Universidad Nacional de Rosario
Facultad de Ciencias Exactas, Ingeniería y Agrimensura
Escuela de Ingeniería Electrónica

(A3) Sistemas y Señales I

Trabajo Práctico Nº 1
Introducción a Matlab

Autores: Cátedra SyS-I

Febrero de 2025

Trabajo Práctico Nº 1
Introducción a Matlab

1. Objetivos

El principal objetivo de este Trabajo Práctico es introducir el programa de cálculo científico **Matlab ®**[4]. El programa **Matlab** (el nombre corresponde a la abreviación de **Mat**rix **Lab**oratory) es una potente herramienta de cálculo numérico y visualización de uso muy difundido a nivel mundial en los ambientes industrial y académico en tareas de investigación, desarrollo y diseño en diversas áreas tales como procesamiento de señales, control y comunicaciones [4].

En este Trabajo Práctico se pretende que el alumno

1. se familiarice con los comandos **Matlab** que posibilitan
   - la representación y cálculo matricial,
   - la generación de señales a valores reales o complejos, y su visualización,
   - la creación de programas **Matlab** (llamados **M-files)** en las modalidades **function-files** y **script-files**,
   - el almacenamiento de los resultados de una sesión **Matlab** y el ingreso de datos en el espacio de trabajo de **Matlab,**
2. se familiarice con las operaciones de convolución y correlación de señales, y su implementación en **Matlab.**
3. se familiarice con el algoritmo de integración numérica Euler de 1er. Orden, y con su aplicación en la simulación digital de sistemas dinámicos simples.

2. Introducción

Matlab es un lenguaje de alto nivel para cálculo científico que integra, en un único ambiente software, rutinas de cálculo, visualización y programación. El programa es de fácil uso ya que los problemas se pueden formular usando una notación matemática estándar. La representación básica de los datos en **Matlab** es en forma matricial, pero con la ventaja de que no es necesario especificar las dimensiones de las matrices involucradas [4].

Usos típicos de **Matlab** incluyen:

- Cálculo numérico.
- Desarrollo de algoritmos.
- Modelado, simulación y desarrollo de prototipos.
- Análisis y visualización de datos.
- Construcción de gráficas.
- Desarrollo de aplicaciones en distintas áreas científicas y tecnológicas.

Matlab es un sistema abierto al cual el usuario puede incorporar nuevas funciones para su uso en aplicaciones particulares. Existen también extensiones de **Matlab** denominadas **Toolboxes**, que son colecciones de funciones **Matlab** que permiten resolver problemas específicos en diversas áreas de ciencia e ingeniería. Actualmente existen Toolboxes en áreas tales como Control [5], Procesamiento de Señales [6], Identificación de Sistemas [7], Procesamiento de Imágenes, Redes Neuronales, Lógica Difusa, Wavelets, etc..

En este primer Trabajo Práctico se introducen los comandos **Matlab** básicos que se emplearán en los Trabajos de Laboratorio de la Asignatura. En la Sección 3., que es de caracter tutorial, se ilustra a través de ejemplos cómo pueden representarse señales en Tiempo Continuo (TC) y en Tiempo Discreto (TD), y cómo las mismas pueden visualizarse mediante el uso de los comandos `plot`, `stem`, y `mesh`. Así mismo se indica cómo pueden ingresarse datos en el espacio de trabajo de **Matlab**, y cómo pueden almacenarse los resultados de una sesión **Matlab**. Se detallan también es esta sección los comandos **Matlab** para control de flujo de programas, y se describen las principales características de los denominados **M-files** en las modalidades **function-file** y **script-file**. Los experimentos que deberán resolverse en la sesión de Laboratorio se plantean en la Sección 4., en tanto que las referencias bibliográficas se detallan en la Sección 5. Los problemas cuya solución deberá reportarse en el Informe de Laboratorio se incluyen en el documento ***Problemas a incluir en el Informe de Laboratorio***, en tanto que las pautas generales para la elaboración del Informe de Laboratorio se incluyen en el documento ***Pautas para la Elaboración del Informe de Laboratorio***.

3. Comandos básicos Matlab

3.1. Ingreso de datos al espacio de trabajo

Matlab trabaja esencialmente con una única clase de objetos, nominalmente matrices, que pueden ser a valores reales o complejos. Todas las variables en **Matlab** representan matrices. Una matriz 1 x 1 es interpretada como un escalar y matrices con una sola columna o una sola fila son interpretadas como vectores (columna o fila respectivamente). Las matrices en **Matlab** pueden básicamente ser ingresadas en las siguientes formas:

- Ingresadas como una lista de elementos. Por ejemplo el comando

```
>> A = [1,2,4;8,16,32;64,128,254]


A =
     1     2     4
     8    16    32
    64   128   254
```

  asignará a la variable `A` la matriz 3 x 3 arriba indicada. El símbolo ',' o un espacio, separa elementos en una fila, en tanto que el símbolo ';' indica el inicio de la siguiente fila.

- Generadas por líneas de comando u otras funciones de **Matlab**. Por ejemplo el comando

```
>> B = rand(3,2)


B =
    0.9501    0.4860
    0.2311    0.8913
    0.6068    0.7621
```

  asignará a la variable `B` la matriz 3 x 2 arriba indicada, cuyos elementos son números aleatorios entre 0 y 1 con distribución uniforme.

