PROCESAMIENTO DIGITAL DE SEÑALES

PROFESOR: HUMBERTO LOAIZA C., Ph.D.

LABORATORIO No. 2. (Software)
Funciones Matlab -Remuestreo – Funciones de Transferencia – Ecuaciones de diferencia

I. Objetivos

- *Estudiar el mecanismo de creación y manipulación de **funciones** en Matlab.*
- *Conocer el mecanismo de creación y manipulación de polinomios racionales en Matlab.*
- *Aplicar las técnicas de diezmado e interpolación para el remuestreo de señales.*
- *Generar y evaluar funciones de transferencia.*
- *Analizar la respuesta de sistemas LTI expresados como funciones racionales en z.*
- *Generar y analizar la respuesta de un sistema LTI representado por ecuaciones de diferencias.*

II. Introducción

1. Funciones

Es posible adicionar funciones a las librerías de Matlab expresadas en términos de funciones existentes. Los comandos y funciones existentes que componen las nuevas funciones residen en archivos texto con extensión .M.

Funciones son archivos .M que pueden aceptar argumentos de entrada y retornar argumentos de salida. El nombre del archivo .M y de la función deben ser los mismos. El nombre empieza con un caracter alfabético. El nombre del archivo sin su extensión es lo que busca Matlab durante el proceso de llamado.

Las funciones operan sobre variables dentro de su propio espacio de trabajo (workspace), diferente al espacio de trabajo creado desde la línea de comandos de Matlab.

- La primera línea de un archivo de función contiene la definición de la sintaxis. Esta empieza con la palabra clave **function,** la cual permite definir el nombre de la función y, el orden y cantidad de argumentos. Por ejemplo, el archivo **stadist.m** que contiene una función para calcular el valor medio y la desviación estándar de un vector está dado por,

```
function  [promed,devst]  = stadist (x)
% Función para calcular el valor promedio y la desviación estándar de un vector
% Entrada:   x. vector a procesar
% Salida:  promed. valor promedio de los datos de x
%            devst. desviación estándar del vector x
n = length(x);
promed = sum(x)/n;
devst = sqrt(sum((x-promed).^2/n));
```

En el ejemplo, la primera línea define el nombre de la función **stadist** que contiene un sólo parámetro de entrada definido entre paréntesis (el vector x) y dos parámetros de salida entre corchetes (promed y devst).

- Las líneas de comentario siguientes, hasta la primera línea en blanco o la primera línea ejecutable, constituyen la información que suministra el comando *help* desde la línea de comandos – siempre y cuando Matlab pueda acceder al directorio donde se ubica la función-
- El resto del archivo lo constituye el código ejecutable que define cada función en particular.

1.1. Subfunciones

Las subfunciones son creadas al definir nuevas funciones dentro de un mismo archivo .M. Las subfunciones se definen al final de la función precedente o subfunción, y solo son visibles a las otras funciones dentro del archivo.

Por ejemplo, si se redefine el archivo **stadist.m** para que tenga una subfunción **promedio** para calcular el valor medio del vector *x*, se tendría,

```
function [pmedio,devst] = stadist(x)        %Función principal
n = length(x);
pmedio = promedio(x,n);
devst = sqrt(sum((x-avg(x,n)).^2)/n);

function [prom] = promedio(x,n)             %Subfunción
prom = sum(x)/n;
```

Las subfunciones no son visibles fuera del archivo de definición. Las funciones retornan a la línea siguiente de donde fueron invocadas, después de finalizar su código ejecutable. La instrucción **return** puede utilizarse para forzar un retorno en cualquier parte del programa.

1.2. Funciones con cantidad de argumentos variables

Las funciones ***varargin*** y ***varargout*** permiten pasar en un programa, definido por el usuario, cualquier número de parámetros de entrada o de salida, respectivamente.

