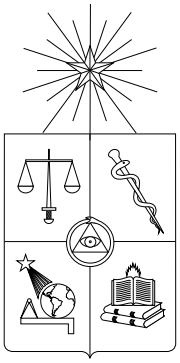

Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ingeniería Eléctrica

Introducción a Matlab y Simulink

Preparado por Diego Sepúlveda J.

Version 1.0, 6 de agosto de 2002

Introducción

MATLAB es un poderoso sistema el cual permite el tratamiento numérico de una gran cantidad de aplicaciones en ingeniería, tales como: procesamiento de señales, análisis estadísticos, interpolación de curvas, series de tiempo, simulación y control de sistemas, lógica difusa, redes neuronales, identificación de parámetros, etc.

El nombre MATLAB se debe a que en realidad consiste en un laboratorio de matrices (MATrix LABoratory), razón por la cual es excelente para el cálculo numérico de vectores y valores propios, descomposiciones de matrices y cualquier aplicación que involucre matrices.

Además de lo mencionado anteriormente, MATLAB también incluye un subprograma llamado SIMULINK, el cual permite la simulación de todo sistema dinámico lineal y casi cualquier sistema no lineal.

A pesar de la gran cantidad de aplicaciones mencionadas anteriormente, este apunte está enfocado hacia las aplicaciones orientadas hacia el control de sistemas.

Si el lector desea ahondar más sobre los aspectos generales de MATLAB entonces deberá referirse a [1]

Índice general

Capítulo 1

Lo básico

Dado que todas las aplicaciones de MATLAB se basan en el uso de matrices, lo primordial en este capítulo es mostrar cómo utilizarlas, posteriormente se verán los distintos tipos de variables y funciones que existen, para finalmente aprender el manejo de gráficos.

1.1. Matrices

1.1.1. Definición y asignación de variables

Para introducir una matriz en MATLAB sólo se debe introducir los números de la matriz entre paréntesis cuadrados ([ ]), las columnas se separan por espacios y las filas por punto y coma (;)[1]. Por ejemplo:

```
>> A=[3 4 5 ; 3 2 7]

A =

     3     4     5
     3     2     7
```

Como se puede ver en el ejemplo anterior las variables se asignan mediante un signo igual (=) de la misma manera que en lenguajes como JAVA o C.

[1]; también se utiliza para suprimir la visualización del resultado

1.1.2. Operando con matrices

Para transponer[2] matrices sólo hay que poner después de la matriz o de la variable un apóstrofe ('), siguiendo con el ejemplo anterior quedaría:

```
>> A'

ans =

     3     3
     4     2
     5     7
```

En la variable `ans` mostrada en el ejemplo anterior, MATLAB guarda el resultado de la última operación ejecutada.

Las operaciones aritméticas son igual que ne la mayoría de los lenguajes, así para sumar (o restar) sólo hay utilizar el signo + (o -), para multiplicar se utiliza el asterisco (*) y para dividir por la derecha (izquierda) se utiliza (/ (\)). [3] Por ejemplo:

```
>> B=[1 2 3; 4 5 6];
>> C=A+B

C =

     4     6     8
     7     7    13

>> D=C*A'

D =

    76    80
   114   126
```

Para potenciar una matriz se utiliza el símbolo (^), seguido del exponente que se desea. Si se desea invertir una matriz se puede hacer de dos maneras: elevando la matriz a -1 o utilizando la función `inv`:

[2] En el caso de que se utilicen números complejos se obtiene la conjugada transpuesta

[3] Obviamente para poder realizar estas operaciones es necesario que las dimensiones de las matrices sean consistentes.

```
>> A=[2 2 ; 0 1]

A =

     2     2
     0     1

>> inv(A)

ans =

    0.5000   -1.0000
         0    1.0000
```

En el cuadro 1.1 se muestra un resumen de los operadores matriciales.

