# Guión 0.4: Un muy breve Resumen de Matlab e IPT

Profesor: Rafael Molina

# Índice

# Introducción

Image Processing Toolbox (IPT) es un toolbox de MATLAB con numerosas funciones que constituye una potente base para el desarrollo de aplicaciones de procesamiento de imágenes. Este guión pretende ser un resumen de todo lo visto hasta ahora, así como una introducción a los operadores y el manejo de funciones. No es, por tanto, un guión estrictamente nuevo. Contiene algunos conceptos que ya hemos estudiado y algunos nuevos. Es importante que repases todas las secciones y prestes atención a las nuevas. Dedica algún tiempo a pensar en la optimización del código cuando programes.

# 1. Representación de imágenes digitales

## 1.1. Convención de coordenadas

Una imagen en Matlab se representa mediante una matriz de tamaño $M \times N$ y que los índices empiezan en $(1, 1)$, observa una imagen que podemos crear y visualizar

```
imagen=zeros(100,200);
imagen(1:50,:)=0.75;
```

visualízala con las funciones que conoces y sálvala en un fichero, si no conoces ninguna utiliza imshow. La figura **??** es la imagen que hemos creado

## 1.2. Imagen como matriz

El ejemplo anterior nos sirve para ilustrar donde está el origen de coordenadas. Observa también que una imagen no es más que una matriz. El ejemplo que hemos visto es de una imagen de una única banda, ya veremos imágenes RGB.

Figura 1: Imagen creada con dos regiones de niveles de gris

# 2. Lectura de imágenes

Ya sabemos como leer imágenes utilizando la función

```
f=imread('filename')
```

recuerda que tenemos que proporcionar el camino de la imagen o tener el directorio en el que está la imagen en el path.

Si usamos

```
[M, N]=size(f)
```

almacenaremos en `M` y `N` el tamaño en filas y columnas de la imagen. Podemos extraer sólo el tamaño de una dimensión utilizando, por ejemplo para extraer el número de filas,

```
M=size(f,1)
```

Podemos extraer información sobre qué es `f` utilizando

```
whos f
```

# 3. Visualización de imágenes

Ya sabemos que la orden básica es `imshow`. Sin embargo, como ya hemos visto, debes tener cuidado con los rangos, en especial si la matriz a visualizar es double. Puedes usar

```
imshow(f,[low high])
```

para definir el rango de interés, por debajo de `low` irá a negro y por encima de `high` irá a blanco. También podemos reescalar con el mínimo y máximo de nuestra matriz usando

```
imshow(f,[ ])
```

Recuerda que si queremos ver varias imágenes simultáneamente debemos usar `figure` antes de cada `imshow`.

Por último, si queremos ver el contenido de una imagen, seleccionar regiones, ajustar el contrate, etc usaremos la función `imtool`.

# 4. Escritura de imágenes

La forma usual de salvar una imagen es utilizar

```
imwrite(f,'nombre_fichero')
```

Para las imágenes que se salvan en formato JPEG existe la opción de indicar la calidad (cantidad) de compresión

```
imwrite(f,'nombre_fichero','quality',q)
```

donde q es un entero entre 0 y 100 (más bajo, mayor compresión).

Para obtener información sobre un fichero de imágenes usaremos

```
imfinfo('nombre_fichero')
```

Esta información es almacenable en una variable K utilizando

```
K=imfinfo('nombre_fichero')
```

La variable K es una variable de tipo estructura cuyos campos son accesibles. Veamos como podemos calcular la razón de compresión de una imagen usando esos campos

```
>> im=imread('cameraman.tif');
>> imwrite(im,'cameraman25.jpg','quality',25);
>> K=imfinfo('cameraman25.jpg');
>> bytes_imagencomp=K.FileSize;
>> bytes_imagen=K.Width*K.Height*K.BitDepth/8
>> comp_ratio=bytes_imagen/bytes_imagencomp

comp_ratio =

   13.9587
```

Consulta las opciones de salvado de imágenes en formato `tif`.

A veces necesitarás exportar imágenes y dibujos en la forma en la que aparecen en el ambiente de MATLAB. para ello, en la ventana de la figura debes usar en el menú **File**, la opción **Save As**. Tendrás más control sobre la figura que salvas si utilizas

```
print -fnum_figura -dformato_fichero -rresol_ppp nombre_de_fichero
```

consulta la ayuda de MATLAB para conocer las diferentes opciones.

