Guía N° 3: *Matlab/Octave*

Objetivo:

Introducir al alumno a la programación en el entorno Matlab/Octave

Fecha de entrega:

29/06/2019

Formato de entrega:

Tanto Matlab como Octave son entornos donde se pueden hacer operaciones directamente por consola y también crear programas en archivos. En este práctico algunas operaciones se realizarán por consola, esas operaciones luego deberán ser copiadas a un programa para su entrega.

Cada ejercicio se corresponde a un archivo .m, con nombre *eN_M.m*, donde N es el nombre de la guía y M el número de ejercicio. El contenido de cada archivo .m debe contener todos los puntos correspondientes a cada ejercicio.

Todos los archivos se deberán comprimir en un solo archivo .zip o .rar y enviarse por correo con título *Entrega Guía 3*

Bibliografía recomendada:

The Scientist and Engineer's Guide to Digital Signal Processing

Para instalar Octave en Debian/Ubuntu:

```
sudo apt update
sudo apt install octave
sudo apt install octave-image
```

Dependencias adicionales

Dependiendo de la instalación particular de Octave en cada caso, puede ser necesario correr los siguientes comandos en consola de octave para obtener los paquetes faltantes

```
pkg install image
pkg install signal
pkg install -forge audio  % Este último solo en caso de que las
```

```
% utilidades de audio por defecto no
% funcionen correctamente
```

1. Operaciones con matrices

Comandos generales de utilidad
help *function* % Devuelve la ayuda de la función <- IMPORTANTE
clear all % Elimina todas las variables
clc % Limpia la pantalla
close all % Cierra todas las figuras

Operaciones comunes con matrices
En los ejemplos supongamos una matriz *M* de 10x10 y *V* un vector de 10 elementos

- Obtener el tamaño de una matriz
Para obtener el tamaño de una matriz se puede utilizar la función **size**

```
size(M) % Devuelve las dimensiones de la matriz M
```

Para una sola dimensión o un vector, se puede usar la función **length**

```
length(V) % Devuelve el tamaño del vector V
```

- Operador paréntesis **()**
El acceso a los elementos de un vector o matriz se realiza mediante el operador paréntesis. Dentro del paréntesis especificaremos las coordenadas de cada dimensión separadas por comas. Tener en cuenta que en Matlab/Octave el primer elemento es el 1, en lugar del 0 usado por ejemplo en C/C++ y Labview.

```
M(1,1)=1        % Asigno el valor 1 a la posición ubicada en fila 1 columna 1
M(10,10)=0      % Asigno el valor 0 a la posición ubicada en la fila 10,
                 % columna 10
M(1,2)=M(2,1)    % Asigno el valor de la posición ubicada en la fila 2,
                 % columna 1, a la posición ubicada en fila 1, columna 2
```

- Operador dos puntos **:**
El operador dos puntos permite acceder a rangos de valores en una matriz. Cuando se usa solo en una dimensión, indica todos los elementos de esa dimensión. También se puede usar con límites y paso, siendo este último opcional y por defecto de valor 1.

```
inicio : paso : fin  % Esto genera un vector que comienza en inicio, termina
                      % en fin y avanza en saltos de paso
M(:,1)=0             % Asigna todas las filas de la columna 1 al valor 0
M(1:4,1:4)=1         % Asigna los elementos ubicados desde la fila 1 hasta
                     % la 4 y desde la columna 1 hasta la 4 al valor 1
```

```
M(10:-2:1,1)=1:2:10  % Asigna a los elementos 10,8,6,4 y 2 los valores
                     % 1,3,5,7 y 9, respectivamente
max(M(:))            % Obtiene el máximo de todos los elementos de M,
                     % independientemente de la dimensión
```

- Operador **end**

El operador **end** se utiliza para indicar la última posición de una dimensión al indexar un arreglo

```
M(end,end) = 0          % Escribe cero en la última fila y columna
M(end:-1:1,1) = 1:10    % Asigna los valores 1,2,...,10 a las posiciones
                        % 10,9,...,1, respectivamente
```

- Operador comilla simple **'**

El operador ' (comilla simple) se utiliza para transponer una matriz

```
Mt = M'               % Asigna a la matriz Mt la matriz M con filas y
                      % columnas intercambiadas
```

- Operador **.**

El operador . (punto) se utiliza para indicar que el operador a continuación debe aplicarse elemento a elemento y no de forma matricial

```
Ms = M.^2     % Eleva cada elemento de M al cuadrado
Mp = M1.*M2   % Multiplica las matrices M1 y M2 elemento a elemento. M1 y M2
              % deben ser del mismo tamaño
```

