Introducción a Matlab

IPDI

2do. cuatrimestre de 2013

En este apunte veremos las operaciones más comunes del entorno de programación *Matlab*. Se aprenderán a manejar los aspectos básicos como saltos condicionales, ciclos, operaciones con vectores y matrices, y otros más avanzados como el manejo de imágenes.

1. Entorno

Para la realización de las primeras guías de ejercicios de este curso usaremos el entorno de programación Matlab. Matlab (*Matrix Laboratory*) es una herramienta especializada en cálculos matriciales. Es de gran ayuda para el procesamiento de imágenes.

Al iniciar el entorno de programación de Matlab se nos muestra una aplicación con 3 áreas de trabajo claramente diferenciadas (Figura 1). La primera zona de la aplicación se compone de los diversos menús y barras de botones. Después se nos presenta una amplia zona denominada *Command Window* donde podremos ejecutar los comandos de Matlab. Una zona que se denomina *Workspace* muestra diversa información tal como las variables que existen en memoria y un listado de los archivos del directorio de trabajo. Finalmente, *History* exhibe los últimos comandos ejecutados.

2. Comandos básicos

Dentro del entorno de Matlab podemos cambiar de directorio, borrar y crear archivos y directorios, visualizar los contenidos de directorios, etc.

Los comandos más comunes son:

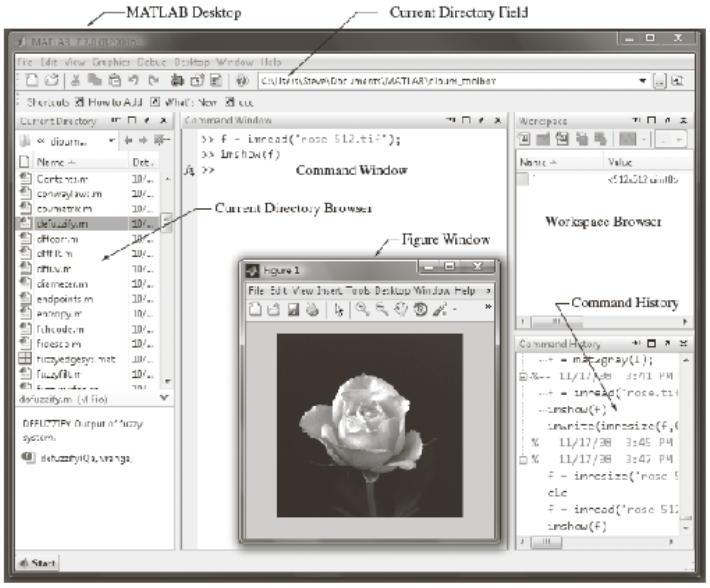

FIGURE 1.1 The MATLAB Desktop with its typical components.

Figura 1: Áreas de trabajo, escritorio Default

- `ls` ó `dir`: para listar los contenidos del directorio donde nos encontramos. Permite el uso de comodines (ej. `ls *.m` nos muestra todos los archivos con extensión *.m*, mientras que `ls ???.m` solamente nos muestra los archivos con extensión *.m* y que su nombre esté compuesto de tres *caracteres*.

- `cd`: Para cambiar de un directorio a otro.

- `pwd`: Para mostrar el directorio actual en el que nos encontramos (esta información también se muestra en la barra).

- `mkdir`: Para crear un directorio (ej. `mkdir directorio1` crear un directorio llamado `directorio1` dentro del directorio actual).

- `rmdir`: Para borrar un directorio, el mismo debe estar vacío. (ej. `rmdir directorio1`) borra el directorio de nombre `directorio1`). Si a `rmdir` se le añade el parámetro `s`, se borra el contenido del directorio y todo subdirectorio que exista dentro del mismo (ej. `rmdir directorio1 s`).

- `delete`: Se usa para borrar un archivo concreto (ej. `delete archivo1` borra el `archivo1`) del directorio actual).

Para utilizar la ayuda de Matlab, tecleando `help` junto al nombre de algún comando nos saldrá la ayuda disponible sobre este último. Si, en cambio, utilizamos el comando `doc` se abrirá el manual de Matlab mostrando esta misma información.

