Fundamentos de control Industrial

Práctica 1

Javier Rico Azagra

Índice

Objetivo

El objetivo de esta práctica es proporcionar al alumno una visión general del software Matlab-Simulink. Este software es el más empleado en el ámbito de la ingeniería de control, y será empleado en prácticas posteriores como herramienta para la resolución de problemas relacionados con la asignatura.
En esta primera práctica se describe el software, familiarizando al alumno con el entorno, tipos de archivo empleados, e instrucciones principales. Posteriormente se incluyen una serie de ejercicios cuya finalidad es que el alumno afiance los conocimientos adquiridos en la primera parte de la práctica.

El programa Matlab

Matlab es un programa de cálculo numérico, su nombre proviene de "**MAT**rix textbfLABoratory" (*laboratorio de matrices*), puesto que esta especialmente indicado para trabajar con matrices y vectores.
Este software incorpora un lenguaje de programación propio (*similar a C*), que nos será muy útil para construir pequeños algoritmos de forma sencilla. Otro de sus puntos fuertes es la cantidad de funciones específicas para solucionar problemas muy variados del ámbito de la ciencia e ingeniería, así como para el tratamiento de gráficos 2-D ó 3-D.
Matlab está organizado en un núcleo y *toolboxes*. El núcleo contiene las funciones básicas y se instala por defecto. Estas instrucciones son completadas mediante *toolboxes* (librerias) diseñadas para propósitos específicos. El usuario determina que *toolboxes* desea instalar en función de las aplicaciones que desea emplear. En esta asignatura trabajaremos con la *Control System Toolbox*, que proporciona múltiples herramientas relacionadas con la ingeniería de control.

Entorno Matlab

Al arrancar Matlab observamos una ventana dividida en varias secciones. Esta organización no siempre será igual y depende de la configuración realizada por el usuario. Por defecto observamos los siguientes elementos:

- **Start:** se encuentra en la esquina inferior izquierda y da acceso a la ayuda, ejemplos, enlaces web, etc. Simula al botón de inicio en Windows (*no suele emplearse*).
- **Command window:** es la ventana de mayor tamaño, en ella se ejecutan los comandos Matlab seguidos del *prompt* ». Puede emplearse para ejecutar pequeños segmentos de código, pero lo normal será emplear scripts (archivos con extensión *.m).
- **Workspace:** muestra las variables definidas actualmente. Además permite algunas opciones para crear, borrar, cargar y guardar estas variables de forma gráfica. Estas operaciones también pueden realizarse empleando comandos.
- **Current directory:** muestra el contenido del directorio o carpeta de trabajo. Los archivos generados serán guardados en este directorio.
- **Command History:** muestra un histórico con los comandos empleados a lo largo de las ultimas sesiones.

Tipos de archivo

Los archivos con los que trabaja Matlab son los siguientes:

- Código Matlab scripts *.m. Contiene código Matlab en formato texto listo para ser ejecutado. Es la forma más apropiada cuando trabajamos con programas complejos

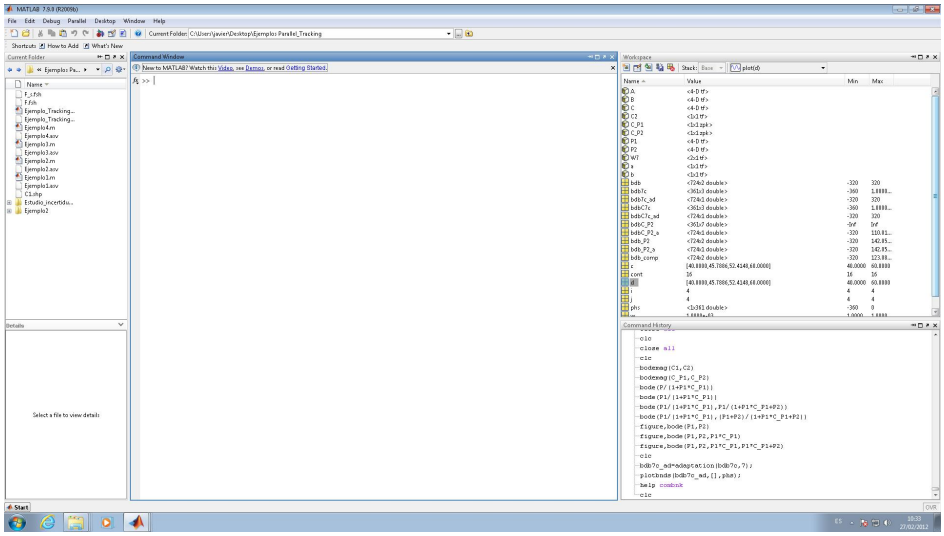

Figura 1: Aspecto del entorno Matlab.