- Operador **[ ]**

El operador **[ ]** (corchetes) permite concatenar matrices, vector y escalares. Los elementos dentro de los corchetes se concatenarán por columnas si están separados por espacios o comas, y se concatenarán por filas si están separados por punto y coma.

```
MC = [ M, M ]; % Genera una matriz de 10 filas y 20 columnas
MF = [ M; M ]; % Genera una matriz de 20 filas y 10 columnas
K  = [ 1 2 ;  3 4 ]; % Genera una matriz de 2x2 con los elementos
                       % 1,2 en la primer fila y los elementos 3,4 en la
                       % segunda
```

1.1. Cargar la matriz *I* con la imagen lena.tiff haciendo [1] [2]

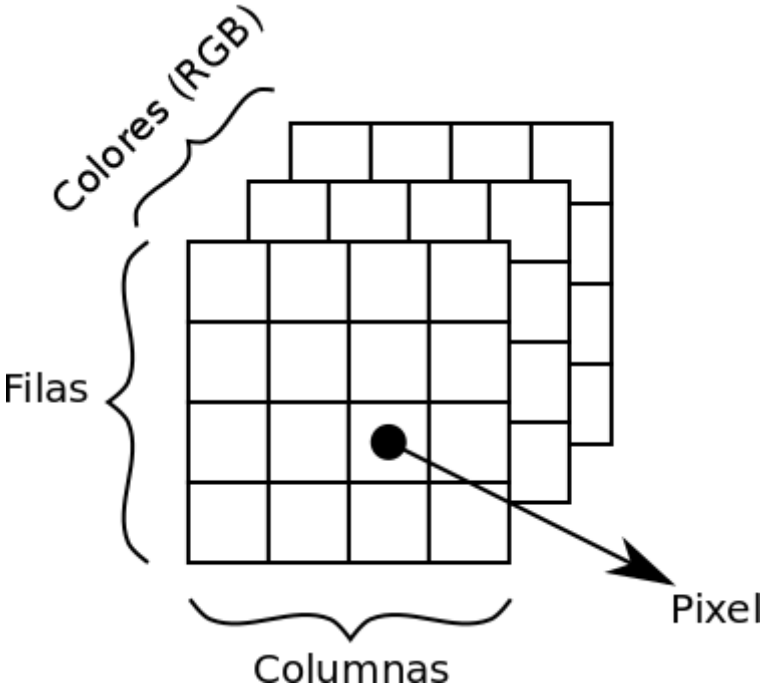
M=imread('lena.tiff');

El punto y coma final no es necesario, pero se usa para evitar la salida por pantalla de toda la operación. Luego consultar el tamaño de la matriz mediante

[1] https://es.wikipedia.org/wiki/Lenna
[2] https://es.wikipedia.org/wiki/TIFF

[F,C,cl]=size(M)

En este caso no usamos punto y coma para poder observar el resultado por pantalla.
La matriz tiene *F* Filas, *C* columnas y *cl* colores, que en general son 3: rojo, verde y azul. Cada elemento de la matriz es en general un entero entre 0 y 255 (8 bits) aunque en algunos casos se utiliza un flotante entre 0 y 1.



El valor más bajo representa la menor intensidad, y el mayor la intensidad más alta. Los colores se forman de acuerdo a la combinación de las intensidades.

	R	G	B
Negro	0	0	0
Blanco	255	255	255
Rojo	255	0	0
Verde	0	255	0
Azul	0	0	255
Cyan	0	255	255
Magenta	255	0	255
Amarillo	255	255	0

Por último para visualizar nuevamente la matriz imagen, se puede hacer **imshow(M)**;

1.2. Espejar la imagen horizontalmente. Para ello primero crear una nueva figura haciendo

figure();

y luego invertir el orden de las columnas mediante el operador end y el operador : . Por último graficar la figura mediante la función **imshow**.

1.3. Repetir el punto anterior pero espejando la imagen verticalmente.

1.4. Se puede obtener la imagen en escala de grises combinando los 3 colores. Para poder operar sin limitaciones numéricas, lo primero a realizar es convertir el tipo de datos de la matriz a flotante, mediante la función **double**

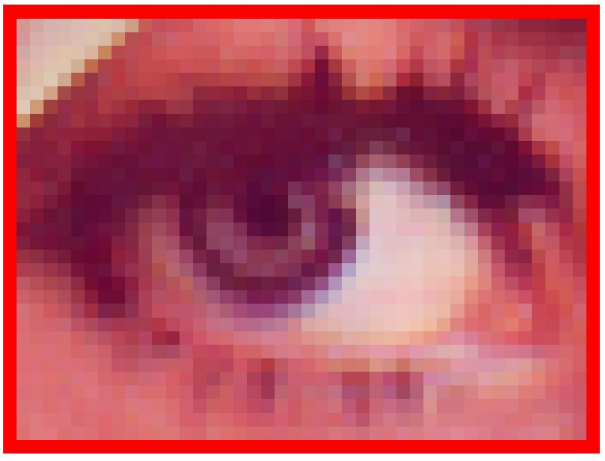
M_d=double(M);

Para obtener la imagen en escala de grises, promedie 3 matrices de cada color entre sí. Luego convierta la imagen nuevamente a entero sin signo de 8 bits mediante

M_u8=uint8(M_d);

y presente la imagen en una nueva figura.