- Creadas en M-files (ver Sección 3.2.). Por ejemplo si se crea un archivo `datos1.m` cuyo contenido es

```
C = [1 4 9; 16 25 36];
D = [1 2; 8 16; 32 64];
```

  y se ejecuta el comando

```
>> datos1
```

  entonces, en el espacio de trabajo quedarán definidas las variables `C` y `D` con los valores arriba indicados.

- Ingresadas desde archivos de datos externos con el comando `load`. Por ejemplo, si externamente se crea un archivo conteniendo

```
16.0     3.0   2.0   13.0
 5.0    10.0  11.0    8.0
 9.0     6.0   7.0   12.0
 4.0    15.0  14.0    1.0
```

  y se lo salva con el nombre `datos2.mat`, entonces el comando

```
>> load datos2.mat
```

ingresará en el espacio de trabajo una variable, `datos2`, a la cual le asignará la matriz almacenada en el archivo `datos2.mat` (siempre que el archivo esté ubicado en un directorio incluído en el Matlab-Path[1]).

3.2. Especificación de elementos de una matriz y de submatrices

Un elemento particular de una matriz o un vector se referencia en la forma matemática usual.

Por ejemplo:

- `A(1,2)` denota el elemento de la primer fila y segunda columna de la matriz A,
- `A(:,2)` denota la segunda columna de A,
- `A(3,:)` denota la tercera fila de A, y
- `A(2:3,1:4)` denota la submatriz formada por las filas 2 y 3 y las columnas 1 a 4 de la matriz A.

Matlab sólo acepta índices enteros positivos para denotar los elementos de una matriz o vector.

3.3. Operaciones elementales con matrices

Las siguientes operaciones básicas con matrices están disponibles en **Matlab**:

+	adición
-	substracción
*	multiplicación
^	potencia
‘	transpuesta conjugada
.’	transpuesta
\	división por izquierda
/	división por derecha

Por supuesto, estas operaciones con matrices también se aplican a escalares (matrices de 1 x 1). Si las dimensiones de las matrices no son compatibles para realizar la operación aparece un mensaje de error, excepto en el caso de realizar una operación entre una matriz y un escalar, en cuyo caso se realiza la operación entre cada elemento de la matriz y el escalar (para las operaciones de suma, resta, multiplicación y división).
Es importante destacar que las operaciones arriba mencionadas son operaciones matriciales, si bien las operaciones de suma y resta actúan, por definición, componente-a-componente. Puede lograrse que las operaciones *, ^, \, y /, actúen también componente-a-componente, precediendo al símbolo correspondiente por un punto ‘.’ . Por ejemplo, si queremos elevar cada elemento de una matriz al cuadrado, tendríamos

```
>> E = [1 2 3; 4 5 6];
>> F = E.^2
F =
   1  4  9
  16 25 36
```

Los símbolos \ y / denotan la división por izquierda y por derecha, respectivamente. La interpretación precisa de los comandos `A\B,` y `A/B` se puede obtener recurriendo al help-on line de **Matlab**, tipeando

```
>> help slash
```

3.4. Gráficas

En **Matlab** hay diversos comandos para la realización de gráficas en 2 y 3 dimensiones. En dos dimensiones, los comandos básicos son las funciones `plot` y `stem,` en tanto que para gráficas en 3 dimensiones los comandos

[1] **Matlab** busca las funciones y archivos de datos en una lista ordenada de directorios que se denomina Matlab-path. Pueden agregarse, eliminarse, o alterar el orden de los directorios 'clickeando' en la opción 'Set Path' del menú 'File' en la 'Command Window' de **Matlab**.

básicos son `mesh` y `surf`. Los siguientes ejemplos ilustran la utilización de los comandos `plot`, `stem` y `mesh`, para la representación de señales en una y dos dimensiones.

Ejemplo 1:

Consideremos la siguiente señal analógica

$$x(t) = sin(2\pi 50t) + 0.5sin(2\pi 150t),$$

y supongamos que se muestrea esta señal con una frecuencia $F_s = 600\,\text{Hz}$ (el lector interesado puede verificar que la frecuencia de muestreo elegida cumple con el Teorema de Muestreo de Shannon, es decir $F_s > F_N$, donde F_N es la tasa de muestreo de Nyquist). La señal en tiempo discreto resultante $x(n)$ se obtiene tomando los valores de $x(t)$ en los instantes $nT_s = \frac{n}{F_s}$, donde T_s es el denominado intervalo de muestreo. Es decir, en el caso del ejemplo se tiene

$$x(n) = sin\left(\frac{2\pi 50}{600}n\right) + 0.5sin\left(\frac{2\pi 150}{600}n\right).$$

Las señales en tiempo continuo $x(t)$ y en tiempo discreto $x(n)$ pueden graficarse usando los comandos **Matlab** `plot` y `stem`, respectivamente. La siguiente sucesión de comandos permite generar y graficar estas señales.

```
» t=0:0.0001:0.04;
» x=sin(2*pi*50*t')+0.5*sin(2*pi*150*t');
» td=0:1/600:0.04;
» xn=sin(2*pi*50*td')+0.5*sin(2*pi*150*td');
» plot(t,x)
» hold on
» stem(td,xn)
» grid;
» xlabel('Tiempo [seg]'); ylabel('x(t)');
```