- Variables matlab *.mat. Contiene las variables guardadas en el workspace, permite cargar datos salvados en sesiones anteriores.
- Figuras Matlab *.fig. Figuras creadas con Matlab, guardan información suficiente para porder ser modificadas posteriormente.
- Modelos Simulink *.mdl , *.slx. Modelos de simulación Simulink.

Instrucciones básicas

A continuación se muestra un listado con algunas de las instrucciones más empleadas.

Comandos de ayuda

Matlab incorpora comandos que nos permiten obtener información sobre librerías funciones o variables. Los principales comandos de ayuda de Matlab son los siguientes:

```
help <toolbox>;  %  Muestra la ayuda de una toolbox
help <instrucción>;  % Muestra la ayuda de una función

% Ejemplo
help control;  % Muestra la ayuda para la Control System Toolbox
help rand;  % Muestra la ayuda de la instrucción rand
```

Instrucciones Workspace

Las instrucciones «save» y «load» nos permiten guardar y cargar respectivamente los datos del Workspace en archivos *.mat. Estas funciones también están disponibles mediante los botones del entorno gráfico. Existen otras instrucciones para borrar variables de forma selectiva, listar variables, visualizar valores, etc.

```
load <archivo>;  % Carga las variables almacenadas en el archivo indicado
load <archivo> <variables>;  % Carga sólo las variables indicadas
save <archivo>;  % Guarda todas las variables del workspace en un archivo
```

```
save <archivo> <variables>; % Idem al anterior pero solo guarda  las variables deseadas
clear <variables>; % Borra las variables indicadas del espacio de trabajo
who; % Muestra las variables del Workspace
whos; % Idem al anterior mostrando las características de las variables
disp(<variable>);    % Muestra en pantalla el valor de la variable
<variable>=input('<cadena de texto>'); % Mustra por pantalla la cadena de texto y
% almacena el dato introducido por teclado por el usuario en la variable indicada

% % Ejemplos
a=rand(3); b=rand(3); c=rand(3); % Crea tres matrices 3x3 aleatorias
save datos; % Guarda el workspace en el fichero datos.m. Contendra a, b y c
clear; % Borra todas las variables del workspace
load datos; % Carga las variables almacenadas en el fichero datos.mat
save datos2 a c; % Guarda en el fichero datos2.mat las variables a y c
clear a; % Borra la variable a
load datos a; % Carga del fichero datos.mat  la variable a
who; % Lista las variables del workspace
disp(a); % Muestra en pantalla la matriz a
disp('Ejemplo'); % Muestra en pantalla la cadena de caracteres Ejemplo
n=input('introduce el valor de n'); % Muestra el texto y espera al valor n
```

Las instrucciones «diary» permiten guardar en un archivo de texto los resultados mostrados por pantalla a lo largo de un sesión de trabajo.

```
diary <fichero.txt>: % Guarda en el fichero todas las órdenes y salidas generadas en la
% ventana de comandos que se ejecuten en la sesión actual
diary off; % Finaliza el diario
diary on; % Reanuda el diario
```

Matrices y vectores

Debemos tener en cuenta que para Matlab todos los elementos son matrices, en el caso más sencillo un número será una matriz $(1, 1)$ (escalar). Para definir matrices o vectores bastará con introducir entre corchetes los elementos. El separador de columnas es el espacio en blanco o la coma (,) y el de filas es el carácter punto y coma (;) .

```
V=[v1 v2 v3 ... vn]; % Crea el vector V de n elementos
M=   [a11 a21 ... a2n;
      a21 a22 ... a2n];
      am1 am2 ... amn]; % Crea la matriz M de  (mxn) elementos

% Ejemplos:
V1=[0 1 2 3 4 5 6 7 8 9];
M=[1 2 3; 4 5 6; 7 8 9];
```