Los parámetros de *entrada* se almacenan en un arreglo de celdas (cell array), donde cada celda puede tener cualquier tipo de dato y tamaño. Esto lo hace automáticamente Matlab.
Los parámetros de *salida* deben ser empaquetados dentro del código de la función en un arreglo para ser devuelto al finalizar la función.
A manera de ejemplo, se tiene la función *trazos.m* que permite dibujar cualquier cantidad de puntos y unirlos uno tras de otros con líneas rectas. Las coordenadas de cada punto se entregan como un vector de dos elementos. No se definen parámetros de salida.

```
function trazos (varargin)

for i = 1:length(varargin)          % repetir según cantidad de puntos
```

```
   x(i) = varargin{i}(1);          % almacenar en el vector x todas las abscisas de los puntos
   y(i) = varargin{i}(2);          % almacenar en el vector y todas las abscisas de los puntos
end

plot(x,y);                         % graficar x vs. y
```

La función trazos puede trabajar con varias listas de entrada, como,

trazos([2 3],[1 5],[4 8],[6 5],[4 2],[2 3])
trazos([-1 0],[3 -5],[4 2],[1 1])
trazos([1 1],[3 -5])

1.2.1. Desempaquetando el contenido de *varargin*

Puesto que *varargin* contiene todos los argumentos de entrada en el arreglo de celdas, se hace necesario utilizar la indexación de celdas para extraer los datos. Por ejemplo, la instrucción del programa precedente,

y(i) = varargin{i}(2);

utiliza dos índices:

{i} indexado de la celda: accede a la i-ésima celda de varargin.
(2) indexado del dato almacenado en la celda i: representa el segundo elemento del contenido de la celda i.

1.2.2. Empaquetando el contenido de *varargout*

Cuando se desea tener cualquier cantidad de argumentos de salida, se debe empaquetar los datos de salida en el arreglo de celdas ***varargout***. Para determinar cuantos argumentos se utilizaron durante el llamado de la función es necesario utilizar la función ***nargout***

El siguiente programa, *pares_ord*, tiene como entrada una matriz de dos columnas, en donde la primera columna contiene un conjunto de coordenadas *x* y la segunda contiene las coordenadas *y*. El programa debe generar un conjunto de **pares ordenados** [x_i, y_i] para poder utilizar el programa *trazos* visto en §1.2.

```
function [varargout] = pares_ord(matriz_in)

[f,c]= size(matriz_in);                    % Verificar que no se pidan mas datos de los disponibles
if   nargout ≤ f
        for i = 1:nargout
           varargout{i} = matriz_in( i, :) ;        % Asignación de datos
        end
end
```

La instrucción dentro de la bucla *for* utiliza **{ }** para indexar el arreglo de celdas, y **( )** para indexar los elementos de la matriz que se copiarán en la celda.

La función *pares_ord* puede invocarse,

```
a = {1 2;3 4;5 6;7 8;9 0};
[p1,p2,p3,p4,p5] = pares_ord (a);
```

donde p1, p2,.. representan vectores de dos elementos que almacenan un par ordenado.

1.2.3. *varargin* y *varargout* como un elemento más de la lista de argumentos

varargin o ***varargout*** pueden aparecer en la parte final de la lista de argumentos, junto con cualquier otra variable en la declaración de una nueva función.

La forma correcta de utilizarlos se muestra en los siguientes ejemplos,

```
a) function [out1,out2] = ejemplo1 (a,b,varargin)
b) function [i,j,varargout] = ejemplo2(x1,y1,x2,y2,flag)
c) function [r,s, t, varargout]= ejemplo3(t1, t2, varargin)
```

1.3. Funciones con Argumentos Incompletos

Matlab permite que una función se invoque sin todos los parámetros declarados durante la creación de la función. Como mínimo se permite llamar a la función con un solo parámetro de entrada. Los parámetros de salida no son obligatorios para la ejecución de la función. Los resultados serán almacenados ordenadamente en las variables de salida. Si no existen argumentos de salida, solo el primer resultado será almacenado en **ans**. En ningún caso se acepta un llamado con más parámetros que los definidos.

Para permitir llamar a funciones con un número *incompleto* de argumentos de entrada, es necesario utilizar la función ***nargin*** (number of function input arguments) dentro de una instrucción de decisión. La función *nargin* dentro de una función declarada por el usuario, regresa el número de argumentos de entrada utilizados en la llamada de la función. [fuera del cuerpo de una función, *nargin('fun')* retorna el número de entradas declaradas para la función 'fun'. Este número es negativo si la función tiene un número *variable* de argumentos de entrada].