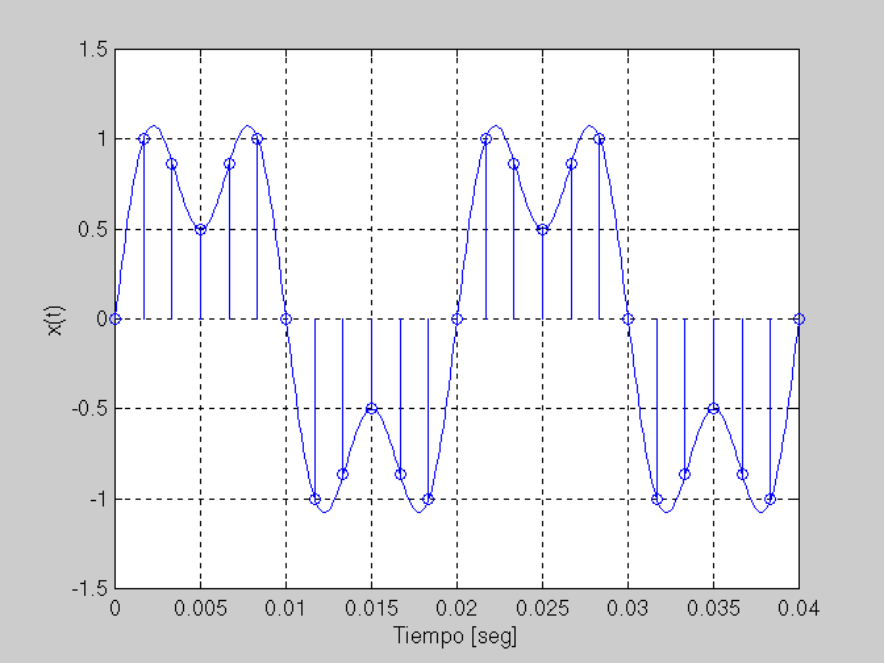

Figura 1: Salida gráfica correspondiente al Ejemplo 1.

La gráfica correspondiente está representada en Figura 1.

❑

Ejemplo 2:

Supongamos que se desea generar y graficar las señales en tiempo continuo $x_1(t)$ y $x_2(t)$ definidas como

$$x_1(t) = \cos(2\pi 50t)$$
$$x_2(t) = \cos(2\pi 550t)$$

y las correspondientes señales en tiempo discreto que se obtienen muestreando $x_1(t)$ y $x_2(t)$ con una frecuencia $F_s = 500$ Hz. Notar que las dos señales tienen asociada la misma señal muestreada (el lector interesado puede probar este hecho). Los comandos **Matlab** para generar y graficar las señales se incluyen a continuación, y la gráfica resultante se muestra en la Figura 2.

```
>> Fs = 500;                    % frecuencia de muestreo en Hz
>> t1 = [0:1/(20*Fs):1]';       % vector tiempo "continuo"
>> F1 = 50;                     % frecuencia de la señal x1
>> F2 = F1 + Fs;                % frecuencia de la señal x2
>> x1 = cos(2*pi*F1*t1);        % señal x1
>> x2 = cos(2*pi*F2*t1);        % señal x2
>> t = [0:1/Fs:1]';             % vector tiempo discreto
>> xm = cos(2*pi*F1*t);         % señal muestreada
>> plot(t1,x1,'b-',t1,x2,'r-',t,xm,'go'); % gráfica
>> axis([0 0.02 -1 1]);         % escalado en los ejes
>> xlabel('Tiempo [s]');        % rotulado del eje de abscisas
>> grid;                        % grilla
>> title('Señales x1(t), x2(t) y muestreadas');      % título
```

