Matlab

Salvador Ramírez
http://www.profc.udec.cl/~sram

5 de marzo de 2002

Índice general

Capítulo 1

Introducción

Matlab es un software para llevar a cabo computaciones numéricas casi de todo tipo, pudiendo manipular vectores y matrices tanto reales como complejos con funciones y fórmulas de variadas ramas de la matemática. Matlab se compone de un programa básico y un conjunto de toolbox para labores más especializadas. Con Matlab básico es posible llevar a cabo cualquier operación aritmética tanto con escalares como vectores y matrices, solucionar sistemas de ecuaciones lineales, manipular imágenes, crear interfaces gráficas, etc. Entre los toolbox más importantes se encuentran:

- **Curve fitting:** ajustes de modelos y análisis.
- **Data Acquisition:** adquiere y envía datos a un instrumento electrónico conectado al computador. (sólo para Windows)
- **Excel link:** permite usar Matlab con datos leídos directamente desde planillas Excel.
- **Image processing:** permite el procesamiento de imágenes, análisis y desarrollo de algoritmos.
- **Partial differential equation:** soluciona y analiza sistema de ecuaciones diferenciales parciales.
- **Signal Processing:** permite el procesamiento de señales, análisis y desarrollo de algoritmos.
- **Spline:** crea y manipula modelos de aproximación de datos Spline.
- **Statistics:** permite aplicar modelos estadísticos y modelos de probabilidades.
- **Structural Dynamics:** analiza modelos de elementos finitos y lleva a cabo análisis modales de sistemas mecánicos.
- **Wavelet:** analiza, comprime y saca el ruido de señales e imágenes usando técnicas de wavelet.

Matlab, como ya se ha dicho, esta orientado al cálculo numérico, a diferencia de otros software como Mapple y Mathematica que están orientados al cálculo simbólico.

Otra característica importante de Matlab es que es orientado al arreglo (vectores y matrices), es decir, las operaciones o funciones matemáticas que son válida para números escalares también lo son para arreglos. Si por ejemplo V es un vector de 5 elementos, entonces cos(V) entregará un vector de 5 elementos con los valores de coseno para cada elemento del vector original.

En las siguientes secciones se dará una introducción a los tópicos de matlab más usados. Para profundizar más en cualquier comando que aquí se muestre se puede ejecutar “help <comando>” en el ambiente matlab.

Capítulo 2

Cálculos simples y gráficos

En esta sección se dará una breve introducción a algunos conceptos básicos de matlab, como son la definición de vectores y matrices (**arreglos**[1]), algunos operaciones entre ellos y la creación de gráficos simples. Nota: Matlab funciona a través de comandos (línea de texto indica alguna orden) que se escriben en la llamada *prompt* de Matlab (>>). Cada comando despliega una salida con la respuesta del comando. Si se quiere evitar esta salida se puede escribir el comando con un signo de punto y coma al final de la línea.

2.1. Vectores, Matrices, números complejos

En Matlab todos las variables son arreglos. Incluso un valor escalar es un arreglo de 1x1 dimensión. Ejemplos:

Definir una variable simple con un valor:

```
>>a = 2
a =
  2
```

Para definir arreglos (vectores y matrices) se usa los paréntesis corchete para especificar los valores dentro y el punto y coma para separar las filas del arreglo, un espacio además separa las columnas. Por ejemplo:

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

[1] Arreglo es la palabra que se usará en adelante para referirse a cualquier composición de números, es decir un vector, una matriz o composiciones de números de mayor dimensión.

```
4   5   6
7   8   9
```

Un comando útil para ver todas las variables definidas es **whos,** que despliega línea por línea las variables con tamaños, dimensiones y tipo de datos.

Las operaciones básicas de suma y multiplicación sobre arreglos se usan de la siguiente forma:

```
>>A = [1 2 3;4 5 6;7 8 9];
>>B = [1 1 1;2 2 2;3 3 3];
>>C = [1 2;3 4;5 6];
>>A + B
ans =
  2    3    4
  6    7    8
 10   11   12
```

Notar por ejemplo que A + C no tiene sentido por que las dimensiones de las matrices no coinciden, Matlab desplegaría un error. La multiplicación de matrices también es posible:

```
>>A * C
ans =
 22   28
 49   64
 76   100
```

Para obtener la transpuesta conjugada (o transpuesta simplemente para matrices con valores reales) de una matriz se usa el operador ':

```
>>A'
ans =
     1   4   7
     2   5   8
     3   6   9
```

2.1.1. Operaciones para arreglos y para los elementos del arreglo

Existen dos formas de especificar operaciones sobre arreglos en Matlab. Por ejemplo si se quiere multiplicar una matriz por sí misma, se puede usar el operador de potencia al cuadrado de la siguiente forma:

```
>>A ^2
```

Sin embargo si se desea elevar cada uno de los miembros de la matriz al cuadrado (operacion por supuesto distinta de elevar el arreglo como matriz al cuadrado), esto

debe expresarse asi:

>>A .^2

En general todas las operaciones, cuando se les antepone un punto, operan elemento a elemento en el arreglo.