1.5. Extraer un rectángulo que contenga el ojo izquierdo de la figura y rodearlo de un rectángulo rojo, como se ve en la siguiente figura. No es necesario definir la ventana de extracción perfectamente.



Mostrar el resultado con **imshow**

1.6. Generar y mostrar en una nueva figura la imagen con los colores intercambiados, de acuerdo al siguiente mapeo

Destino	Origen
R	G
G	B

B | R

1.7. Crear una función que permita ajustar el brillo y el contraste de la imagen
Para crear una función se debe utilizar el siguiente código

```
function Q=brillo_contraste(I,B,C)
% Agregar el codigo aqui
end
```

Y guardar el script con el mismo nombre que la función.

La función deberá tomar la imagen I y aplicar las siguientes funciones:

$$I_d = 128 \left[\frac{I_d - \min(I_{byn})}{\max(I_{byn}) - \min(I_{byn})} (C+1) + (1-C) \right]$$

$$I_d = I_d + 255\,(2B-1)$$

La primer función ajusta el contraste de la imagen a partir del parámetro C, y para ello necesita los valores mínimos y máximos de la imagen en blanco y negro como factores de normalización. Un valor C = 0 dará el contraste mínimo y un valor C = 1 dará el contraste máximo.
La segunda función ajusta el nivel de brillo de la imagen mediante el parámetro B. Un valor B = 0 dará el brillo mínimo, B=1 el brillo máximo y B = 0.5 no cambia el brillo de la imagen.

Los pasos entonces son:

- Convertir la imagen a flotante
- Obtener la versión en escala de grises
- Obtener mínimo y máximo de la imagen en escala de grises
- Aplicar las fórmulas sobre la imagen a color
- Convertir de nuevo a entero
- Actualizar el valor de la variable de retorno (Q)

Probar la función desde la consola de comandos.

2. Filtros de respuesta finita al impulso - FIR

En este ejercicio vamos a implementar un filtro FIR operando muestra a muestra. El filtro FIR es una aplicación directa de la operación matemática de convolución, que permite combinar una secuencia de entrada con una secuencia de coeficientes para producir una secuencia de salida con características de señal diferentes. Estas características que se modifican se definen a partir del conjunto de coeficientes (también llamados taps) que componen el filtro.
La fórmula general de la convolución es:

$$y[n] = h[n] * x[n]$$

$$y[n] = \sum_{i=-\infty}^{\infty} x[i]h[n-i]$$

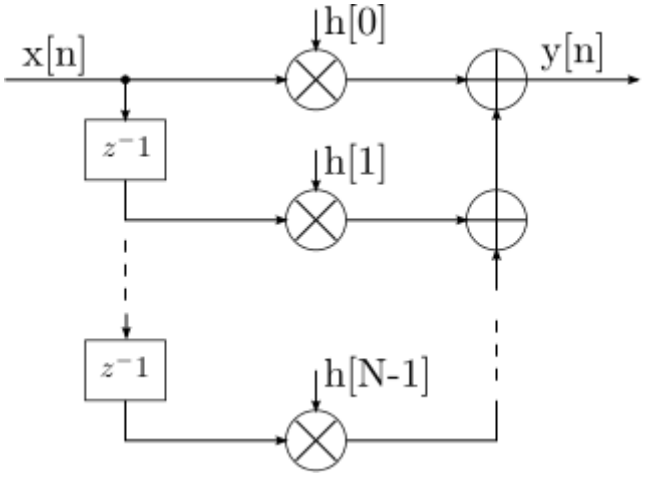
La fórmula general nos dice que para cada muestra de salida debemos sumar una cantidad infinita de productos, que incluyen las muestras pasadas y futuras de la señal *x[n]*. Por supuesto, esto no es realizable en la práctica, por ello se ponen dos condiciones sobre la secuencia *h[n]*:
- Cantidad finita de muestras: *h[n]* tendrá *N* muestras
- Causalidad: Las muestras de *h[n]* existirán para valores positivos de *n*. (De manera que sólo se usen muestras pasadas de *x[n]* en la convolución).
Con estas condiciones, la ecuación de la convolución se simplifica a

$$y[n] = \sum_{i=0}^{N-1} x[n-i]h[i]$$

A la secuencia *h[n]* se la conoce entonces como coeficientes, taps o kernel del filtro FIR y esta secuencia es exactamente igual a la respuesta al impulso del sistema.
Si ampliamos la sumatoria nos queda

$$y[n] = h[0]x[n] + h[1]x[n-1] + \cdots + h[N-1]x[n-N+1]$$

Como vemos, cada muestra de salida se genera multiplicando uno a uno los últimos *N* valores de la entrada *x* con la secuencia *h* y sumando todos los productos. Esto puede representarse de manera gráfica de la siguiente manera:



Los rectángulos con leyenda z^{-1} representan retardos de 1 muestra y se implementan con memoria. Estos retardos almacenan el estado del filtro. Los círculos representan las operaciones de productos y sumas.