❑

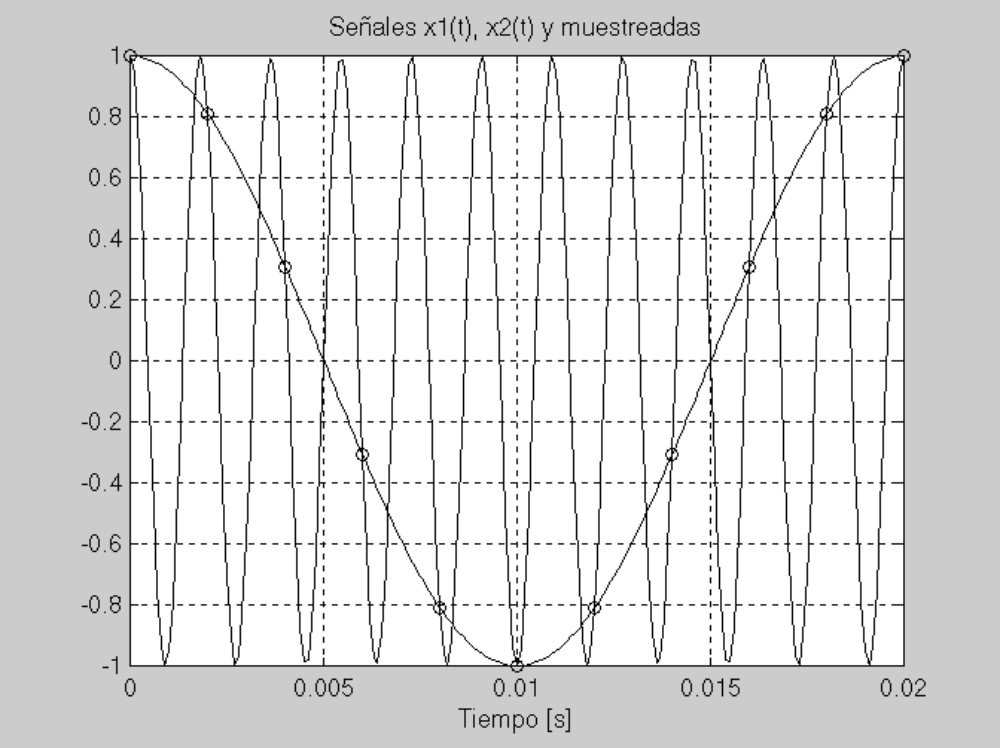

Figura 2: Salida gráfica correspondiente al Ejemplo 2.

Ejemplo 3:

En este ejemplo mostramos cómo pueden realizarse varias gráficas en una misma figura usando el comando `subplot`.

Consideremos la siguiente señal exponencial compleja

$$x(n) = e^{j\frac{n}{3}}.$$

La siguiente sucesión de comandos **Matlab** permite generar la señal y graficar la parte real y la parte imaginaria de la señal en una misma figura.

```
>> n=0:30;
>> x=exp(j*n/3);
>> subplot(211), stem(n,real(x));
>> xlabel('Indice "n"');title('Parte Real');
>> subplot(212), stem(n,imag(x));
>> xlabel('Indice "n"');title('Parte Imaginaria');
```

La salida gráfica correspondiente se representa en la Figura 3. ❑

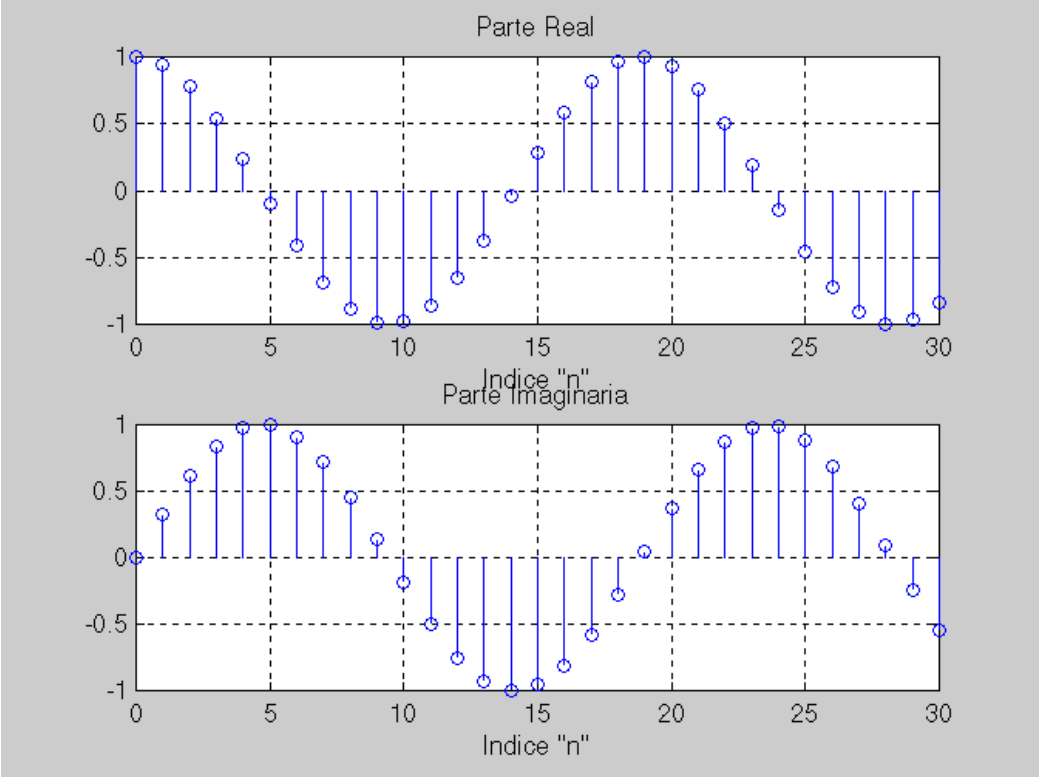

Figura 3: Salida gráfica correspondiente al Ejemplo 3. La gráfica superior corresponde a la parte real de $x(n)$ *, y la gráfica inferior corresponde a la parte imaginaria de* $x(n)$ *.*

Ejemplo 4:

En este ejemplo ilustramos el uso del comando `mesh` para realizar gráficas en 3 dimensiones. Consideremos la función *sinc* en dos dimensiones, definida como

$$f(x,y) = \frac{sin(x^2+y^2)}{x^2+y^2},$$

y supongamos que queremos graficar $f(x,y)$ para x e y en el rango $[-8,8]$. La siguiente sucesión de comandos **Matlab** permitirá realizar esta operación [5]. La correspondiente salida gráfica está representada en la Figura 4.

```
>> [x,y]=meshgrid(-8:0.5:8);
>> r=sqrt(x.^2+y.^2)+eps;
>> z=sin(r)./r;
>> mesh(x,y,z,ones(size(z)));       % en colores: mesh(x,y,z)
>> xlabel('x'); ylabel('y'); zlabel('f(x,y)');
```

