MATLAB

Prof. Lino Coria Mendoza

1. Inicio

Operaciones básicas

Con Matlab puedes hacer cálculos simples como si trabajaras con una calculadora.

Operación	Símbolo	Ejemplo
Suma, a+b	+	5+3
Resta, a-b	-	20-9
Multiplicación,a*b	*	8.7*9.2
División, a÷b	/ o \	40/8 = 8\40
Potencia, a^b	^	5^3

A Matlab no le importan los espacios la mayoría de las veces y la multiplicación tiene preferencia sobre la suma.

```
» 2 + 1 + 3

ans =

     6

» 8*9 - 6*5 + 3*4

ans =

    54
```

Pueden guardarse los valores en variables:

```
» manzanas = 2

manzanas =

     2

» peras = 1

peras =

     1

» duraznos = 3

duraznos =
```

```
     3

» fruta = manzanas + peras + duraznos

fruta =

     6
```

El agregar un punto y coma al final de la instrucción le indica a Matlab que efectúe la operación pero que no despliegue la respuesta.

Para ver una lista de las variables almacenadas se utilizan los comandos **`who`** o **`whos`**.

```
» who

Your variables are:

ans         fruta       peras
duraznos    manzanas

» whos
  Name            Size         Bytes  Class

  ans             1x1              8  double array
  duraznos        1x1              8  double array
  fruta           1x1              8  double array
  manzanas        1x1              8  double array
  peras           1x1              8  double array

Grand total is 5 elements using 40 bytes
```

Las variables son sensibles a las mayúsculas y pueden contener hasta 19 caracteres. Deben comenzar con una letra y pueden seguir letras, números o guiones bajos.

Variables especiales	**Valor**
ans	Se usa por default para resultados
pi	3.141592...
eps	Número más pequeño usado por Matlab
inf	Infinito
NaN	Indica que no es un número
i (y) j	i = j = $\sqrt{(-1)}$

Para borrar variables se usa el comando **`clear`**.

Números complejos

```
» c1 = 1 - 2i

c1 =

   1.0000 - 2.0000i

» c2 = 6 - 9j

c2 =

   6.0000 - 9.0000i

» c3 = sqrt(-2)

c3 =

        0 + 1.4142i

» c4 = c1 + c2 - c3

c4 =

   7.0000 -12.4142i
```

Funciones Comunes	
abs(x)	Valor absoluto
acos(x)	Coseno inverso
angle(x)	Ángulo de número complejo
asin(x)	Seno inverso
atan(x)	Tangente inversa
ceil(x)	Redondear hacia más infinito
conj(x)	Complejo conjugado
cos(x)	Coseno
exp(x)	Exponencial
fix(x)	Redondear hacia cero
floor(x)	Redondear hacia menos infinito
imag(x)	Parte imaginaria de número complejo
log(x)	Logaritmo natural
log10(x)	Logaritmo base 10
real(x)	Parte real de número complejo
round(x)	Redondear hacia entero más cercano
sign(x)	Función signo
sin(x)	Seno
sqrt(x)	Raíz cuadrada
tan(x)	Tangente

Archivos de texto

Matlab permite colocar los comandos en un simple archivo de texto. Después al darle a Matlab el nombre del archivo de texto se evalúan los comandos como si se estuvieran escribiendo directamente sobre la pantalla de Matlab. Estos archivos se llaman *archivos "M"* porque deben tener la extensión **`.m`**, por ejemplo, **`primer.m`**.

`primer.m`

```
% Primer programa en Matlab

% Variables
a = 3;
b = -4;
c = 8;

% Operación
d = a*b-c
```

Esto aparece en Matlab al ejecutar `primer.m`:

```
» primer

d =

   -20
```

Funciones de archivos M	
`disp(ans)`	Despliega resultados sin identificar nombres de variables
`input`	Permite al usuario escribir información
`keyboard`	Da control al teclado de manera temporal (escríbase `return` para salir)
`pause`	Pausa hasta que se presione el teclado
`pause(n)`	Pausa durante `n` segundos
`waitforbuttonpress`	Pausa hasta que se presione teclado o se haga click en el mouse

Ayuda en línea

Para poder localizar comandos, Matlab tiene capacidad de brindar ayuda en línea. Existen tres opciones: el comando **`help`**, el comando **`lookfor`**, y el uso de la ayuda de la barra del menú.

