Tutorial de MATLAB
para MA45C: Ecología Microbiana

Héctor Ramírez C. *

6 de abril de 2010

Introducción a MATLAB

a) Descripción del ambiente MATLAB.

Command Window. Ésta es la ventana en la que se ejecutan interactivamente las instrucciones de MATLAB y en donde se muestran los posibles resultados, si corresponde. Luego, se puede considerar como la ventana más importante de MATLAB.

Launch Pad. Ventana de la cual se puede acceder a otros componentes de MATLAB como el *Help Browser* (o Buscador de ayuda), o las *Demos* asociadas a distintos tópicos de MATLAB. Esto permite trabajar de una forma más dinámica sin pasar por menús o por otros comandos.

Workspace Browser. Es el espacio de trabajo de MATLAB donde se nos muestra el conjunto de variables y funciones que actualmente están definidas en la memoria de MATLAB, y por lo tanto se puede acceder y trabajar con esta información desde el *Command Window*.

Command History Browser. Permite el acceso a las sentencias que se han ejecutado anteriormente en el *Command Window*. Esto mediante un doble clic de la sentencia deseada. También se puede acceder a estas sentencias usando las teclas $\uparrow$ y $\downarrow$.

Current Directory Browser. Permite acceder y determinar el *directorio actual* de MATLAB, el cual se define a grandes rasgos como el directorio (o

*Centro de Modelamiento Matemático & Departamento de Ingeniería Matemática, Universidad de Chile, Casilla 170-3, Santiago 3, Chile, e-mail: hramirez@dim.uchile.cl

carpeta) que MATLAB utiliza para "funcionar" en la actual sesión. En efecto, sólo se podrán usar los archivos o ejecutar los programas, procedimientos y funciones que se encuentran en este directorio o en uno de uso interno de MATLAB que ya se encuentre especificado. Ésta ventana también permite ver y trabajar con los archivos que se encuentra en el directorio de trabajo.

Observación 1. El usuario puede incorporar una de sus carpetas como una carpeta de uso interno de MATLAB, o lo que usualmente se llama "incorporar una carpeta al *Path* de MATLAB". La manera más fácil de hacer esto es mediante el menú, siguiendo las opciones: FILE → SET PATH.

b) Acceso al directorio de trabajo.

Estos comandos nos permiten acceder directamente a la carpeta de trabajo o *directorio actual* desde la ventana *Command Window*, lo cual se puede ver como una manera alternativa a trabajar con la ventana *Current Directory Browser*. La mayoría de ellos funcionan igual que en el sistema operativo MS-DOS.

dir. Nos muestra todos los archivos del *directorio actual*.

mkdir (Make Directory). Crea un directorio (o carpeta). Por ejemplo:

```
>> mkdir C:\Curso_MATLAB_UDP
```

crea el directorio o carpeta "Curso_MATLAB_UDP" en el disco duro "C:".

cd (Change Directory). Cambia el *directorio actual*. Por ejemplo:

```
>> cd C:\Curso_MATLAB_UDP
```

cambia el *directorio actual* al directorio o carpeta "Curso_MATLAB_UDP" en el disco duro "C:".

Observación 2. El directorio al que se desea cambiar debe existir.

pwd (Print Working Directory). Nos muestra cual es el *directorio actual*. Por ejemplo:

```
>> pwd

ans =

C:\Curso_MATLAB_UDP
```

type. Nos muestra el contenido de un archivo.

copyfile. Copia un archivo. Por ejemplo:

```
>> copyfile sesion_1.m sesion__2.m
```

copia el archivo "sesion_1.m" en el archivo "sesion_2.m" dentro del *directorio actual.*

delete. Borra un archivo.

c) Comandos básicos de asignación y comparación.

En esta sección veremos como usar los operadores relacionales y lógicos en MATLAB, estos son los que nos permiten efectuar una o varias comparaciones asignando un grado de verdad a una condición matemática.

Para esto primero debemos establecer la forma de asignar valor a las variables. En MATLAB esto se hace simplemente con el comando "=". Por ejemplo:

```
>> a=2

a =

     2
```

asigna el valor 2 a la variable "a". Nótese que la variable "a" no necesita ser definida previamente. Si en el anterior ejemplo se tipea el símbolo ";" después de la asignación, esta será efectuada pero MATLAB no mostrara ninguna respuesta. El valor de la variable puede ser luego consultado en el *Workspace Browser.*

Los operadores relacionales (o de relación) son los siguientes:

$<$	menor que,
$>$	mayor que,
$<=$	menor o igual que,
$>=$	mayor o igual que,
$==$	igual que,
$\sim=$	distinto que.