3. Operaciones Matemáticas

3.1. Vectores y matrices

Veremos unas pocas operaciones básicas con matrices que nos serán de ayuda a la hora de trabajar con imágenes. En Matlab todo son matrices, incluso los números y los vectores, los primeros son una matriz 1×1 y los segundos son una matriz ($1 \times N$ en el caso de que sea un vector fila ó $N \times 1$ en el caso de que sea un vector columna). Por ejemplo:

```
>> v = [1 2 3 4]
v =
1 2 3 4
```

Se puede trasponer dicho vector usando el operador `'`, por ejemplo:

```
>> w = v'
w =
1
2
3
4
```

Para acceder a un elemento concreto de un vector sencillamente usamos:

```
>> v(2)
ans =
2
```

donde *ans* es una variable temporal de Matlab que almacena el último resultado si no lo asignamos a ninguna variable. Para asignar un valor a una variable en Matlab sencillamente escribimos `a = v(2)`. En general, a la variable `a` se le asignará el resultado de la operación presente después del `=`. Ahora, si lo que queremos es recuperar un rango de resultados, por ejemplo los que van del 2do. al 4to. elemento del vector v, haremos:

```
>> v(2:4)
ans =
2 3 4
```

Esta última sentencia la podríamos haber sustituido por:

```
>> v(2:end)
ans =
2 3 4
```

Por otro lado, también se pueden mostrar elementos en orden no continuo, por ejemplo:

```
v =
1 2 3 4 5 6 7 8 9 10
>> v(1:2:end)
ans =
1 3 5 7 9
```

Esto hará que se vaya recorriendo el vector v de dos en dos desde la posición 1 hasta el final del mismo.

También se puede usar otro vector como índice de un vector, por ejemplo:

```
>> v([1 4 7])
ans =
1 4 7
```

Definir una matriz en Matlab es tan fácil como definir un vector, por ejemplo, para definir una matriz 3×3:

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

La selección de elementos de una matriz es tan sencilla como la selección de elementos de un vector, con la única diferencia de que ahora tenemos que escribir dos índices. Por ejemplo:

```
>> A(2, 3)
ans =
6
```

Para seleccionar un vector columna:

```
>> A(1:2,3)
ans =
3
6
```

Para seleccionar un vector fila:

```
>> A(2,:)
ans =
4 5 6
```

Observar que aquí el uso de `:` es igual a haber escrito `1:3`.

Para seleccionar una submatriz:

```
>> A(1:2,:)
ans =
1 2 3
4 5 6
```

Para crear una matriz B igual a A, y hacer que todos los elementos de la 3era columna de B sean igual a cero podemos seguir el siguiente ejemplo:

```
>> B=A;
>> B(:,3)=0
B =
1 2 0
4 5 0
7 8 0
```

Por supuesto, podemos indexar elementos no continuos al igual que con los vectores, por ejemplo:

```
>> A(1:end,1:2:end)
ans =
1 3
4 6
7 9
```

También podemos usar vectores y matrices lógicas (matrices *verdadero* o *falso*, donde el número 1 representa verdadero y el 0 representa falso) para indexar matrices:

```
>> E = A([1 3],[2 3])
E =
2 3
8 9
>> D = logical([1 0 0; 0 0 1; 0 0 0])
D =
1 0 0
0 0 1
0 0 0
>> A(D)
ans =
1
6
```

Solamente se muestran como resultado en forma de vector los casos *verdadero*.

El operador : también nos devuelve todos los elementos de una matriz en un único vector columna:

```
>> A(:)
ans =
1
4
7
2
5
8
```

```
3
6
9
```

Por último, el comando `size(A)` devuelve las dimensiones de la matriz A:

```
>> A = [1 1 1; 1 2 3]
A =
1 1 1
1 2 3
>> [M, N] = size(A)
M =
2
N =
3
```

En el ejemplo anterior, almacenamos en una variable `M` el número de filas de A y en una variable `N` el número de columnas de A.

3.2. Tipos de datos

Matlab soporta los siguientes 8 tipos de datos numéricos, 1 char y uno lógico.

Name	Description
`double`	Double-precision, floating-point numbers in the approximate range $\pm 10^{308}$ (8 bytes per element).
`single`	Single-precision floating-point numbers with values in the approximate range $\pm 10^{38}$ (4 bytes per element).
`uint8`	Unsigned 8-bit integers in the range [0, 255] (1 byte per element).
`uint16`	Unsigned 16-bit integers in the range [0, 65535] (2 bytes per element).
`uint32`	Unsigned 32-bit integers in the range [0, 4294967295] (4 bytes per element).
`int8`	Signed 8-bit integers in the range [−128, 127] (1 byte per element).
`int16`	Signed 16-bit integers in the range [−32768, 32767] (2 bytes per element).
`int32`	Signed 32-bit integers in the range [−2147483648, 2147483647] (4 bytes per element).
`char`	Characters (2 bytes per element).
`logical`	Values are 0 or 1 (1 byte per element).