2.1.2. Creación de Arreglos

En Matlab existen muchas formas de crear arreglos. Una forma útil es la de crear los elementos del arreglo con una sucesión de números. Por ejemplo:

>>t = 1:6
t =
1 2 3 4 5 6

También es posible dar la diferencia en el salto de los números de la sucesión:

>>t = 0:0.2:1
t =
0 0,2 0,4 0,6 0,8 1

Otra manera de crear arreglos es a través de las siguientes funciones:

- zeros(m, n) crea una matriz mxn de ceros
- ones(m, n) crea una matriz mxn de unos
- eye(n) crea la matriz identidad nxn

2.1.3. Direccionamiento de los elementos de un arreglo

Sea la matriz:

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{pmatrix}$$

El modo general de referenciación de los elementos de un arreglo es: A(f,c), donde f y c son los números de fila y columna respectivamente. Si uno de estos valores es un ':' entonces se asume la totalidad de los elementos en esa dimensión. En la matriz A anterior si uno imprime el valor A(3, 4) se mostrará el valor 12, pues 12 es el elemento de la fila 3 columna 4 del arreglo A.

Si se desea utilizar la primera fila completa de la matriz A entonces puede referenciarse como A(1,:). Así por ejemplo si escribimos en Matlab:

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

éste responderá imprimiendo la primera fila de A: 1 2 3 4

De la misma forma, para imprimir la columna cuarta, se hace así:

```
>>A(:, 4)
ans =
4
8
12
16
```

Notar que este último entrega los valores como una columna y no como una fila de elementos (horizontales). En muchos casos Matlab no trabaja con vectores columna sino con vectores fila y puede ser entonces necesario transformar este arreglo columna a su equivalente fila. Para esto se usa el comando **reshape**. Por ejemplo, para cambiar la columna cuarta (impresa en el ejemplo anterior) a su correspondiente vector fila (es decir un vector con elementos 4 8 12 16) se puede usar el siguiente comando:

```
>>n_vector = reshape( A(:, 4), 1, 4)
n_vector =
4 8 12 16
```

Como se puede apreciar n_vector es una nueva variable creada para almacenar la cuarta columna en forma de fila. Notar que en este caso reshape fue dado 3 argumentos: A(:, 4), 1 y 4. Esto quiere decir, que el vector columna A(:, 4) sea transformado en un vector de 1 fila y 4 columnas.

2.1.4. Números complejos

Matlab usa la letra i o j para indicar la unidad compleja ($\sqrt{-1}$).

```
>>sqrt(-1)
ans =
    0 + 1.0000i
```

Notar que en el comando anterior, el resultado no fue asignado a una variable directamente, por lo que Matlab asignará este resultado momentáneamente a la variable predefinida ans, que luego puede ser usada como cualquier variable.

```
>>ans + 1
    1 + 1.0000i
```

Como se puede ver también del ejemplo anterior, sqrt es una función predefinida de matlab, que obtiene la raíz cuadrada de un número. Así como sqrt existen un conjunto de funciones básicas también tales como: sin, cos, log, atan.

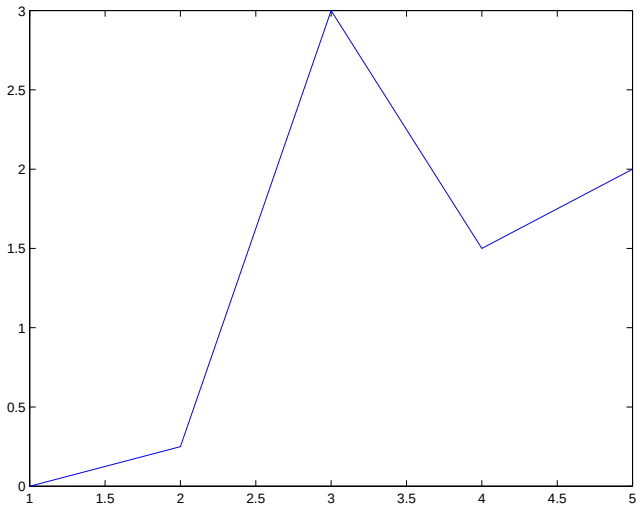


Figura 2.1: Primer plot

2.2. Gráficos

La forma de gráfico más simple en Matlab es plot, a través del comando plot. Este comando espera vectores de valores como argumentos que corresponden a los pares ordenados x,y. Ejemplo:

```
>>x = [1; 2; 3; 4; 5]
>>y = [0; .25; 3; 1.5; 2]
>>plot(x,y)
```

(ver resultado en figura 2.1)

El comando plot tiene un argumento para indicar como se desea que se unan los puntos del gráfico. Por ejemplo con plot(x,y,'o') el gráfico anterior sería dibujado con 'o' en los puntos y sin líneas uniéndolos.

Existe también una serie de comandos destinados a cambiar la apariencia del gráfico desplegado:

- xlabel('título eje x'), ylabel('título eje y'): respectivamente configuran el texto de título de tanto el eje x como el y.
- title('título'): pone un título al gráfico desplegado por plot.
- grid: pone una grilla al gráfico.