MATLAB asigna un número 1 para el valor "verdadero" y 0 para "falso". Por ejemplo:

```
>> 1>=2

ans =

      0
```

Los operadores lógicos son definidos como sigue:

&	and (y)
|	or (o)
$\sim$	negación lógica

Ejemplifiquemos su uso. Consideremos la variable "a", a la cual le fue asignado anteriormente el valor 2. Se obtiene así que:

```
>> a>0 & ~(a~=2)

ans =

     1
```

d) Sentencias lógicas.

MATLAB nos permite programar rutinas usando sentencias que se ejecutaran si ciertas condiciones son ciertas. Estos comandos funcionan similarmente a cualquier otro lenguaje de programación y sólo su sintaxis es levemente distinta. Procederemos a explicar brevemente cada una de ellas.

if, elseif, else. Se describe su uso como sigue:

```
if condicion1
    rutina1
elseif condicion2
    rutina2
%....
elseif condicionN
    rutinaN
else
    rutina_final
end
```

for. Repite un conjunto de sentencias (es decir, una rutina) un número predeterminado de veces dada por la primera sentencia.

```
for j=1:n
    rutina
end
```

while. Se ejecuta una rutina mientras la condición dada en la primera sentencia sea cierta (es decir tenga valor "verdadero"). Su sintaxis es la siguiente:

```
while condicion
    rutina
end
```

break. Interrumpe la actual rutina.

try...catch...end. Permite gestionar errores que se pueden producir en la ejecución de un programa. Su sintaxis es:

```
try
   rutina1
catch
   rutina2
end
```

En el caso de que en la primera rutina ("rutina1") se produzca un error, MATLAB pasa a ejecutar la segunda rutina ("rutina2"). Si no se produciera un error en la primera rutina, la segunda rutina no se ejecutaría nunca.

e) Lectura y escritura de variables.

input. Permite imprimir un mensaje en la línea de comandos de MATLAB y al mismo tiempo recuperar un valor numérico o un texto (o "string") que ha sido ingresado por el usuario. Veamos su modo de uso:

```
>> b=input('ingrese numero ');
ingrese numero
```

De este modo MATLAB espera que el usuario ingrese una valor para la variable "b". Para recuperar un texto se debe incorporar la opción 's'. Por ejemplo, ejecutando

```
>> nombre=input('Cual es tu nombre? ','s');
Cual es tu nombre?
```

MATLAB espera que el usuario ingrese un texto que identifique su nombre, el cual será guardado en la variable "nombre".

display (o disp). Estos comandos nos permiten imprimir en pantalla un mensaje de texto o el valor de una variable. El comando *display* imprime además el nombre de la variable mientras que *disp* no lo hace. Veamos un ejemplo:

```
>> display(b)

b =

     4

>> display('Hola')

ans =

Hola
```

o bien

```
>> disp(b)
     4

>> disp('Hola')
Hola
```

f) Guardar variables y sesiones.

Muchas veces se desea abandonar una sesión de MATLAB pero se desea volver a ella más adelante. Los siguientes comandos nos permiten solucionar este problema y recuperar la información que se ha acumulado en el *workspace*.

save, load. El primer comando nos permite guardar una sesión mientras que el segundo nos permite cargar una que ya está guardada.

diary. Almacena en un archivo de texto especificado por el usuario lo que se va haciendo en la sesión. Este comando funciona como sigue:

```
>> diary nombre_archivo.txt
%...
>> diary off
```

```
%...
>> diary on
```

Así "diary off" suspende la ejecución del comando mientras que "diary on" lo reanuda. Se puede acceder al archivo de texto donde se guarda la sesión solamente cuando "diary" esta en "off".

Funciones

a) Definición de funciones y procedimientos.

Para definir funciones o procedimientos en MATLAB se usa un *texto plano* que debe terminar en la extensión ".m". Una manera simple de editar y crear un archivo ".m" es el editor de texto plano que se incluye en MATLAB, y al cual se puede acceder desde su menú.

Para crear una función se debe tener en cuenta que el archivo ".m" debe tener el mismo nombre que el de la función. Además, para poder usar la función previamente definida se debe guardar el archivo en el *directorio actual* (o directorio de trabajo) o bien en algún directorio de uso interno de MATLAB.

Veamos como crear la función

$$f(x) = e^x + x^2 - x + 1.$$

En algún editor de texto plano como el que tiene MATLAB se escribe:

```
function y=f(x);

y=exp(x)+x^2-x+1;
```

y luego se guarda con el nombre "f.m".