La función de transferencia en transformada z resulta:

$$\frac{Y[z]}{X[z]} = h[0] + h[1]z^{-1} + \cdots + h[N-1]z^{-N+1}$$

Y está compuesta únicamente de ceros, por lo que un filtro FIR siempre es estable.

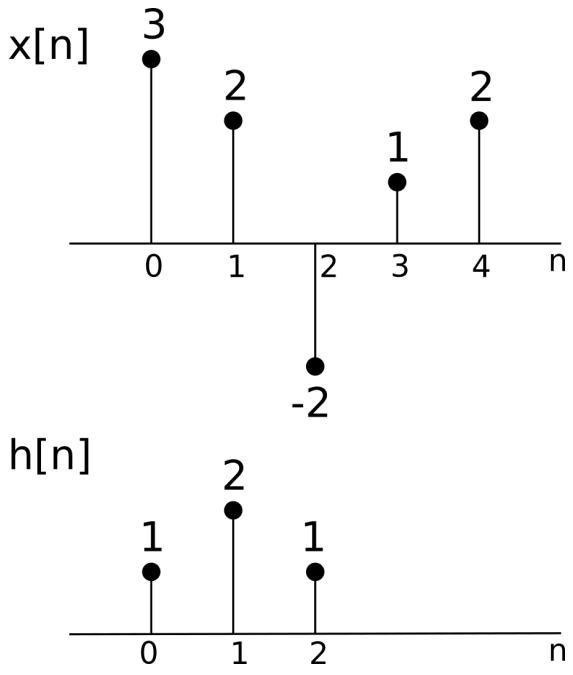
Ejemplo de convolución:
Supongamos una secuencia *x[n]*, en este caso ejemplo, con los siguientes valores

```
x = [ 3, 2, -2, 1, 2 ]
```

y una secuencia *h[n]* también finita, con los siguientes valores

```
h = [ 1, 2, 1]
```

Podemos representar estas secuencias gráficamente desde Matlab mediante la función **stem**



Para calcular el resultado de la convolución aplicaremos la fórmula, expresada como la sumatoria ampliada. Empezamos calculando la salida para *y[-1]*

y[-1] = h[0] x[-1] + h[1] x[-1-1] + h[2] x [-1-2]

Cómo la secuencia *x[n]* no está definida para *n<0*, vamos a asumir su valor igual a 0. Lo mismo haremos para valores de *n>4*

y[-1] = 1 . 0 + 2 . 0 + 1 . 0 = 0

Para y[0] tenemos

y[0] = h[0] x[0] + h[1] x[0-1] + h[2] x [0-2]