Podemos acceder a los elementos de una matriz indicando las “coordenadas de dicho elemento”. El operador (:) nos permite definir rangos de datos.

```
Item=M(2,3);    % Guarda en la variable Item el elemento (2,3) de M
Col=M(:,1);     % Guarda en Col los elementos de la primera columna de M
Fil=M(2,:);     % Guarda en Col los elementos de la segunda fila de M
S=M(1:3,2:3);   % Guarda en S la matriz que contiene filas de 1 a 3, columnas de 2 a 3
```

Existen instrucciones para crear matrices o vectores automáticamente, sin necesidad de introducir manualmente los elementos uno a uno.

```
V1=0:9; % Crea el vector V1 con los valores del 0 al 9
V2=0:0.5:9; % Crea el vector V2 con los valores del 0-9 espaciados 0.5. V2=[0 0.5 1 ...

V3=linspace (ini,fin,num); % Genera un vector entre el elemento ini y el elemento fin
% con num elementos. Por ejemplo para crear el vector de los ejemplos anteriores
% es V3=linspace(0,9,10);
V4=logspace(ini,fin,num); % Igual que linspace pero en escala logarítmica

eye (n);        % Crea una matriz unidad (nxn)
zeros(n,m);     % Crea una matriz de ceros  (nxm)
zeros (n);      % Crea una matriz de ceros (nxn)
ones (n);       % Crea una matriz de unos (nxn)
ones (n,m);     % Crea una matriz de unos (nxm)
rand (n);       % Crea matriz  de valores pseudo-aleatorios. Valores 0 y 1
randn (n,m);    % Crea la matriz aleatoria (nxm)
magic (n);      % Crea una matriz en la que filas y columnas suman lo mismo
```

A continuación se muestran las operaciones algebraicas que se pueden realizar con matrices en Matlab.

```
M1+M2;     % Suma matricial de M1 y M2
M1-M2;     % Resta matricial de M1 y M2
M1*M2;     % Producto matricial de M1 y M2
M1.';      % Transpuesta de M1
M1^n;      % Elevar a potencia n-ésima (matrices cuadradas)
M1/M2;     % Divide M1 entre M2 por la derecha
M1\ M2;    % Divide M1 entre M2 por la izquierda
M1.*M2;    % Producto elemento a elemento
M1./M2;    % Divide elemento a elemento
M1.^M2;    % Elevar a potencia elemento a elemento
abs(M);    % Calcula el valor absoluto de una matriz
angle(M);  % Calcula el ángulo de una matriz
real(M);   % Obtiene la parte real de una matriz
imag(M);   % Obtiene la parte imaginaria de una matriz
```

Otras instrucciones empleadas habitualmente con matrices son:

```
size(dato);     % Devuelve el número de filas y columnas de la matriz
length(dato);   % Calcula el número de elementos de un vector
diag(vector);   % Crea una matriz diagonal con los elementos del vector
min(vector);    % Obtiene el valor mínimo de un vector o matriz
max(vector);    % Obtiene el valor máximo de un vector o matriz
triu(matriz);   % Crea una matriz triangular superior
tril(matriz);   % Crea una matriz triangular inferior
```

Polinomios

Para Matlab un polinomio es un vector que contiene los coeficientes de dicho polinomio ordenado de forma descendente.

$$P(x) = a_n x^n a_{n-1} + x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1 x + a_0$$

$$P = [a_n \quad a_{n-1} \quad a_{n-2} \quad \ldots \quad a_1 \quad a_0]$$

Definido un vector con los coeficientes del polinomo podemos aplicar las funciones para polinomios.

```
conv (P1,P2);         % Convolución. Multiplica los polinomios P1 y P2
deconv (P1,P2);       % Deconvolución. Divide los polinomios P1 y P2
poly (C);             % Polinomio a partir de las raíces indicadas en C
polyder(P1);          % Derivada del polinomio
polyval(P1,xo);       % Evalúa el polinomio para el valor xo
polyvalm(P,X);        % Evalúa el polinomio con los valores dados en una matriz
residue(B,A);         % Calcula la expresión de los residuos para B/A
roots(P);             % Calcula las raíces del polinomio
```