Para poner varios gráficos en una sola ventana se puede usar el comando subplot. El comando subplot(m, n, i) crea m*n gráficos y el tercer argumento es su ubicación. A continuación se muestra un ejemplo de esto:

```
>>t = (0:.1:2*pi)';
>>subplot(2,2,1)
```

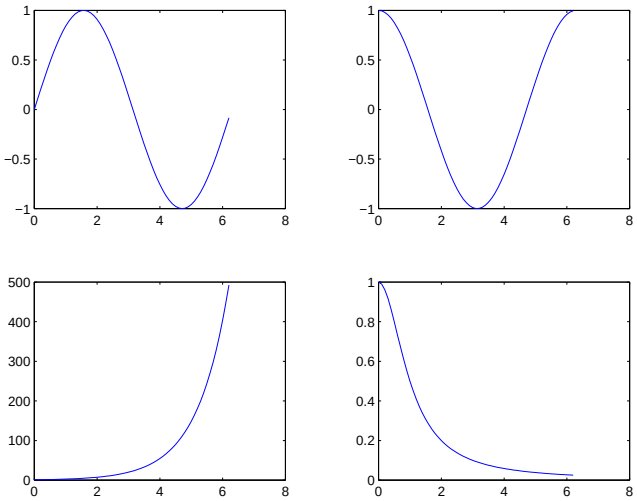


Figura 2.2: Múltiples gráficos en una ventana

```
>>plot(t,sin(t))
>>subplot(2,2,2)
>>plot(t,cos(t))
>>subplot(2,2,3)
>>plot(t,exp(t))
>>subplot(2,2,4)
>>plot(t,1./(1+t.^2))
```

el resultado de lo anterior se puede ver en la figura 2.2.

2.2.1. Impriendo gráficos

Para imprimir la figura del plot a un archivo o a una impresora directamente, se puede usar el comando print, que guarda la imagen en diferentes formatos dependiendo de los parámetros pasados a print. Los parámetros para formato de salida de print más usados en matlab son:

-dps2: Postscript nivel 2 blanco y negro

-dpsc2: Postscript nivel 2 color

-deps2: Figura en Postscript encapsulado nivel 2 blanco y negro

-depsc2: Figura en Postscript encapsulado nivel 2 color

-djpeg: Figura en JPEG calidad 75

-djpeg90: Figura en JPEG calidad 90

-dtiff: Figura en TIFF

-dpng: Figura en PNG (Portable Network Graphic) en 24-bit

La diferencia entre Postscript y Postscript encapsulado (encapsulated postscript o eps) es que Postscript es la descripción de una página y no una figura propiamente tal, como EPS. Así, cuando se quiere una figura para incluir en algún documento se debiera usar EPS, JPEG, PNG o TIFF. Si se desea imprimir la figura a una impresora Postscript entonces se debe usar el formato Postscript.

Ejemplos:

- Imprimir el gráfico actual a un archivo para su posterior impresión en una impresora postscript a color:
  print -dpsc2 migrafico.ps

  Nota: el comando anterior grabará el gráfico a un archivo llamado migrafico.ps que luego puede ser impreso con el comando Unix: "lp migrafico.ps" (si lp no funciona probar lpr).

- Imprimir el gráfico actual del plot a un archivo llamado test.jpg en formato JPEG:
  print -djpeg test.jpg

2.3. Resolución de sistemas de ecuaciones lineales

Supongamos un sistema de ecuaciones lineales como el siguiente:
12x + 7y + 4z = 3
82x + 6y = 45
3y + 97z = -10

Este sistema de ecuaciones se puede representar con matrices así: Ax = b, donde A es la matriz de coeficientes, x es el vector de soluciones (x = [x, y, z]) y b es el vector de valores del lado derecho de las ecuaciones. Es decir:

$$A = \begin{matrix} 12 & 7 & 4 \\ 82 & 6 & 0 \\ 0 & 3 & 97 \end{matrix}$$

$$b = \begin{matrix} 3 \\ 45 \\ -10 \end{matrix}$$

Estas matrices son ingresadas a Matlab así:

```
>>A = [12 7 4; 82 6 0; 0 3 97];
>>b = [3; 45; -10]
```

La solución de este sistema de ecuaciones, matricialmente, es: $x = A^{-1}b$. Sin embargo en Matlab hay una manera de resolver este sistema sin obtener la inversa de la matriz A. Esto se puede hacer de la siguiente forma:

```
>>x = A\b
x =
   0,5875
  −0,5290
  −0,0867
```

que representa la solución del sistema de ecuaciones lineales. Es decir, x = 0.5875, y = -0.5290, z = -0.0867.

2.4. Comandos útiles

A continuación se listan una serie de comandos de gran utilidad en Matlab:

- max(x), si x es un vector se retorna el máximo valor de éste.
- min(x), si x es un vector se retorna el mínimo valor de éste.
- abs(x), retorna un arreglo del mismo tamaño de x cuyos elementos son los valores absolutos de los elementos del arreglo original
- size(A), retorna un vector con el número de filas, columnas, etc del arreglo A.
- lenght(x), retorna el tamaño del arreglo.
- save varsfile, graba todas las variables definidas en un archivo llamado varsfile.mat
- load varsfile, carga todas las variables previamente guardadas en el archivo varsfile.mat
- quit, sale de Matlab

Capítulo 3

Programación en Matlab

En esta sección se revisarán los comandos de control de Matlab como lenguaje de programación. Esto es, las condiciones *if-then-else* y los ciclos *for* y *while.*

3.1. If-then-else

En Matlab este control tiene la sgte forma:

```
if expr1
   comandos_matlab
elseif expr2
   otro_comandos_matlab
...
else
   aun_otros_comandos_matlab
end
```

expr es una abreviación de expresión. Una expresión es una sentencia cuyo valor es verdadero o falso. Por ejemplo a >b , esta expresión puede ser verdadera o falsa dependiendo de los valores de a y b en un momento determinado. Ejemplo:

```
t = rand(1)
if t >0.5
   disp(“valor es mayor que 0.5”)
elseif t <0.25
   disp(“valor es menor que 0.25”)
else
   disp(“valor está entre 0.25 y 0.5”)
end
```

Nota: el comando disp simplemente despliega los argumentos pasados dentro del paréntesis, ya sea este un texto o valores directamente.