Por ejemplo, el archivo *promedio.m* que contiene una función para calcular el promedio de datos,

```
function [res] = promedio (a,b,c)
        if nargin == 3
                res= (a+b+c)/3;
        elseif  nargin == 2
                res=(a+b)/2;
        else
                res=a;
end
```

puede invocarse de una de las siguientes formas:

```
>> promedio(25,40,35 )
>> promedio(48,82)
>> promedio (12)
```

La función ***nargout*** (number of function output arguments) dentro de una función declarada por el usuario, regresa el número de argumentos de salida utilizados en la llamada de la función. [fuera del cuerpo de una función, *nargout('fun')* retorna el número de salidas declaradas para la función 'fun'. Este número es negativo si la función tiene un número *variable* de argumentos de salida]. Esta función no es necesaria para la manipulación de funciones con argumentos *incompletos*.

2. Evitar buclas *for*

Existe una tendencia casi natural por parte de los programadores de lenguajes estructurados, como el C y Pascal, de utilizar Matlab de la misma forma que un lenguaje de alto nivel. Lo anterior puede conducir a la elaboración de programas poco eficientes cuando se utilizan buclas *for* para procesar los elementos de un vector. En su lugar se recomienda utilizar *funciones* de Matlab que realizan estas operaciones. El ejemplo siguiente permite observar la simplicidad del código al utilizar estas funciones de Matlab (en el ejemplo, ***sum***).

Ejemplo: sumar los N elementos del vector Dat y almacenar el resultado en la variable Res.

i. Programación iterativa

```
Res=0
for i=1: N
        Res=Res+Dat(i);
end
```

ii. Programación con funciones de Matlab

```
Res=sum (Dat);
```

Otra alternativa para evitar las buclas *for* es la utilización de *operaciones* vectoriales, en donde puede aprovecharse las propiedades de las operaciones para obtener los resultados. Continuando con el ejemplo anterior, la suma de los elementos se puede lograr con la multiplicación entre el vector dato y un vector de unos:

iii. Programación con operaciones vectoriales

```
uno= ones(N,1);        % creación de un vector columna con todos sus elementos iguales a uno.
Res=Dat*uno;
```

La razón para evitar los bucles *for* es su extremada ineficiencia en Matlab, por ser éste un lenguaje interpretado. Por lo tanto, las buclas *for* sólo deben utilizarse como último recurso y probablemente sólo en operaciones de control y no como alternativa de cálculo.

III. Funciones prácticas de Matlab

- **subplot**. Permite visualizar en una misma pantalla varios gráficos. La función divide la ventana actual en planos rectangulares enumerados en forma de filas. Cada plano contiene un eje. Las funciones para graficar visualizan sus gráficos en el plano recién activado.
- **clf**. Borra todos los gráficos presentes en la ventana activa.
- **figure**. Permite generar nuevas ventanas y/o dirigir la salida gráfica hacia la ventana especificada por su manejador (handle).
- **func2str**. Construye una cadena que almacena el nombre de la función a la cual su manejador pertenece.
- **strtok**. Retorna el primer caracter delimitador en cadenas.

IV. Procedimiento

1. Remuestreo de Señales

En procesamiento digital de señales es frecuente recurrir al re-muestreo (resample). Por ejemplo, si dos señales, o más, de un sistema han sido capturadas con diferentes periodos de muestreo, se necesitará de un re-muestreo para poder procesar simultáneamente a estas dos señales.

La función ***resample( )*** de Matlab permite efectuar un cambio de la frecuencia de muestreo utilizando los procesos de interpolación y de diezmado.

La *interpolación* toma la señal de entrada y produce una señal de salida muestreada a una frecuencia entera, **L** veces *mayor* que la señal de entrada. La función **interp( )** permite realizar este cambio en la frecuencia de muestreo.

El *diezmado* toma la señal de entrada y produce una señal de salida que es muestreada a una frecuencia entera **M** veces *menor* que la señal de entrada. La función **decimate( )** permite realizar este cambio en la frecuencia de muestreo.

Para cambiar la frecuencia de muestreo de una señal por una cantidad no entera, es suficiente realizar una combinación de las operaciones de interpolación y diezmado.