Ejercicio 1. Crear la función

$$g(x) = \begin{cases} \cos(x) + x^3 + \sqrt{x} - 1, & \text{si } x \geq 0, \\ \log(x+1) - x, & \text{si } x < 0 \end{cases}$$

De esta misma forma se pueden crear funciones con rutinas más complicadas o procedimientos que no necesariamente entregan un valor determinado.

La mejor manera de evaluar una función previamente definida es usando el comando *feval*

```
>> feval('f',0)

ans =

     2
```

Un procedimiento es creado mediante un archivo ".m" cuyo nombre será el del procedimiento o rutina a ejecutar, y que contendrá las instrucciones y sentencias a seguir.

b) Funciones predefinidas.

MATLAB tiene varias funciones matemáticas predefinidas. Veamos algunas de las más importantes.

sqrt. raíz de un número real positivo.

cos, sin, tan. Representan las funciones trigónometricas seno, coseno y tangente.

max, min, sum, mean, sort. Estas funciones se aplican a un conjunto de valores que generalmente son guardados como un vector (esto se vera en detalle en la proxima sesión).

El comando "max" nos entrega el maximo valor, "min" el mínimo, "sum" la suma de todos los valores, "mean" el promedio y "sort" ordena los valores de menor a mayor. Ejemplo:

```
>> t=[1 23 3 -5 45];
>> sum(t)

ans =

    67

>> sort(t)

ans =
    -5     1     3    23    45

>> min(t)

ans =
```

```
    -5

>> mean(t)

ans =

   13.4000
```

floor, round. El primero nos da el número entero menor más cercano al valor real al cual se le aplica. Mientras que el segundo aproxima el valor.

```
>> floor(3.6)

ans =

     3

>> round(3.6)

ans =

     4
```

rand. Genera matrices o vectores cuyas componentes son números aleatorios entre cero y uno.

c) Gráfico de funciones.

El gráfico de una función es una herramienta muy importante en el analísis matemático. Explicaremos brevemente los comandos que nos permiten realizar un gráfico.

plot. Este comando nos permite graficar un conjunto de datos de la forma $(x_i, y_i),\ i = 1, ..., n.$. Para esto se debe trabajar con dos vectores, por ejemplo x e y, que contengan la información del eje x e y respectivamente. Veamos como dibujar la función seno en el intervalo [-10, 10] .

```
>> x=-10:0.1:10; y=sin(x);
>> plot(x,y);
```

En este caso hemos creado una grilla de puntos equiespaciados entre -10 y 10, con un paso igual a 0.1, usando el comando ":". Esto ha sido finalmente guardado en la variable "x".

Veamos ahora ciertas opciones que nos permiten individualizar nuestro gráfico. Se puede dar un nombre a los ejes x e y usando los comandos

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

El tipo de línea se determina con

Tipo de línea	Símbolo
continua (por defecto)	-
guiones	- -
punteada	:
guiones y puntos	-.

Se pueden usar marcas especiales en vez de líneas mediante los cdigos

Tipo de marca	Símbolo
punto	.
más	+
estrella	*
círculo	o
marca x	x

Podemos también especificar el color del trazado usando los siguientes cdigos:

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

Por ejemplo, reproduciremos nuevamente el gráfico anterior con una marca de círculo de color azul.

```
>> plot(x,y,'ob');
```

Para dibujar varias funciones al mismo tiempo se debe utilizar una *matriz* de valores. Este tópico será estudiado en la proxima clase. Veamos un ejemplo:

```
>> A= [sin(x); cos(x)];
>> plot(x,A);
```

Observación 3. Cuando se calculan los valores, ya sea "y" o la matriz "A" en los ejemplos anteriores, se deben reemplazar las operaciones usuales multiplicación "*" división "/" y elevado " ^ " por las versiones "punto" o "componente a componente" de estas operaciones ".* ./ .^ ". Esto evita errores en el manejo de las dimensiones. Esto será explicado en profundidad en la proxima sección.

fplot. Este comando grafica directamente una función sin pasar por un grillaje. Su sintaxis es la siguiente:

```
>> fplot('nombre_funcion',[xmin xmax],'opciones');
```

Los valores "xmin" y "xmax" nos dan los valores mínimos y mximos del eje x, las "opciones" de gráficos son las mismas que las del comando *plot.*

subplot. Permite hacer varios gráficos al mismo tiempo haciendo divisiones de la pantalla original. Por ejemplo

```
>> subplot(2,2,1), plot(x,y);
>> subplot(2,2,2), plot(x,A);
>> subplot(2,2,3); fplot('sin',[-10 10],'+g');
>> subplot(2,2,4); fplot('f',[-1 1],':k');
```

divide la pantalla gráfica en cuatro partes dibujando en cada uno de los segmentos las funciones correspondientes.

d) Definición de polinomios.

