1)
a =
   1.0000 +22.0000i
```

Tanto en la parte real como en la imaginaria puede utilizarse la notación científica,

```
» a = 2e1 + 2e5i
a =
  2.0000e+001 +2.0000e+005i
```

Aclaremos que en la expresión `2e5i` MATLAB no lo considera como $2e^{5i}$, sino $2 \cdot 10^5 i$. Para que lo tome como un exponente imaginario deberemos usar la función **exp**,

```
» a = 2e1 + 2*exp(5i)
a =
  20.5673 - 1.9178i
```

2.3. Funciones

Como hemos mencionado MATLAB dispone de pocas palabras claves, la mayor parte de las operaciones se realizan a través funciones. La forma general del llamado a una función es la siguiente:

```
» [var1,var2,...] = funcion(arg1,arg2,...);
```

Esta tiene argumentos de salida (`var1`, `var2`, etc.) los cuales se encierran entre corchetes y argumentos de entrada (`arg1`, `arg2`, etc.) los cuales se encierran entre paréntesis. La función tomará las variables de entrada, hará los cálculos y devolverá los resultados en las variables de salida.

La cantidad de argumentos ya sea de entrada como de salida depende de la función en particular. No es necesario que en el llamado estén presente todos los argumentos de salida, por ejemplo la función **cart2pol**, la cual convierte de coordenadas cartesianas a polares, tiene dos argumentos de salida y dos de entrada. Una llamada a esta función puede incluir los dos argumentos de salida:

```
» [theta,rho] = cart2pol(3,4)
theta =
    0.9273

rho =
     5
```

o solo uno:

```
» theta = cart2pol(3,4)
theta =
    0.9273
```

En este último caso, sólo tendremos el primero de los dos argumentos devueltos por la función.

Por su parte, pese a que la cantidad de argumentos de entrada puede variar en una función, siempre deben estar los mínimos necesarios. Siguiendo con el ejemplo anterior si solo llamamos a la función con un solo argumento, MATLAB dará error:

```
» theta = cart2pol(3)
??? Input argument 'y' is undefined.

Error in ==> C:\MATLABR11\toolbox\matlab\specfun\cart2pol.m
On line 20  ==> th = atan2(y,x);
```

pues es necesario las dos componentes, x e y, para calcular ρ y θ. Por tanto, cada vez que utilicemos una función deberemos consultar la documentación (o la ayuda en línea) de MATLAB para ver los argumentos necesarios.

Por último, mencionemos que las funciones pueden ser utilizadas dentro de operaciones matemáticas,

```
» c = b + sqrt(a)
c =
    6.7321
```

y también ser argumento de entrada de otras funciones.

2.4. Signos de puntuación

Terminamos esta sección con los signos de puntuación utilizados en MATLAB. Podemos incluir varias instrucciones en una misma línea separándolas por "," o ";", por ejemplo

```
» sqrt(a),b
ans =
    1.7321

b =
     5
```

Si en cambio utilizamos ";" MATLAB no presentará el resultado en el *Command Window*, por ejemplo modificando la línea de instrucciones anterior,

```
» sqrt(a),b;
ans =
    1.7321
```

solo aparece el resultado de `sqrt(a)` pues `b` está seguido de ";".

Una serie de instrucciones puede ser continuada en otra línea utilizando "...", por ejemplo

```
» c = sqrt(3)/...
2
c =
    0.8660
```

Los "..." no debe cortar el nombre de una variable o de una función, observemos que si en el ejemplo anterior hubiésemos hecho

```
» c = sqr...
t(3)/2
??? c = sqrt
           |
Missing operator, comma, or semi-colon.
```

MATLAB emitirá un mensaje de error .

3. Matrices

Los datos básicos en MATLAB son los arreglos, nos concentraremos en las matrices y vectores (llamaremos vectores a las matrices que tengan una de sus dimensiones igual a uno), cuyos elementos son en general números complejos. Tanto la parte real como la imaginaria se almacenan en lo que sería el tipo double de C. Primeramente veremos como definir matrices y luego como se puede acceder a cada uno de sus elementos.

3.1. Definición de matrices

MATLAB permite varias formas de definir matrices, comenzaremos por la más simple en la cual se ingresa cada uno de los elementos. Así para definir una matriz `A` de 3×3 haremos

```
» A = [1,2,3;4 5 6
2+i,2 8i]
A =
   1        2    3
   4        5    6
   2 + 1i   2    0 + 8i
```

Aquí podemos ver las cuatro reglas para construir una matriz ingresando cada uno de los elementos,

- ▷ Los elementos dentro de una fila se separan por espacios en blanco o una coma.
- ▷ Las filas se separan por un punto y coma o por un ENTER.
- ▷ Los límites de la matriz se indican con corchetes.
- ▷ Los elementos de la matriz se deben ingresar por fila y todas deben tener la misma cantidad de elementos

MATLAB emitirá un mensaje de error en caso que algunas de estas reglas no se verifiquen, por ejemplo:

```
» A = [1,2;4 5 6
2+i,2 (8i)]
???  All rows in the bracketed expression must have the same
number of columns.
```

Como la fila uno contiene solo dos columnas mientras que las restantes filas tienen tres columnas, MATLAB presenta un mensaje de error indicando que todas filas deben tener la misma cantidad de columnas.

Otra posibilidad es definir matrices utilizando algunas de las funciones que dispone MATLAB. Por ejemplo, podemos utilizar la función **eye** para obtener la matriz identidad de 3×3,

```
» eye(3)
ans =
     1     0     0
     0     1     0
     0     0     1
```

la función **rand** para matrices cuyos elementos son números aleatorios entre 0 y 1 con una distribución uniforme,

```
» rand(3,2)
ans =
    0.9501    0.4860
    0.2311    0.8913
    0.6068    0.7621
```

la función **ones** para construir matrices cuyos elementos son todos unos,

```
» ones(1,4)
ans =
     1     1     1     1
```

etc. Como podemos observar en los ejemplos anteriores, cuando solo se ingresa un argumento la matriz resultante es cuadrada. En caso de desear una matriz rectangular se debe incluir la cantidad de filas y de columnas. En la Tabla 2 se da un listado de algunas de las funciones que generan matrices.

Función	**Descripción**
`zeros(m,n)`	Matriz de ceros de $m \times n$
`ones(m,n)`	Matriz de unos de $m \times n$
`eye(m)`	Matriz identidad de $m \times m$
`rand(m,n)`	Matriz de $m \times n$ con elementos aleatorios entre 0 y 1, con distribución uniforme
`randn(m,n)`	Matriz de $m \times n$ con elementos aleatorios entre 0 y 1, con distribución normal
`diag(n)`	Matriz diagonal, cuyos elementos de la diagonal son especificados por el vector `n`
`[]`	Matriz vacía

Tabla 2: Algunas funciones para generar matrices

Una tercer alternativa, para los casos de definición de vectores de grandes dimensiones, es utilizar la notación ":". Por ejemplo para definir un vector cuyos elementos estén comprendidos en el intervalo $[1, 25]$ con un espaciado entre ellos de cinco unidades utilizaremos:

```
» x = 1:5:25
x =
     1     6    11    16    21
```

La instrucción **x = inicio:incremento:final** construye un vector cuyo primer elemento es **inicio**, el segundo es **inicio** más **incremento**, el tercero es el segundo más **incremento**, y así siguiendo hasta superar el **final** (este último elemento no se incluye en el vector). El incremento puede ser cualquier número real inclusive valores negativos (si el incremento es 1 se puede omitir):

```
» x = 20:-2:10
x =
    20    18    16    14    12    10
```

También es posible utilizar **linspace** la cual cumple una función similar a la notación ":", por ejemplo:

```
» x = linspace(1,25,5)
x =
     1     7    13    19    25
```

construye un vector de 5 elementos cuyo primer elemento es 1 y el último es 25. Podemos observar que el resultado difiere del obtenido con la notación ":", en el primer caso se especifica el incremento pero no la cantidad de elementos y en el segundo al contrario se especifica la cantidad pero no el incremento. MATLAB tiene también una función similar a **linspace** para generar vectores con espaciado logarítmico,

```
» x = logspace(1,2,5)
x =
   10.0000   17.7828   31.6228   56.2341  100.0000
```

donde 1 y 2 son los exponentes del primer y último elemento respectivamente.

En la Tabla 3 se resumen las reglas para generar vectores.

Sintaxis	**Descripción**
`x = [1 2 3 4]`	Crea un vector con los elementos especificados.
`x = inicio:incremento:final`	Crea un vector que comienza en **inicio**, aumentando por **incremento** y terminando en o antes de **final**.
`x = inicio:final`	Crea un vector que comienza en **inicio** y termina en o antes de **final**, con un incremento igual a uno.
`x = linspace(inicio,final,cant)`	Crea un vector que comienza en **inicio** y termina en **final** que tiene **cant** elementos.
`x = logspace(inicio,final,cant)`	Crea un vector que comienza en $\mathbf{10^{inicio}}$ y termina en $\mathbf{10^{final}}$ que tiene **cant** elementos.

Tabla 3: Resumen de reglas para construir vectores

Una cuarta posibilidad es definir matrices a partir de otras, así dada la matriz:

```
» A = [1 5;6 7]
A =
     1     5
     6     7
```

podemos generar la matriz B de la siguiente manera

```
» B = [A,zeros(2);zeros(2),diag([1 2])]
B =
     1     5     0     0
     6     7     0     0
     0     0     1     0
     0     0     0     2
```

Aquí hemos construido la matriz B utilizando la matriz A y las funciones **zeros** y **diag**. Esta última función genera una matriz diagonal, cuyos elementos de la diagonal son especificados por el vector [1 2]. Observemos la utilización de los símbolos "," y ";" para indicar como se deben concatenar las matrices. A,zeros(2) y zeros(2),diag([1 2]) forma matrices de 2×4, las cuales luego son unidas por ";" para formar una matriz 4×4. Por supuesto debemos cumplir las reglas de dimensiones que hemos indicado antes, si hubiesemos hecho

```
» B = [A,zeros(2,1);zeros(2),diag([1 2])]
???  All rows in the bracketed expression must have
the same number of columns.
```

MATLAB indicará un error pues tratamos de concatenar una matriz de 2×3 con una de 2×4, una arriba de la otra.

3.2. Direccionamiento

Ya sabemos definir matrices y ahora nos resta aprender como acceder a un elemento de la matriz o una submatriz. Esto resulta bastante intuitivo pues, nuevamente, las expresiones son similares a la notación matemática. Así dada una matriz `A` de 4×6,

```
» A
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
```

para acceder al elemento correspondiente a la segunda fila cuarta columna usaremos la siguiente expresión:

```
» A(2,4)
ans =
    21
```

Es decir, para acceder al elemento A_{mn}, haremos `A(m,n)`. Pero MATLAB permite cosa aún más complejas, es posible reemplazar los escalares `m` y `n` por vectores y así obtener submatrices o construir matrices a partir de cierta combinación de filas y columnas de otras matrices. Por ejemplo, si deseamos acceder a la submatriz comprendida entre la fila 2 y 3, y la columna 2 y 4 de la matriz `A` haremos

```
» A(2:3,2:4)
ans =
    32     7    21
     9     2    22
```

En este caso hemos utilizado la notación ":", vista anteriormente, para construir dos vectores. Observemos que la siguiente secuencia de comandos da el mismo resultado:

```
» m = 2:3;
» n = 2:4;
» A(m,n)
ans =
    32     7    21
     9     2    22
```

No es necesario que los vectores `m` y `n` estén formado por números consecutivos:

```
» m = [1 3];
» n = [2 4 6];
» A(m,n)
ans =
     1    26    24
     9    22    20
```

donde hemos seleccionado las filas 1 y 3 y las columnas 2, 4 y 6. Tampoco es necesario que los elementos de los vectores `m` y `n` estén ordenados en orden ascendente:

```
» m = 4:-1:2;
» n = 6:-2:2;
» A(m,n)
ans =
    15    17    28
    20    22     9
    25    21    32
```

Es importante destacar que **los elementos del vector `m` deben estar comprendido entre 1 y la cantidad de filas de la matriz `A` y los elementos del vector `n` entre 1 y la cantidad de columnas**. En caso contrario MATLAB emitirá el correspondiente mensaje de error. Observemos también que si alguno de los vectores `m` y `n` contiene números no enteros MATLAB los redondea y emite un mensaje de advertencia,

```
» m = 1:0.5:2
m =
    1.0000    1.5000    2.0000

» n = 1:3;
» A(m,n)
Warning: Subscript indices must be integer values.
ans =

    35     1     6
     3    32     7
     3    32     7
```

En el ejemplo anterior MATLAB repitió la fila 2 pues redondeó el segundo elemento de `m`.

En caso que deseemos obtener todas las filas o columnas se puede utilizar simplemente ":",

```
» A(:,[2 6])
ans =
     1    24
    32    25
     9    20
    28    15
```

donde obtenemos las columnas 2 y 6 de la matriz `A`. El lector puede probar otras combinaciones que se le ocurra y observar los mensajes de MATLAB en caso de no ser correctas.

Continuemos viendo dos funciones que permiten determinar las dimensiones de un matriz, una de ellas es **size** la cual da la cantidad de filas y columnas:

```
» size(A)
ans =
     4     6
```

la otra es **length** la cual devuelve la longitud de un vector (en caso de matrices da la dimensión de mayor valor).

```
» length(n)
ans =
     3
```

Estas funciones resultan útiles, por ejemplo, para acceder a la submatriz formada por todas las filas de las últimas tres columnas de `A`:

```
» [ii,jj] = size(A);
» A(:,4:jj)
ans =
    26    19    24
    21    23    25
    22    27    20
    17    10    15
```

Nota: Para versiones 5.0 y superiores es posible abreviar esto utilizando la palabra **end**, el ejemplo anterior resulta:

```
» A(:,4:end)
ans =
    26    19    24
    21    23    25
    22    27    20
    17    10    15
```

Hasta ahora hemos accedido a los elementos de una matriz indicando la fila y la columna, pero MATLAB también permite utilizar un solo índice. En el caso de un vector coincide con el número de elemento, por ejemplo

```
» n = 1:3
n =
     1     2     3

» n(2)
ans =
     2
```

Para las matrices este índice corresponde al número de elemento contándolos por columnas, así el elemento $\mathtt{A}_{24}$ se direcciona con el índice `14`:

```
» A(14)
ans =
    21

» A(2,4)
ans =
    21
```

Este índice se puede calcular con la siguiente expresión:

$$ind = (n-1) \cdot n_f + m$$

donde n_f es la cantidad de filas de la matriz y m y n corresponde a la fila y columnas, respectivamente, del elemento que se desea acceder. En el ejemplo anterior

$$(4-1) \cdot 4 + 2 = 14.$$

Utilizando esta notación, la función **find** permite obtener los índices de los elementos que verifican alguna condición lógica. Por ejemplo para seleccionar los elementos de `A` mayores de 30,

```
» A(find(A>30))
ans =
    35
    31
    32
    33
```

3.3. Manipulación de matrices

No solo podemos "leer" los elementos de una matriz sino también modificarlos, así por ejemplo para hacer igual a cero los elementos comprendidos entre las filas 2 y 3 y las columnas 2 y 4 asignaremos una matriz de ceros a este sector de la matriz `A`,

```
» A(2:3,2:4) = zeros(2,3)
A =
    35     1     6    26    19    24
     3     0     0     0    23    25
    31     0     0     0    27    20
     8    28    33    17    10    15
```

Es importante que en una asignación como la anterior que las dimensiones del arreglo del lado izquierdo sean iguales al del lado derecho.

Esta notación combinada con la matriz vacía sirve para eliminar filas o columnas de una matriz, por ejemplo para eliminar la columna 4 de la matriz `A` haremos:

```
» A(:,4) = []
A =
    35     1     6    19    24
     3    32     7    23    25
    31     9     2    27    20
     8    28    33    10    15
```

Por último, notemos que si deseamos acceder al elemento A_{51} MATLAB emitirá el correspondiente mensaje de error, pues estos índices exceden las dimensiones de `A`,

```
» A(5,1)
???  Index exceeds matrix dimensions.
```

Por el contrario, si queremos hacer que este elemento tome el valor 1:

```
» A(5,1) = 1
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
     1     0     0     0     0     0
```

MATLAB se encarga automáticamente de expandir las dimensiones de la matriz.

4. Operaciones con matrices

Las operaciones con matrices pueden ser divididas en tres grupos, operaciones **escalar-matriz**, **matriz-matriz** y **elemento-elemento**. Pasemos a ver cada una de estos grupos y como se realizan estas operaciones en MATLAB.

4.1. Operaciones escalar-matriz

Las operaciones **escalar-matriz** son las más sencillas, simplemente cada uno de los elementos de la matriz son afectado por el escalar. Por ejemplo, dado la matriz `A`

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

si la multiplicamos por el escalar 4 resulta:

```
» A*4
ans =
     4     8    12
    16    20    24
```

es decir, cada elemento de `A` ha sido multiplicado por 4. En forma similar podemos restar 8 a esta matriz,

```
» A - 8
ans =
    -7    -6    -5
    -4    -3    -2
```

Lo mismo sucederá con las operaciones de suma (`+`) y división (ya sea por derecha, `/`, o por izquierda, `\`). Observemos que MATLAB automáticamente detecta que las operaciones anteriores son escalar-matriz, sino fuese así hubiera discrepancia en las dimensiones de los operandos.

4.2. Operaciones matriz-matriz

Dentro las operaciones matriz-matriz se encuentran la suma, la resta, la multiplicación, la división y la potenciación matricial. Recordemos de algebra matricial que estas no son tan simples como las anteriores, que tienen restricciones respecto de las dimensiones y orden de los operandos y que las dimensiones del resultado pueden diferir de las de los operandos.

Comencemos por las más simples, la suma y la resta. Dada las matrices `A` y `B`, las cuales deben tener dimensiones coincidentes,

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

la suma de ambas se expresa de la siguiente manera:

```
» C = A + B
C =
     6     8
    10    12
```

La matriz resultante `C` tiene las mismas dimensiones que `A` y `B`.

Sigamos con una operación un poco más compleja, la multiplicación. Tomemos otras dos matrices `A`, de dimensiones $m_a \times n_a$, y `B`, de dimensiones $m_b \times n_b$. Para poder realizar el producto matricial AB el número de columnas de `A` y el número de filas de `B` deben ser coincidentes ($n_a = m_b$). La matriz resultante tendrá la misma cantidad de filas que `A` y la misma cantidad de columnas que `B`, es decir, sus dimensiones serán $m_a \times n_b$. Por ejemplo dadas las siguientes matrices

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

```
» B = [3 4 5;6 7 8]
B =
     3     4     5
     6     7     8
```

el producto `A` por `B` resultará:

```
» C = A*B
C =
    15    18    21
    33    40    47
    51    62    73
```

Por otro lado, para el producto `BA` necesitamos que $n_b = m_a$ y el resultado será de $m_b \times n_a$. Siguiendo con el ejemplo anterior tenemos

```
» C = B*A
C =
    40    52
    67    88
```

Como podemos ver el orden de los operandos produce distintos resultados. Inclusive no siempre podremos hacer ambos productos, por ejemplo se cambiamos la matriz `B`

```
» B = [3;6]
B =
     3
     6
```

el producto `A*B` resulta

```
» A*B
ans =
    15
    33
    51
```

pero cambio el producto `B*A`

```
» B*A
??? Error using ==> *
Inner matrix dimensions must agree.
```

causa que MATLAB emita un mensaje de error debido a que n_b no coincide con m_a.

Pasemos ahora a la potenciación, consideremos la siguiente matriz

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

para elevarla a la potencia 3 la expresión es la siguiente:

```
» A^3
ans =
    37    54
    81   118
```

Esta operación corresponde al producto matricial `A*A*A`, lo cual implica que la matriz `A` debe ser cuadrada. Pues para hacer `A*A`, n_a debe ser igual a m_a. El exponente también puede ser negativo, por ejemplo

```
» A^(-3)
ans =
  -14.7500    6.7500
   10.1250   -4.6250
```

En este caso, primero de invierte la matriz `A` y luego se hacen los sucesivos productos. Debemos tener cuenta que como hay una inversión de la matriz `A` esta debe ser no singular, es decir, el determinante debe ser no nulo.

Concluimos esta sección con la operación de división. Esto es, dada dos matrices

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

» B = [3;6]
B =
     3
     6
```

la división por izquierda de `B` por `A` (`A\B`) corresponde al producto $A^{-1}B$. Hay dos expresiones para realizar esta operación en MATLAB, una utiliza la función **inv** que calcula la matriz inversa,

```
» inv(A)*B
ans =
         0
    1.5000
```

y la otra los operadores `\` y `/`,

```
» A\B
ans =
    0.0000
    1.5000
```

Ambas expresiones realizan la misma operación pero no utilizan el mismo algoritmo. La primera utiliza las fórmulas de inversión de matrices y la segunda la eliminación gaussiana. En general, la segunda opción realizará la operación de división en forma más rápida y con menos errores de cálculo.

La división matricial nos permite resolver sistemas de ecuaciones lineales, por ejemplo dado el siguiente sistema

$$\begin{aligned} 1x_1 + 2x_2 &= 5 \\ 3x_1 + 4x_2 &= 9 \end{aligned}$$

Para encontrar la solución primero definimos dos matrices `A` y `b`,

```
» A = [1 2;3 4];
» b = [5;9];
```

y luego utilizando el operador `\` obtenemos las soluciones x_1 y x_2 como los elementos del siguiente vector,

```
» x=A\b
x =
  -1
   3
```

Siempre que el determinante de `A` sea no nulo. El lector interesado puede probar cual es la respuesta cuando se tiene más incógnitas que ecuaciones, caso subdeterminado y cuando se tiene más ecuaciones que incógnitas, caso sobredeterminado. Ver `help mldivide` o la ayuda en `helpdesk`.

4.3. Operaciones elemento-elemento

Dada las siguientes matrices

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

» B = [5 6;8 9]
B =
     5     6
     8     9
```

la suma de estas dos matrices,

```
» C = A + B
C =
     6     8
    10    12
```

se realiza haciendo una operación escalar entre los elemento A_{ij} y B_{ij} y el resultado se asigna al elemento C_{ij} de la matriz resultante. Esto constituye una operación elemento-elemento.

Como hemos visto en la sección anterior la multiplicación, división y potenciación no operan en forma elemento-elemento, como la suma y la resta. Por esto MATLAB distingue a este tipo de operaciones de las matriciales anteponiendo un punto a los símbolos utilizados en las operaciones de multiplicación, división y potenciación. Por ejemplo, para hacer el producto elemento-elemento de `A` por `B` usaremos,

```
» A.*B
ans =
     5    12
    24    36
```

Puesto que en realidad son operaciones entre escalares, el orden de los operandos no altera el resultado

```
» B.*A
ans =
     5    12
    24    36
```

La operación de división trabaja en forma similar, pero la potenciación tiene algunas variantes. Por ejemplo si deseamos elevar al cuadrado cada elemento de `A` haremos

```
» A.^2
ans =
     1     4
     9    16
```

En cambio si queremos elevar el escalar 4 a la potencia de cada elemento de `B` haremos,

