

Procesamiento Digital de Señales  
**Laboratorio 01: Introducción a MATLAB**  
Código: DSP2022II\_LAB01

Name: \_\_\_\_\_

Profesor: Marco Teran  
Deadline: 8 de abril

**Abstract**

En el presente laboratorio se se presentan los fundamentos del uso de MATLAB, así como una serie de ejercicios para graficar señales y realizar operaciones matemáticas básicas. Se implementarán algunas herramientas necesarias para generar señales directamente en MATLAB. Se buscará apropiarse del entorno de trabajo de MATLAB. En el presente laboratorio se realizará código de MATLAB que permita la generación, tratamiento y almacenamiento de imágenes, audio y vídeo. En el presente laboratorio se implementarán algunas herramientas necesarias para el procesamiento de archivos *multimedia* en MATLAB. Se realizarán códigos en MATLAB para la generación, transformación y almacenamiento de archivos de audio e imágenes. Se abrirán en MATLAB archivos de video y se obtendrá información de estos.

## 1 Introducción a MATLAB

### 1.1 MATLAB

MATLAB es un programa de cómputo basado en operaciones matriciales y orientado al cálculo numérico por computadora. Este programa resulta muy útil para ingenieros y para el área científica ya que puede resolver problemas numéricos de una manera más sencilla que utilizando lenguajes de programación de alto nivel.

MATLAB es un sistema interactivo cuyo elemento básico son las matrices y no requiere dimensionamiento. El nombre proviene de "laboratorio de matrices". Sobre la base de MATLAB se han construido conjuntos de funciones específicas para diferentes problemas y campos de la ingeniería y ciencias, denominadas *toolboxes*.

También hace referencia al lenguaje de alto nivel para computación e ingeniería utilizado por este programa. Integra computación, visualización y programación.

Aplicaciones típicas de MATLAB son:

- Matemáticas y computación
- Desarrollo de algoritmos
- Modelado, simulación y prototipado
- Análisis de datos, exploración y visualización
- Gráficos científicos y de ingeniería.
- Desarrollo de aplicaciones

### 1.2 El escritorio MATLAB

La actual Interfaz Gráfica de Usuario (GUI) de MATLAB maneja el estilo de pestañas y *ribbons*, introducido la primera vez por *Microsoft Office 2007*. Si está familiarizado con la interfaz moderna de *Microsoft Office*, no tendrás inconvenientes cuando utilice MATLAB.

Una vez ejecutado MATLAB se presentan diferentes ventanas. El escritorio de MATLAB es parecido al que se aprecia en la figura 1. De forma predeterminada se muestran 5 ventanas:

**Ventana de Comandos (Command Window)** Se puede ingresar y ejecutar comandos MATLAB y se muestran las salidas. La línea de comandos que puede identificarse mediante el símbolo `>>`.

**Ventana del Editor (Editor Window)** En esta ventana es posible editar los archivos de MATLAB, `.m`. Estos archivos pueden de tipo *script* y contengan una secuencia de instrucciones en MATLAB, para su posterior ejecución. Es posible también en estos definir funciones de usuario.

**Ventana del Historial de comandos (Command History Window)** Contiene la lista de los últimos comandos ejecutados desde la Ventana de comandos.

**Espacio de trabajo o Ventana de variables (Workspace/Variables Window)** Se encuentra la lista de los objetos y variables creados durante la sesión actual. Las variables que se desean utilizar se generan y almacenan en la memoria del programa al asignarles una cierta expresión. Al hacer doble clic en un elemento en esta ventana, se abre el elemento en el Editor donde puede ser examinado o editado.

**Carpeta actual o Carpeta de trabajo ( Current/Work Folder Window)** Al comenzar a trabajar, debe ser especificado el directorio de proyecto.

Al incluir al final de la línea el símbolo `;`, se omite el despliegue del resultado en la pantalla y solamente se asigna el resultado a la variable designada.