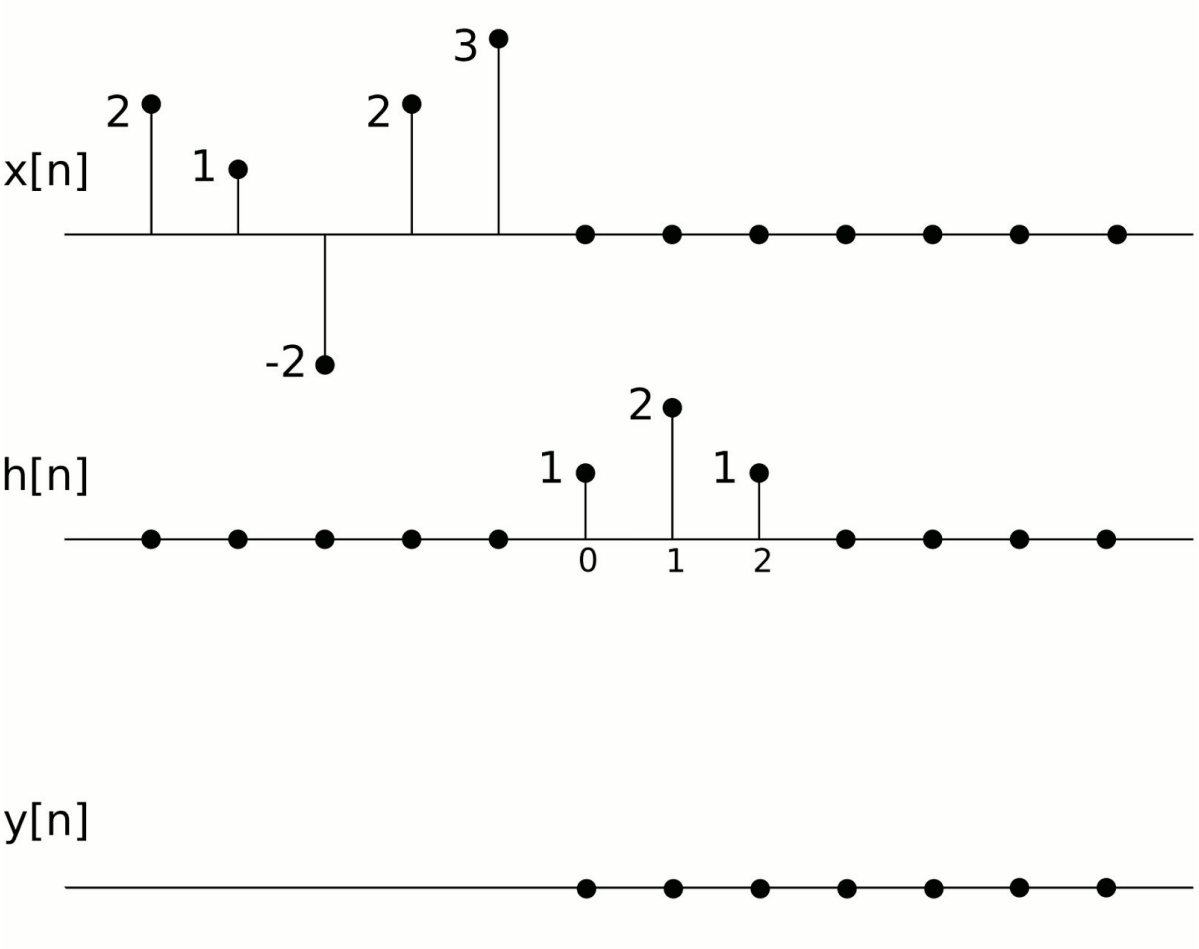
y[0] = h[0] x[0] + h[1] x[-1] + h[2] x [-2] = 1 . 3 + 2 . 0 + 1 . 0 = 3

Realizando estas operaciones para todos los valores de *n* nos queda:

```
y[-1] = h[0] x[-1] + h[1] x[-2] + h[2] x [-3] = 1 . 0 + 2 . 0 + 1 . 0 =  0
y[ 0] = h[0] x[ 0] + h[1] x[-1] + h[2] x [-2] = 1 . 3 + 2 . 0 + 1 . 0 =  3
y[ 1] = h[0] x[ 1] + h[1] x[ 0] + h[2] x [-1] = 1 . 2 + 2 . 3 + 1 . 0 =  8
y[ 2] = h[0] x[ 2] + h[1] x[ 1] + h[2] x [ 0] = 1 .-2 + 2 . 2 + 1 . 3 =  5
y[ 3] = h[0] x[ 3] + h[1] x[ 2] + h[2] x [ 1] = 1 . 1 + 2 .-2 + 1 . 2 = -1
y[ 4] = h[0] x[ 4] + h[1] x[ 3] + h[2] x [ 2] = 1 . 2 + 2 . 1 + 1 .-2 =  2
y[ 5] = h[0] x[ 5] + h[1] x[ 4] + h[2] x [ 3] = 1 . 0 + 2 . 2 + 1 . 1 =  5
y[ 6] = h[0] x[ 6] + h[1] x[ 5] + h[2] x [ 4] = 1 . 0 + 2 . 0 + 1 . 2 =  2
y[ 7] = h[0] x[ 7] + h[1] x[ 6] + h[2] x [ 5] = 1 . 0 + 2 . 0 + 1 . 0 =  0
```

Notar en el cálculo como las muestras de *x[n]* (en gris) van ingresando secuencialmente, como si pasaran por un registro de desplazamiento. También notar que hay sólo 7 salidas distintas de cero. Esto es, porque en general la convolución de 2 secuencias finitas de tamaño *N* y *M*, es otra secuencia de tamaño *N+M-1*.

Podemos visualizar la misma operación de la siguiente manera (Imagen GIF animada, la señal *y[n]* se representa en escala reducida)



Programación de la convolución en Matlab/Octave

Para programar utilizaremos algunas estructuras de control de flujo, por lo que los siguientes ejemplos nos serán de utilidad:

Estructura **switch**

```
switch variable
   case 1
      % Se ejecuta si variable vale 1
   case 2
      % Se ejecuta si variable vale 2
   case 3
      % Se ejecuta si variable vale 3
   otherwise
      % Cláusula por defecto
end
```

Estructura **if...else**

```
if valor > 0
   % Código a ejecutar si valor es mayor que 0
elseif valor == 0
   % Código a ejecutar si valor es igual a 0
else
   % Código a ejecutar si no se cumplen las condiciones previas (valor < 0)
end
```

Estructura **while**

```
while cond
   % Código a ejecutar mientras la expresión contenida en cond se
   % evalúe como verdadero
end
```