```
» C = 4.^B
C =
        1024         4096
       16384        65536
```

es decir el elemento $\texttt{C}_{\texttt{ij}}$ es $4^{\texttt{B}_{\texttt{ij}}}$.

Quizás la aplicación más común de este tipo de operaciones sea con vectores. Por ejemplo para evaluar la función $y = 1/(x^2 + 1)$ en el intervalo $[-1, 1]$ primero construimos el vector `x`

```
» x = linspace(-1,1,100);
```

luego utilizando las operaciones de elemento-elemento obtenemos los valores de y correspondiente a cada punto del vector `x`

```
» y = 1./(x.^2+1);
```

En la Tabla 4 se resumen las operaciones matriciales y elemento-elemento:

Operación	**Operador matricial**	**Operador elem.-elem.**
Suma	`+`	`+`
Resta	`-`	`-`
Producto	`*`	`.*`
División izq.	`\`	`.\`
División der.	`/`	`./`
Potencia	`^`	`.^`

Tabla 4: Operadores aritméticos

4.4. Orden de precedencia

El orden de precedencia de las operaciones son las habituales. Las expresiones se evalúan de izquierda a derecha. La potenciación es evaluada primeramente seguido del producto y la división y por último, la suma y la resta. También podemos utilizar paréntesis, en cuyo caso se evalúa primero el par de paréntesis más interno.

Es conveniente utilizar paréntesis en toda expresión en la cual pueda haber posibilidad de confusión, por ejemplo la siguiente expresión puede llegar a dar error en algunas versiones de MATLAB

```
» A^-3
```

en este caso es mejor usar

```
» A^(-3)
```

También, la siguiente expresión puede tener un resultado inesperado:

```
» 4/2*3
ans =
     6
```

El resultado no corresponde a $\frac{4}{2\cdot 3}$. Debido a que MATLAB evalúa las expresiones de izquierda a derecha, la expresión anterior corresponde a $\frac{4}{2} \cdot 3$. El problema se resuelve usando:

```
» 4/(2*3)
ans =
    0.6667
```

Operador	Descripción
`>`	Mayor que
`>=`	Mayor o igual que
`<`	Menor que
`<=`	Menor o igual que
`==`	Igual a
`~=`	Distinto a

Tabla 5: Operadores relacionales

Operador	Descripción
`&`	AND
`|`	OR
`~`	NOT

Tabla 6: Operadores lógicos

o

```
» 4/2/3
ans =
    0.6667
```

5. Operadores relacionales y lógicos

Además de las operaciones aritméticas, MATLAB dispones de los operadores relacionales y logicos comunes. En la Tabla 5 se listan los operadores relaciones y en la Tabla 6 los operadores lógicos.

Como entradas a las expresiones relacionales y lógicas, MATLAB considera a cualquier valor no nulo como verdadero y al cero como falso. El valor devuelto por estas expresiones será `1` para verdadero y `0` para falso. Por ejemplo,

```
» 4 > 2
ans =
     1

» 1 > 2
ans =
     0
```

Todos estos operadores pueden utilizarse con vectores y matrices. Así dados los siguientes vectores, **de las mismas dimensiones**:

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

» y = 2*x-4
y =
    -2     0     2     4     6
```

podemos ver cuales de los elementos de `x` son mayores que los de `y` con la siguiente expresión:

```
» z = x > y
z =
     1     1     1     0     0
```

Como podemos ver la respuesta es un vector de las mismas dimensiones que `x` e `y`, cuyo elemento z_i es 1 si $x_i > y_i$ o 0 si $x_i \leq y_i$.

Las matrices o vectores que resultan de las operaciones relacionales y lógicas son arreglos numéricos y pueden ser utilizados en operaciones matemáticas y en funciones como cualquier otro.

Podemos utilizar estos operadores para evitar la división por cero, por ejemplo dado el siguiente vector:

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

si queremos calcular $y = sen(x)/x$ MATLAB presentará un mensaje de advertencia debido a que el cuarto elemento de `x` es nulo y asigna `NaN` a y_4 pues corresponde a $0/0$,

```
» y = sin(x)./x
Warning: Divide by zero.
y =
    0.0470    0.4546    0.8415       NaN    0.8415    0.4546    0.0470
```

Si en cambio hacemos:

```
» x = x+(x==0)*eps;
» y = sin(x)./x
y =
    0.0470    0.4546    0.8415    1.0000    0.8415    0.4546    0.0470
```

Donde `eps` es una de las variables especiales de MATLAB que hemos visto, la cual tiene el valor de la mínima representación.

6. Descripción de algunas funciones

MATLAB cuenta con una gran cantidad de funciones, posiblemente encontraremos una para cualquier procesamiento numérico que deseemos hacer. No es la intención de este trabajo hacer una descripción detallada de cada grupo de funciones, solo nos dedicaremos hacer algunos comentarios de aquellas que implementan las funciones matemáticas más comunes y las de algebra matricial.

6.1. Funciones matemáticas

Dentro de este grupo de funciones se encuentra el seno, el coseno, la tangente, los logaritmos, etc. Estas funciones operan en forma similar a las operaciones elemento-elemento. Por ejemplo, para obtener el $\log(x)$ debemos construir primeramente el vector `x` (el cual puede ser complejo),

```
» x = linspace(0,10,100);
```

y luego utilizando la función **log10** calculamos los valores del logaritmo de base 10 para cada elemento del vector `x`

```
» y = log10(x);
Warning: Log of zero.
> In C:\MATLABR11\toolbox\matlab\elfun\log10.m at line 13
```

Podemos ver como MATLAB advierte que hemos incluido el cero en el vector `x`, si vemos el valor del primer elemento de `y` (el cual corresponde al primer elemento de `x`)

```
» y(1)
ans =
  -Inf
```

MATLAB le ha asignado $-\infty$.

Las demás funciones de este grupo se utilizan de la misma manera, en la Tabla 7 hemos indicado el nombre de las funciones más usadas, el lector puede referirse a la ayuda en línea para conocer más detalles de cada una de estas.

Función	**Descripción**
`sin`, `sinh`	Seno trigonométrico, hiperbólico
`cos`, `cosh`	Coseno trigonométrico, hiperbólico
`tan`, `tanh`	Tangente trigonométrico, hiperbólico
`asin`, `asinh`	Arco seno trigonométrico, hiperbólico
`acos`, `acosh`	Arco coseno trigonométrico, hiperbólico
`atan`, `atanh`	Arco tangente trigonométrico, hiperbólico
`log`	Logaritmo natural
`log2`	Logaritmo de base 2
`log10`	Logaritmo de base 10
`exp`	Exponencial
`real`	Parte real de un número complejo
`imag`	Parte real de un número complejo
`abs`	Modulo de un número complejo o valor absoluto
`angle`	Fase de un número complejo

Tabla 7: Funciones matemáticas

6.2. Funciones matriciales

La cantidad de funciones dedicadas a problemas de álgebra matricial es muy importante, recordemos que MATLAB es un programa orientado para trabajar con matrices. Aquí pasaremos revista sólo a algunas de la funciones matriciales, para mayores detalles el lector puede referirse a la ayuda en línea.

Comencemos con la matriz traspuesta, para ello dada:

```
» A = [1 2i 3;4 (2+i) 6;1 7 8i]
A =
   1    0 + 2i      3
   4    2 + 1i      6
   1    7           0 + 8i
```

la conjuga transpuesta la obtenemos utilizando el operador `'`,

```
» A'
ans =
   1        4        1
   0 - 2i   2 - 1i   7
   3        6        0 - 8i
```

En cambio, la transpuesta la obtendremos agregando un punto al símbolo `'`,

```
» A.'
ans =
   1        4        1
   0 + 2i   2 + 1i   7
   3        6        0 + 8i
```

Claro está que si `A` es real ambas operaciones concuerdan.