Representación gráfica en 2D

```
plot(x,y); % Representa gráficos bidimensionales
subplot (n,m,p);  % Divide la ventana en diferentes gráficas
hold;      % Retiene o libera el grafico actual
figure;    % Crea la figura una ventana gráfica
clf;       % Borra la ventana gráfica

title (<'Texto'>); % Titulo del gráfico
xlabel (<'Texto'>); % Testo para el eje x
ylabel (<'Texto'>);  % Testo para el eje y
legend;  % Muestra la leyaenda
axis;  % Control del escalado de los ejes

% Ejemplo. Traza los valores de p(x)=x^3 +5x^2-x+4 entre -5 y 5
x=-5:0.1:5;
P=[1 5 -1 4];
y=polyval(P,x);
plot(x,y);
```

Programación en Matlab

Matlab incorpora un lenguaje de programación propio que permite el empleo de las estructuras típicas de programación. Este lenguaje es similar a C, pero su menor número de opciones simplifica la complejidad de los programas.

Sentencia if

La sentencia *if* es similar a la empleada en C, salvo que en este caso los paréntesis en la condición son optativos. Tampoco son necesarias las llaves para agrupar sentencias.

```
if <condición>
     <sentencias>
end

% Formato de la condición con if else
if <condición>
     <sentencias si cierto>
else
     <sentencias si falso>
end

```

```
% Formato de la instrucción completo (múltiples condiciones)
if <condición 1>
     <sentencias si es cierta la condición 1>
elseif <condición 2>
     <sentencias si es cierta la condición 2>
elseif <condición 3>
     <sentencias si es cierta la condición 3>
else
     <sentencias si no se cumple ninguna de las anteriores>
end

% Ejemplo simple
if P1==P2
     disp('Los polinomios son iguales')
else
     disp('Los polinomios son diferentes')
end
```

Sentencia switch-case

Realiza la misma operación que una estructura *if* concatenada, seleccionando un bloque de código en función del estado de la condición. Es una forma más compacta y legible que la instrucción anterior, sobre todo cuando se presentan muchas opciones.

```
switch <variable>
   case <valor 1>
      <sentencias si variable = 1>
   case <valor 2>
      <sentencias si variable = 2>
   otherwise
      <sentencias si variable diferente a las opciones dadas>
end

% Ejemplo
n=input ('Introduce el tipo de operación a realizar: ')
switch n
   case n==1
      P3=P1+P2;
   case n==2
      P3=P1-P2;
   otherwise
      P3=P1*P2;
end
```

Sentencia for

Ejecuta, un número de veces conocido, una serie de instrucciones agrupadas dentro de esta sentencia.

```
for <variable = expresión>
     <sentencias>
end

% Ejemplo
```

```
[n,m]=size(M);c=1;
for i=1:n % Crea un vector con todos los elementod de la matriz M
     for j=1:m
          V(c)=M(i,j);
          c=c+1;
     end
end
```

Sentencia while

Ejecuta, mientras sea cierta una condición de prueba, una serie de instrucciones agrupadas dentro de esta sentencia.

```
while <condición>
     <sentencias>
end

% Ejemplo
a=0;
while a<=100
     a=a+1;
end
```

Scripts y Funciones

El trabajo con Matlab supone ejecutar secuencias de comandos desde el prompt obteniendo unos resultados numéricos y/o gráficos. Sería interesante que todas estas sentencias se pudiesen recoger en un fichero y ejecutarlas automáticamente llamando al mismo. Esto es posible gracias a los "scripts" o ficheros guión. Si además pudiésemos llamarlos pasándoles parámetros de entrada, podríamos hacer módulos de código similar a los de cualquier lenguaje de programación de alto nivel. Esto último, es posible mediante las "funciones" de Matlab.
Tanto los guiones como las funciones son ficheros de texto con extensión *.m. Se deberán ubicar en el directorio de trabajo actual o en los *path* donde Matlab busca las fuentes. Los ficheros funcionales sólo podrán contener una única función que se llamará del mismo modo que el fichero *.m.

```
% Definición de una función
function [datos de retorno]=función (lista de argumentos)
% ayuda  de la función mostrada cuando empleamos help nombre funcion
```