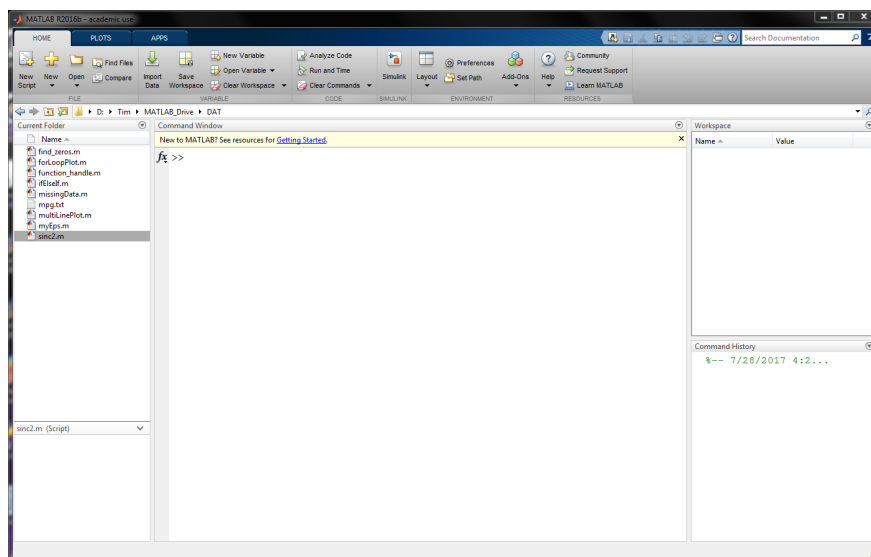


Fig. 1 – Interfaz de desarrollo de MATLAB.

Comenzaremos introduciendo manualmente una matriz, para ello utilizamos las siguientes reglas:

- Separar elementos de una fila con espacios o comas.
- Usar *punto y coma* ; para indicar final de fila.
- Incluir la lista completa de elementos dentro de corchetes, [ ].

### 1.3 Formas de introducir matrices en MATLAB

Se puede introducir matrices en MATLAB:

- Lista explícita de elementos.
- Desde un fichero de datos externo.
- Utilizando funciones propias.
- Creando un fichero .m.

Para introducir la matriz de se escribe: Escribimos el siguiente código para borrar memoria, cerrar todas las ventanas abiertas y limpiar el *Command Window*

```
A = [ 1 6 3 2 1 3 ; 5 10 11 8 ; 9 6 7 1 2 ; 4 15 14 1 ]
```

```
A =  
    16     3     2    13  
     5    10    11     8  
     9     6     7    12  
     4    15    14     1
```

Una vez introducida la matriz, esta queda guardada en el entorno de trabajo de MATLAB. La matriz A es un cuadrado mágico: Todas sus filas, columnas y diagonales suman lo mismo. Para comprobarlo hacemos

```
sum(A)  
  
ans =  
    34    34    34    34
```

El comando `sum(A)` calcula la suma de las columnas de la matriz A, obteniéndose un vector de dimensión el número de columnas.

## 1.4 Expresiones en MATLAB

Al igual que muchos otros lenguajes de programación, MATLAB dispone de expresiones matemáticas, pero al contrario que en la mayoría de los lenguajes de programación, estas expresiones hacen referencia a matrices. Los bloques constructivos de las expresiones son

- Variables
- Números
- Operadores
- Funciones

### 1.4.1 Variables

MATLAB no requiere ningún tipo de declaración o indicación de la dimensión. Cuando MATLAB encuentra un nuevo nombre de variable la crea automáticamente y reserva la cantidad de memoria necesaria. Si la variable ya existe, MATLAB cambia su contenido y si es necesario modifica la reserva de memoria. Por ejemplo, la expresión

```
num_est = 15
```

crea una matriz 1 por 1 llamada `num_est` y almacena el valor 15 en su único elemento. MATLAB no requiere declarar ni dimensionar variables”

- Las variables se auto-declaran al inicializarlas
- La memoria se reasigna dinámicamente

```
x=5;           % poniendo ; se realiza la asignacion pero no se muestra el resultado  
y=20;  
z=x*y          % sin poner ; se puestra el resultado final  
  
z =  
    100
```

Otro ejemplo,

```
datos=load('datos.txt');  
cadena='hola';
```

Realice la multiplicación de los vectores de datos:  $x = [0.5 \ 0.3 \ 0.2 \ 0.8 \ 0.4 \ -0.2]$ ,  $y = [0.8 \ 0.2 \ 0.6 \ 0.3 \ 0.9 \ 0.1]$ . Encuentre el resultado de la multiplicación término a término de la serie de datos.

```
x = [0.5 0.3 0.2 0.8 0.4 -0.2];  
y = [0.8 0.2 -0.6 -0.3 0.9 0.1];  
z = x.*y  
  
z =  
0.4000 0.0600 -0.1200 -0.2400 0.3600 -0.0200
```

En el resultado anterior se observa que la multiplicación se realizó en una sola instrucción, además hay que destacar que la multiplicación punto a punto de dos vectores se indica mediante el punto (.) antes del símbolo de multiplicación \*.