Continuemos con el cálculo del determinante, en este caso la función será **det**. Así dada la siguiente matriz cuadrada

```
» A = [1 2 3;4 11 6;1 7 8]
A =
     1     2     3
     4    11     6
     1     7     8
```

el determinante será

```
» det(A)
ans =
    45
```

El lector puede utilizar lo aprendido hasta aquí para verificar algunas de las siguientes propiedades de los determinantes:

- ▷ $det(A) = det(A^T)$.
- ▷ $det(AB) = det(A)det(B) = det(B)det(A)$.
- ▷ $kdet(A) = k^n det(A)$, donde k es un escalar y A es de $n \times n$.
- ▷ El intercambio dos filas o columnas de A solo cambia el signo del determinante.
- ▷ El determinate de una matriz que tenga una fila o columna combinación lineal de la otra es nulo.

El rango de una matriz lo calcularemos con **rank**, recordemos que este es la máxima cantidad de columnas o filas linealmente independiente de la matriz. Así dada la matriz

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

el rango será

```
» rank(A)
ans =
     2
```

En este ejemplo, el rango es inferior a 3 pues la fila dos es combinación lineal de la uno. El lector puede verificar las siguientes propiedades

- ▷ si A es de $m \times n$ entonces $rango(A) \leq min(m, n)$.
- ▷ si A es de $n \times n$ y el rango es menor de n entonces $det(A) = 0$.
- ▷ $rango(AB) \leq min(rango(A), rango(B))$.

Otra característica útil de las matrices son los autovalores, estos son las raíces del determinante de $\lambda I - A$. En MATLAB los podemos calcular con la función **eig**, por ejemplo dada la matriz

```
» A = [0 1 0;0 0 1;-6 -11 -6]
A =
     0     1     0
     0     0     1
    -6   -11    -6
```

los autovalores serán

```
» V = eig(A)
V =
  -1.0000
  -2.0000
  -3.0000
```

Asociados a los autovalores están los autovectores, así si utilizamos nuevamente la función **eig** pero con dos argumentos de salida

```
» [U,V] = eig(A)
U =
   -0.5774    0.2182   -0.1048
    0.5774   -0.4364    0.3145
   -0.5774    0.8729   -0.9435

V =
   -1.0000         0         0
         0   -2.0000         0
         0         0   -3.0000
```

obtendremos dos matrices, en las columnas de `U` tendremos los autovectores y en los elementos de la diagonal de `V` los autovalores. Los autovectores tiene la propiedad de diagonalizar la matriz `A`, verifiquémoslo

```
» U\A*U
ans =
   -1.0000    0.0000    0.0000
    0.0000   -2.0000    0.0000
    0.0000   -0.0000   -3.0000
```

También si P es una matriz no singular, los autovalores de $P^{-1}AP$ y los de PAP^{-1} coinciden con los de A. Verifique usando la función **eig**.

El determinante de $\lambda I - A$ resulta ser un polinomio en λ, el cual se lo denomina polinomio característico de A. En MATLAB este polinomio puede ser obtenido con la función **poly**. Así dada la misma matriz `A` de los ejemplos anteriores,

```
» pc = poly(A)
pc =
    1.0000    6.0000   11.0000    6.0000
```

MATLAB utiliza para representar polinomios vectores filas, donde el primer elemento corresponde a la potencia más alta, es decir el `pc` representa el polinomio

$$\lambda^3 + 6\lambda^2 + 11\lambda + 6$$

Las raíces de este polinomio deben coincidir con los autovalores. Esto lo podemos verificar con la función **roots**, la cual calcula las raíces de un polinomio,

```
» roots(pc)
```

```
ans =
   -3.0000
   -2.0000
   -1.0000
```

En la Tabla 8 se indican las funciones matriciales vistas y algunas más, las cuales son simples de usar. El lector puede hacer `help función`, para obtener más información.

Función	Descripción
`rank(A)`	Rango de la matriz A
`det(A)`	Determinante de la matriz A
`[U,V]=eig(A)`	Calcula los autovalores y autovectores de A
`norm(A)`	Norma de la matriz A
`trace(A)`	Traza de la matriz A, suma de los elementos de la diagonal
`inv(A)`	Calcula la inversa de la matriz A
`poly(A)`	Polinomio característico de la matriz A
`roots(p)`	Raíces de un polinomio

Tabla 8: Algunas funciones para álgebra matricial

7. Texto

En MATLAB las cadenas de caracteres son consideradas como un vector, solo que los números de cada elemento son convertidos según el código ASCII. Para que MATLAB interprete de esta manera un vector cuyos elementos son números enteros entre 0 y 255, es necesario ingresar la cadena de caracteres entre comillas simple. Por ejemplo,

```
» a = 'esta es una prueba de texto'
a =
esta es una prueba de texto
```

Podemos ver los números enteros correspondientes a cada carácter por medio de la función **abs**,

```
» abs(a)
ans =
  Columns 1 through 12
   101   115   116    97    32   101   115    32   117   110    97    32

  Columns 13 through 24
   112   114   117   101    98    97    32   100   101    32   116   101

  Columns 25 through 27
   120   116   111
```

Nota: En versiones anteriores a la 5.0 las cadenas se almacenaban en arreglos tipos double como cualquier otro vector, es decir se empleaba 8 bytes para almacenar cada carácter. En las versiones posteriores se modificó y ahora cada carácter se almacena en un byte lo cual constituye un mejor aprovechamiento de la memoria

---

Ya que las cadenas son vectores podemos usar todas las técnicas de direccionamiento que hemos visto en las secciones anteriores. Así para obtener parte de la cadena de caracteres anterior usaremos la notación de ":",

```
» b = a(1:12)
b =
esta es una

» c = a(22:end)
c =
 texto
```

De la misma manera, utilizando lo que aprendimos para construir matrices a partir de otras podemos concatenar cadenas. Por ejemplo usando `b` y `c`,

```
» d = [b 'modificación del' c]
d =
esta es una modificación del texto
```

obtenemos la nueva cadena de caracteres `d`.

Las cadenas no necesariamente son vectores también pueden ser matrices. En este caso el texto tendrá varias líneas, por ejemplo:

```
» A = ['Este es un texto';'de dos líneas   ']
A =
Este es un texto
de dos líneas
```

Al igual que cuando trabajamos con matrices, tanto la fila uno como la dos deben tener la misma cantidad de columnas. Es por esto que hemos rellenado con espacios en blancos la cadena de caracteres de la segunda fila.

MATLAB cuenta con una serie de funciones para manipular cadenas, los nombre suelen coincidir con los del lenguaje C. Aquí nos detendremos solamente en la función **sprintf**, la cual permite generar una cadena transformando datos numéricos de acuerdo a un formato especificado. Por ejemplo,

```
» x = 10.5;
» cad = sprintf('El valor de x es %3.2f',x)
cad =
El valor de x es 10.50
```

El texto `%3.2f` indica el formato que se debe utilizarse para transformar el número almacenado en `x` en una cadena. Los códigos de formatos puede encontrarse haciendo `help sprintf`.

8. Command Window

En las secciones anteriores hemos visto como definir matrices, acceder y manipular sus elementos, hacer operaciones matemáticas y utilizar funciones. En esta sección pasaremos revista a una serie de funciones que sirven para modificar algunos aspectos del *Command Window* y obtener información respecto de las variables disponibles.

8.1. Formato de los resultados

El formato numérico de los resultados presentados en el *Command Window* puede ser modificado utilizando la función **format**. Es importante que tengamos en cuenta que esto modifica solo la presentación de los números pero no altera la precisión de la cálculos (MATLAB sigue utilizando los datos tipo double para todas las cuentas).

El formato por defecto utilizado es tipo **short**, el cual presenta el resultado como un número de punto fijo con cuatro decimales. Es decir, si tomamos el número π en este formato MATLAB lo presenta de la siguiente manera

```
» pi
ans =
    3.1416
```

Si cambiamos el formato a **long** se presenta con catorce decimales

```
» format long
» pi
ans =
   3.14159265358979
```

en cambio en formato **short e** los resultados se muestran en punto flotante:

```
» format short e
» pi
ans =
  3.1416e+000
```

Hay por lo menos nueve opciones para la presentación de los resultados, el lector puede recurrir a la ayuda en línea de la función **format** para obtener todas las variantes (Ver `help format`).