En las funciones de Matlab no se puede incluir la sentencia return, las variables se calculan en el interior de la función, finalizando la misma cuando terminan las sentencias de su interior.
Las variables definidas en el interior de las funciones actúan como variables locales, lo que las hace inaccesibles al resto del programa. Matlab toma los argumentos de las funciones como si se pasasen por referencia, es decir, cuando modificamos los argumentos estos se copian, modificándose únicamente las copias realizadas por la función.
Cuando busquemos la ayuda de una función definida por el usuario (help función) Matlab mostrará las líneas definidas como comentarios (las que comienzan con el símbolo %) hasta alcanzar la primera línea de código.

```
function P=maximosM(M)
% Función que calcula un vector con el valor máximo de cada fila de una matriz
% M -> matriz de partida
% P -> polinomo de salida
```

```
[n,m]=size(M)
for i=1:n
      P(i)=max(M(i,:));
end
```

Ejercicios propuestos

Ejercicio 1

El prototipo de moto eléctrica prestado por el equipo *Unirioja Motostudent* en la cuarta edición de la competición *Motostudent*, incorporaba un novedoso sistema de cambio de marchas automático. Este emplea dos sistemas de control, uno encargado de controlar las revoluciones (rpm) del motor, y otro encargado de controlar la posición del selector de marchas.

Figura 2: Prototipo de moto eléctrica en el circuito *Motorland* (Alcañiz).

El sistema de control de revoluciones del motor es el encargado de gobernar la velocidad del motor expresada en rpm, para que esta coincida con una demandada por el usuario (referencia). Para ello, un sistema electrónico compara las rpm actuales con las deseadas, y actúa sobre el acelerador electrónico en función del error detectado. En la Figura 3 se muestra un esquema de funcionamiento. El archivo *Datos_motor.csv* contiene los resultados **reales** obtenidos por el sistema de control en uno de los experimentos realizados. La información se encuentra organizada por columnas, donde cada columna contiene la siguiente información:

- Columna 1. Tiempo en el que se recogen los datos. Expresado en milisegundos.
- Columna 2. Consigna para las rpm del motor (valor deseado).
- Columna 3. Acción de control aplicada. Corresponde con la posición del acelerador. Expresada en u*100. Es decir, 300 corresponde con acelerador al 3 % de su valor máximo.
- Columna 4. Revoluciones del motor.

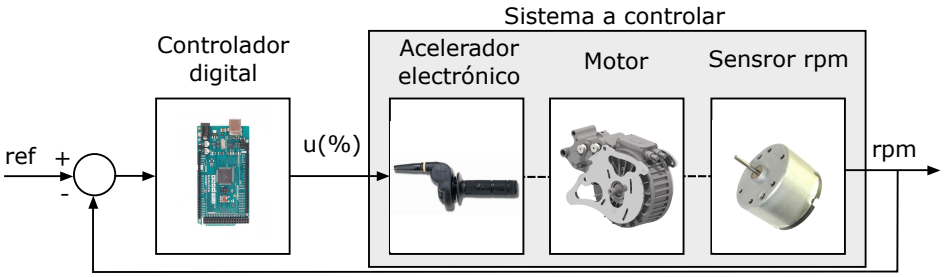

Figura 3: Esquema de funcionamiento del sistema de control de rpm.

Se pide:

- Importar los datos a Matlab en formato matriz con nombre *Datos_motor*. Se recomienda arrastrar el archivo al workspace y seguir el asistente.

a. Extraer los datos de *Datos_motor*, de forma que cada variable se quede almacenada en un vector (t, ref, u, rpm).

```
%% a) Extraemos los datos en vectores
t=Datos_motor(:,1)/1000;
ref=Datos_motor(:,2);
u=Datos_motor(:,3)/100;
rpm=Datos_motor(:,4);
```

b. Genera una gráfica que muestre la velocidad del motor frente al tiempo. Etiqueta los ejes de forma adecuada. Eje x: de 1 a 6 segundos. Eje y: de -100 a 4200 rpm.

```
%% b) Representación de las rpm del motor
plot(t,rpm);
title('Respuesta del motor');
xlabel('Tiempo (s.)');
ylabel ('rpm');
axis([1 6 -100 4200]);
```

c. Comprueba de forma automática si los datos se han tomado a intervalos de tiempo regulares. $t(2) - t(1) = t(3) - t(2) = \cdots = t(n) - t(n-1)$

```
%% c) Analiza el intervalo de muestreo
t_aux=t(2:length(t))-t(1:length(t)-1);
Tso=t(2)-t(1);
% la instrucción sum, suma todos los elementos de un vector
if sum (abs(t_aux-Tso))==0
    disp('Intervalo de muestreo regular');
else
    disp('Intervalo de muestreo no regular');
end
```

d. Calcula los valores característicos de la respuesta.