### 1.4.2 Números

MATLAB utiliza notación decimal convencional, con punto decimal opcional y signo + ó -. Es posible utilizar notación científica. La letra *e* especifica un factor de escala de potencia de 10, ej.  $1.25e-4$ , que equivale a  $1.25 \times 10^{-4}$ . La variable compleja para expresar números imaginarios y complejos se especifica con la combinación *1i* ó *1j*.

## 1.5 Scripts: Ficheros .m

Los *scripts* son

ficheros de texto que contienen código de MATLAB. Un *script* es una secuencia de instrucciones de MATLAB guardada en un archivo con extensión .m. Se ejecuta escribiendo su nombre

```
ejem_script
```

## 1.6 Funciones en MATLAB

MATLAB proporciona un gran numero de funciones matemáticas elementales, por ejemplo, **abs**, **sqrt**, **exp**, **sin**, **cos**, etc. Por defecto, MATLAB utiliza números complejos: La raíz cuadrada o el logaritmo de un numero negativo no producen error, sino que dan como resultado un numero complejo. Los argumentos de las funciones pueden ser números complejos MATLAB proporciona también funciones avanzadas: Funciones de *Bessel* o funciones *gamma*.

Las funciones definidas por el usuario se escriben en archivos .m que deben encontrarse en el directorio actual (o en un directorio definido en el *PATH*).

Ejemplo de una función,

```
function [med,des]=med_des(x)  
% Funciona para calcular la media y la desviacion a la vez  
% [med,des]=med_des(x)  
%  
% Rafael Palacios (nov/2004)  
med=mean(x(:));  
des=std(x(:));
```

El nombre de la función debe ser igual al nombre del archivo .m.

## 2 Desarrollo de la práctica de laboratorio

1. (5 points) Responda brevemente en la sección de Marco Teórico de su plantilla de laboratorio las siguientes preguntas:

- (a) Definir los siguientes comandos de MATLAB y describir sus funciones: *print*, *clear*, *close*, *clc*, *round*, *sin*, *cos*, *'LineWidth'*, *'Color'*, *hold*, *xlabel*, *ylabel*, *legend*, *xlim*, *ylim*, *grid*.
- (b) Investigar como abrir y guardar variables del espacio de memoria de MATLAB en archivos tipo *.dat*.
- (c) Investigar como guardar las imágenes del plot en formato *.png* y alta calidad, para exportarlas a  $\text{\LaTeX}$ .
- (d) Investigue acerca de la generación de señales aleatorias y las características estocásticas del ruido blanco Gaussiano aditivo (AWGN, *ing.* Additive White Gaussian Noise). ¿Como se genera en MATLAB una señal contaminada con ruido especificando una relación señal ruido (SNR, *ing.* Signal to noise ratio) predeterminada?
- (e) En el marco teórico es necesario definir las características, tipo de codificación y propiedades de un archivo de audio con extensión *.wav*.
- (f) ¿Que diferencia hay entre archivos de audio *mono* y *estéreo* al ser leídos y guardados en MATLAB?
- (g) Para leer una imagen en MATLAB es necesario conocer el tipo de formato que maneja la imagen y su codificación. Describa los formatos *.PNG*, *.BMP* y *.JPG*, diferencie, preferiblemente con una tabla. ¿Que son los modelos de color RGB y CMYK?

**2. (5 points) Generar una gráfica en MATLAB y guardarla:** MATLAB es un paquete de análisis matemático que trabaja solo con funciones vectoriales y matriciales, **discretas**.

Procedemos a abrir la herramienta de software matemático MATLAB.

Procedemos a crea un nuevo archivo *script* (CTRL+N) y lo guardamos.