3.3. Vectores y matrices estándar

- `zeros(M,N)`: genera una matriz $M \times N$ de ceros de tipo `double`.
- `ones(M,N)`: genera una matriz $M \times N$ de unos tipo `double`.
- `true(M,N)`: genera una matriz $M \times N$ lógica de unos.
- `false(M,N)`: genera una matriz $M \times N$ lógica de ceros.
- `magic(M)`: genera una matriz $M \times M$ como un cuadrado mágico. En un cuadrado mágico la suma de cualquier fila, columna y diagonal principal vale lo mismo.
- `rand(M,N)`: genera una matriz $M \times N$ con elementos aleatorios tipo `double` dentro del rango $[0, 1]$, con distribución uniforme.
- `randn(M,N)`: idem caso anterior, pero con distribución normal.

3.4. Operadores y expresiones lógicas

Para operar entre vectores y matrices, los operadores usuales (`+`, `-`, `*`, `/`) se encuentran definidos de la forma esperable en términos de álgebra lineal. Sin embargo, para el caso del operador `*` se debe tener mayor cuidado. Al multiplicar con `*` dos matrices (o un vector y una matriz, o cualquiera de estos dos con un escalar) se estará efectuando un producto matricial, por lo que aplican las reglas algebráicas correspondientes. Sin embargo, si se quiere hacer un producto elemento-a-elemento, es necesario utilizar el operador `.*`.

```
>> A = [1 2 3; 4 5 6];
>> v = [0 1 0];
>> A * v'
ans=
2
5
>> w = [3 3 3];
>> v * w'
3
>> v .* w
0 3 0
```

En cuanto a los operadores relacionales (es decir, los que permiten comparar dos variables), se encuentran definidos `<`, `>`, `==`, `!=` y las combinaciones de los mismos. Al usar estos operadores, lo que se obtiene es en realidad una matriz lógica, indicando con 1 o con 0 si la comparación fue verdadera o falsa para cada elemento.

Operator	Name
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Equal to
~=	Not equal to

Si lo que uno quiere es comparar dos elementos y obtener un valor lógico único (indicando si las dos variables son iguales o no) se debe usar el comando `isequal(A,B)`.

4. Control de flujo

Si bien la mayoría de las operaciones en Matlab pueden realizarse mediante comandos que operen directamente sobre las matrices sin necesidad de iterar sus elementos, hay situaciones en donde algoritmos complejos requerirán inevitablemente un control de flujo concreto.

4.1. Bucle: `for`

El bucle `for` sirve para hacer una tarea un número determinado de veces, siguiendo la siguiente sintaxis:

```
for indice = comienzo:incremento:final
<Cuerpo del ciclo>
end
```

Lo que en realidad se está haciendo en este ciclo es, primero, construir un array que va de `comienzo` a `final`, con un paso de `incremento` y, segundo, ejecutar el cuerpo del ciclo para cada elemento del mismo asignando este último a `indice` cada vez. Por ello, no es necesario seguir exactamente la sintaxis anterior, sino que se puede generar un array arbitrario de la forma que sea adecuada para el lado derecho del `=`.

Un ejemplo simple de este tipo de bucle es el siguiente, en el cual se multiplica 3 veces una matriz por si misma:

```
% Ejemplo bucle for
A = [1 1 1; 1 1 1; 1 1 1]
for q = 0:1:3
  A = A * A
end
```

4.2. Condicionales: `if - elseif - else`

Los condicionales nos permiten evaluar diversas expresiones según se cumplan o no ciertas condiciones. Su sintaxis es la siguiente:

```
if expresion1
  <sentencias1>
elseif expresion2
  <sentencias2>
else
  <sentencias3>
end
```

Su funcionamiento es el siguiente: si se cumple la primera expresión condicional, se ejecutan solamente las `sentencias1` y se continua con el programa a partir de la línea `end`. Si la condición no se cumple, se chequea la expresión del `elseif` y, si ésta sí se cumple, se ejecuta el código `sentencias2` y se continua con el programa a partir de la línea `end`. Si no se cumple ni la condición del `if` y tampoco la de ningún otro `elseif` se ejecutará el código `sentencias3`. Se pueden tener varios `elseif` o ninguno, y se puede tener o no un `else`. Un ejemplo sencillo (y solo con fines didácticos) es el siguiente:

```
for q = 0:1:4
  if q == 0
    disp('q = 0')
  elseif q == 1
    disp('q = 1')
  elseif q == 2
    disp('q = 2')
  else
```

```
    disp('q distinto de 0, 1 \'o 2')
  end
end
```

La función `disp` permite imprimir un texto arbitrario en pantalla.