Los operadores para comparación mas usados son: <(menor que), >(mayor que), == (igual a), <= (menor o igual que), >= (mayor o igual que) y ~= (distinto a).

Los ciclos de control son usados para repetir un conjunto de comandos, por lo general usando condicionales if dentro. Como ya se ha dicho, los ciclos de control en Matlab son for y while. A continuación se presenta su forma general:

```
for variable = expresión
   comandos_matlab
end

while expresión
   comandos_matlab
end
```

Ejemplos:

- Imprimir el cuadrado de los 5 primeros numeros naturales:
  ```
  for i=[1, 2, 3, 4, 5]
  disp(i^2)
  end
  ```
  Notar que el for anterior tambien pudo ser escrito asi: “for i=1:5”

- Imprimir el valor de x, mientras este sea positivo, decrementando cada vez x en 17:
  ```
  x = 100
  while x >0
  disp(x)
  x = x - 17;
  end
  ```

- Imprimir el cuadrado de los números decimales de una cifra decimal de 0 a 1, en forma decreciente, es decir empezando por 1, 0.9, 0.8, ... 0.1, 0:
  ```
  for i=1:-0.1:0
  disp(i^2)
  end
  ```

Si mientras dentro de un ciclo o loop (for o while) se quiere terminar éste sin haberse cumplido la condición de término (argumento de while) o sin haber recorrido todos los elementos (argumento de for) se puede usar el comando break, que saldrá inmediatamente del ciclo.

3.2. Scripts y funciones

Un script es una colección de comandos Matlab escritos en un archivo-m (archivos con extensión .m) y que pueden ser ejecutados todos de una vez tan sólo escribiendo el nombre del archivo (sin la extensión .m). Esto es útil cuando se tiene un procedimiento que se debe aplicar repetitivamente y no se quiere estar escribiendo cada vez todos estos comandos, en su lugar se escriben en un archivo y luego se llama a través del nombre de este archivo.

Para poder ejecutar estos archivos-m es necesario localizar este archivo en los directorios en donde Matlab buscará por ellos. El conjunto de estos directorios se puede ver con el comando path dentro de Matlab (ejecutar "help path" para ver información de como agregar otros directorios para búsquedas).

Veamos un ejemplo: crear un archivo (con la ayuda de cualquier editor de texto, en unix vi o nedit; en windows notepad por ejemplo) llamado primerm.m con las tres siguientes líneas:

```
x = 0: 0.8 * pi/N: 2*pi;
y = tan(0.3 * x);
plot(x, y)
```

Ahora en la prompt de Matlab ejecutar:

```
>>N = 50
>>primerm
```

y el resultado es la figura 3.1.

Dentro de estos archivos-m se pueden también definir funciones. Una función es un conjunto de líneas de comandos Matlab pero que poseen un espacio de variables aparte[1], además de retornar un valor específico al final de la función. Por ejemplo para definir una función que obtenga coseno del cuadrado de los elementos de un vector se puede escribir dentro de un archivo-m (coscuad.m) las siguientes líneas:

```
function y = mi_primera_funcion(x)
    y = cos(x.^2);
```

Luego, desde la prompt de Matlab:

```
>>vec = [1, 2, 5, 7, 8]
>>res = mi_primera_funcion(vec)
```

res entonces contendrá un vector de elementos con los resultados de la función. El '.' delante de la operación '^' indica que la operación debe ser vectorizada. Esto debe

[1] Esto quiere decir que dentro de una función no serán "visibles" las variables definidas fuera de esta, por lo que si se quiere esto así sea es necesario de paserle a la función tal argumento, por ejemplo un vector V. Ver ejemplo.

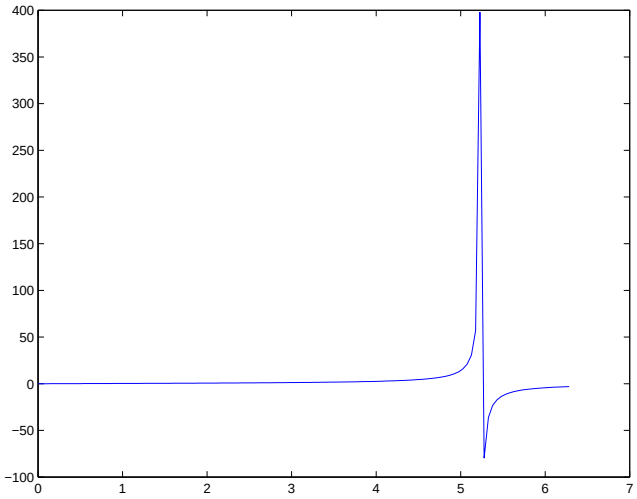


Figura 3.1: Ventana resultado de primerm.m

indicarse cuando la operación también está definida para vectores (multiplicación de dos vectores en este caso) y en lugar de multiplicar x por x como vectores, se deben elevar sus componentes al cuadrado.