Escribimos el siguiente código para borrar memoria, cerrar todas las ventanas abiertas y limpiar el *Command Window*

```
clear all; close all; clc;
```

Luego *seteamos* los parámetros por defecto de las gráficas.

MATLAB no trabaja variables continuas en su dominio, por tanto se hace necesario realizar aproximaciones de señales discretas en el tiempo a continuas, es decir, que si se toma un intervalo de muestreo que tienda a cero (o una frecuencia de muestreo que tienda a infinito), a *simple vista* la señal parecerá *como* de tiempo continuo, es decir aparentemente *aproximadamente continua*. Esta frecuencia se llamará **frecuencia aparente de MATLAB**  $f_s$ .

```
settings.fs = 1e5;
settings.ts = 1/settings.fs;      % step in time
```

Se aconseja tomar la costumbre de guardar parámetros característicos en forma de estructuras (orientadas a objetos).

A continuación se generará el vector de tiempos. Este se puede generar:

- Con un tiempo inicial  $t_i$  y un tiempo final dado  $t_f$ :

```
signal.ti = 0;      % initial time
signal.tf = 0.1;    % final time
t=signal.ti:settings.ts:signal.tf;
```

- Con una duración determinada  $t_m$ :

```
signal.tm = 1e-1;      % signal duration
L = round(signal.tm/settings.ts); % Samples
t = (0:L-1)*settings.ts;
```

Se utilizará la primera forma en el laboratorio.

Procedemos a generar dos señales sinusoidales, una con frecuencia  $f_1 = 2.42kHz$  y la otra con el doble de esa frecuencia  $f_2 = 2f_1$

```
f_1 = 1.5e3; % [Hz]
f_2 = 2*f_1; % [Hz]
s1 = cos(2*pi*f_1*t); % Carrier based on time vector;
s2 = cos(2*pi*f_2*t); % Carrier based on time vector;
```

Se procede a la generación de gráficas en una primera figura 1.

```
figure(1)
plot(t/1e-3,s1);
```

Para dibujar las dos gráficas superpuestas se utilizan los comandos

```
hold on
plot(t/1e-3,s2, '- ', 'LineWidth',2, 'Color',[0.8500 0.3250 0.0980]);
hold off
```

Recreamos los parámetros del gráfico de tipo tamaño de la caja y las rejillas, etiquetas y leyendas

```
xmin=0; xmax=0.5; ymin=-1.1; ymax=1.1; % Box size
Box=[xmin xmax ymin ymax];
labels = {'t, [ms]', 's_n(t)'}; %labelx, labely
legends = {'$s_1(t)$', '$s_2(t)=-\int_0^{\pi} \{s_1(t) dt\}$'};
h=legend(legends);
set(h, 'Interpreter','latex', 'Location','NorthEast','FontSize',16,'...
    FontWeight','bold','Orientation','vertical');
xlabel(labels(1),'fontsize',16,'FontAngle','Italic');
ylabel(labels(2),'fontsize',16);
xlim([Box(1) Box(2)]); ylim([Box(3) Box(4)]);
grid(gca,'minor');
```

Para guardar la imagen, es preferible guardarla con los formatos *.fig* y *.png* automáticamente para futuras modificaciones. Se implementan los siguientes comandos

```
set(gcf, 'Position', get(0,'Screensize')); % Maximize figure.
saveas(gca, 'example1','png');
print('example2', '-dpng', '-r600'); % Save as PNG with 600 DPI
saveas(gca, 'example1.fig');
```

A continuación se generará una imagen aleatoria. Vale recordar que cada vez que se compile este *script*, la línea de generación de ruido se volverá a interpretar y a generar un nuevo ruido. Muchas veces es necesario generar una sola iteración de una señal aleatoria, y debe ser utilizada más adelante sin perder esos datos, por eso guardamos los valores de la variable en un archivo tipo *.DAT*

```
noise=0.1*randn(1,length(s1));
save('noisesignal.mat','noise');
```

Para cargar la variable con el archivo guardado con anterioridad se utiliza el código

```
load('noisesignal.mat','noise');
```