El comando `help`

Escribe `help tema` para obtener ayuda sobre el tema de interés.

Ej.: `help pause`

El comando `lookfor`

Escribe `lookfor tema` y Matlab regresa toda la ayuda que tenga donde aparezca el tema de interés.

Ej.: `lookfor complex`

MATLAB

Prof. Lino Coria Mendoza

2. Arreglos

Vectores

En Matlab pueden hacerse arreglos de números de forma directa e intuitiva:

```
» a = [2 4 6 9 -1 3]

a =

     2     4     6     9    -1     3
```

Los números del arreglo pueden separarse por espacios (como en el ejemplo anterior) o comas. En caso de que se trabaje con números complejos deben evitarse los espacios y utilizarse las comas.

Hay formas sencillas de crear arreglos horizontales (vectores renglón) que van creciendo o decreciendo de manera lineal. Por ejemplo, supóngase que se desea obtener un vector `t` que contenga once valores igualmente espaciados que vayan de cero a π. Existen tres caminos para hacer esto:

- `» t = [0 .1*pi .2*pi .3*pi .4*pi .5*pi .6*pi .7*pi .8*pi .9*pi pi]`
- `» t = (0:0.1:1)*pi`
- `» t = linspace(0,pi,11)`

En cualquiera de estos tres casos el resultado es el mismo:

```
t =

  Columns 1 through 7

         0    0.3142    0.6283    0.9425    1.2566    1.5708    1.8850

  Columns 8 through 11

    2.1991    2.5133    2.8274    3.1416
```

- El primer camino no es más que colocar todos los valores uno por uno, lo cual se vuelve tedioso.
- El segundo camino lo que hace es crear un arreglo que comienza en 0, se incrementa 0.1 cada vez hasta llegar a 1. Cada elemento de este arreglo es posteriormente multiplicado por π.

- El tercer caso consiste en utilizar la función `linspace` de Matlab que tiene los siguientes argumentos:
  `linspace(valor_inicial, valor_final, número_de_valores)`

Obviamente las dos últimas instrucciones son más prácticas que la primera.

Construcción de Vectores	
`x = [1 3 9 33 0 -2]`	Se crea vector renglón `x` con los elementos especificados
`x = primero:último`	Se crea vector renglón `x` comenzando con `primero`, aumentando en intervalos de uno, terminando en o antes de `último`
`x = primero:incremento:último`	Se crea vector renglón `x` comenzando con `primero`, aumentando en intervalos de `incremento`, terminando en o antes de `último`
`x = linspace(primero,último,n)`	Se crea vector renglón `x` comenzando con `primero`, terminando en `último`, teniendo `n` elementos
`x = logspace(primero,último,n)`	Se crea vector renglón `x` espaciado logarítmicamente comenzando con $10^{\texttt{primero}}$, terminando en $10^{\texttt{último}}$, teniendo `n` elementos

Para accesar elementos individualmente en un vector se escribe entre paréntesis el lugar que el dato ocupa en el arreglo. Por ejemplo, **`x(1)`** es el primer elemento en **`x`**.

Para accesar un bloque de elementos se usa la notación siguiente: **`x(1:5)`**. En este ejemplo se accesan los primeros 5 elementos del arreglo. A continuación se muestran otros ejemplos:

`x(4:-1:1)`	Cuarto, tercer, segundo y primer elementos en ese orden.
`x(2:2:7)`	Segundo, cuarto y sexto elemento, y se detiene al llegar al 7.

Hasta ahora se ha trabajado únicamente con vectores renglón. Para crear un vector columna se separan los elementos con punto y coma.

```
» b = [1; 2; 3; 4; 5]
```

```
b =

     1
     2
     3
     4
     5
```

Para crear un vector columna usando la notación de los dos puntos `inicio:incremento:final`, o bien las funciones `linspace` o `logspace`, se debe *transponer* el resultado con el operador (`'`).

```
» a = 1:5

a =

     1     2     3     4     5

» b = a'

b =

     1
     2
     3
     4
     5
```

Si se aplica el operador (`'`) otra vez se obtiene el vector renglón. Si los números del vector son complejos, al transponerlo se obtiene el complejo conjugado transpuesto. Para que el vector sea transpuesto sin obtener su conjugado se usa el operador (`.'`).