Notar en la definición de la función (*function y = mi_primera_funcion(x)*) que 'y' es la variable a retornar, por lo que el valor de retorno dentro de la función debe asignarse a 'y'. La variable 'x' en cambio es el argumento de la función, es decir, es una variable que se le pasa a la función para que compute algo con ella. Esto se hace dado que de no pasar esa variable como argumento entonces dentro de la función no podría usarse (la variable x).

En el caso de querer definir una variable que pueda ser usada en todas las funciones y scripts de Matlab, se puede declarar esa variable con el **global**. Ejemplo:

- Crear una variable llamada HJ que pueda ser usada dentro de cualquier función de Matlab
  >>global HJ

3.2.1. Archivos-p

Los archivos con extensión .p o archivos-p en Matlab son archivos pre-compilados a partir de un archivo-m. Las ventajas de un archivo-p respecto de un archivo-m son:

1. Dado que los archivos-p son pre-compilados, entonces se ejecutan más rápidamente que los archivos-m

2. En un archivo-p se puede esconder el código de un algoritmo si éste se desea mantener secreto.

La desventaja de los archivos-p es que dependen de la plataforma sobre la cual fueron pre-compilados y por lo tanto no podrán ser ejecutados en una plataforma distinta.

Para construir un archivo-p a partir de un archivo-m se puede usa el comando Matlab pcode. Ejemplo:

- Crear un archivo-p a partir archivo-m llamado prueba.m y dejar ese archivo prueba.p en el directorio actual:
  >>pcode -inplace prueba.m

Capítulo 4

Entrada y Salida

En esta sección se introducirá la entrada y salida de datos tanto de teclado/pantalla como de archivos.

4.1. Entrada y salida de teclado/pantalla

Básicamente existen dos funciones Matlab para ingresar y recibir datos de teclado y pantalla. Estas funciones son input y disp. La función input despliega un mensaje, espera por el ingreso de datos desde teclado y guarda tal dato en una variable. La función disp simplemente despliega los argumentos que se le pasan. Vemos ejemplos:

- Pedir ingresar el número cualquiera y guardar ese número en la variable n:
  >>n = input('Ingrese un número: ')
- Desplegar el número ingresado anteriormente, n:
  >>disp('El número ingresado fue: '), disp(n);

4.2. Entrada y salida de archivos

En Matlab existen variadas formas de leer archivos. A continuación se presentan algunas de estas formas.

4.2.1. Uso de *load*

El comando load permite leer archivos de texto plano con datos formateados en filas y columnas. Este comando creará una matriz de datos con las filas y columnas del archivo. Esta matriz tendrá el nombre del archivo (sin extensión). Si el archivo de lectura tiene un encabezado de texto que no quiere leerse entonces debiera modificarse el archivo de texto de tal manera de que esas líneas de encabezado comiencen con el caracter ' %'. Ejemplo: Supóngase el archivo prueba.txt con el siguiente contenido:

%tiempo	*posición*
0	12
1	16
2	23

Ahora se ejecuta desde la prompt de Matlab los siguiente:

>>load prueba.txt

con lo que Matlab generará una matriz (o arreglo) llamada prueba y que tendrá 2 columnas y 3 filas, las mismas que se muestran en el archivo anterior. Notar que no se incluyen las líneas que comienzan con %. Si se quiere asignar el contenido de un archivo a una variable con nombre específico A por ejemplo, se puede usar load así:

>>A = load prueba.txt

Por último hay que notar que una matriz creada de esta manera tiene las mismas características que los arreglos vistos en secciones anteriores y por lo tanto se les pueden aplicar todas las operaciones que hasta aquí se han visto para arreglos.

4.2.2. Uso de textread

El comando textread también lee datos desde archivos de texto planos formateados en filas y columnas y guarda cada columna en una variable vector. El uso general de este comando es el siguiente:

```
[a, b, c, ... ] = textread(archivo, formato, N)
```

lo que quiere decir que las columnas del archivo quedarán en las variables a, b, c, etc respectivamente, con el formato indicado. N es la cantidad de veces que el formato y la lectura deben ser usados, esto se usa comúnmente para indicar la cantidad de filas del archivo que se quieren leer. Este paŕametro puedo no especificarse en la llamada a textread, en cuyo caso Matlab leerá todas las líneas del archivo.

Ejemplo: Leer el archivo prueba.txt de la subsección anterior (eso sí sin el encabezado, textread parece no reconocer que una línea empezada por un % no debe tomarse en cuenta, por lo que se supone ahora que prueba.txt no tiene la primera línea que se escribió anteriormente):

```
>>[tiempo, posicion] = textread('prueba.txt', '%f%f')
```

el comando anterior creará un vector tiempo y otro posicion con las columnas del archivo. El formato '%f%f' indica que ambos datos deben leerse como datos reales o de punto flotante. Otros formatos son enteros (%d), string de texto (%s), etc.