- Tiempo en el que comienza a cambiar la velocidad del motor.
- Valor máximo de la respuesta y tiempo en el que se alcanza.

- Valor medio final. Calcular el valor medio entre los segundos 4 y 6.

```
%% d) Calculo de valores característicos
clc;
% Tiempo de inicio
c=1;
while rpm(c)==0
    c=c+1;
end
t_ini=t(c);
disp('Tiempo de inicio: '),disp(t_ini);

% Valor máximo
[rpm_max,i]=max(rpm);
t_max=t(i);
disp('Valor máximo: '),disp(rpm_max);
disp('En el tiempo : '); disp(t_max);

% Valor medio
suma=0;
n_datos=0;
for i=1:length(t)
    if (t(i)>=4) && (t(i)<=6)
        suma=suma + rpm(i);
        n_datos=n_datos+1;
    end
end
rpm_rp_media=suma/n_datos;
disp('Valor medio en régimen permanente: '),disp(rpm_rp_media);
```

e. Traza lineas en el gráfico con el valor medio y máximo para comprobar que los cálculos anteriores son correctos.

```
%% e) Traza una linea en el gáfico
hold on
plot([t(1) t_max], [rpm_max rpm_max],'k--');
plot([t(1) t(length(t))], [rpm_rp_media rpm_rp_media],'k--');
hold off
```

f. Genera una nueva gráfica en la que se representen en la parte superior las rpm y la referencia frente al tiempo. En la parte inferior la acción de control frente al tiempo.

```
%% f) Genera una nueva gráfica en la que se representen: arriba rpm, ref; abajo u
figure(2)
subplot(2,1,1),plot(t,ref,t,rpm);
xlabel('Tiempo (s.)');
ylabel ('rpm')
subplot(2,1,2),plot(t,u);
xlabel('Tiempo (s.)');
ylabel ('Accion de control (%)');
```

g. Para comprobar si el control es adecuadosuelen emplease índices de error $e = ref - rpm$ que cuantifican el área del error cometido. Un valor más pequeño corresponde con un mejor funcionamiento . Algunos

de los índices más conocidos son:

$$IE = \int_0^t e(t)dt \tag{1}$$

$$IAE = \int_0^t |e(t)|dt \tag{2}$$

$$ISE = \int_0^t e^2(t)dt \tag{3}$$

```
%% g) Índice de error
n=length(t_aux);
IE=sum(t_aux.*(ref(1:n)-rpm(1:n)));
IAE=sum(t_aux.*(abs(ref(1:n)-rpm(1:n))));
ISE=sum(t_aux.*((ref(1:n)-rpm(1:n)).^2));

disp('Integral del error: '  ),disp(IE);
disp('Integral del valor absoluto del error: ' ); disp(IAE);
disp('Integral del cuadrado del error: ' ); disp(ISE);
```

h. En este experimento, el porcentaje del acelerador se regula en función del error entre las rpm del motor y el valor deseado. La ley de control sigue el formato $u = (ref - rpm) * k$. El % del acelerador solo puede tomar valores positivos.

- Calcular el valor de k.
- Analizar los resultados.
- Pintar sobre la gráfica anterior el valor que tendría la acción de control si el acelerador pudiese presentar valores negativos.

```
%% h) Calcula la ley de control
k=u./(ref-rpm);
disp(k);
u_teorica=(ref-rpm)*k(1);
hold on
subplot(2,1,2),plot(t,u_teorica,'k--');
legend('Real', 'Teórica')
hold off
```

i. Diseña una función *[y]=filtra_datos(x,n)* que dada una colección de datos x los filtre empleando una media móvil de n valores. Cada valor debe generarse según:

$$y(i) = \frac{x(i) + x(i-1) + \cdots + x(i-n+1)}{n}$$

```
function [y]=filtra_datos(x,n)
y(1:n)=x(1:n);
for i=n:length(x)
    y(i)=sum(x(i+1-n:i))/n;
end
```

j. Utiliza la función anterior con diferentes valores de n. Muestra todos los resultados de forma conjunta y extrae conclusiones.

```
%% j) Analiza los datos filtrados
n=input('Vector con los valores a probar: ');
figure,plot(t,rpm,'k');
hold on
for i=1:length(n);
    rpm2=filtra_datos(rpm,n(i));
    plot(t,rpm2);
end
hold off
```