# 5. Clases de datos

Como sabemos, las clases de datos más importantes que soporta MATLAB son `double, single, uint8, uint16, uint32, int8, int16, int32, char, logical`. También soporta `uint64` y `int64` pero estas dos clases no son soportadas por los toolboxes.

Como sabemos el IPT también soporta cuatro tipos de imágenes fundamentalmente: `niveles de gris, binarias, indexadas, RGB`.

El IPT tiene, como sabemos, funciones para convertir una imagen de un tipo a otro. Es bueno comprender bien como lo hace. Veamos algunos ejemplos adicionales a los que hemos visto

```
>> f=[-0.5 0.5; 0.75 1.5];
>> g=im2uint8(f)

g =

    0  128
  191  255
```

valores menores que cero se ponen a cero, mayores de uno a 255 y los restantes se multiplican por 255.

La función `im2double` convierte la entrada en clase `double`. Para variables del tipo `uint8, uint16` o `logical` los lleva al rango [0,1]. Con un ejemplo

```
>>A=uint8([2 100; 240 150]);
>> im2double(A)

ans =

    0.0078    0.3922
    0.9412    0.5882
```

Sin embargo, observa

```
>> A=double([2 100; 240 150]);
>> im2double(A)

ans =

     2   100
   240   150
```

lo que ocurre también para variables tipo `single`.

```
>> A=single([2 100; 240 150]);
>> im2double(A)

ans =

     2   100
   240   150
```

y también ocurre si estamos en el rango [0,1].

```
>> A=double([0.2 0.100; 0.240 0.150]);
>> im2double(A)

ans =

    0.2000    0.1000
    0.2400    0.1500
```

La función `mat2gray` nos llevará a una imagen en [0,1] mediante escalado. Su sintaxis es

```
g=mat2gray(A,[Amin, Amax]);
```

Si no especificamos el rango toma el mínimo y el máximo de la imagen.

Por último la conversión a variable lógica se hace usando

```
g=im2bw(f,T)
```

`T` es un umbral en [0,1] que se reescala teniendo en cuenta el tipo de matriz. Con un ejemplo

```
>> A=uint8([2 170; 140 200]);
>> B=im2bw(A,0.5)

B =

     0     1
     1     1
```

Los ejemplos siguientes aclaran las conversiones entre tipos de imágenes y a tipos de datos

```
>> f=[1 2; 3 4]

f =

     1     2
     3     4
```

queremos que el 0 y el 1 se conviertan en cero y en 3 y 4 en 1 usando la coversión entre imágenes. La convertimos al rango [0,1]

```
>> g=mat2gray(f)

g =

         0    0.3333
    0.6667    1.0000
```

Ahora umbralizamos en 0.6

```
>> gb=im2bw(g,0.6)

gb =

     0     0
     1     1
```

Obviamente hubiésemos conseguido lo mismo con

```
gb= f> 2
```

Si queremos convertir esta imagen a una de tipo double sólo tenemos que escribir

```
>> gbd=im2double(gb)

gbd =

     0     0
     1     1
```

Aunque tienes que tener cuidado ya que

```
>> gbd=im2double(uint8(gb))
```

```
gbd =

         0         0
    0.0039    0.0039
```

En muchas aplicaciones no es necesario trabajar con formato double y single es más que suficiente. Ten cuidado con el espacio que ocupas cuando creas una imagen.

# 6. Acceso a arrays

## 6.1. Vectores

Como ya sabemos, un vector tiene la forma

```
v=[1 3 5 7 9]
```

y podemos acceder a una posición dada usando por ejemplo

```
v(3)

ans =

     5
```

Es, como ya sabemos, muy fácil convertirlo en un vector columna y acceder a varias componentes utilizando los dos puntos, por ejemplo

```
 v(1:3)

ans =

     1     3     5
```

Fíjate en algunos ejemplos adicionales

```
>> v(3:end)

ans =

     5     7     9

>> v([1 3 5])
```

```
ans =

     1     5     9
```

Prueba ahora algún ejemplo con la función

```
x=linspace(a,b,n)
```

que genera n valores entre a y b igualmente espaciados (conteniendo a a y b).