1.1. Interpolación de una señal

El siguiente programa permite generar y visualizar una señal discreta antes y después de aumentar la frecuencia de muestreo mediante la técnica de interpolación.

```
clear all
TT=1.5                                  % tiempo total
fact =5;                                % factor de interpolación
dt=0.001;                               % periodo de muestreo en segundos
t = 0:dt:TT;                            % vector de instantes
x = sin(2*pi*30*t) + sin(2*pi*60*t);    % generación de la señal
y = interp(x,fact);                     % interpolacion
stem(x(1:30));
```

```
title('Señal original');
figure
stem(y(1: (30*fact)));
title('Señal Interpolatada');
```

1.1.1. Analice cada uno de los comandos del programa anterior y explique el papel de las instrucciones más importantes.

1.1.2. Implemente una función con el programa de §1.1. Ejecútelo y observe las señales obtenidas. Indique si la señal remuestreada conserva las mismas características de la señal original (compare gráficamente la señal discreta y la señal continua). Si considera necesario efectúe un *zoom* a las zonas de interés de la imagen. Observe el eje horizontal y modifique el programa para que se visualice la escala de tiempo adecuadamente en segundos.

1.1.3. Realice un programa que permita obtener, a partir de la señal de §1.1, una señal remuestreada a 20, 60 y 120 Hz (cambiar dt por 1/20, 1/60, 1/120, 1/700). El programa debe visualizar simultáneamente las señales en *tiempo discreto* antes y después del remuestreo. Compare con los resultados anteriores en *periodos iguales* de tiempo. Qué efecto se aprecian y cuales pueden ser sus orígenes? Justifique matemáticamente su respuesta.

1.1.4. Indique si en cada caso la señal remuestreada conserva la misma información que la señal original. Qué indica el eje horizontal? Justifique cada respuesta.

1.2. Diezmado de una señal

Con base en el programa del numeral §1.1, efectúe un programa para aplicar la operación de *diezmado* mediante la función *decimate* de Matlab. Luego, repita los numerales del §1.1.1 hasta el §1.1.4 .

1.3. Remuestreo por un factor racional (no entero)

1.3.1. Realice una función **remuestreo** que permita aplicar las operaciones de diezmado e interpolación a una señal discreta utilizando las funciones *interp* y *decimate* (no se debe utilizar la función *resample* de Matlab). El programa debe recibir como parámetros de entrada la señal a remuestrear, la frecuencia original de muestreo y la nueva frecuencia. El algoritmo debe calcular el factor de modificación y determinar si es racional o no antes de aplicar el remuestreo (ver función *rat* de Matlab). El programa debe visualizar la señal original y las procesadas en una misma ventana. El eje horizontal debe presentarse en unidades de tiempo.
Es indiferente el orden de aplicación de las operaciones de *interp* y *decimate*?
Explique el algoritmo y consigne el código del programa.

1.3.2. Pruebe la función desarrollada para por lo menos tres señales. Consigne, analice y compare los resultados al aplicar la función *resample* de Matlab. Tenga en cuenta en la

comparación aspectos tales como facilidad, tiempo de ejecución (ver funciones tic,toc, clock, etime) y precisión numérica.

1.3.3. Explique la importancia del remuestreo en el tratamiento digital de señales y cite por lo menos dos ejemplos prácticos donde se puede hacer uso del remuestreo.

2. Generación y Evaluación de Funciones de Transferencia

Cualquier señal digital que pueda representarse como la solución de una ecuación de diferencia con coeficientes constantes puede expresarse en Matlab como una función racional. En forma general la ecuación de diferencia está dada por:

$$\sum_{k=0}^{N} a_k \; y(n-k) = \sum_{k=0}^{M} b_k \; x(n-k) \qquad \text{[ec. 1]}$$

luego su transformada z estará dada por,

$$H(z) = \frac{\sum_{k=0}^{M} b_k \; z^{-k}}{\sum_{k=0}^{N} a_k \; z^{-k}} = \frac{B(z)}{A(z)} \qquad \text{[ec. 2]}$$

En Matlab los polinomios $B(z)$ y $A(z)$ se representan como vectores b y a que contienen sus coeficientes.