Estructura **for**

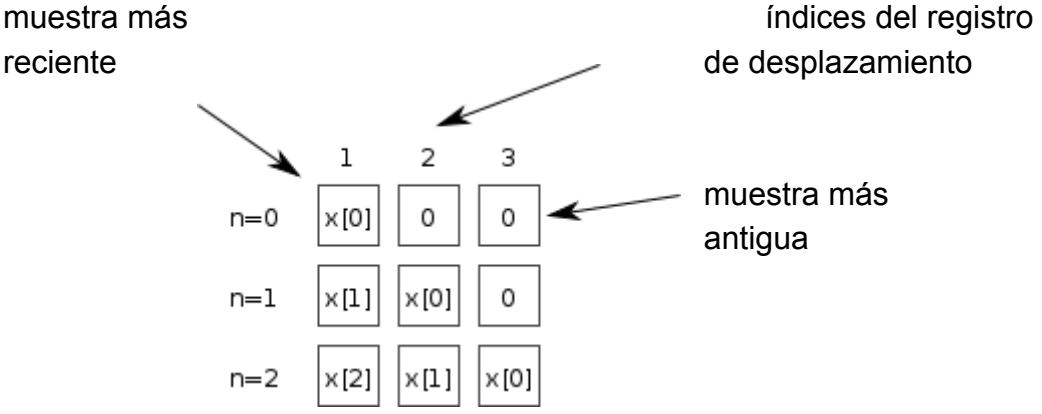
```
for i=1:N
   % Se correrán tantas iteraciones elementos como tenga el vector de
   % entrada (N en este caso)
end
```

2.1. Crear un programa que realice la convolución entre una señal *x de* longitud *M* y un kernel *h* de longitud *N* mediante un bucle **for** y operando muestra a muestra, de acuerdo a la ecuación de la convolución.

- Como la secuencia resultante tiene *M+N-1* muestras, lo primero a realizar es crear un vector *y* de 1 fila y *M+N-1* columnas, para contener el resultado. Para ello se puede utilizar la función **zeros**.
- Inicializar a cero un vector de tamaño *N*. Este vector funcionará como registro de desplazamiento.
- Extender el vector *x* concatenando *N-1* muestras en cero al final mediante el operador de concatenación.

- Crear un bucle **for** que itere las *M+N-1* muestras de salida, con índice *n*. En cada iteración, actualizar el registro de desplazamiento con las *N-1* muestras más recientes del registro de desplazamiento y la muestra actual de la señal *x*, leída utilizando el índice *n* del bucle.

  Ejemplo para N=3

  En una aplicación de tiempo real, este bucle no se realiza, pues no se cuenta con todas las muestras de entrada. En ese caso lo que se hace es simplemente ingresar la muestra nueva al registro de desplazamiento.
  Por eso es que usamos un registro de desplazamiento para almacenar las muestras en lugar de operar directamente sobre la secuencia x, porque este enfoque es aplicable en ambas situaciones.
- Agregar otro bucle **for**, interno al bucle anterior y ubicado luego de la actualización del registro de desplazamiento, que barra *N* iteraciones y con índice *i*. En cada iteración del bucle anterior multiplicar muestra a muestra el registro de desplazamiento con una muestra del filtro *h* y acumular el producto en la muestra *y[n]*.
- Usando las secuencias mostradas en la introducción, verificar el funcionamiento correcto del algoritmo.
- Graficar el vector *y[n]* mediante la función **stem**.

2.2. Generar una nueva secuencia *x*, de longitud *M=7*, mediante la función **randn**. Calcular la convolución con la secuencia h anterior y graficarla mediante el siguiente comando

```
plot(y,'o-')  % Grafica el vector y identificando las muestras con
              % círculos conectados por líneas
```

Calcular la convolución nuevamente pero usando la función **conv**. Grafique el nuevo resultado con los siguientes comandos

```
hold all      % Mantiene el gráfico anterior
plot(z, 'x-') % Grafica el vector z identificando las muestras con
              % cruces conectados por líneas
```

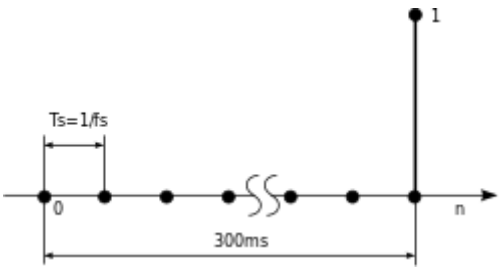
Verifique que ambos resultados son iguales

2.3. Reemplazar el bucle **for** interno por una multiplicación matricial de una matriz de 1 fila x *N* columnas con otra matriz de *N* filas y 1 columna. Usar el operador de transposición para transponer el segundo elemento del producto.
Generar dos vectores aleatorios de 10000 (diez mil) muestras cada uno.
En el código de la convolución, poner en la primer línea la función **tic**, y en la última la función **toc**. Esto al correr nos dará una estimación del tiempo de procesamiento. Convolucionar ambos vectores y registrar el tiempo como comentario.
Repetir los pasos para la versión de la convolución con el bucle **for**, correr con los mismos vectores y registrar el tiempo como comentario.