4.3. Bucle: `while`

Es un bucle que ejecuta una serie de sentencias mientras la condición que controla el bucle se siga cumpliendo. Su sintaxis es la siguiente:

```
while expresion
  <sentencias>
end
```

Un ejemplo del funcionamiento del bucle `while` se puede ver a continuación (para entender el funcionamiento de dicho ejemplo es necesario saber que matlab considera todo número, cuando lo evalúa lógicamente, como verdadero, a no ser que sea 0).

```
a = 10;
b = 5;
while a
  a = a - 1;
  disp('bucle a')
  while b
    b = b - 1;
    disp('bucle b')
  end
end
```

4.4. Condicional: `switch`

Se utiliza para controlar la ejecución de un programa dependiendo de diversos casos. Su sintaxis es la siguiente:

```
switch expresion
case valor
  <sentencias>
```

```
case valor
  <sentencias>
otherwise
  <sentencias>
end
```

Matlab irá evaluando si el valor de la expresión utilizada equivale a alguno de los valores de los `case`, yendo en orden. Si esto sucede con alguno, se ejecutarán las sentencias correspondientes y se continuará desde el `end`. Si ningún case satisface la expresión, se ejecutarán las sentencias del caso `otherwise`. Sin embargo, no es necesario incluir éste, en cuyo caso no se ejecutará ninguna sentencia del `switch`. Un ejemplo:

```
for q = 0:4
  switch q
  case 0
    disp('q = 0')
  case 1
    disp('q = 1')
  case 2
    disp('q = 2')
  case 3
    disp('q = 3')
  otherwise
    disp('q distinto de 0, 1, 2 o 3')
  end
end
```

4.5. Pensando en forma matricial

Como se mencionó previamente, la idea general de Matlab es utilizar comandos que operen directamente sobre matrices en vez de iterar mediante bucles los elementos de las mismas.

Para ciertas tareas que uno quiera realizar, a veces es necesario encadenar distintos comandos que operen entre matrices de forma de lograr la expresión final. Para descubrir estos comandos se recomienda utilizar la ayuda del Matlab (menu *Ayuda*).

Un ejemplo de un comando que permite lograr una tarea relativamente compleja es el de `repmat`. Este permite crear una matriz a partir de repetir

otra una cantidad de veces determinada en cada dimensión. Si uno, por ejemplo, quisiera generar una matriz en donde cada fila sea igual a un vector dado v (donde v puede ser resultado de una expresión compleja), puede hacerse lo siguiente:

```
>> v = 1:3:9;
>> repmat(v, [4 1])
ans=
1 4 7
1 4 7
1 4 7
1 4 7
```

Cabe aclarar que, para este ejemplo simple, se podría haber obtenido un resultado idéntico utilizando el hecho de que

$$\begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \begin{pmatrix} 1 & 4 & 7 \end{pmatrix} = \begin{pmatrix} 1 & 4 & 7 \\ 1 & 4 & 7 \\ 1 & 4 & 7 \\ 1 & 4 & 7 \end{pmatrix}$$

En Matlab, esto se puede obtener de la siguiente forma:

```
>> ones(4, 1) * v
ans=
1 4 7
1 4 7
1 4 7
1 4 7
```

5. Escribiendo archivos .m

En vez de estar constantemente trabajando en la consola, evaluando comandos de a uno por vez (lo cual puede servir para experimentar y hacer pruebas rápidas), para algoritmos minimamente complejos ya es necesario hacer un script o programa en un archivo aparte. Matlab interpreta y ejecuta archivos con extensión `.m`. Dentro del entorno mismo, se incluye un editor con ciertas facilidades para simplificar el desarrollo de scripts. Para invocar