8.2. Información sobre las variables

Existen dos funciones para obtener un listado de las variables disponibles en el *MATLAB Workspace*. Una es **who** la cual da un listado de los nombres de las variables,

```
» who
Your variables are:
A         B         x         z
```

y la otra es **whos** la cual nos da un listado de las variables más detallado, que incluye las dimensiones, la cantidad de memoria utilizada y el tipo de arreglo que es cada variable,

```
» whos
  Name      Size         Bytes  Class
  A         2x2             32  double array
  B         1x2             16  double array
  x         1x17           136  double array
  z         1x2             32  double array (complex)

Grand total is 25 elements using 216 bytes
```

8.3. Guardando las variables

MATLAB nos permite almacenar las variables que tenemos en *MATLAB Workspace* para luego poder utilizarlas en otra sesión de trabajo. Las variables son almacenadas en archivos con extensión "mat", los cuales pueden ser de formato binario o de texto. Con este fin MATLAB cuenta con las funciones **save**, para guardar las variables y **load**, para volver a cargarlas en el *Workspace*.

Suponiendo que tenemos las variables `A`, `B`, `x` y `z` en el *MATLAB Workspace*, para guardarlas en el archivo "misdatos.mat" utilizaremos la siguiente expresión,

```
» save misdatos
» save misdatos -ascii
```

En la primera almacenaremos las variables en un archivo de extensión mat con formato binario y en la segunda con formato texto. Si solo deseamos guardar algunas de las variables bastará con listarlas a continuación del nombre del archivo,

```
» save misdatos A B z
```

De esta manera solo las variables `A`, `B` y `z` se almacenarán. Si luego necesitamos agregar la variable `x` al archivo anterior podemos usar,

```
» save misdatos x -append
```

El modificador `-append` le indica a MATLAB que agregue la variable sin borrar las anteriores. Es importante que recordemos que si no utilizamos `-append` el archivo será sobrescrito sin ninguna advertencia, perdiendo la información de las otras variables que habíamos guardado previamente.

En otra sesión de trabajo podemos utilizar el comando opuesto, **load**, para cargar en el *Workspace* las variables `A` y `B`,

```
» load misdatos A B
```

o quizás todas todas las variables utilizando simplemente `load`.

MATLAB también cuenta con una serie de funciones para realizar operaciones con archivos y directorios similares a los comandos del DOS. Por ejemplo para cambiar de directorio está **cd**, para un listado de archivos **dir**, etc.

8.4. Ayuda

A lo largo de este trabajo hemos visto que MATLAB cuenta con una cantidad muy grande de funciones, cada uno de ellas con diversas variantes. Como es imposible que recordemos como utilizar cada función, MATLAB cuenta con unos comandos que nos provee de ayuda en línea para cada función.

Si conocemos el nombre de la función que realiza la operación que deseamos, podemos acceder a la ayuda en línea con siguiente expresión,

```
» help find

 FIND   Find indices of nonzero elements.
    I = FIND(X) returns the indices of the vector X that are
    non-zero. For example, I = FIND(A>100), returns the indices
    of A where A is greater than 100. See RELOP.

    [I,J] = FIND(X) returns the row and column indices of
    the nonzero entries in the matrix X.  This is often used
    with sparse matrices.

    [I,J,V] = FIND(X) also returns a vector containing the
    nonzero entries in X.  Note that find(X) and find(X~=0)
    will produce the same I and J, but the latter will produce
    a V with all 1's.

    See also SPARSE, IND2SUB.
```

De esta manera, obtenemos en el *Command Window* la información sobre la función **find** con todas las variantes que podemos utilizar. También presenta una última línea que nos indica el nombre de otras funciones que realizan operaciones relacionadas.

Si en cambio no sabemos cual es el nombre de la función que realiza la operación que necesitamos se puede utilizar **lookfor**. Por ejemplo, si deseamos calcular el rango de la matriz `A`, puesto que rank es rango en inglés hacemos

```
» lookfor rank
CHOLUPDATE Rank 1 update to Cholesky factorization.
QRUPDATE Rank 1 update to QR factorization.
RANK   Matrix rank.
SPRANK Structural rank. CRANKDEM Demonstration - system
FRANKE Franke's bivariate test function.
RANK   Symbolic matrix rank. FRANK  Frank matrix.
```

Vemos que devuelve un listado de funciones las cuales hacen mención a la palabra "rank", entre ellas se encuentra `rank`. Luego podemos utilizar `help rank` para obtener mayores detalles.

Actualmente las versiones superiores a 5.0 cuenta además de ayuda en formato html, para acceder se debe utilizar `helpdesk`. También a través de esta interfaz se puede acceder a los manuales en formato pdf.

9. Gráficos

MATLAB cuenta con un gran diversidad de tipos de gráficos, los cuales resultan muy simples de usar comparado a un entorno de programación. Nos concentraremos en los gráficos de dos dimensiones, el lector interesado puede recurrir a `help graph3d` para un listado de funciones relacionadas a gráficos en tres dimensiones y a `help specgraph` para los gráficos especiales.

Supongamos que deseamos hacer un gráfico de la arcotangente hiperbólica, $tgh^{-1}(kx)$ en el intervalo $[-20, 20]$ para distintos valores de k. Lo primero que tenemos que hacer es construir un vector para las abscisas (eje x) y luego utilizando la función **atanh** calculamos los valores del vector correspondiente a la ordenada (eje y),

```
» x = -20:0.1:20;
» y = atanh(x);
```

Una vez que contamos con los vectores correspondientes para los ejes x e y, podemos hacer uso de la función **plot** para obtener un gráfico en dos dimensiones con ejes en escala lineal,

```
» plot(x,real(y))
```

Al ejecutar esta instrucción MATLAB creará una nueva ventana la cual se denomina *Figure Window*. Dentro de esta ventana se crea un par de ejes y se une por rectas los puntos correspondientes a las coordenadas `(x(n),y(n))`. Los ejes son escalados en forma automática para que la curva pueda ser observada en forma completa. El resultado lo podemos ver en la Fig. 1. Siempre debemos tener presente que los vectores `x` e `y` deben ser ambos de la misma longitud.

Podemos completar el gráfico indicando a que variable corresponde cada eje agregándole etiquetas,

```
» xlabel('eje x');
» ylabel('eje y');
```

y un título,

```
» title('Este es un gráfico de la arcotangente hiperbólica')
```

Los textos no se limitan solamente a etiquetas en los ejes y un título, también es posible agregar texto en cualquier punto del gráfico utilizando las funciones **text** y **gtext**. Por ejemplo,

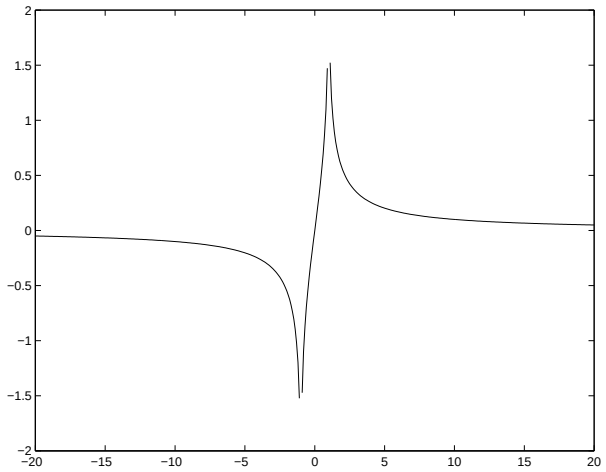

Figura 1: Gráfico 2D utilizando la función **plot**.

```
» text(5,1,'tgh^{-1}');
```

pone el mensaje `'tgh^{-1}'` en las coordenadas `(5,1)` del gráfico que antes habíamos hecho. Por otro lado, la función **gtext** representa una forma más interactiva de hacer lo mismo, usando

```
» gtext('tgh^{-1}');
```

el texto se ubicará donde hagamos click con el mouse. En la Fig. 2 tenemos la Fig. 1 con los textos que le hemos agregado.