Procedemos a compilar el siguiente resto de código, y describimos cada uno de los comandos utilizados.

```

subplot(2,2,1)
plot(t/1e-3,noise,'-','LineWidth',1,'Color',[0.3010 0.7450 0.9330]);
labels = {'t, [ms]', 'n(t)'}; %labelx, labely
xmin=0; xmax=0.5; ymin=min(noise)-0.2; ymax=max(noise)+0.2; % Box size
Box=[xmin xmax ymin ymax];
xlim([Box(1) Box(2)]); ylim([Box(3) Box(4)]);
grid(gca,'minor');
title('Ruido aleatorio')
subplot(2,2,2)
h = histogram(noise);
h.Normalization = 'probability';
h.BinWidth = 0.01;
h.FaceColor = [0.3010 0.7450 0.9330];
h.EdgeColor = 'k';
title('Histograma del ruido aleatorio')
subplot(2,2,[3 4])
plot(t/1e-3,s1+noise,'-','LineWidth',1,'Color',[0.3010 0.7450 0.9330]);
hold on
plot(t/1e-3,s1,'-','LineWidth',2,'Color',[0 0.4470 0.7410]);
hold off
xmin=0; xmax=0.5; ymin=min(s1+noise)-0.3; ymax=max(s1+noise)+0.3;
Box=[xmin xmax ymin ymax];
labels = {'t, [ms]', 's_n(t)'}; %labelx, labely
legends = {'$s_1(t)+n(t)$', '$s_1(t)$'};
h=legend(legends);
set(h,'Interpreter','latex','Location','NorthEast','FontSize',16,'FontWeight','...','bold','Orientation','vertical');
xlabel(labels(1),'fontsize',16,'FontAngle','Italic');
ylabel(labels(2),'fontsize',16);
xlim([Box(1) Box(2)]); ylim([Box(3) Box(4)]);
grid(gca,'minor');
title('Señal sinusoidal contaminada con ruido')

```

Todas las gráficas generadas se deben guardar e incluir en el informe. Las líneas de código relevantes deben ser comentadas. Debe existir una correcta *identación*.

### 3. (10 points) Generar señales continuas y discretas en MATLAB:

Los siguientes comandos aparecen en versión de estándar, pero usted deberá aplicar el estilo y guardar las gráficas tal cual como se mostró anteriormente. Recuerde comentar cada línea de código.

En el siguiente *script* se generará una señal sinusoidal *aparentemente continua* (no se volverá a utilizar esta denominación, se debe sobreentender) con una frecuencia de  $f_m = 10 \text{ kHz}$ . Para que la señal discreta sea *aparentemente continua* se utilizará una frecuencia de muestreo de  $f_s = 500 \text{ kHz}$ . En el siguiente código representaremos el osciloscopio de la señal (comportamiento de la señal en el dominio del tiempo) y la magnitud de su espectro.

```

fs=1e6; % Frecuencia de muestreo se la señal de Matlab 1 MHz
f=10e3; % Frecuencia de la señal sinusoide
ti=0; % Tiempo incial de la señal [s]
nCyl=5; % Numero de ciclos (periodos) a generar de la sinusoidal
t=ti:1/fs:nCyl*1/f; % Vector de tiempo conntinuo
x=cos(2*pi*f*t); % Señal sinusoide

figure
plot(t,x) % Dibujar la señal
title('Señal sinusoidal continua');

```

```

xlabel('Tiempo, [s]');
ylabel('Amplitud');
figure
[f,X]=spectrumGen(x,fs,2);           % Espectro bidireccional de la señal y ...
    vector de frecuencia
plot(f,X)                          % Dibujar la señal
title('Espectro de la señal sinusoidal');
xlabel('Frecuencia, [Hz]');
ylabel('Amplitud');

```

Código 1 – Representación de una señal continua en Matlab

Para encontrar el espectro de la señal utilizamos la función `[freq,X]=spectrumGen(Signal,fs,side)`. La cual tiene como entradas la señal *Signal*, *fs* la frecuencia de muestreo aparente y *side* para establecer si el espectro es de un lado o dos lados (*side* = 1 o *side* = 2). Descargue `spectrumGen.m` en la carpeta */laboratory/introtomlab/code/* del repositorio del repositorio del curso [aquí](#).

Para trabajar con secuencias de tiempo discreto en MATLAB, en la practica es mucho más sencillo. Para gráficas de señales de tipo discreto, se utiliza el comando **stem** en lugar de **plot**, ambos son análogos.