Ejercicios complementarios

Definición y cálculo con matrices

a. Define por teclado las matrices A y B.

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}; \qquad B = \begin{pmatrix} -1 & 1 & 1 \\ 2 & -2 & 2 \\ 3 & 3 & -3 \end{pmatrix}$$

b. Guarda en C la matriz traspuesta de A.

c. Crea una matriz D aleatoria de tamaño (3,3) con números comprendidos entre 0 y 10.

d. Calcula las siguientes operaciones:

- Resta elemento a elemento las matrices A y B. Resultado en A_B.
- Producto matricial de de A y B. Resultado AporB.
- División izquierda A entre B. Resultado en AdivB.

e. Guarda todas las variables en un archivo de nombre *Salva_workspace.mat*.

f. Borra posteriormente todas las matrices del espacio de trabajo excepto AporB y D.

g. Obtén el valor mínimo y máximo de A*B

Resolución de sistemas de ecuaciones

a. Define el siguiente sistema de ecuaciones empleando su forma matricial.

$$\begin{cases} 3x + 2y + z = 1 \\ 5x + 3y + 4z = 2 \\ x + y - z = 1 \end{cases}$$

b. Calcula las soluciones en las variables X, Y, Z y añádelas al archivo .mat de ejercicio anterior.

Matrices con números complejos

a. Define la siguiente matriz compuesta por números complejos.

$$A = \begin{pmatrix} 1+i & -2i & 3 \\ 2+2i & 5+5i & 2+2i \\ -8i & 3-4i & 9 \end{pmatrix}$$

b. Calcula las siguientes operaciones:

- Calcula la parte real.
- Calcula la parte imaginaria.
- Calcula el modulo.
- Calcula el argumento.
- Calcula el argumento en grados.

Definición de matrices

a. Matriz aleatoria (3,3).

b. Matriz identidad (4,4).

c. Matriz todo unos (4,4).

d. Añade a la matriz aleatoria la última fila y columna de la matriz unitaria, obteniendo una matriz (4,4).

e. Crea un vector t comprendido entre (0-100) con valores cada 0.2.

f. Crea un vector f con 100 valores espaciados logarítmicamente entre 1HZ y 100KHz

g. Crea una matriz aleatoria de dimensiones 10x10, con valores comprendidos entre (0-100). Realiza las siguientes operaciones:

- Toma la cuarta fila, almacénala en un vector y obtén el valor máximo
- Toma la última columna, almacénala en un vector y calcula el valor mínimo
- Toma los elementos de la quinta fila comprendidos entre la segunda y séptima columna
- Calcula un vector con los elementos inferiores a 50 de la matriz

Operaciones con polinomios

a. Define los siguientes polinomios:

$$P_1(x) = x^5 + 3x^4 + 2x^3 - 10x^2 - 5x + 10$$
$$P_2(x) = x^3 + 2x^2 - 3x + 7$$

b. Realiza las siguientes operaciones con polinomios:

- Calcula las raíces de P_1 y P_2.
- Evalua P_1 y P_2 para $x = 0.25$.
- Calcula el producto de P_1 y P_2.
- Calcula la división deP_1 entre P_2.
- Obtén los residuos del cociente P_1/P_2.
- Obtén la derivada de P_1.

Funciones de usuario

a. Crea una función llamada *borra.m* que tome un vector (definido previamente), y elimine los elementos menores a un valor introducido por teclado.

b. Crea una función *ordenav.m* que tome un vector y lo ordene de mayor a menor.

c. Crea una función *integra.m* que dado un vector de datos f y el tiempo de muestreo Ts calcule la integral de dicha señal.