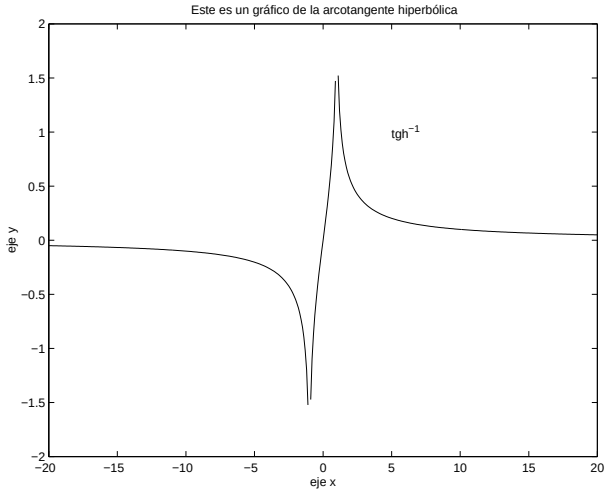

Figura 2: Fig. 1 con el agregado de texto.

La función **grid** nos permite agregar o quitar una grilla al gráfico, la expresión `grid on` la aplica y `grid off` la quita (en caso de usar solo `grid` alterna entre `on` y `off`).

Tenemos hasta ahora el gráfico de $tgh^{-1}(kx)$ para $k = 1$, calculemos los vectores que corresponden a $k = 0{,}5$ y $k = 0{,}75$,

```
» y1 = atan(0.75*x);
» y2 = atan(0.5*x);
```

La función **plot** abre una *Figure Window* solo cuando no hay ninguna abierta, en caso contrario dibujará en la *Figure Window* activa borrando cualquier gráfico existente (Se dice *Figure Window* activa a

última creada o la última sobre la cual se ha hecho click con el mouse). Para poder dibujar las curvas correspondientes a `y1` e `y2` tenemos dos opciones. Una de ellas es utilizando **hold**, esta función hace que **plot** dibuje sin borrar las curvas anteriores. Usando `hold on` activa esta opción y `hold off` lo desactiva. Por tanto para dibujar las tres curvas haremos,

```
» hold on;
» plot(x,real(y1))
» plot(x,real(y2))
```

La otra alternativa es utilizar la función **plot** con la siguiente variante,

```
» plot(x,y,x,y1,x,y2);
```

Con esta sola llamada de la función **plot** MATLAB dibujará las tres curvas, dándoles colores diferentes a cada una. En ambos casos el resultado será el mismo y puede verse en la Fig 3.

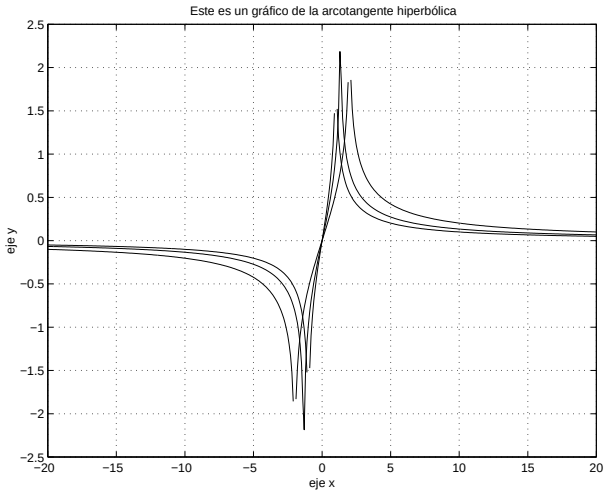

Figura 3: Fig. 2 con el agregado de las curvas correspondientes a $tgh^{-1}(0{,}5x)$ y $tgh^{-1}(0{,}75x)$.

En este caso, donde el vector correspondiente a las abscisas es el mismo para todos vectores de las ordenadas, la línea de instrucciones anterior se puede simplificar haciendo

```
» plot(x,[y',y1',y2']);
```

La función **plot** dibujará tres curvas correspondientes a cada columna del segundo argumento versus el vector `x`. Observemos que para formar la matriz del segundo argumento hemos traspuesto los vectores `y`, `y1` e `y2`.

Al tener varias curvas necesitamos saber cual corresponde a cada variable. Si bien utilizando el último método para dibujar varias curvas automáticamente asigna diferentes colores esto puede ser insuficiente, además tendremos que recordar el orden de colores asignados por MATLAB. La función **plot** tiene opciones para controlar el estilo de las líneas, por ejemplo la expresión

```
» plot(x,y,'k-');
```

dibuja una curva con línea continua de color negro. El estilo de línea se indica con la cadena de caracteres `'k-'`, la letra `k` es color negro y `-` es línea continua. En la Tabla 9 se da un listado de las letras que corresponde a cada color, en la Tabla 10 de los símbolos correspondientes a cada tipos de línea y en la Tabla 11 los tipos de marcas.

Siguiendo con nuestro ejemplo, dibujemos con línea continua la curva que corresponde a $k = 1$, con línea punteada la de $k = 0{,}75$ y con marcas `+` la de $k = 0{,}5$; todas de color negro.

Símbolo	Color
y	amarillo
m	magenta
c	cían
r	rojo
g	verde
b	azul
w	blanco
k	negro

Tabla 9: Opciones de colores de líneas

Símbolo	Estilo de línea
-	sólida
:	puntos
-.	trazos y puntos
--	trazos

Tabla 10: Opciones de estilos de líneas

```
» plot(x,y,'k-',x,y1,'k:',x,y2,'k+');
```

Luego podemos agregar una leyenda para indicar sobre el gráfico a que curva corresponde a cada línea,

```
» legend('tanh^{-1}(x)','tanh^{-1}(0.75*x)','tanh^{-1}(0.5*x)');
```

El orden de los textos le indican a MATLAB con cual línea debe hacer coincidir cada mensaje. Si deseamos eliminarla del gráfico bastará con usar `legend off`. El resultado podemos verlo en la Fig. 4.

Cada vez que se utiliza la función **plot** MATLAB automáticamente fija los límites de los ejes de manera de dibujar toda la curva. Estos límites pueden resultar inconvenientes en algunos casos, por esto MATLAB cuenta la función **axis** para modificar estos límites. Siguiendo con el ejemplo pongamos los límites del eje x en -10 y 10, usando la función **axis** la instrucción será

```
» axis([-10 10 -2.5 2.5]);
```

donde el argumento de esta función es un vector construido según $[x_{lim_{inf}}\ x_{lim_{sup}}\ y_{lim_{inf}}\ y_{lim_{sup}}]$. El resultado lo podemos ver en la Fig. 5.

Otra forma de modificar los límites del gráfico en forma más interactiva es usar la función **zoom**. Con `zoom on` al hacer click con el botón izquierdo del mouse sobre la *Figure Window* se amplia en un factor dos la zona centrada en la posición del mouse; por el contrario haciendo click con el botón derecho se reduce en un factor dos. También si se mantiene presionado el botón izquierdo del mouse y se arrastra aparece un cuadro indicando la zona que se ampliará cuando se suelte el botón del mouse. Utilizando `zoom xon` y `zoom yon` solo se amplia un solo eje. Por último, `zoom off` devuelve la función normal al mouse.

En la Tabla 12 hemos enumerado las distintas opciones de uso de la función **plot** a modo de resumen de lo expuesto hasta aquí. Además de la función **plot** MATLAB cuenta con las funciones **semilogx**, **semilogy** y **loglog**, las cuales se utilizan de la misma manera que **plot** pero hacen gráficos con uno o ambos ejes en escala logarítmica.