Operación	*Símbolo*
Multiplicación	*
División por la derecha	/
División por la izquierda	\
Potenciación	^
Transposición conjugada	'

Cuadro 1.1: Operadores para álgebra matricial

1.1.3. Matrices especiales

Dado que existen matrices que son muy utilizadas en la práctica MATLAB incluye funciones específicas para crearlas:

- `ones` crea una matriz de unos
- `zeros` crea una matriz de ceros
- `eye` crea la matriz identidad

La utilización de las tres es muy similar, se introduce primero el número de filas y posteriormente el número de columnas; como muestra el siguiente ejemplo:

```
>> A=ones(3,2)

A =

     1     1
     1     1
     1     1

>> B=zeros(3,5)

B =

     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0

>> C=eye(3)

C =

     1     0     0
     0     1     0
     0     0     1

>> C=eye(3,4)

C =

     1     0     0     0
     0     1     0     0
     0     0     1     0
```

Nótese que si se desea la matriz identidad de orden n sólo hay que introducir n, pero si se desea una rectangular se utiliza igual que las funciones anteriores.

Otro tipo de matrices comúnmente utilizadas son los vectores, los cuales se pueden definir como cualquier matriz. La importancia radica en que muchas veces se utilizan para indexar alguna serie de elementos, para definirlos de esta manera se utiliza (:), de la siguiente manera:

```
número_inicial:paso:cota_superior
```

Por ejemplo:

```
>> 1:4:19

ans =

     1     5     9    13    17
```

1.1.4. Manipulación de matrices

Una aplicación de uso frecuente consiste en seleccionar algunas columnas, filas o simplemente elementos de alguna matriz; esto se logra con la utilización de paréntesis después del nombre del nombre de la variable y de dos puntos (:). A continuación se muestran algunos ejemplos para la matriz definida anteriormente:

```
>> A(1,1)

ans =

     2

>> A(:,1)

ans =

     2
     0

>> A(2,:)

ans =

     0     1

>> A(4)
```

```
ans =

     1
```

En el primer ejemplo se obtiene el elemento a_{11} de la matriz, en el segundo ejemplo la primera columna, el tercer ejemplo se obtiene la segunda fila, y para el último ejemplo se obtiene el cuarto elemento de la matriz. La numeración del último ejemplo es por columnas (así a_{11} es 1°, a_{21} es 2°, a_{12} es 3°, a_{22} es 4°).

Además de seleccionar elementos, muchas veces es útil eliminar algún(os) elemento(s) de la matriz[4], para lograr lo anterior se utilizan un par de paréntesis cuadrados. Así, por ejemplo, si se desea borrar la segunda columna de una matriz dada:

```
>> A=[1 2 6; 3 4 8; 5 7 9; 3 4 9];
>> A(:,2)=[ ]

A =

     1     6
     3     8
     5     9
     3     9
```

Es importante notar que tanto la selección como la eliminación de elementos de una matriz se puede realizar utilizando un vector como conjunto índices a utilizar. Por ejemplo:

```
>> A=0:1:10

A =

     0     1     2     3     4     5     6     7     8     9    10

>> indice=1:2:11

indice =
```

[4]Sin embargo, sólo se puede hacer si la estructura resultante sigue siendo una matriz

```
     1     3     5     7     9     11

>> B=A(indice)

B =

     0     2     4     6     8     10
```

obtiene los números que están en las posiciones impares del vector A.

Muchas veces es útil concatenar matrices, lo cual se puede utilizar utilizando los paréntesis cuadrados ([ ]). Por ejemplo para concatenar horizontalmente:

```
>> B=[1 2 3; 4 5 6];
>> C=[1 ; 2];
>> [B C]

ans =

     1     2     3     1
     4     5     6     2
```

y para concatenar verticalmente:

```
>> B=1:1:4;
>> C=4:-1:1;
>> [B ; C]

ans =

     1     2     3     4
     4     3     2     1
```

Obviamente las dimensiones de las matrices deben ser consistentes con la concatenación.

1.1.5. Arreglos

Los arreglos son matrices, pero poseen una aritmética distinta en cuanto a la multiplicación y división. Estas operaciones se ejecutan elemento a

elemento, y para que sean consistentes los arreglos deben ser de las mismas dimensiones.

Para diferenciar las operaciones matriciales de las operaciones de arreglos los operadores van precedidos por un punto (.), como se muestra en el cuadro 1.2.