Ejemplo: (sacado de 'help textread'): Supóngase el siguiente archivo de texto llamado 'datos.txt':

Juan	*tipo*1	2,34	42	*si*
Pedro	*tipo*2	24,1	83	*no*
Jose	*tipo*2	2,1	73	*si*

estos datos pueden ser leídos de la siguiente forma en Matlab:

```
>>[nombres, tipos, x, y, respuesta] = textread('datos.txt', '%s %s %f %d %s')
```

ahora se despliegan algunas de estas variables se verá el resultado del comando textread:

```
>>disp(nombres)
Juan
Pedro
Jose
>>disp(y)
42
83
73
```

En el ejemplo anterior se leyeron los datos de 'y' como enteros (%d), sin embargo para evitar confusiones o errores es posible leer cualquier número como real (%f) dado que los reales es un conjunto que incluye a los reales, por lo que en el comando anterior se pudo reemplazar perfectamente %d por %f y obtener el mismo resultado[1].

Por último, con textread es posible evitarse leer alguna columna. Por ejemplo si no se quisiera leer la columna de tipo en el archivo anterior se puede hacer de la siguiente forma:

```
>>[nombres, x, y, respuesta] = textread('datos.txt', '%s %*s %f %f %s')
```

4.2.3. Uso de funciones f/io

He llamado funciones f/io a las siguientes funciones: fopen, fclose, fprintf, fscanf, fwrite, fread que se inspiran en funciones originales del lenguaje C. A continuación veremos una a una estas funciones a través de ejemplos.

La función fopen sirve para abrir un archivo que luego se utilizará para lectura, escritura o ambos, dependiendo de los parámetros que se pasen a esta función. La forma general de esta función es:

```
[fi, mensaje] = fopen('archivo', 'opciones')
```

aqui fi es un número identificador del archivo recién abierto y que es usado para las posteriores lecturas y/o escrituras en este archivo. La variable mensaje es un texto que en caso que fi sea negativo (problemas al abrir el archivo ya sea por que no existe o por no tener permisos para hacerlo) se llena automáticamente con un mensaje que indica el problema por el cual no pudo abrirse el archivo. El parámetro de opciones puede ser alguno de los siguientes:

'r': lectura (read)

[1] Usando más memoria eso sí.

’w’: escritura sobreescribiendo si es que el archivo existía (write)
’a’: escritura pero añadiendo a lo ya existente si es que el archivo existía (add)
’r+’: lectura y escritura

Para cerrar un archivo se usa fclose(fi). Si se desean cerrar todos los archivos abiertos se puede usar close(’all’).

Las funciones fprintf y fscanf permiten escribir y leer archivos de texto formateados. A continuación un ejemplo de escritura a un archivo con fprintf.
Primero creamos una matriz cualquiera de 20 filas y 5 columnas de valores al azar:

```
>>A = rand(20, 5)
```

Luego abrimos el archivo salida.txt para escribir esta matriz A:

```
>>fi = fopen('salida.txt, 'w')
```

Luego recorremos cada elemento del arreglo por filas, al final de imprimir cada fila se imprimirá un salto de línea, para dar la forma de matriz a la salida en el archivo:

```
>>for i:1:20
    for j:1:5
      fprintf(fi, ' %f', A(i,j))
    end
    fprintf(fi, '\n')
  end
```

Por último cerramos el archivo salida.txt:

```
>>fclose(fi)
```

Capítulo 5

Introducción al procesamiento de imágenes

En este capítulo se darán los conceptos básicos acerca de representación de imágenes digitales además de como leer, escribir y desplegar estas imágenes en Matlab.

5.1. Conceptos básicos de imágenes

Existen dos formas de representar imágenes digitalmente, formato *vectorial* y formato *raster*. El formato vectorial representa la imagen por sus formas internas, tratando de representar figuras geométricas que se acomoden a la imagen a representar. El formato raster en cambio representa la imagen a través de un conjunto rectangular (matriz o arreglo) de puntos coloreados llamados pixeles. Este formato es el más usado y de este se hablará en lo que sigue.

5.1.1. Color

Cada pixel tiene un valor[1] que corresponde a su color. La cantidad de bits (dígitos binarios 0 o 1) usados para representar dicho color se llama *profundidad de color* o *colordepth*[2] (ver figura 5.1).

El color de un pixel puede ser representado de tres maneras:

1. Color indexado (indexed-color): En este caso, el pixel posee un valor que en realidad no es un código de color, sino un índice del color real en una paleta de colores, también llamado mapa de colores o colormap. El formato GIF usa esta representación. Una paleta de colores no es más que una tabla indexada con los valores de colores a usar en la imagen. Esta paleta se incluye en la imagen. El

[1] Este valor puede ser un simple valor o un conjunto de valores dependiendo de la cantidad de canales de color. En RGB por ejemplo cada pixel tendrá 3 valores de colores, uno para Red, otro para Green y otro para Blue.

[2] Mientras más bits de colordepth mayor será el tamaño del archivo conteniendo la imagen.

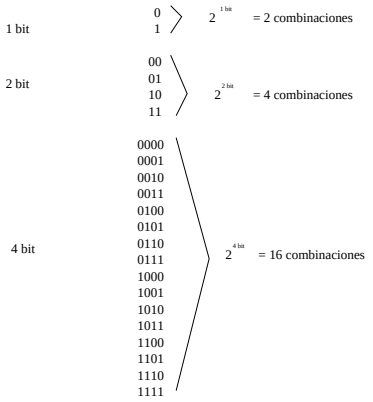


Figura 5.1: Bits y combinaciones de códigos para colores.