❑

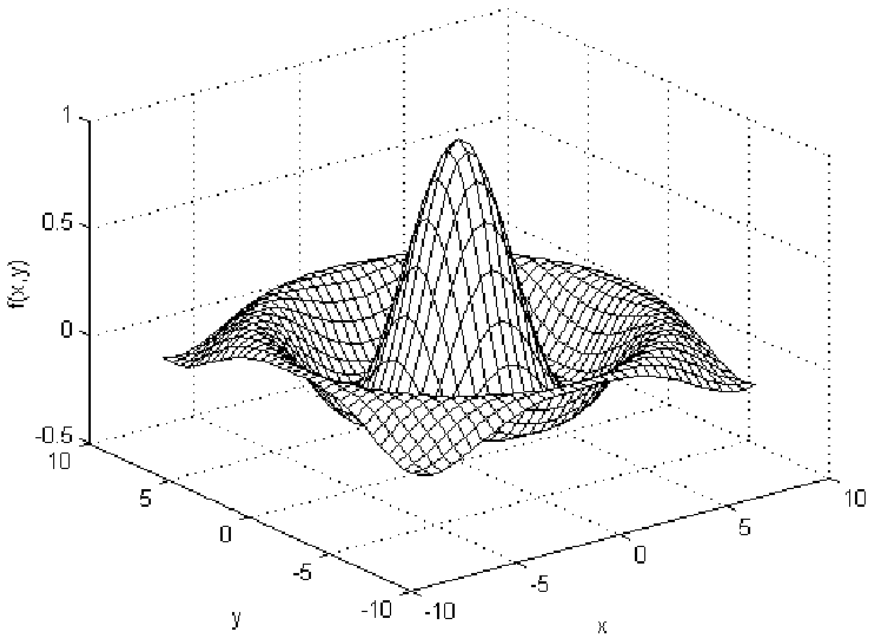

Figura 4: Salida gráfica correspondiente al Ejemplo 4.

3.5. Control de Flujo de Programas Matlab

Hay cuatro comandos básicos para el control de flujo en un programa **Matlab**. Los comandos, que se describen a continuación, son muy similares a los respectivos comandos de control de flujo de otros lenguajes de programación.

- **if** (junto con **else**,y **elseif**): Evalúan una expresión lógica y ejecutan un grupo de sentencias de acuerdo al resultado de la expresión. En su forma más simple la sintaxis es:

```
if expresión_lógica1
      sentencias1;
elseif expresión_lógica2
      sentencias2;
else
      sentencias3;
end
```

  Un ejemplo de aplicación donde se emplea este comando es el script-file `positividad22.m` que se presenta en la Subsección 3.2. (Ejemplo 5).

- **for**: Permite la ejecución reiterada de un grupo de sentencias un número fijo y predeterminado de veces. La sintaxis es:

```
for indice = vinicial:incremento:vfinal
sentencias;
end;
```

- **while**: Permite la ejecución reiterada de un grupo de sentencias un número indefinido de veces bajo control de una condición lógica. La sintaxis es:

```
while expresión_lógica
sentencias;
end;
```

- **switch**: Permite la ejecución de grupos de sentencias dependiendo del valor de una variable o expresión. La sintaxis es:

```
switch caso
caso1
sentencias1;
caso2
sentencias2;
        ⋮
otherwise
sentencias3;
end;
```

Como puede verse, estos comandos operan en forma similar a los análogos en otros lenguajes de programación (ver [4] por mayores detalles).

3.6. Creación de M-files

Matlab permite ejecutar secuencias de comandos almacenados en un archivo. Estos archivos deben tener la extensión '.m' , y por eso se denominan **M-files.** Existen básicamente dos tipos de **M-files**: los denominados **function-files** y **script-files**.

- **Script-files**

  Un script-file consiste de una sucesión de líneas de comando **Matlab**. Si por ejemplo el archivo tiene el nombre `positividad22.m` , entonces el comando **Matlab**

  ```
  >> positividad22
  ```

  hace que se ejecuten los comandos del archivo en el orden en que aparecen. Las variables en el script-file son globales, y por lo tanto cambiarán el valor de las variables del mismo nombre que estén definidas en el espacio de trabajo (en la misma sesión), por el nuevo valor calculado.

Ejemplo 5:

El siguiente script-file (lo llamaremos `positividad22.m`) permite determinar si una matriz 2x2 es definida positiva[2] usando el Test de Sylvester [3] :

```
A = [1,2;3,4];
D = det(A);
if (A(1,1)> 0) & (D > 0),
disp('A es definida positiva');
else
disp('A no es definida positiva');
end;
```

❑

[2] Una matriz A se dice **definida positiva**, y se nota $A > 0$, si $x^T Ax > 0$, donde x es un vector columna de dimensiones apropiadas.

[3] **Test de Sylvester:** Una matriz A es definida positiva si todos los menores principales sucesivos son positivos.