Matrices

Los arreglos de números pueden también tomar la forma de matrices. Para crear una matriz los espacios y las comas se utilizan para separar los elementos de un mismo renglón, y los puntos y comas se usan para separar filas.

```
» g = [1 2 3 4; 5 6 7 8]

g =

     1     2     3     4
     5     6     7     8
```

En este caso **g** es una matriz de 2 renglones y cuatro columnas, es decir, es una matriz de 2x4. El punto y coma le indica a Matlab que inicie un nuevo renglón entre los valores 4 y 5.

Importante: Todos los renglones de una matriz deben tener el mismo número de columnas.

Matemáticas con arreglos y escalares

Para un arreglo, ya sea vector o matriz, la suma, la resta, la multiplicación y la división por un escalar equivale a aplicar la operación en todos los elementos del arreglo.

```
» g+3

ans =

     4     5     6     7
     8     9    10    11

» 2*g-2

ans =

     0     2     4     6
     8    10    12    14
```

Matemáticas entre arreglos

En Matlab cuando dos arreglos tienen las mismas dimensiones, la suma, la resta, la multiplicación y la división se aplican elemento por elemento. Por ejemplo:

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

A =

     1     2     3     4
     5     6     7     8
     9    10    11    12

» B = [1 1 1 1; 2 2 2 2; 2 2 2 2]

B =

     1     1     1     1
     2     2     2     2
     2     2     2     2
```

```
» A + B

ans =

     2     3     4     5
     7     8     9    10
    11    12    13    14

» 2*A - B

ans =

     1     3     5     7
     8    10    12    14
    16    18    20    22
```

Para la multiplicación y la división elemento por elemento se usa una notación ligeramente distinta. Lo mismo para elevar a alguna potencia.

```
» A.*B

ans =

     1     2     3     4
    10    12    14    16
    18    20    22    24

» A./B

ans =

    1.0000    2.0000    3.0000    4.0000
    2.5000    3.0000    3.5000    4.0000
    4.5000    5.0000    5.5000    6.0000

» A.^2

ans =

     1     4     9    16
    25    36    49    64
    81   100   121   144
```

Direccionamiento de Arreglos	
`A(r,c)`	Direcciona un subarreglo dentro de `A` definido por el vector índice de renglones deseados `r` y el vector índice de columnas deseadas `c`
`A(r,:)`	Direcciona un subarreglo dentro de `A` definido por el vector índice de renglones deseados `r` y todas las columnas
`A(:,r)`	Direcciona un subarreglo dentro de `A` definido por todos los renglones y el vector índice de columnas deseadas `c`
`A(:)`	Direcciona todos los elementos de `A` como un vector columna tomado columna por columna
`A(i)`	Direcciona un subarreglo dentro de `A` definido por el vector índice individual de elementos deseados en `i`, como si `A` fuera el vector columna `A(:)`

Funciones de Matrices	
`det(A)`	Determinante
`inv(A)`	Matriz inversa
`norm(A)`	Norma de matriz o vector
`rank(A)`	Número de renglones o columnas linealmente independientes
`trace(A)`	Suma de los elementos de la diagonal

Matrices Especiales	
`eye`	Matriz identidad
`magic`	Cuadrado Mágico
`ones`	Matriz que contiene solamente unos
`rand`	Matriz aleatoria uniformemente distribuida con elementos entre 0 y 1
`randn`	Matriz aleatoria normalmemente distribuida con elementos con media cero y varianza unitaria
`zeros`	Matriz que contiene solamente unos

MATLAB

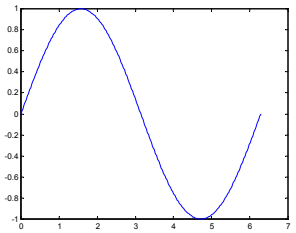
Prof. Lino Coria Mendoza

3. Gráficas en 2D

La función `plot`

El comando más utilizado para graficar datos en dos dimensiones es `plot`.

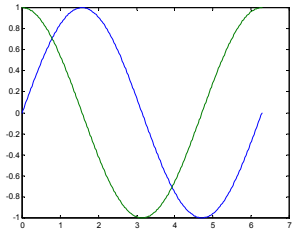
```
» t = linspace(0,2*pi,300);
» x = sin(t);
» plot(t,x)
```



Este ejemplo genera 300 puntos en el rango de $0 \leq t \leq 2\pi$ para formar el eje horizontal y crea un vector `x` que contiene el seno de los puntos de `t`.

Se pueden graficar un seno y un coseno en la misma figura.