Por ejemplo, a=[1 –1.5 0.99] representa el polinomio $A(z) = 1 - 1.5\, z^{-1} + 0.99\, z^{-2}$

2.1. El siguiente conjunto de instrucciones evalúa una función de transferencia de un filtro discreto en el círculo unitario del plano-z. (lo cual equivale a tener su respuesta en frecuencia). El filtro tiene función de transferencia $H(z) = \frac{1}{1 + 0.8\, z^{-1}}$

```
clear all
n= 1024;          % número de muestras (preferiblemente una potencia de 2)
b=[1];            % coeficientes del polinomio numerador
a=[1  0.8];       % coeficientes del polinomio denominador
freqz(b,a, n);    % graficar respuesta en frecuencia
```

Analice cada una de las instrucciones del programa anterior y explique el papel de las sentencias más importantes. Realice una función de Matlab para que acepte a cualquier función de transferencia.

2.2. Ejecute el programa de §2.1. y a partir de los diagramas obtenidos determine la frecuencia de corte de –3 dB y la fase en la misma frecuencia de corte. Verifique con los valores obtenidos en los vectores resultantes de las funciones. (Ver help de la función *freqz*). Qué tipo de filtro implementa H(z)?

2.3. Cómo se relaciona la información del eje horizontal con la frecuencia en Hertz? Justifique su respuesta. Calcule analíticamente la frecuencia de corte en hertz.

2.4. Desarrolle una función en Matlab que realice la misma operación que el código del numeral §2.1., pero que además presente como parámetro a la frecuencia de muestreo, fs, y visualice las gráficas en una misma ventana y con el eje horizontal expresado en hertz.

2.5. Repita los numerales §2.1. hasta §2.4, pero modifique el programa para evaluar la función de transferencia $H(z) = \frac{1}{1 - 0.8\,z^{-1}}$

2.6. Investigue cómo se puede obtener la respuesta en frecuencia H(w) a partir de una función de transferencia expresada en el dominio, H(z). Justifique su respuesta. (Analice el resultado y la función de los programas del numeral §2)

3. Respuesta de un sistema LTI expresado como una función racional en z

3.1. La respuesta de un sistema caracterizado por su función de transferencia puede obtenerse en Matlab con ayuda del comando **filter( )**. El siguiente programa permite calcular la respuesta de un sistema discreto *al impulso* a partir de los polinomios de la función de transferencia H(z).

```
%H(z) = 1/(1 + 0.8/ z)      Función de transferencia de un solo polo (F.P.A.)
clear all
nm=65;                      % Numero de muestras
b=[1];                      % Coeficientes del polinomio
a=[1 0.8];

ceros(1:nm-1)=0;            % Señal  impulso
impulso=[1 ceros];

h= filter(b,a,impulso);     % Calculo de la respuesta del sistema

subplot(2,1,1); stem(impulso);  % Visualizacion entrada
xlabel('Instantes de muestreo');
title('Señal de entrada')
ylabel('Impulso');

subplot(2,1,2); stem(h);    % Visualizacion salida
xlabel('Instantes de muestreo');
title('Respuesta al impulso');
ylabel('h(n)');
```

3.2. Analice cada instrucción y ejecute el programa de §3.1 en una función. Consigne los resultados.

3.3. Interprete los resultados del programa anterior. Cómo se puede calcular la frecuencia de corte teórica en Hertz a partir de h(n)? Analice la transformada z inversa de H(z).

3.4. Repita los numerales §3.1 a §3.3, para la siguiente función de transferencia $H(z) = 1/(1- 0.8/ z)$.

3.5. Con base en el programa de §3.1, implemente una función en Matlab que permita visualizar en una misma ventana la señal de entrada, $x(n)$, y de salida, $y(n)$, del filtro en rango de muestras indicado por el usuario (menor o igual al número máximo de las muestras que presenta la señal de entrada). La función debe permitir obtener la respuesta para cualquier tipo de señal discreta y visualizar mensajes de error con la función ***msgbox***. Consigne el código y explique el algoritmo usado.

3.6. Pruebe el programa implementado en §3.5 para las señales de entrada dadas y verifique matemáticamente las respuestas obtenidas.

a. Un escalón unitario
b. Una senoidal de amplitud 1 y frecuencia 1.

Consigne resultados.