3. Convolución con filtros de retardo

3.1. Experimentaremos con un filtro muy simple: el filtro de retardo. Este filtro consiste en un impulso desplazado y su único efecto es retardar la señal de entrada. Para ello sigamos los siguientes pasos

- Leer el archivo numeros.wav, mediante la función **wavread** o **audioread**(en caso de usar MATlab 2012 en adelante) y guardar la señal resultante en una variable *r*. Obtener al mismo tiempo con esta función la frecuencia de muestreo y cantidad de bits del archivo wav(para hacer esto en Matlab 2012 en adelante se usa **audioinfo**). Utilizar la ayuda de ser necesario.
- Crear una señal *h* que consista en un impulso desplazado 300ms con respecto al origen, como se muestra en la siguiente figura:

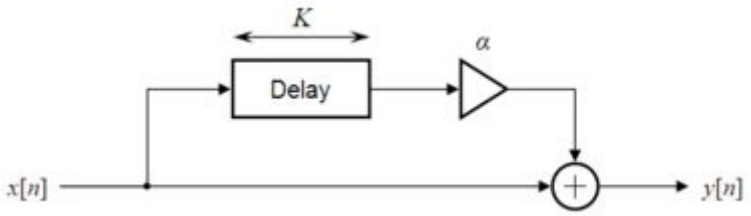


Utilice la frecuencia de muestreo para calcular la cantidad de muestras que equivalen a dicho retardo.

- Convolucionar la señal *r* con la señal *h* para crear una señal *l*. Eliminar las últimas muestras de la señal *l* de manera que las señales *l* y *r* tengan la misma longitud.

- Combinar ambas señales en una matriz de dos columnas mediante la operación *[r , l]* . De esta manera generamos una señal estéreo a partir de la señal mono original.
- Escribir la señal resultante del punto anterior a un archivo .wav utilizando la función **wavwrite** o **audiowrite** (Matlab 2012) . Utilizar la misma definición de frecuencia de muestreo y cantidad de bits de la señal original. Reproducir con un reproductor de audio para comprobar el resultado. También se puede reproducir desde matlab con la función **sound**

3.2. A continuación utilizaremos una extensión del filtro de retardo llamada filtro peine (comb filter)[3]. El filtro peine posee la siguiente estructura:



Se puede expresar la salida como

$$y[n] = x[n] + ax[n - K]$$

Y posee la siguiente respuesta al impulso

$$h[n] = \begin{cases} 1 & \text{si } n = 0 \\ a & \text{si } n = K \\ 0 & \text{de otro modo} \end{cases}$$

- Crear un filtro peine con N secciones de retardo, de acuerdo a la siguiente ecuación.

$$y[n] = x[n] + \sum_{i=1}^{N} a_i x[n - iK]$$

Calcular *K* para que cada retardo individual equivalga a 20ms. Calcular *N* de manera que el mayor retardo (para *i=N*) equivalga a un retraso total de 100ms. Calcular el valor de *a* para que cada versión retrasada de *x[n]* posea 3dB menos de potencia que la anterior.

- Convolucionar la señal leída en el ejercicio anterior con el filtro. Normalizar la amplitud de la señal para que el máximo valor absoluto sea menor que 1 y grabar el resultado con la función **wavwrite** o

[3] https://es.wikipedia.org/wiki/Filtro_comb

audiowrite (Matlab 2012) .

4. Convolución en dos dimensiones

La operación de convolución se puede aplicar en dos dimensiones, por ejemplo para utilizar filtros en imágenes
Supongamos una matriz *x[n,m]* con *F* filas y *C* columnas, y otra matriz *h[n,m]* con *N* filas y *M* columnas, la convolución entre ambas es

$$y[n,m] = \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} x[n-i, m-j] h[i,j]$$

Donde *y[n,m]* es la matriz resultado y tiene *F+N-1* filas y *C+M-1* columnas.