En los ejemplos anteriores hemos dibujado solo la parte real de $tgh^{-1}(kx)$, supongamos ahora que deseamos graficar tanto la parte real como la imaginaria pero no en los mismos ejes. Para esto tenemos dos opciones, una es dibujar en *Figure Window* separadas usando la siguiente secuencia de comandos,

```
» h1 = figure,
» plot(x,real(y),'k-',x,real(y1),'k:',x,real(y2),'k+');
```

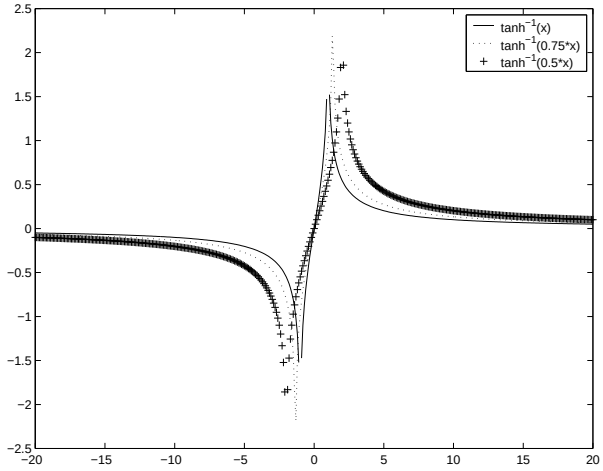

Figura 4: Fig. 3 con distintos tipos de línea y leyenda.

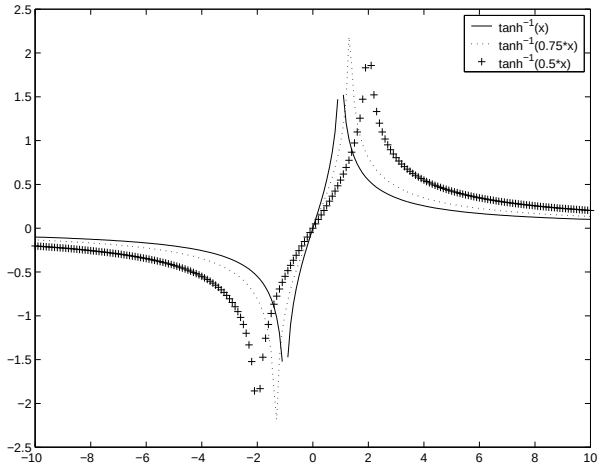

Figura 5: Fig. 4 con los límites del eje entre -10 y 10.

Símbolo	Marcas
o	círculo
x	x
+	Suma
*	Estrella
s	Cuadrado
d	Diamante
v	Triángulo (abajo)
^	Triángulo (arriba)
<	Triángulo (izquierdo)
>	Triángulo (derecho)
p	Pentágono
h	Hexágono

Tabla 11: Opciones de marcas de líneas

Expresión	Descripción
`plot(x2,y1,'op1',x2,y2,'op2',...)`	Dibuja una curva para cada grupo de vectores, con las opciones de líneas dada por la cadena `opi`.
`plot(x,[y1' y2' ..],'op')`	Dibuja una curva para cada columna de la matriz `[y1' y2'..]` todas con el mismo vector en las abscisas y con la opciones de líneas dada por la cadena `op` para todas las curvas igual
`plot(x,'op')`	Dibuja una curva de `x` versus el número de elemento, con las opciones de líneas dada por la cadena `op`.
`plot(x,'op')`, con `x` complejo	Dibuja una curva correspondiente a la parte imaginaria en función de la parte real de `x`, con las opciones de líneas dada por la cadena `op`.

Tabla 12: Distintas opciones para el uso de la función **plot**

```
» h2 = figure,
» plot(x,imag(y),'k-',x,imag(y1),'k:',x,imag(y2),'k+');
```

La función **figure** abre una nueva *Figure Window* y devuelve un número que se llama *Handle Graphic* el cual identifica en forma única la *Figure Window*. Este número nos sirve para poder controlar sobre cual ventana se dibuja, usando `figure(h1)` la ventana activa pasa a ser las correspondiente al *Handle Graphic* `h1`.

Otra alternativa para graficar en ejes separados es la función **subplot**. Esta función divide la *Figure Window* en forma de una matriz, por ejemplo la expresión `subplot(1,2,1)` crea un par de ejes en la mitad superior de *Figure Window*. Donde `1` indica la cantidad de filas de la división, `2` la cantidad de columnas y `1` indica que se refiere al elemento 1 de esta matriz en la que se dividido la *Figure Window*. Utilicemos esta función en nuestro ejemplo,

```
» h1 = subplot(2,1,1);
» plot(x,real(y),'k-',x,real(y1),'k:',x,real(y2),'k+');
» ylabel('y');xlabel('x');
» title('Parte real de thg^{-1}');
» h2 = subplot(2,2,3);
» plot(x,abs(y),'k-',x,abs(y1),'k:',x,abs(y2),'k+');
» ylabel('y');xlabel('x');
» title('Módulo de thg^{-1}');
» h3 = subplot(2,2,4);
» plot(x,angle(y),'k-',x,angle(y1),'k:',x,angle(y2),'k+');
» ylabel('y');xlabel('x');
```

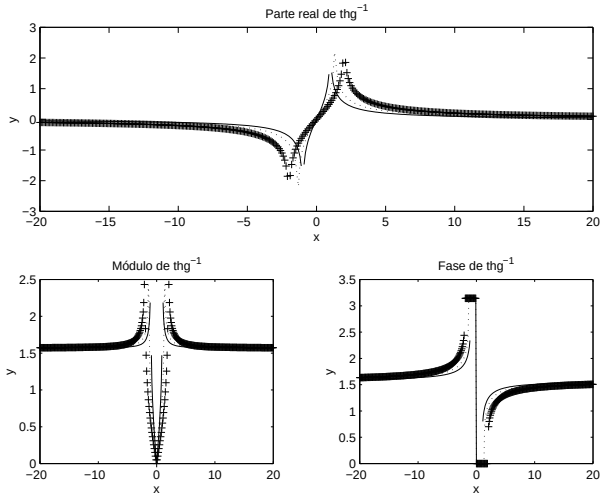

Figura 6: Ejemplo de uso de la función **subplot**.

Expresión	**Descripción**
`close`	Cierra la *Figure Window* activa, si se usa `close all` cierra todas las ventanas.
`clf`	Borra todos los pares de ejes de la *Figure Window* activa.
`gcf`	Devuelve el *Handle Graphic* de la ventana activa.
`gca`	Devuelve el *Handle Graphic* del par de ejes activo.
`print`	Permite exportar la *Figure Window* a un archivo de formato gráfico.

Tabla 13: Algunas funciones para manipular *Figure Window*

```
» title('Fase de thg^{-1}');
```

En la Fig. 6 se muestra el resultado de esta serie de instrucciones. Podemos ver que la función **subplot** también devuelve una *Handle Graphic* que identifica al par de ejes. También podemos ver como es posible combinar diferentes divisiones de la *Figure Window*.

MATLAB también dispone de una serie de funciones para manipular las *Figure Window*, las cuales se enumeran en la Tabla 13. Detallaremos solamente **print** pues las demás son muy simples. La función **print** en las primeras versiones de MATLAB era utilizada para imprimir los gráficos, actualmente se utiliza para exportar la *Figure Window* activa a un archivo de formato gráfico como tiff, postscript, window metafile, etc. Por ejemplo, para exportar el gráfico que hemos hecho en los ejemplos anteriores a un archivo con formato tiff haremos,

```
» print -dtiff mifigura.tif
```

El lector interesado puede recurrir a `help print` para conocer todas las opciones disponible de esta función. Por último, también se puede exportar utilizando la opciones de **copy & paste** que dispone todo programa en entorno Window.