```
» y = cos(t);
» plot(t,x,t,y)
```



Funciones para graficar en 2D	
`bar`	Gráfica de barra
`fill`	Dibujo de polígono en 2D relleno
`hist`	Histograma
`image`	Imagen
`loglog`	Gráfica con ambos ejes en escala logarítmica
`plot`	Gráfica simple
`polar`	Gráfica en coordenadas polares
`semilogx`	Gráfica con eje `x` en escala logarítmica
`semilogy`	Gráfica con eje `y` en escala logarítmica
`stairs`	Gráfica de escalera
`stem`	Gráfica de secuencia discreta

Herramientas para graficar en 2D	
`axis`	Modifica las propiedades de los ejes
`clf`	Borra la ventana *Figure*
`close`	Cierra la ventana *Figure*
`figure`	Crea o selecciona la ventana *Figure*
`grid`	Coloca una rejilla
`gtext`	Coloca texto con el mouse
`hold`	Mantiene la gráfica actual
`subplot`	Crea subgráficas dentro de la ventana *Figure*
`text`	Coloca texto en un lugar dado
`title`	Coloca el título
`xlabel`	Coloca etiqueta del eje `x`
`ylabel`	Coloca etiqueta del eje `y`
`zoom`	Magnifica o disminuye los ejes

MATLAB

Prof. Lino Coria Mendoza

4. Procesamiento de Señales de Voz con Matlab

Con Matlab podemos trabajar en campos tan variados como comunicaciones, control, electrónica de potencia, procesamiento digital de imágenes y de señales. En este capítulo nos concentraremos en el Procesamiento Digital de Señales, trabajando con señales de voz.

Cargar una señal de voz

Para cargar un archivo ".wav" a Matlab se utiliza la siguiente instrucción:

```
[x,fs,bits] = wavread('archivo.wav');
```

Donde `x` es la variable que guarda las muestras de la señal de voz, `fs` es la frecuencia de muestreo y `bits` indica el número de bits utilizados en cada muestra para codificar la señal.

Para escuchar la señal de voz en Matlab se usa esta instrucción:

```
soundsc(x,fs);
```

Procesamiento digital de señales

Una señal de voz discreta no es más que una tabla con datos de las muestras tomadas de la señal analógica original. Por lo tanto, podemos trabajar con ella como con cualquier otro vector y utilizar todas las funciones y operaciones que hasta ahora se han visto.

Podemos sumar señales de voz, multiplicarlas, recortarlas, filtrarlas, etc. Pueden pues procesarse fácilmente.

Para grabar una señal procesada se usa el siguiente comando:

```
wavwrite(x2,fs,bits,'nuevo.wav');
```

Donde `x2` es la variable que guarda las muestras de la señal de voz ***procesada***, `fs` es la frecuencia de muestreo y `bits` indica el número de bits utilizados en cada muestra para codificar la señal.

Algunos comandos para procesar señales	
`y = conv(x,h)`	La convolución de `x` con `h`
`X = fft(x)`	La transformada discreta de Fourier del vector `x`
`fftshift`	Util para visualizar la transformada de Fourier con el componente de dc a la mitad del espectro
`b = fir1(n,w)`	Diseña un filtro digital pasabajas FIR de orden `n` con frecuencia de corte `w` y regresa los coeficientes al vector `b`
`freqz`	Muestra la respuesta en frecuencia de un filtro digital
`x = ifft(X)`	La transformada discreta inversa de Fourier del vector `X`