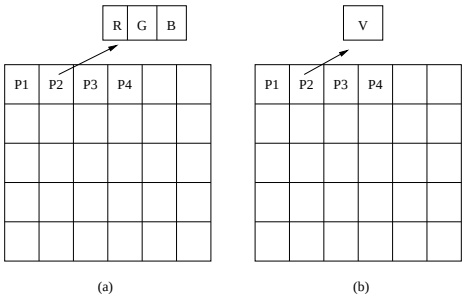


Figura 5.2: Imagen RGB (a) e imagen de escala de grises (b).

colordepth en este caso indica la cantidad máxima de colores en la paleta pero no la calidad de los colores de esta.

2. Escala de grises (grayscale): El valor del pixel en este caso es un tono de gris, donde cero indica negro y el mayor valor ($2^{\text{colordepth}}$) indica blanco.

3. Color verdadero (truecolor): El valor del pixel es representado por un conjunto de tres valores Rojo, Verde, Azul (RGB).

En la figura 5.2 (a) se representa una imagen de 6 x 5 pixeles, donde por supuesto cada celda es un pixel. En este caso cada pixel tiene un valor triple, es decir su color es representado por tres números enteros que van desde 0 hasta ($2^{\text{colordepth}}$), pudiendo así representar cualquier color con la combinación de los tres canales de colores. En la figura (b) en cambio la imagen es representada por pixeles con valores de color que son un simple entero que va desde 0 hasta ($2^{\text{colordepth}}$). En esta imagen, como el valor es único, sólo se puede representar el degradamiento de un color 0 a otro ($2^{\text{colordepth}}$).

Bits	Cantidad de Colores
1	2
4	16
8	256
16	65.536
24	16.777.216
32	4.294.967.296

Una imagen con un bit de colordepth puede solo tener dos colores, por ejemplo blanco y negro. El formato GIF representa imágenes con 8 bits de colordepth, por lo que la máxima cantidad de colores posibles presentes en una imagen GIF es 256. Por esta razón, para imágenes de alta calidad no es recomendable usar GIF. Otros formatos de imagen raster que soportan mayor colordepth son JPEG, PNG y TIFF por ejemplo.

5.1.2. Números enteros y ordenamiento de bytes

Tal como se ha dicho, una imagen digital es un conjunto de valores ordenados como una matriz rectangular. Cada uno de estos valores son números enteros que van desde 0 hasta un cierto valor que depende del colordepth ($2^{\text{colordepth}}$). Estos números enteros (simples, no largos) en un computador son representados con 2 o 4 bytes[3] y dependiendo de la plataforma computacional en uso (procesador, sistema operativo, etc) estos bytes son ordenados desde el más significativo (MSB[4]) al menos significativo (LSB[5]) o bien desde el LSB al MSB.

Supóngase un entero de 4 bytes: B0, B1, B2 y B3. Un entero con el MSB primero sería “B3 B2 B1 B0”. Un entero con el LSB primero sería “B0 B1 B2 B3”.

Es importante notar que formatos como PNG[6] usan un ordenamiento predeterminado, lo que permite que las imágenes puedan ser interpretadas en todos los sistemas (independiente de la plataforma hardware y software) de la misma forma.

5.1.3. Canal Alfa o transparencia

Algunos formatos de imagen soportan un canal de transparencia para cada pixel. Por ejemplo en imágenes RGB, cada pixel tiene un conjunto RGB de valores, y soportando un canal alfa entonces ahora cada pixel tendrá un conjunto RGBA de valores, A representando el grado de transparencia (u opacidad) del pixel. Esto es útil cuando se desean mezclar imágenes.

5.2. Lectura de imágenes en Matlab

En Matlab se soportan los sgtes formatos de imagen: JPEG, TIFF, GIF, BMP, PNG, HDF, PCX, XWD, ICO y CUR.

[3] Cada byte posee 8 bits.
[4] Most Significant Byte.
[5] Less Significant Byte.
[6] Portable Network Graphic.

La función imread en Matlab se puede usar para leer imágenes. Si la imagen es grayscale entonces imread devuelve una matriz bidimensional. Si la imagen es RGB entonces imread devuelve un arreglo tridimensional.

Ejemplo paso a paso:

Leer y Desplegar una Imagen:

Leeremos a continuación una imagen que viene con el toolbox Image Processing de Matlab. La imagen se llama pout.tiff y guardaremos esa imagen (la información de los pixeles de esa imagen en realidad) en una variable arreglo Matlab llamada I.

>>I = imread('pout.tif');

Ahora para desplegarla en pantalla se puede usar el comando imshow:

>>imshow(I)

Cabe notar que si pout.tif hubiera sido una imagen indexada (ver sección anterior) entonces la sintaxis correcta para leer esa imagen hubiera sido:

>>[I, map] = imread('pout.tif');

lo que indica que los valores de pixeles de la imagen quedarán en el arreglo I y la paleta quedará en el arreglo map.

Revisar la Imagen en Memoria:

Para ver esto se usa el comando whos, como ya se ha visto:

>>whos

Name	Size	Bytes	Class
I	291x240	69840	uint8 array

Grand total is 69840 elements using 69840 bytes.