La manera más simple de definir polinomios es a través de un conjunto de valores reales que determinaran los coeficientes del polinomio, por ejemplo, los valores del vector "t" definido anteriormente nos entregan el polinomio

$$t(x) = x^4 + 23x^3 + 3x^2 - 5x + 45.$$

e) Comandos para polinomios.

Describamos brevemente los comandos relacionados con el uso de los polinomios.

polyval. Permite evaluar un polinomio. Consideremos el vector "t" definido anteriormente.

```
>> polyval(t,0)

ans =

     45
```

roots. Nos entrega las raíces del polinomio correspondiente.

```
>> roots(t)

ans =

 -22.8554
  -1.3873
   0.6214 + 1.0164i
   0.6214 - 1.0164i
```

polyder. Calcula la derivada de un polinomio. El resultado es un vector donde las componentes son los coeficientes del polinomio derivado.

```
>> polyder(t)

ans =

     4    69     6    -5
```

Note que el resultado tiene una dimensión menor que el polinomio original.

conv, deconv. Nos entrega la multiplicación y la división usual de polinomios. En el caso de la división también nos entrega el polinomio residuo. Revisemos rpidamente su sintaxis.

```
>> q=conv(p,t);
>> [q,r]=deconv(p,t);
```

Ejercicio 2. Calcular el producto entre los polinomios "t", definido anteriormente, y el polinomio

$$p(x) = 1 - x^2 + 5x^3 + 13x^4 - 6x^5.$$

Ejercicio 3. Calcular el polinomio residuo de la división del polinomio "p" por el polinomio "t".

Matrices

a) Definición de Matrices.

Las matrices se crean introduciendo sus elementos entre corchetes cuadrados "[" y "]". Se procede primero por filas las cuales se separan por el símbolo ";". Luego, cada elemento en una fila es separado por un espacio o coma. Ilustremos su uso:

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

A =

     1     2     3
     4     5     6
```

Los vectores se consideran como casos partículares de matrices donde una de las dimensiones es igual a 1.

Existen comandos que también permiten definir matrices, por ejemplo el operador traspuesto definido por la comilla simple '.

```
>> A'

ans =
     1     4
     2     5
     3     6
```

Los comandos "zeros" y "ones" permiten crear matrices de una dimensión asignada por el usuario cuyos elementos son sólo ceros y unos, respectivamente. Por ejemplo:

```
>> zeros(1,3)

ans =

     0     0     0

>> ones(3,2)
```

```
ans =

     1     1
     1     1
     1     1
```

El comando “diag” tiene dos funciones en dependencia si se aplica a un vector o a una matriz. Si se aplica a un vector, este comando nos entrega una matriz diagonal, donde los elementos de la diagonal son las componentes del vector.

```
>> x=[1 3 4]; diag(x)

ans =

     1     0     0
     0     3     0
     0     0     4
```

Si se aplica a una matriz, este comando extrae la diagonal de la matriz y nos da como respuesta un vector columna cuyas componentes son los elementos de esta diagonal.

```
>> diag(A)

ans =

     1
     5
```

Una manera muy útil para definir y crear matrices es el “concatenar” diferentes submatrices. De esta forma se crea una matriz de un tamaño mayor cuyos bloques corresponden a los valores concatenados. Por ejemplo:

```
>> B=[A ; x]

B =

     1     2     3
     4     5     6
     1     3     4
```

crea una nueva matriz "B" donde sus dos primeras filas corresponden a la matriz "A", y su tercera fila es el vector "x".

Ejercicio 4. Crear directamente la matriz utilizada en el cálculo de la *spline cúbica* asociada a una malla de 51 puntos equiespaciados donde la distancia entre los puntos es igual a 1. Esta matriz viene dada por

$$\begin{pmatrix} 2 & 1 & & & & & \\ 1 & 4 & 1 & & & & \\ & \ddots & \ddots & & & & \\ & & & \ddots & \ddots & & \\ & & & & 1 & 4 & 1 \\ & & & & & 1 & 2 \end{pmatrix}.$$

Los espacios vacíos son considerados con el valor 0 y el tamaño de la matriz es 50.

Ejercicio 5. Recordando el comando *rand* enseñado en la sesión anterior, crear una matriz de 7 filas y 9 columnas cuyas componentes sean enteros entre -5 y 10.

Para ver o utilizar en algún cálculo el elemento (i, j) de la matriz "A" se debe usar la sintaxis:

```
>>  A(i,j)
```

Así también podemos ver o utilizar una fila o columna completa de una matriz con el comando ":". Veamos un ejemplo de su uso.