A continuación, se representará en MATLAB la siguiente secuencia de tiempo discreto:

$$x[n] = \dots, 0, 0.5, 1, 1.5, 2, 2.5, \underset{\uparrow}{3}, 2.5, 2, 1.5, 1, 0.5, 0, \dots$$

Para ello primero debemos definir el vector tiempo *n*. En este caso, para tener un panorama completo de la señal, lo ideal es crear un vector temporal entre  $[-8, 8]$ , de la siguiente forma: `n=-8:1:8`, lo cual significa que generará un vector de  $-8$  a  $8$ , con un paso (*step*) de una unidad. El siguiente código se utilizara para representar de forma gráfica la señal, el espectro no presenta ningún tipo de interés.

```

n=-8:1:8; % Vector de tiempo discreto
x_n=[0,0,0,0.5,1,1.5,2,2.5,3,2.5,2,...
    ...1.5,1,0.5,0,0,0]; % señal x_n
stem(n,x_n); % Dibujar la señal
title('Señal de tiempo discreto');
xlabel('n, [sample]');
ylabel('Amplitud');

```

Código 2 – Señal de tiempo discreto en Matlab

Dibuje de forma discreta, y para un intervalo adecuado donde se puedan apreciar todas las características, las siguientes funciones **sign**, **sawtooth**, **pulstran**, **rectpuls**, **tripuls** y **gauspuls**.

### 3 Procesamiento de imágenes, audio y vídeo en MATLAB

#### 4. (10 points) Exportar señales de MATLAB a archivos *.wav*:

Un archivo *.wav* es un estándar de audio para computadores<sup>1</sup>.

A continuación se muestran los comandos utilizados para abrir, leer y guardar archivos *.wav* en MATLAB. La línea de comando

```
[x,fs] = audioread('filename')
```

Lee un archivo llamado 'filename.wav' y lo convierte en el vector *x* como variable de MATLAB; *fs* extrae la frecuencia de muestreo del archivo. El comando

```
x = audioread('filename',Nsamples)
```

<sup>1</sup>WAV, artículo Wikipedia <https://en.wikipedia.org/wiki/WAV>



lee las primeros  $N_{samples}$  muestras del archivo *.wav* en caso de que no se desee extraer el audio completo. El comando

```
audiowrite('filename',x,fs)
```

crea a partir del vector  $x$  un archivo de audio *.wav* con una frecuencia de muestreo  $f_s$  y llamado *'filename.wav'*.

La resolución estándar para archivos de audio es de 16 *bits* o  $\pm 32768$  niveles. La amplitud se encuentra escalada y restringida a un intervalo de  $[-1, 1]$  en el vector  $x$ . El comando

```
audiowrite('filename',x,Fs,bit)
```

cambia la resolución de bits del archivo.

A continuación genere en MATLAB una señal senoide con una amplitud de  $A = 0.1$ , una frecuencia  $f = 100 \text{ Hz}$ , para  $N = 100000$  muestras, y una frecuencia de muestreo de  $f_s = 22050 \text{ Hz}$ .

Escuche la señal, y luego guarde esta en un archivo *.wav*, cuyo nombre de archivo es el código asignado al laboratorio. Adjunte el archivo *.wav* en el proyecto *.zip* de su laboratorio.

El tiempo total de la señal:  $N_{samples}/f_s = 4.535 \text{ s}$ .

```
N_samples = 100000;
fs = 22050;
fc = 100;
t = (0:N-1)/fs;
x = 0.1*sin(2*pi*fc*t);
sound(x, fs)
audiowrite('sinsound.wav',x,fs);
```

Código 3 – Generación de una señal *.wav* a partir de un armónico

Genere y guarde una versión ruidosa de esta señal, es decir adicione ruido blanco Gaussiano con una potencia igual a la mitad de la potencia de la señal de audio original. Llame esta versión del audio contaminado tal cual como la anterior, pero agregue sin espacios al final del nombre la palabra *NOISED*.