3.7. Repita §3.6 b, para señales senoidales con frecuencias f_1=1 KHz, f_2=10 KHz, f_3= 50KHz y amplitud igual a 1. (Para este análisis es mejor seleccionar una sola frecuencia de muestreo? Que valor mínimo de frecuencia de muestreo se requiere?) Explique si los resultados obtenidos son consistentes con la respuesta dada al numeral §3.3. Es decir, se aprecia la acción de filtrado? Visualice las señales con el eje horizontal expresado en unidades de tiempo.

3.8. Indique qué se debe hacer para incluir condiciones iniciales y/o conocer las condiciones finales (Qué significado presentan?) del sistema en la función desarrollada en §3.5. Implemente, valide y consigne su programa. Incluya análisis de los resultados obtenidos.

4. Respuesta de un sistema LTI representado por ecuaciones de diferencia

4.1. Implemente una función en Matlab para calcular la respuesta de un sistema LTI representado por su ecuación de diferencias con coeficientes constantes utilizando la *evaluación recursiva*. La función debe presentar el siguiente formato,

function [y] = eqdif_rec (b,a, x, y0)

donde,

y → vector que contiene la secuencia de salida
x → vector que contiene la secuencia de entrada
b → vector que contiene los coeficientes de x(n) en la ecuación de diferencia (ec. 1)
a → vector que contiene los coeficientes de y(n) en la ecuación de diferencia (ec. 1)
y0 → vector que contiene las condiciones iniciales.

El programa debe verificar si el número de condiciones iniciales corresponde con el orden de la ecuación de diferencia.

La función *eqdif_rec( )* debe también graficar en una misma ventana la secuencia de entrada x(n) y la secuencia de salida y(n).

Consigne el código y explique el algoritmo utilizado.

4.2. Pruebe la función *eqdif_rec( )* con las ecuaciones:

a. $y(n) - 1.8\cos\left(\frac{\pi}{16}\right) y(n-1) + 0.81\, y(n-2) = x(n) + 0.5\, x(n-1)$

b. $y(n) = \frac{5}{6} y(n-1) - \frac{1}{6} y(n-2) + x(n)$

Para las señales de entrada dadas, considerando *condiciones iniciales* diferentes de cero tanto de valores pequeños como grandes. Genere suficientes puntos (>60) para obtener una buena visualización de los resultados.

a. Impulso unitario
b. Escalón unitario
c. Exponencial de la forma x(n)= a^n u(n) para | a | < 1

Consigne y analice los resultados.

4.3. Verifique matemáticamente que algunos valores (por lo menos 4) de las respuestas obtenidas en §4.2. corresponden a las esperadas. Consigne cálculos.

4.4. Repita los numerales §4.1. a §4.3. y genere una función denominada *eqdif_hpt* que permita calcular la respuesta ante la entrada nula, en estado nulo, la homogénea, la particular y la total del sistema. Visualice las respuestas simbólica y gráficamente.

Resuelva el problema únicamente para casos de raíces de multiplicidad uno y forma de la respuesta particular no repetida.

El programa debe indicar en una ventana de diálogo cuantas raíces de la ecuación característica son diferentes y si existen repetidas.

Observación: conserve las funciones desarrolladas durante la práctica, puesto que estas pueden servirle en los laboratorios siguientes.

5. Consulta

5.1. Realice el resumen de un artículo que describa una aplicación de procesamiento digital de señales: Señales fisiológicas, señales sísmicas, señales sensoriales o aplicación en comunicaciones.
Consigne la referencia bibliográfica.

6. Informe

6.1. Presente un informe escrito claro en donde se consigne el procedimiento, los programas, las señales, las justificaciones de las respuestas y los resultados obtenidos. Igualmente incluya el análisis e interpretación de los resultados. También consigne las conclusiones, observaciones y la literatura consultada. Utilice en el informe la misma numeración de la guía de laboratorio.

6.2. Desarrolle ayudas para cada función implementada que puedan invocarse desde el comando *help* de Matlab.

Notas: La omisión de alguno de los ítems en el informe representa una disminución de la nota.
El informe debe hacer referencia ordenada a cada uno de los puntos de la guía.
Para facilitar la sustentación ante el profesor, realice scripts donde se definan los datos y se invoquen las funciones desarrolladas.