La operatoria es muy similar a la convolución de una dimensión, y lo vamos a demostrar con el siguiente ejemplo:
Supongamos la siguiente matriz *x[n,m]*

n \ m	0	1	2	3
0	1	1	1	1
1	1	-2	-1	0
2	1	0	0	0
3	1	-1	-1	2

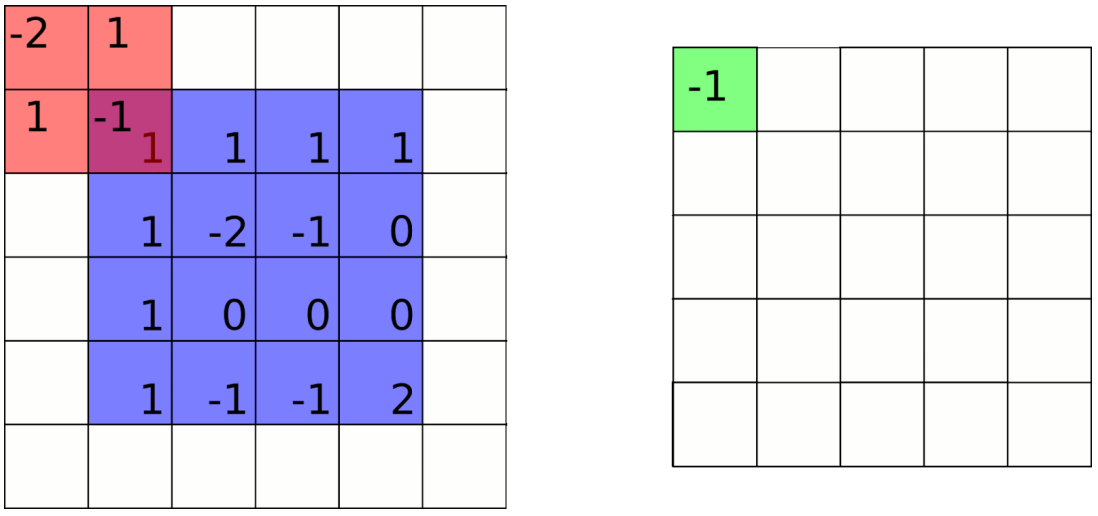
Y la siguiente matriz *h[n,m]*

n \ m	0	1
0	-1	1
1	1	-2

Los pasos para convolucionar ambas matrices son:
a) Crear la matriz resultado con *N+F-1* filas y *M+C-1* columnas.
b) Invertir la matriz *h* en filas y también en columnas.

c) Inicializar los índices de posición de salida en [0,0].
d) Ubicar la matriz *x* al centro de una matriz de ceros de tamaño *2N+F-2* filas y *2M+C-2* columnas (matriz ampliada).
e) Tomar los primeros *NxM* elementos de la matriz ampliada y multiplicar uno a uno todos los elementos de esta submatriz con la matriz *h* invertida.
f) Sumar todos los productos y guardar la suma en la posición de salida.
g) Desplazarse una columna. Si ya estamos en la última columna posible, saltar a la próxima fila. Al mismo tiempo incrementar los índices de salida de igual forma.
h) Repetir los pasos del e) al h) hasta completar todas las posiciones de la matriz de salida.

Podemos visualizar esta operatoria con la siguiente imagen animada. En azul se ve la matriz *x* al centro de la matriz ampliada. En rojo la matriz *h* invertida en filas y en columnas. Y en verde la matriz de salida. Los elementos en morado se multiplican uno a uno y luego se suman para generar cada posición de salida.



4.1. Crear un programa que permita convolucionar dos matrices entre sí. Utilice un bucle **for** para recorrer las filas, otro bucle **for** para recorrer las columnas y el operador **.*** para multiplicar elemento a elemento. No es necesario el uso de un registro de desplazamiento, pudiéndose leer los elementos directamente de la matriz ampliada. Luego utilice la función **sum** para sumar los productos y generar la salida. Pruebe el funcionamiento con las matrices de ejemplo.

4.2. Genere una nueva matriz *h* aleatoria con números aleatorios entre 0 y 5 mediante la función **rand** y la función de redondeo **floor**. Convolucione con la matriz *x* del ejemplo y compare la salida con el resultado de la misma operación realizada con la función **conv2**.

4.3. Convolucione la imagen en escala de grises del ejercicio 1.4, con la siguiente matriz:

$$h[n,m] = \begin{bmatrix} 1 & -1 \\ 0 & 0 \end{bmatrix}$$

Presente el resultado mediante **imshow**. ¿Qué efecto tuvo la convolución en el resultado? Comente como comentario.

4.4. Convolucione la imagen en escala de grises del ejercicio 1.4, con la siguiente matriz

$$h[n,m] = \begin{bmatrix} 1 & 0 \\ -1 & 0 \end{bmatrix}$$

Presente el resultado mediante **imshow**. ¿Qué efecto tuvo la convolución en el resultado? Comente como comentario.

4.5. Combine ambas imágenes mediante la expresión:

$$G = \sqrt{G_x^2 + G_y^2}$$

Aplique un umbral al resultado, de manera que los píxeles que sean mayores al umbral tengan el valor blanco (255) y de ser menores el valor negro (0). Ajuste el umbral a criterio propio.

4.6. Repita los pasos desde el 4.3 al 4.5 para las siguientes matrices *h*[4]:

$$h_x[n,m] = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \qquad h_y[n,m] = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

[4] https://en.wikipedia.org/wiki/Sobel_operator