Los script-files son muy útiles cuando se tienen que ingresar datos en la forma de matrices de grandes dimensiones, ya que pueden editarse fácilmente para corregir errores. Esto sería muy trabajoso si los datos se ingresan por teclado.

- **Function-files**

Los function-files permiten extender la biblioteca de funciones **Matlab** para el uso en aplicaciones específicas. A diferencia de los script-files, las variables en un function-file son locales, por lo que variables del mismo nombre en el espacio de trabajo de **Matlab** no son modificadas cuando la función es ejecutada.

El siguiente ejemplo muestra la estructura típica de un function-file.

Ejemplo 6:

Un function-file que permita calcular la norma de Frobenius[4] de una matriz sería

```
function  F = Frobenius(A)
% Esta función calcula la norma de Frobenius de una matriz A
% La norma de Frobenius de una matriz (m x n) A=(a_ij) esta
% definida como:
%        F^2= Sum_{i=1,m} Sum_{j=1,n} |a_ij|^2
F = sqrt(sum(sum(abs(A).^2)));
```

❑

La primera línea de un function-file siempre comienza con la palabra *function* seguida de una expresión donde se declara el nombre de la función (en el caso del Ejemplo 6, `Frobenius`), y se indica cómo la función debe ser llamada desde el espacio de trabajo de **Matlab**, y cuáles son los argumentos de entrada y salida de la función. En el Ejemplo 6, el argumento de salida es F y el argumento de entrada es la matriz A. Las líneas que comienzan con el caracter '%' corresponden a comentarios. Las líneas de comentarios que están inmediatamente debajo de la primer línea pueden utilizarse para aprovechar las facilidades de **help on-line** de **Matlab**. El texto correspondiente a esas líneas aparecerá en el espacio de trabajo cuando se tipee

```
>> help function-name
```

y por lo tanto las mismas se utilizan generalmente para describir la operación que realiza la función. Es recomendable incluir *siempre* esta documentación en un function-file. El nombre del function-file debe ser de la forma `function-name.m` (en el caso del Ejemplo 6, `Frobenius.m`).
Una vez que el function-file ha sido guardado en el disco, y el Matlab-path[5] ha sido "seteado" convenientemente, la función se puede llamar desde el espacio de trabajo. Por ejemplo para el caso de la función Frobenius podríamos tener

```
>> A = randn(5,4);
>> F = Frobenius(A)

F =
    3.8594
```

En este caso la matriz A, de dimensiones 5 x 4, es generada por la función `randn` de **Matlab**, y sus elementos son variables aleatorias entre 0 y 1 con distribución normal.
Una función **Matlab** puede también tener múltiples argumentos de salida. Por ejemplo, la siguiente función computa la norma de Frobenius y la norma-1 de una matriz[6].

---

[4] La **norma de Frobenius** de una matriz $A=(a_{ij}) \in \Re^{m\times n}$ se define como: $\|A\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|^2}$.

[5] **Matlab** busca las funciones en una lista ordenada de directorios que se denomina Matlab-path. Pueden agregarse, eliminarse, o alterar el orden de los directorios 'clickeando' en la opción 'Set Path' del menú 'File' en la 'Command Window' de Matlab.

[6] Por supuesto, existe una función **Matlab** que computa distintas normas (incluídas la norma de Frobenius y la norma-1) de una matriz. El nombre de esta función es `norm`.

Ejemplo 7: Function-file para el cómputo de la norma de Frobenius y la norma -1 de una matriz.

```
function  [F,N1] = Norma(A)
% Esta función calcula la norma de Frobenius y la norma-1 de la
% matriz  A.
% La norma de Frobenius de una matriz (m x n) A=(a_ij) esta
% definida como:
%        F^2= Sum_{i=1,m} Sum_{j=1,n} |a_ij|^2
% La norma-1 esta definida como
%           N1 =  max_j sum_i |a_ij|
F = sqrt(sum(sum(abs(A).^2)));
N1 = max(sum(abs((A))));
```

❑

La forma de editar M-files, ya sea de tipo function-files (implementado por el ususario) o script-file es usando un editor incorporado a **Matlab** (versión 5 o posteriores), el denominado **Matlab Editor/Debugger**, al cual se accede desde la opción 'File' de la barra de menú, en la ventana de comando, cada vez que se abre un M-file (nuevo o ya existente).

4. Experimentos de Laboratorio

En esta sección se plantean los experimentos de laboratorio que deberán resolverse usando **Matlab**. La resolución de los problemas requerirá la implementación de script-files y/o function-files, y el uso de los comandos básicos explicados en la sección anterior.

Problema 1:

La figura 5 corresponde a un circuito RC.