Operación	*Símbolo*
Multiplicación	.*
División por la derecha	./
División por la izquierda	.\
Potenciación	.^
Transposición no conjugada	.'

Cuadro 1.2: Operadores para álgebra de arreglos

Por ejemplo:

```
>> A=[1 3 4; 4 2 6];
>> B=[3 4 8; 7 8 0];
>> A.\B

ans =

    3.0000    1.3333    2.0000
    1.7500    4.0000         0
```

corresponde a la división de arreglos por la izquierda de A por B, i.e. $\frac{b_{ij}}{a_{ij}} \quad \forall\, i, j$.

1.2. Variables y Funciones

1.2.1. Variables

Existen varios tipos de variables en MATLAB, las más comunes son:

double corresponden a las matrices y arreglos numéricos.

char son los arreglos de caracteres, se definen entre apóstrofes (''):

```
>> Z='hola'

Z =

hola
```

1.2.2. Funciones

Las funciones son, al igual que en la mayoría de los lenguajes, subrutinas que facilitan el trabajo, por ejemplo la función `mean` calcula el promedio o media de un set de datos:

```
>> A=1:1:4;
>> mean(A)

ans =

    2.5000
```

Lo importante con respecto a las funciones en MATLAB es que vienen algunas incluidas en el programa (built-in functions) y otras viene dentro de los distintos toolboxes que trae MATLAB (por ejemplo `mean`).

Generalmente las funciones vienen con alguna ayuda de su utilización, la cual se puede visualizar a través de la función `help`.

1.2.3. Algunas Variables y Funciones de utilidad

MATLAB trae muchas variables y funciones predefinidas, algunas de estas variables se muestran en el cuadro 1.3, mientras que algunas funciones más utilizadas aparecen en el cuadro 1.4.

Si se aplica alguna de las funciones matemáticas a alguna matriz se obtiene una matriz en la que los elementos han sido evaluados por la función. Por ejemplo:

```
>> X=0:0.1:1;
>> exp(-X)

ans =
```

Nombre	*Descripción*
`pi`	El número más famoso del mundo
`i`	Unidad imaginaria
`j`	Lo mismo que `i`, pero para los eléctricos
`Inf`	Infinito
`NaN`	No es un número
`eps`	Precisión relativa de punto flotante, 2^{-52}

Cuadro 1.3: Variables Predefinidas

Nombre	*Descripción*
`sin(X)`	Función seno de `X`
`cos(X)`	Función coseno de `X`
`tan(X)`	Función tangente de `X`
`exp(X)`	Función exponencial de `X`
`log(X)`	Función logaritmo natural de `X`
`plot(X,Y)`	Gráfica `Y` versus `X`
`clear(A)`	Borra la variable `A`
`det(A)`	Calcula el determinante de la matriz `A`
`eig(A)`	Calcula los valores y vectores propios de la matriz `A`
`poly(A)`	Calcula los coeficientes del polinomio característico de la matriz `A`
`roots(coef)`	Calcula los raíces del polinomio cuyos coeficientes vienen en `coef`
`sum(X)`	Suma los elementos del vector `X`
`length(X)`	Retorna el largo del vector `X`
`size(A)`	Retorna las dimensiones de la matriz `A`
`help funcion`	Entrega ayuda sobre la función `funcion`
`lookfor palabra`	Retorna las funciones en las que aparece el string `palabra`

Cuadro 1.4: Funciones Básicas más comunes

```
Columns 1 through 7

  1.0000    0.9048    0.8187    0.7408    0.6703    0.6065    0.5488
```

```
  Columns 8 through 11

    0.4966    0.4493    0.4066    0.3679
```

1.3. Gráficos

Para graficar en MATLAB fundamentalmente se utiliza la función `plot` mencionada anteriormente, sin embargo también es posible graficar en forma escalonada, utilizando sólo líneas verticales, utilzando números complejos o en 3D, entre muchas maneras de graficar. A continuación se detallan las más utilizadas y los comandos más útiles relacionados.

1.3.1. Figuras

Una figura es una ventana en la cual se desplegan los gráficos obtenidos mediante MATLAB. Esto presenta varias ventajas las cuales se mostrarán más adelante.