## 6.2. Matrices

Veamos ahora algunos ejemplos para matrices

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

A =

     1     2     3
     4     5     6
     7     8     9

>> A(2,3)

ans =

     6

>> T2=A([1 2], [1 2 3])

T2 =

     1     2     3
     4     5     6
```

Observa que podemos cambiar el orden

```
>> A([1 3], [3 2])

ans =
```

```
     3     2
     9     8
```

Dos ejemplos más

```
R2=A(2,:)

R2 =

     4     5     6

>> A(2:end,end:-2:1)

ans =

     6     4
     9     7
```

## 6.3. Indexado con :

Observa los siguientes ejemplos

```
col_sums=sum(A)

col_sums =

    12    15    18

>> total_sum=sum(col_sums)

total_sum =

    45

>> total_sum2=sum(A(:))

total_sum2 =

    45
```

## 6.4. Indexado lógico

Observa el ejemplo siguiente

```
>> D=logical([1 0 0; 0 0 1; 0 1 0])

D =

     1     0     0
     0     0     1
     0     1     0

>> A(D)

ans =

     1
     8
     6
```

Como ves `A(D)` devuelve un vector columna con los valores en las posiciones en los que la variable lógica es 1, recorriendo la matriz por columnas. Observa ahora

```
>> A(D)=[10 20 30]

A =

    10     2     3
     4     5    30
     7    20     9
```

y también podemos ejecutar

```
>> A(D)=-20

A =

   -20     2     3
     4     5   -20
     7   -20     9
```

## 6.5. Matrices Sparse

Para matrices con un número pequeño de valores no nulos podemos utilizar su representación sparse. Con un ejemplo

```
>> A=[1 0 0; 0 3 4; 0 2 0]

A =

     1     0     0
     0     3     4
     0     2     0

>> S=sparse(A)

S =

   (1,1)        1
   (2,2)        3
   (3,2)        2
   (2,3)        4
```

Podemos volver al formato original utilizando

```
>> Original=full(S)

Original =

     1     0     0
     0     3     4
     0     2     0
```

aunque es, normalmente, más cómodo utilizar los índices de las filas r, los índices de las columnas c, los valores en cada par de valores y la dimensión de la matriz mxn en la sintaxis

```
S=sparse(r,c,values, m,n)
```

con un ejemplo

```
>> S=sparse([1 2 3 2], [1 2 2 3], [ 1 2 3 4], 4,4)

S =

   (1,1)        1
   (2,2)        2
   (3,2)        3
   (2,3)        4
```

```
>> Sfull=full(S)

Sfull =

     1     0     0     0
     0     2     4     0
     0     3     0     0
     0     0     0     0
```

# 7. Algunos arrays standards

Ya hemos visto, y no insistiremos en ello, algunos arrays standards: `zeros(M,N)`, `ones(M,N)`, `true(M,N)`, `false(M,N)`, `magic(M,N)`, `eye(M,N)`, `rand(M,N)`, `randn(M,N)`

# 8. Introducción a la programación de M-funciones

## 8.1. M-ficheros

Ya conocemos los conceptos básicos de los M-ficheros y como obtener ayuda sobre ellos.

## 8.2. Operadores

### 8.2.1. Operadores Aritméticos, Relacionales, Lógicos y algunas funciones

Hemos visto los operadores aritméticos, relacionales y lógicos básicos entre arrays. Algunas funciones interesantes las vemos con unos ejemplos

```
>> A=[1 2 0; 0 4 5]

A =

     1     2     0
     0     4     5

>> B=[1 -2 3; 0 3 1]

B =
```

```
     1    -2     3
     0     3     1

>> xor(A,B)

ans =

     0     0     1
     0     0     0

>> all(A) %opera en cada columna

ans =

     0     1     0

>>  any(A) %opera en cada columna
ans =

     1     1     1
```

Algunas funciones lógicas interesantes son: `iscell(C)`, `iscellstr(s)`, `ischar(s)`, `isempty(A)`, `isequal(A,B)`, `isfield(S,'name')`, `isfinite(A)`, `isinf(A)`, `isinteger(A)`, `isletter(A)`, `islogical(A)`, `ismember(A,B)`, `isnan(A)`, `isnumeric(A)`, `isprime(A)`, `isreal(A)`, `isscalar(A)`, `isspace(A)`, `issperase(A)`, `isstruct(S)`, `isvector(A)`.

# 9. Control de Flujo

Matlab proporciona los ocho modelos de control de flujo que discutieremos a continuación. Recuerda que un 1 lógico o cualquier valor no nulo es tratado como verdadero en MATLAB.

## 9.1. if, else y elseif

Tienen lo siguientes formatos