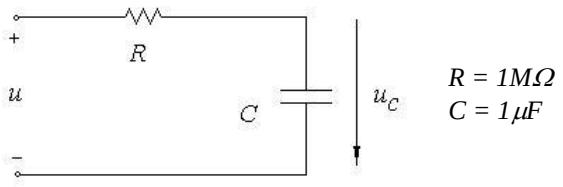

Figura 5: Circuito RC.

a. **Trabajo Previo:** Escriba la ecuación diferencial que describe el comportamiento entrada-salida del circuito. Tome como entrada la tensión $u(t)$, y como salida la tensión $u_c(t)$.

b. **Trabajo Previo:** Una forma de discretizar una ecuación diferencial de primer orden de la forma

$$\dot{x} = f(x,u)$$

de manera que pueda ser implementada en un algoritmo, es discretizar el tiempo continuo de la forma $t = nT$ (donde n es entero y T representa el período de muestreo), y aproximar la derivada respecto al tiempo por el cociente incremental

$$\frac{dx}{dt} \approx \frac{x(n+1)-x(n)}{T}$$

donde por simplicidad de notación se denota $x(n) = x(nT)$. La aproximación mejora a medida que T se hace más pequeño. Esta aproximación es conocida como la aproximación de Euler de primer orden. Resulta entonces la ecuación en diferencias:

$$\frac{x(n+1)-x(n)}{T} = f\big(x(n),u(n)\big) \quad \Rightarrow \quad x(n+1) = x(n) + T\ f\big(x(n),u(n)\big)$$

Usando esta aproximación escriba la ecuación en diferencias para el sistema dado en Figura 5.

c. Escriba un **script-file** usando **Matlab** que implemente el sistema en tiempo discreto descripto, de manera de obtener la respuesta $u_c(n)$ cuando el sistema es excitado con un escalón de amplitud A (es decir, con una entrada $u(n) = A\mu(n)$), a partir de una condición inicial $u_c(0)$ que representa

la tensión inicial del capacitor. Adopte como periodo de muestreo $T = RC / 20$. Repita para $T = RC$. ¿Que conclusión puede sacar respecto a la elección del intervalo de muestreo? Realice la simulación con un tiempo final tal que el capacitor alcance el 99% de su valor final de carga.

Nota: Recuerde que el nombre del **script-file** debe ser de la forma: **script_name.m** .

d. Escriba un function-file usando Matlab que calcule la tensión en el capacitor C a partir de los siguientes parámetros de entrada: condición inicial de tensión ($u_c(0)$) y la señal de entrada $u(n)$ (vector finito) . Incluya un help que explique como se usa la función.

e. Usando el comando `plot` (o si prefiere el comando `stem`) grafique la respuesta del sistema a la entrada escalón para diferentes condiciones iniciales. Usando los comandos `xlabel, ylabel, title,` y `legend`, rotule los ejes, ponga un título, e identifique las distintas curvas, respectivamente.

f. Resuelva (analíticamente) la ecuación diferencial del apartado a. para una entrada $u(t) = A\mu(t)$ y muestree la solución usando el mismo intervalo de muestreo que en los apartados anteriores. Compare gráficamente la respuesta obtenida con la correspondiente al apartado c. .

❑

Problema 2:

La solución de ecuaciones en diferencias, lineales a coeficientes constantes viene dada por la combinación lineal de exponenciales decrecientes de la forma

$$x(n) = a^n ,$$

donde a es un escalar que puede tomar valores complejos.

a. Escriba un **function-file** usando **Matlab** que permita generar una señal exponencial en tiempo discreto de la forma arriba descrita, y que grafique la señal. Los argumentos de entrada de la función deberán ser:

- La constante a ,
- El valor inicial del índice $n : n_0$,
- La longitud L de la señal generada,

en tanto que el argumento de salida deberá ser el vector x que representa la señal generada.

b. Genere 2 gráficas, una para un valor de a real puro y otra para un valor de a complejo (en cuyo caso deberá dibujarse parte real e imaginaria por separado)

❑

Problema 3:

La respuesta a una entrada arbitraria $u(n)$ de un sistema lineal estacionario en TD caracterizado por su respuesta al impulso $h(n)$, puede calcularse como la (suma de) convolución de las dos secuencias, definida como ([1], [2]):

$$y(n) = h(n) * u(n) = \sum_{k=-\infty}^{\infty} h(n-k)u(k) \qquad (1)$$

a. Escriba un **function-file** usando **Matlab** para computar la respuesta $y(n)$ de un SLE a una entrada arbitraria $u(n)$. Asuma que el sistema es causal (lo que implica que $h(n) = 0$ para $n < 0$) y que la entrada también lo es, por lo que la suma infinita (1), en la definición de la convolución, se convierte en una suma con un número finito de términos y de esta forma es implementable.

NB: Una forma eficiente de implementar la suma de convolución es escribiendo (1) en forma matricial. En este caso, un vector Y conteniendo las salidas $y(n)$ se obtendría como el producto de una matriz (que resulta tener una estructura Toeplitz[7] , y cuyos elementos son la respuesta al impulso $h(n)$) con un vector U conteniendo las entradas $u(n)$.