Aunque, generalmente, las figuras se generan por defecto al crear un gráfico, a veces es necesario pedir otra figura a MATLAB, para esto se utiliza el comando `figure`, el cual genera otra figura en la pantalla. El modo de utilizarlo es:

```
>> figure
```

Si se desea cerrar alguna figura se utiliza la función `close`, seguida del número de la figura. Si se quiere cerrar todas las figuras entonces se ejecuta:

```
>> close all
```

lo cual cierra todas las figuras existentes[5].

1.3.2. Gráficos en 2D

Como se menciona anteriormente se utiliza la función PLOT, tal como se muestra en el siguiente ejemplo:

```
>> t=0:0.1:5;
>> plot(t,exp(-t))
```

lo cual produce como resultado la fig. 1.1.

Para que `plot` funcione ambos vectores deben tener el mismo largo. Si `X` o `Y` es una matriz entonces el vector es graficado versus las filas o columnas de la matriz, dependiendo de con cual se alinee.

[5]De manera análoga `clear all` borra todas las variables

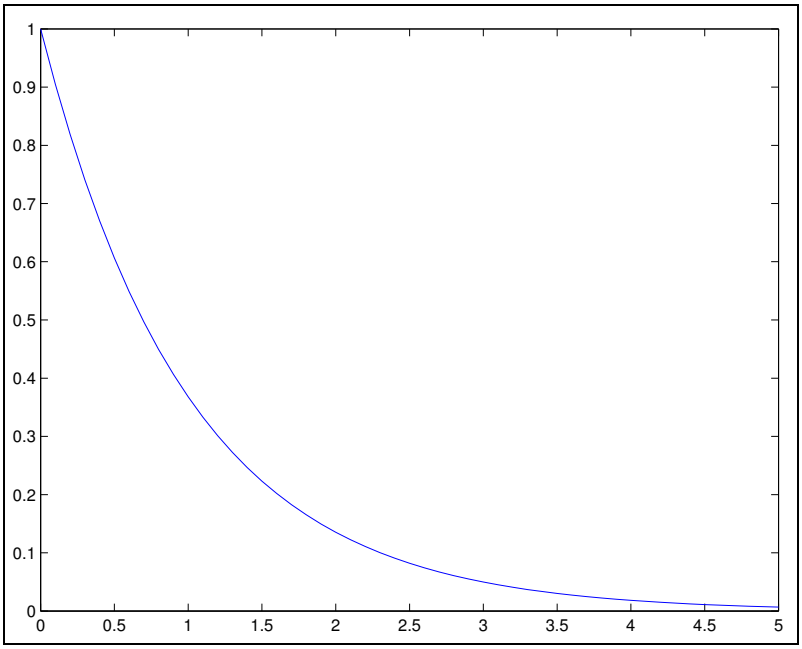

Figura 1.1: Ejemplo de `plot`

Además `plot` tiene más opciones las cuales se pueden ver en la ayuda de la función.

Una forma de graficar comúnmente usada es aquella en la cual se encuentran dos gráficas en la misma figura . Lo anterior se puede lograr de varias maneras, dos de ellas son:

```
>> plot(t,exp(-t),t,sin(t))
>> plot(t,[exp(-t); sin(t)])
```

las cuales producen el mismo resultado (fig. 1.2). Sin embargo, hay otra opción la cual consiste en utilizar la función `hold` que retiene el gráfico actual y agrega el gráfico deseado a la figura actual. Para el ejemplo anterior:

```
>> plot(t,exp(-t))
>> hold on
>> plot(t,sin(t))
>> hold off
```

cuyo resultado se muestra en la fig. 1.3. La utilización de `hold off` es para soltar la figura.