```
>> A(1,:)

ans =

     1     2     3
```

muestra la primera fila de la matriz "A". Similarmente, "A(:,j)" muestra la columna "j" de la matriz "A".

Otra aplicación interesante del comando ":" nos permite mostrar solo una parte de cierta fila o columna. Ejemplifiquemos su uso.

```
>> A(:,1:2)
```

```
ans =

     1     2
     4     5
```

muestra las dos primeras columnas de la matriz "A". También podemos ver o utilizar la primera y tercera columna de "A" de la siguiente forma

```
>> A(:, [1 3])

ans =

     1     3
     4     6
```

Finalmente veremos el uso de la variable indicadora "end" que hace referencia a la última fila o columna de una matriz. Por ejemplo,

```
>> A(end,:)

ans =

     4     5     6
```

muestra la última fila de la matriz "A".

b) Operaciones básicas sobre matrices.

MATLAB permite operar con las operaciones matriciales clásicas: + - * y ^. Para realizar estas operaciones se deben respetar las reglas matemáticas básicas sobre las dimensiones de las matrices implicadas en estas operaciones.

Otro tipo de operaciones aritméticas que se pueden realizar sólo con matrices o vectores de la misma dimensión son las llamadas "operaciones punto o componente a componente". Como su nombre lo dice, estas operaciones se realizan aritméticamente componente a componente, obteniendo un resultado con la misma dimensión que los vectores o matrices a los cuales fue aplicada la operación. Ejemplo:

```
>> y=[2 -1 4];
>> x.*y

ans =
```

```
     2    -3    16

>> x.^y

ans =

    1.0000    0.3333  256.0000

>> x./y

ans =

    0.5000   -3.0000    1.0000
```

c) Comandos especiales para matrices.

Procederemos a describir brevemente algunos de los principales comandos matriciales en MATLAB.

trace. Calcula la traza de una matriz.

det. Calcula el determinante de una matriz cuadrada.

size. Calcula el tamaño de una matriz.

```
>> size(A)

ans =

     2     3
```

inv. Calcula la inversa de una matriz cuadrada no singular.

norm. Calcula la norma (inducida) a una matriz. Existe un segundo parámetro que nos permite especificar la norma utilizada. Por defecto se usa la norma "2", es decir, la norma inducida euclidiana.

```
>> norm(A)

ans =

    9.5080
```

```
>> norm(A,inf)

ans =

    15
```

cond. Calcula el condicionamiento de una matriz.

pinv. Calcula la pseudo-inversa de una matriz. La pseudo-inversa existe para cualquier matriz, incluso una que no sea cuadrada.

```
>> pinv(A)

ans =

   -0.9444    0.4444
   -0.1111    0.1111
    0.7222   -0.2222
```

rank. Calcula el rango de una matriz. Esto es la cantidad de filas o columnas linealmente independientes.

```
>> rank(A)

ans =

     2
```

d) Factorización y descomposición matricial.

Revisaremos ciertos comandos asociados a descomposiciones y factorizaciones de matrices.

chol. Realiza la descomposición de Choleski de una matriz definida positiva.

null. Devuelve una base ortonormal del subespacio nulo o "Kernel" de una matriz.

qr. Nos entrega la descomposición QR de una matriz.

orth. El resultado es una matriz cuyas columnas son una base ortonormal del espacio generado por las columnas de la matriz a la cual es aplicada esta función.

eig. Calcula la descomposición espectral de una matriz. Ilustremos su uso.

```
>> C=A'*A;
>> [Q,D]=eig(C)

Q =

   -0.4082   -0.8060    0.4287
    0.8165   -0.1124    0.5663
   -0.4082    0.5812    0.7039


D =

   -0.0000         0         0
         0    0.5973         0
         0         0   90.4027
```

Nos da como resultado una matriz ortogonal "Q" cuyas columnas son los vectores propios de la matriz "C" y una matriz diagonal "D" con los valores propios de "C".

poly. Devuelve un vector con los coeficientes del polinomio característico de una matriz cuadrada.
lu. Calcula la descomposición LU de una matriz. Veamos un ejemplo de su uso.

```
>> [L,U]=lu(C)

L =

    0.6296    1.0000         0
    0.8148    0.5000    1.0000
    1.0000         0         0


U =

   27.0000   36.0000   45.0000
         0   -0.6667   -1.3333
         0         0         0
```