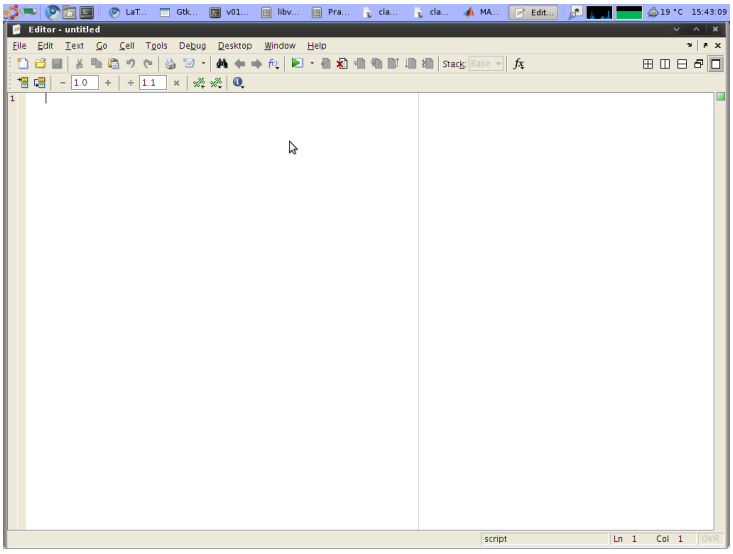

Figura 2: Editor de Matlab

el editor usaremos el comando `edit` en la consola de comandos de Matlab. Esto genera una ventana como la que se muestra en la Figura 5.

Un ejemplo de un script de Matlab es el siguiente:

```
% Practica 1, entendiendonos con el entorno.
% Primero se lee una imagen.
f = imread('lena.tif');
% Almacenamos el tamaño de la imagen f.
% M contiene la informacion del numero de filas.
% N contiene la informacion del numero de columnas de la imagen.
% c contiene la informacion del numero de colores de la imagen.
[M, N, c] = size(f);
```

Para mostrar información mas detallada de la imagen Notar que no se pone; por que sino no mostraria nada.

```
>> whos f
```

El comando `whos f` permite obtener información sobre una variable. En este caso permite obtener datos sobre la imágen que se leyó.

Ejemplo de función:

```
function [y] = promedio(x)
%encabezado
%function palabra reservada
%[y] salida
%promedio(x) el nombre de la funci n
  if ~isvector(x)
  %svector(A) devuelve 1 l gico (true) si size(A)=[1 n]
sino el 0 l gico (false)
        error('Input must be a vector')
  end
  y = sum(x)/length(x);
end
```

Al ejecutar la función en la línea de comandos.

```
>>z = 1:99;
>>promedio(z) %se llama a la función
ans =
    50
```

Todo texto a continuación de un `%` será considerado comentario en script o en función.

Para ejecutar un script simplemente nos debemos colocar en el directorio donde se encuentra el programa y escribir el nombre del archivo *.m* en la consola de comandos (pero no incluyendo dicha extensión).

6. Manejo de imágenes

Matlab permite un manejo muy simple de archivos de imagen, sin importar el formato en el que estén guardadas. El cuadro siguiente muestra los formatos aceptados por Matlab, para las .raw se utiliza fopen y fwrite, como en lenguaje c.

Format Name	Description	Recognized Extensions
TIFF	Tagged Image File Format	`.tif, .tiff`
JPEG	Joint Photographic Experts Group	`.jpg, .jpeg`
GIF	Graphics Interchange Format†	`.gif`
BMP	Windows Bitmap	`.bmp`
PNG	Portable Network Graphics	`.png`
XWD	X Window Dump	`.xwd`

† `GIF` is supported by `imread`, but not by `imwrite`.

Raw .raw

El comando `imread(archivo)` permite cargar una imagen y obtener una matriz de $M \times N \times K$ (donde M es el alto de la imagen, N el ancho y K la cantidad de bandas). Si la imagen es de escala de grises (un solo canal o sea K=1). Si la imagen es color R G B, entonces K=3.

```
>> A = imread('1.png');
>> size(A)
ans=
480 640
```

Ahora A es una matriz, se puede mostrar con el comando `imshow(A)` en una figura de Matlab, si A es `uint8` sino poner `imshow(uint8(A))` o `imshow(A,[])` o `imshow(A,[0 255])`, `[0 255]` es el *rango dinánico*. Dado que dentro de Matlab cualquier imagen es interpretada como una matriz, el comando `imshow` en realidad puede recibir una matriz generada dentro de Matlab mismo (que será interpretada como una imagen).
El comando `subplot(filas,columnas,n)` muestra `n` imágenes, donde $1 \leq n \leq (filas * columnas)$.

Finalmente, si se quiere guardar una matriz a un archivo de imagen, se puede utilizar el comando `imwrite(matriz,archivo)`.

```
>> imwrite(A, '1.png');
```

Para más información sobre cómo hacer un manejo más avanzado de imágenes, consultar el *help* de Matlab.

```
>> help funcion
```