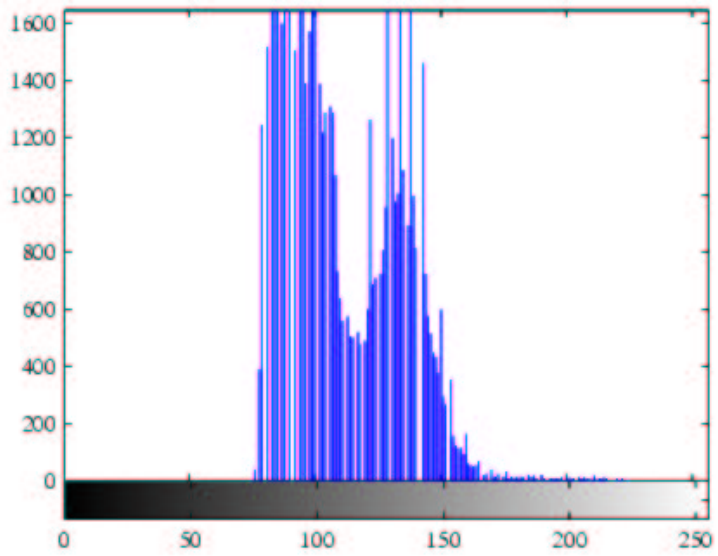
Como se puede apreciar, I es una matriz de 291 filas por 240 columnas, donde cada elemento corresponde al respectivo valor de color del pixel de la imagen. Cada elemento de la matriz además es de tipo uint8, que significa un entero sin signo de 8

bits, dado que la imagen es de 8 bits. Si la imagen hubiera sido de 16 bits entonces whos habría mostrado uint16.

Ecualizando el histograma de colores de la imagen:

Tal como se puede apreciar, la imagen desplegada tiene bajo contraste de color. Para visualizar la distribución de intensidades de la imagen se puede crear un histograma, usando el comando **imhist**. A continuación un comando para mostrar el histograma de I en una ventana nueva (en la anterior está la imagen pout.tif), con figure delante del comando se logra esto:

>>figure, imhist(I)



Este histograma nos revela que el rango de la intensidad de color usado en la imagen es un tanto angosto y no cubre todo el rango potencial de 0 a 255 (8 bits, 2^8 - 1 = 255). Esto provoca el poco contraste de la imagen.

Existe un comando dentro del Image Processing toolbox de Matlab que dispersa los valores de intensidad de una figura a todo su rango. Este comando se llama **histeq.** A continuación se crea una segunda imagen I2, con la modificación de contraste a través de histeq:

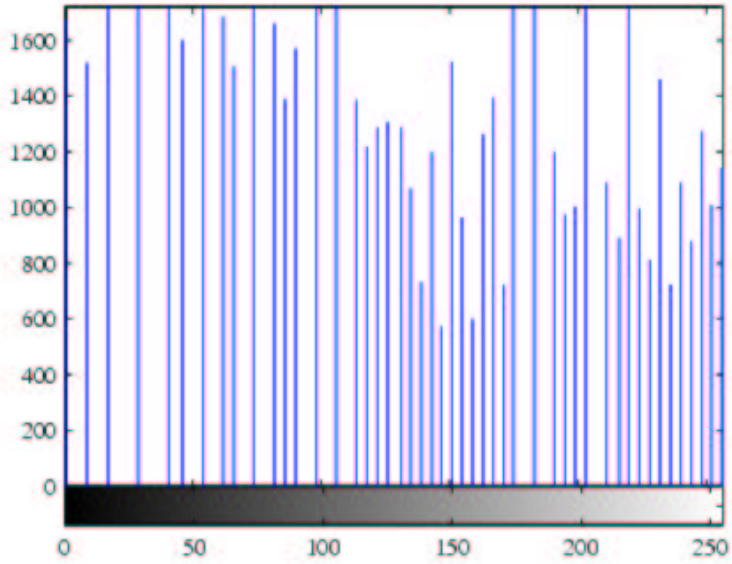
>>I2 = histeq(I);

Ahora desplegamos la imagen en una ventana nueva:

>>figure, imshow(I2)

Ahora se puede ver que el histograma de I2 es más distribuído que I:

>>figure, imhist(I2)



Escribiendo la nueva imagen

Ahora que tenemos la imagen I2 ajustada, la podemos grabar a un archivo en algún formato de imagen, por ejemplo PNG. Para esto se usa el comando **imwrite** de la sigu-

iente forma:

>>imwrite (I2, 'pout2.png');

Notar que ahora debería existir en el disco duro un archivo llamado pout2.png. Existe un comando llamado **imfinfo** que se usa para obtener información de una imagen antes de abrirla. El formato del comando es: imfinfo('archivo.png') por ejemplo.

5.3. Recomendaciones para manejo de imágenes

Como se vió en la sección anterior, imread, la función de Matlab para leer imágenes retorna un arreglo de elementos tipo uint8 o uint16. Estos tipos de datos funcionan bien con las funciones de Matlab para procesamiento de imágenes pero no funcionan para otras funciones, por ejemplo funciones matemáticas o estadísticas que se desearan aplicar al arreglo imagen. Por esta razón es recomendable trabajar las imágenes en Matlab como tipo de dato double. Para esto, se debería transformar inmediatamente los valores del arreglo después de retornados por imread de la siguiente forma:

>>I = double(imread('figura.jpg')) / 256;

Así I queda con los valores en el rango de 0 a 1 en tipo de dato double en lugar de 0 a 255 en tipo de dato uint8. Por supuesto este es el caso para imágenes de 8 bits, por la división por 256. Para imágenes de 16 bits se debería dividir por 2^{16}.