Los argumentos de entrada de la función podrían ser dos vectores conteniendo las secuencias $h(n)$ y $u(n)$, y el argumento de salida, un vector conteniendo la convolución de las dos secuencias. Si se omite el argumento de salida la función sólo debe mostrar las gráficas de $u(n)$, $h(n)$ e $y(n)$ en una

[7] Una matriz se dice Toeplitz si es constante a lo largo de las diagonales paralelas a la diagonal principal. La función `toeplitz` de **Matlab** permite generar matrices con esa estructura.

misma ventana mediante el uso del comando `subplot`. Para realizar esto puede utilizar la variable `nargout` que cuenta el número de argumentos de salida con el que fue llamada la función.

b. Emplee la función desarrollada en **a.** para calcular la respuesta a una entrada escalón unitario de un sistema caracterizado por una respuesta al impulso de la forma

$$h(n)=\left(\frac{1}{4}\right)^n \mu(n).$$

c. Usando el comando `subplot` grafique, en una misma ventana, las señales de entrada, salida y respuesta al impulso para el caso del apartado **b.**

❑

Problema 4:

La secuencia de **correlación cruzada** $r_{xy}(\ell)$ de dos señales de energía finita $x(n)$ e $y(n)$ está definida como

$$r_{xy}(\ell)=\sum_{n=-\infty}^{\infty} x(n)y(n-\ell), \qquad \ell=0,\pm1,\pm2,\cdots,$$

o equivalentemente

$$r_{xy}(\ell)=\sum_{n=-\infty}^{\infty} x(n+\ell)y(n), \qquad \ell=0,\pm1,\pm2,\cdots$$

en tanto que la secuencia de **autocorrelación** de la señal $x(n)$ está definida como

$$r_{xx}(\ell)=\sum_{n=-\infty}^{\infty} x(n)x(n-\ell)=\sum_{n=-\infty}^{\infty} x(n+\ell)x(n).$$

Cuando se trata de señales causales de longitud finita N, las secuencias de correlación cruzada y autocorrelación se definen como

$$r_{xy}(\ell)=\sum_{n=i}^{N-|k|-1} x(n)y(n-\ell),$$

y

$$r_{xx}(\ell)=\sum_{n=i}^{N-|k|-1} x(n)x(n-\ell),$$

respectivamente, donde $i=\ell, k=0$, para $\ell\geq 0$, y $i=0, k=\ell$, para $\ell<0$.

a. Escriba un **function-file** usando **Matlab** para computar la secuencia de correlación cruzada de dos señales de energía finita $x(n)$ e $y(n)$. Note que recurriendo a la propiedad

$$r_{xy}(\ell)=x(\ell)*y(-\ell)$$

podría usarse la rutina que calcula la convolución de dos secuencias para calcular la secuencia de correlación cruzada de las mismas.

b. Emplee la función desarrollada en el apartado **a.** para calcular la secuencia de auto-correlación de la siguiente señal.

$$x(n) = \left(\frac{1}{2}\right)^n \mu(n) .$$

Grafique la señal y la secuencia de autocorrelación.

c. En algunas aplicaciones prácticas la correlación se puede usar para identificar la periodicidad de una señal $x(n)$, a partir de muestras de la señal inmersa en ruido. Se tiene entonces

$$y(n) = x(n) + w(n) ,$$

donde $y(n)$ son las muestras de la señal, $x(n)$ es la señal periódica con período desconocido N, y $w(n)$ es una variable aleatoria que representa al ruido aditivo. Si se toman M muestras de la señal $y(n)$, con $M >> N$, y se asume que $y(n) = 0$ para $n < 0$, y $n \geq M$, entonces la secuencia de autocorrelación resulta [1]

$$r_{yy}(\ell) = \frac{1}{M} \sum_{n=0}^{M-1} y(n) y(n-\ell) = r_{xx}(\ell) + r_{xw}(\ell) + r_{wx}(\ell) + r_{ww}(\ell).$$

Como $x(n)$ es periódica, su secuencia de autocorrelación contendrá picos relativamente grandes en $\ell = 0, N, 2N, \ldots$. Además, las correlaciones cruzadas entre las señales $x(n)$ y $w(n)$ tendrán valores relativamente pequeños como consecuencia de que no es esperable que las señales estén relacionadas. Finalmente, la secuencia de autocorrelación del ruido, contendrá un pico en $\ell = 0$, pero debido a su naturaleza aleatoria es esperable que decaiga a cero rápidamente.

Basado en estas observaciones, indique cómo puede estimarse el período de la señal $x(n)$ a partir del análisis de la secuencia de autocorrelación de $y(n)$.

d. Usando las conclusiones del apartado **c.** , y la función del apartado **a.**, determine el período de la señal inmersa en ruido cuyas muestras se incluyen en el archivo `tp1_1.mat`.

❑

5. Referencias

[1] Proakis, John G. & Manolakis, Dimitris G. (1992): ***Digital Signal Processing – Principles, Algorithms, and Applications*** , Second Edition, Macmillan Publishing Company, New York.

[2] Ziemer, Rodger E. & Tranter, William H. & Fannin, D. Ronald (1993**): *Signal and Systems: Continuous and Discrete,*** Third Edition, Macmillan Publishing Company, New York.

[6] The MathWorks, Inc. (1997): ***Matlab, The Language of Technical Computing – Getting Started with Matlab, Version 5.*** – 24 Prime Park Way, Natick, MA 01760-1500.

[7] The MathWorks, Inc. (1996): ***Matlab, Control System Toolbox – User´s Guide, Version 4*** – 24 Prime Park Way, Natick, MA 01760-1500.

[8] The MathWorks, Inc. (1996): ***Matlab, Signal Processing Toolbox – User´s Guide, Version 4*** – 24 Prime Park Way, Natick, MA 01760-1500.

[9] Lennart Ljung (1997): Matlab, System Identification Toolbox - User´s Guide, Version 4 – 24 Prime Park Way, Natick, MA 01760-1500.