## 5. (10 points) Tratamiento de imágenes en MATLAB:

Para abrir la imagen *lena.png*, de tipo RGB que se encuentra en el repositorio, se debe utilizar la función:

```
RGB = imread('Lena.png');
```

En esta línea de código anterior, la imagen<sup>2</sup> se guarda en una variable-vector denominado RGB. Describa las dimensiones de la variable RGB y su significado. ¿Que tipo de valores toma el vector? Defina la estructura, y por ultimo genere las gráficas de los 3 histogramas correspondientes a los datos de las tres matrices. Utilice las funciones especializadas de MATLAB de DIP (*ing.* Digital Image Processing) si así lo desea.

Para mostrar los datos de la imagen RGB se utiliza la función `imshow(RGB)`.

Para convertir y mostrar una una imagen a escala de grises utilicemos las siguientes líneas de comando:

```
gray = rgb2gray(RGB);
imshow(gray)
```

Código 4 – Convertir y mostrar una una imagen a escala de grises

Describa las dimensiones de la variable *gray*. ¿Que tipo de valores toma? Defina su estructura, y genere un histograma con sus valores.

<sup>2</sup>Descargar la imagen de prueba de la carpeta */laboratory/introtomlab/pictures* del repositorio ([Descargar](#)).

El siguiente código extrae las componentes en rojo, verde y azul de la imagen `peppers.png` que se encuentra en la carpeta de `laboratory/introtomatlab/pictures` en el repositorio<sup>3</sup>:

```
close all; clear all; clc;
myimg=imread('peppers.png','PNG');
size(myimg)
nbc1=512; mcol=[0:nbc1-1]/(nbc1-1);
mypal=zeros(nbc1,3,3); mypal(:,1,1)=mcol;
mypal(:,2,2)=mcol; mypal(:,3,3)=mcol;
for k=1:3
    figure(k), imagesc(myimg(:,:,k))
    colormap(mypal(:,:,k)); axis('image')
end
```

Código 5 – Extracción RGB de una imagen

1. Escriba una función código `.m` que lea una imagen, sea de color o en escala de grises, y la convierta en una imagen en blanco y negro (solo 1 bit). **Nota:** No utilice la función `rgb2bw`.
2. (**Investigación**) A una imagen en escala de grises agregue ruido blanco, de acuerdo a las siguientes relaciones señal ruido (SNR, *ing.* Signal to noise rate):  $-4\text{ dB}$ ,  $0\text{ dB}$ ,  $2\text{ dB}$ . Muestre en *subplots* la imagen original, y las tres versiones ruidosas de esta.

## 6. (10 points) Archivos de video en MATLAB

EL programa matemático MATLAB puede leer distintos formatos de archivos de video como `.avi`, `.mpg` y `.wmv`. Para reproducir un video con MATLAB, basta con utilizar la línea de código `implay('rbsp_launch_720p.mp4');`

Descargar el archivo de video [aquí](#).

Para obtener la información de un archivo de video se puede utilizar el siguiente código:

```
movieObj = VideoReader('grail_launch_720p.wmv');
get(movieObj)
nFrames = movieObj.NumberOfFrames;
width = movieObj.Width;
height = movieObj.Height;
```

Código 6 – Extraer información de un archivo de video en MATLAB

Incluya en el informe la información del video.

Utilice el siguiente código para abrir, mirar la información de un archivo de video y mostrar de forma individual cada uno de sus *frames* (descargar archivo de video `oneCCC.wmv` que se encuentra en la carpeta `introtomatlab/video/` del repositorio del laboratorio [enlace](#)):

```
clear all
close all
movieObj = VideoReader('oneCCC.wmv');
get(movieObj)
nFrames = movieObj.NumberOfFrames;
for iFrame=1:nFrames
    I = read(movieObj,iFrame);
    fprintf('Frame %d\n', iFrame);
    imshow(I,[]);
    pause(0.1);
end
```

Código 7 – Abrir video en MATLAB y mirar sus frames

<sup>3</sup>[Descargar peppers.png](#)

Comente cada una de las líneas del código anterior e incluyalas en el informe, muestre los resultados.

Para más ejemplos de código MATLAB utilizando videos, visitar:

<http://www.mathworks.com/help/matlab/ref/videoreader.read.html>

## 4 Recursos adicionales

- Curso de MATLAB completo en YouTube [enlace](#).
- CheatSheet (hoja de lista de comandos útiles) de MATLAB [descargar](#).
- Libro **MATLAB for Engineers** [descargar](#).