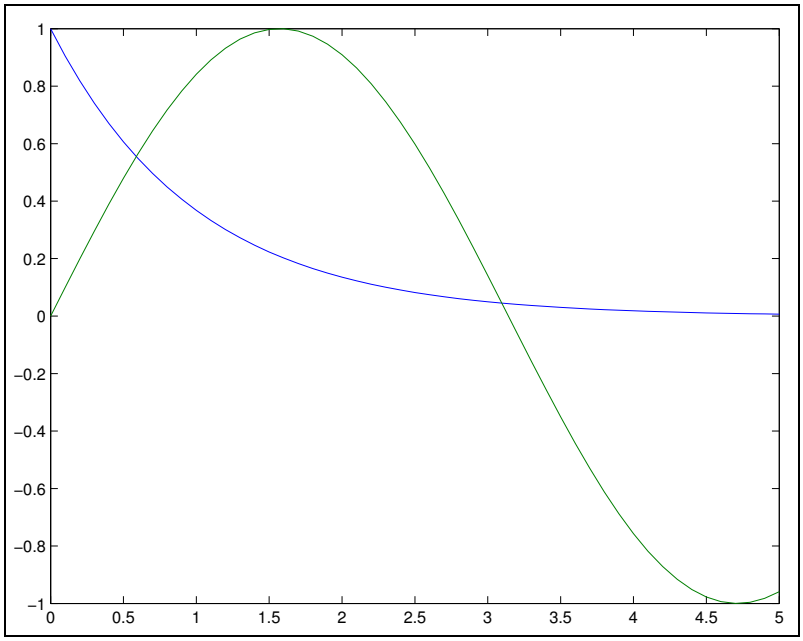

Figura 1.2: Dos gráficas en la misma figura

Para crear una leyenda de las gráficas se utiliza la función`legend(string1, string2, ...)`, donde los `string_i` son los textos de cada gráfico para la leyenda, así para el ejemplo anterior, la instrucción:

```
>> legend('exponencial', 'seno')
```

produce la fig. 1.4.

También es posible graficar en escalonada utilizando la función `stairs(X,Y)` de manera análoga al uso de la función `plot`, con la diferencia que los elementos del vector `X` deben ser equiespaciados.

Si se quiere graficar señales de tiempo discreto se puede utilizar la función `stem(X,Y)`, cuyo uso es análogo a las funciones anteriores.

Muchas veces es muy útil agrupar varios gráficos en una misma figura, lo cual se consigue fácilmente con la función `subplot(m,n,i)`, la cual divide la figura en una matriz de `m`×`n` y el gráfico se agrega en el elemento `i`-ésimo. Además, tiene la ventaja de agregar gráficas que ocupen distinto tamaño en la figura resultante. Por ejemplo, las siguientes instrucciones:

```
>> X=-pi:0.1:pi;
>> subplot(3,2,1)
```

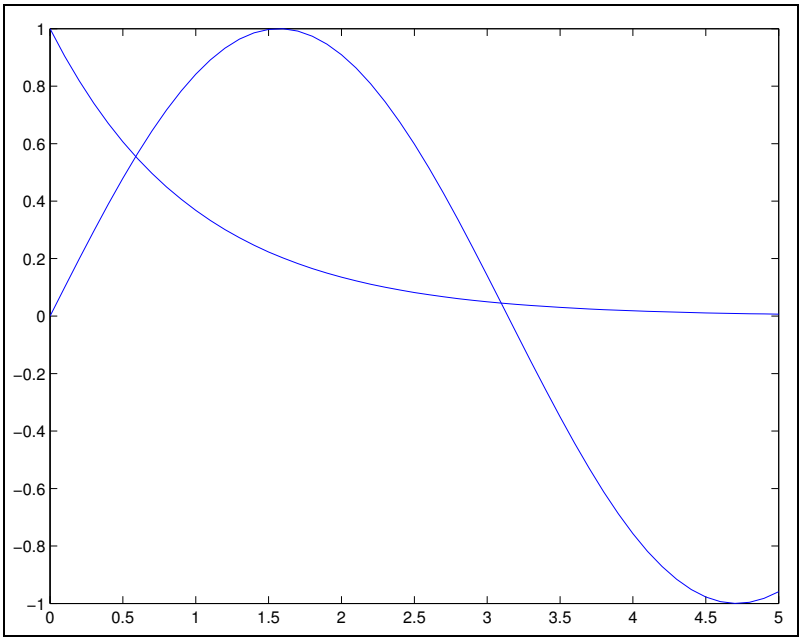

Figura 1.3: Ejemplo de `hold`

```
>> plot(X,sin(X));
>> subplot(3,2,2)
>> plot(X,cos(X));
>> subplot(3,1,2)
>> plot(X,cos(X)+sin(X));
>> subplot(3,1,3)
>> plot(X,[cos(X);sin(X)]);
```

generan como resultado la fig. 1.5.