```
if expresión
  órdenes
end
```

Ya sabemos el funcionamiento, MATLAB evalua la expresión y si es verdadera ejecuta las órdenes.

También podemos utilizar

```
if expresión1
  órdenes1
elseif expresion2
  órdenes2
else
  órdenes3
end
```

## 9.2. for

El formato es bastante obvio

```
for indice=comienzo:incremento:final
  órdenes
end
```

diferentes for pueden incluirse unos dentro de otros.

## 9.3. while

Su formato es

```
while expresion
  órdenes
end
```

También pueden incluirse unos dentro de otros.

## 9.4. break

Como indica el nombre `break` termina la ejecución de un `for` o `while` e indica la salida del ciclo en el que está.

### continue

Pasa el control a la siguiente iteración del `for` o `while`. En bucles anidados pasa el contro a la siguiente iteración del bloque más interno en el que se encuentra.

## 9.5. switch-case

Su formato es

```
switch expresion
  case comp_expresion1
    ordenes1
  case {comp_expresion2,comp_expresion3} % varios valores incluidos en {}
    ordenes2
  otherwise
    ordenes3
end
```

## 9.6. Bloque Try-Catch

Podemos realizar un conjunto de órdenes, si no se produce un error continuamos y si se produce mandamos un mensaje de error. Con dos ejemplos

```
x=ones(4,2);
y=4*eye(2);
try
    z=x*y;
catch
    z=nan;
    disp('error de tamaños')
end
z

z =

     4     4
     4     4
     4     4
     4     4


%Ahora con un error

x=ones(4,2);
y=4*ones(3);
try
    z=x*y;
```

```
catch
    z=nan;
    disp('error de tamaños')
end
z
error de tamaños

z =

   NaN
```

# 10. Manejador de funciones

En problemas será necesario pasar un manejador de funciones como un argumento a otras funciones. Veamos dos formas de hacerlo. La primera es crear un manejador de función con nombre. Para ello despues de @ escribimos el nombre de la función. Con un ejemplo

```
 f=@sin

f =

    @sin

 f(pi/4)

ans =

    0.7071

 sin(pi/4)

ans =

    0.7071
```

El segundo es un manejador anónimo. Está formado por una expresión en MATLAB en lugar del nombre de una función. Su formato es

```
@(lista-argumentos-entrada) expresion
```

Veamos algunos ejemplos

```
 g=@(x) x.^2;
g(2)

ans =

     4
```

Otro ejemplo

```
w=@(x,y) sqrt(x.^2+y.^2);
 w(1,1)

ans =

    1.4142
```

Un ejemplo adicional interesante. La función `quad` realiza integración numerica sobre la función que se proporcione. Si queremos calcular la integral entre 0 y $\pi/4$ del seno podemos utilizar

```
quad(@sin,0,pi/4)

ans =

    0.2929
```

o con la definición de `f` que teníamos

```
quad(f,0,pi/4)

ans =

    0.2929
```

# 11. Optimización de código

En este breve tutorial no vamos a abordar la optimización de código en programación en Matlab. No obstante, es conveniente que visites la sección *Techniques for Improving Peformance* en el menú de ayuda y entiendas los conceptos de preasignación de espacio, vectorización, etc.

# 12. I/O interactivo

La función `disp` se usa para mostrar información en pantalla. Con algunos ejemplos

```
A=[1 2; 3 4];
disp(A)
     1     2
     3     4

sc='Procesamiento Digital';
disp(sc)
Procesamiento Digital
```

Para la entrada de datos podemos usar la función `input` Su formato es

```
t=input('mensaje')
```

veamos su uso con varios ejemplos

```
>> t=input('introduce el dato: ')
introduce el dato: 20

t =

    20

>> class(t)

ans =

double

>> t=input('introduce el dato: ')
introduce el dato: 'abc'

t =

abc

>> class(t)

ans =
```

```
char

>> t=input('introduce el dato: ')
introduce el dato: [1 2; 3 4]

t =

     1     2
     3     4

>> size(t)

ans =

     2     2
```

Para leer las entradas de una hilera de caracteres usaremos la función `strread`, utliza `help strread` para entender su funcionamiento. Si necesitas comparar hileras de caracteres utiliza `strcmp`.

# 13. Arrays de celdas y estructuras

No vamos a insistir en esta revisión sobre los conceptos que ya hemos visto en otros guiones. Revisa como se crean y manejan los arrays de celdas y estructuras.