También hay funciones para poner títulos, formatear los ejes, poner textos en cualquier parte de la figura, nombrar el eje x y el eje y, utilizar grilla, etc. La mayoría de las funciones anteriores aparece en la ayuda de la función `plot`.

No hay que olvidar que todas las instrucciones anteriores pueden combinarse, para producir las figuras que uno desea.

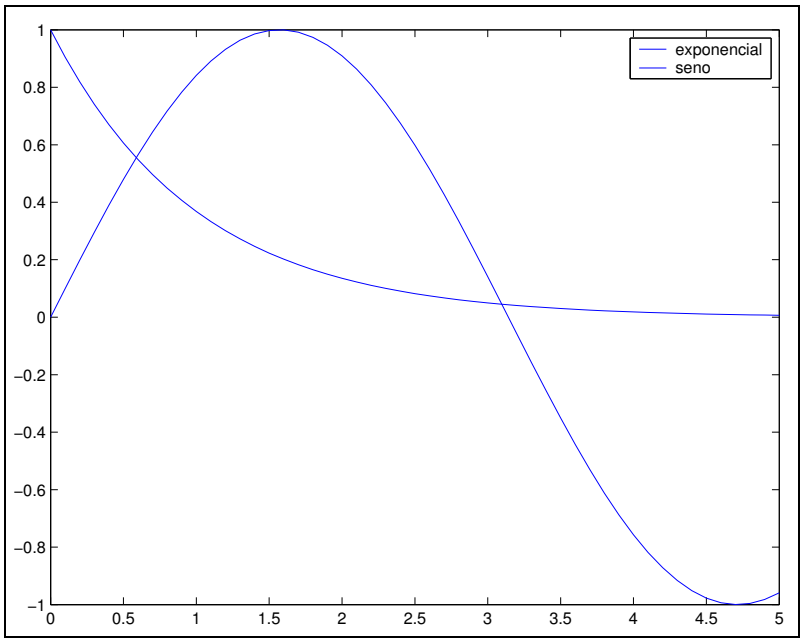

Figura 1.4: Ejemplo de `legend`

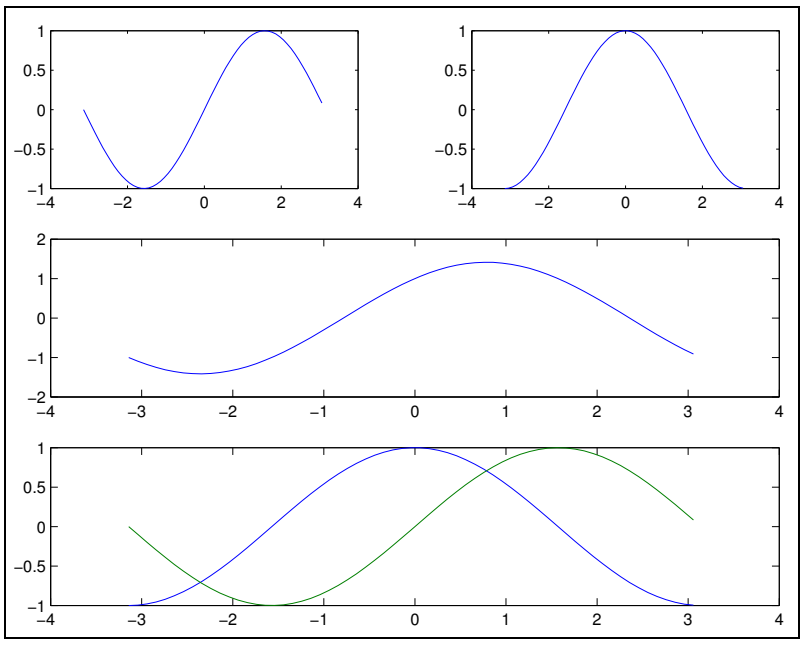

Figura 1.5: Ejemplo de `subplot`