Examen de MATLAB Fundamentos de la Automática primer cuatrimestre:

Práctica 1: Introducción a MATLAB (comandos básicos, polinomios, gráficos y creación de scripts)

Vectores y matrices:

Generación de un vector:

```
vector_fila = [1 2 3];
```

Generación de una matriz:

```
matriz = [1 3 5; 2 4 6; 7 9 0];
```

Generación de un vector con patrón de incremento:

```
vector_incremento = 1:2:19;
```

Generación de un vector columna:

```
vector_columna = [4;3;2];
```

linspace(x1,x2): Devuelve un vector fila de 100 puntos equidistantes entre x1 y x2.

logspace(a,b): Genera un vector fila de 50 puntos espaciados logarítmicamente entre las décadas 10^a y 10^b.

Definición de polinomios:

Definición de un polinomio:

```
polinomio = [1 6 5 -3];
```

Raíces de un polinomio:

```
raices = roots(polinomio);
```

Obtener polinomio mediante sus raíces:

```
r = [-1;0.5+i;0.5-i];
p = poly(r);
```

Creación de gráficos:

```
t = 0:0.1:10;           % Variable independiente
y = exp(-t);            % Variable dependiente
plot(t,y,'blue');       % Función a representar
hold on                 % Solo si hay más de una función
legend('exp(-t)')       % Título de la función
axis([0,1,0,1])         % Tamaño de ejes
grid on                 % Gráfica de rejilla
hold off                % Solo si hay más de una función
```

Ejemplos importantes:

1. Realiza la gráfica de f(t) = e^(-at)*cos(t) en el intervalo t=[0,5] para distintos valores de a.

```
t = 0:0.1:5;
for a = 1:1:5
    y = exp(-a.*t).*cos(t);
    plot(t,y); hold on
end
hold off
```

2. Crea una gráfica con subplot que contenga cuatro subgráficas de las funciones trigonométricas sen(t), cos(t), tan(t) y sen(t)*cos(t).

```
t = 0:0.1:5;
figure
subplot(2,2,1);
plot(t,sin(t))
subplot(2,2,2);
plot(t,cos(t))
subplot(2,2,3);
plot(t,tan(t))
subplot(2,2,4);
plot(t,sin(t).*cos(t))
```

Práctica 2: Modelo de sistemas de control con MATLAB (funciones de transferencia con Control System Toolbox, transformada inversa de Laplace y álgebra de bloques con Symbolic Math Toolbox)

Modelado de sistemas LTI mediante funciones de transferencia:

Función de transferencia polinómica (tf):

```
G1 = tf([2 3],[4 5 6]);                          % G1 = 2s+3/4s^2+5s+6
```

Función de transferencia no polinómica:

```
G2 = tf([1 0 3],conv([1 1 1],[1 0 2 3]));    % G2 = s^2+3/(s^2+s+1)(s^3+2s+3)
```

Función de transferencia factorizada (zpk):

```
G3 = zpk(-1,[-2 -3 -4],2);                       % G3 = 2(s+1)/(s+2)(s+3)(s+4)
```

Conversión entre funciones de transferencia polinómicos y factorizados:

Conversión polinómica-factorizada:

```
Gtf = tf([2 3],[4 5 6]);
Gzpk = zpk(Gtf);
```

Conversión factorizada-polinómica:

```
Gzpk = zpk(-1,[-2 -3 -4],2);
Gtf = tf(Gzpk);
```

Función de transferencia simplificada:

```
s = tf('s');
Gsimp = (s+1)/(s^3+3*s+1);
```

Descomposición (expansión) en fracciones simples:

```
[r,p,k] = residue(32,[1 12 32 0]);            % [residuos,polos,ceros]
```

Transformadas simbólicas usando el Symbolic Math Toolbox:

```
syms s
X = entrada
G = Función de transferencia
Y = X * G
y = ilaplace(Y)
```

Gráficas de funciones simbólicas:

```
syms s
fplot([y],[i,f])
legend('y')
```

Transformadas de Laplace con MATLAB:

```
syms s
y = Respuesta
Y = laplace(y)
```

Álgebra de bloques mediante MATLAB:

Asociación en cascada o serie:

```
series(G1,G2)
```

Asociación en paralelo:

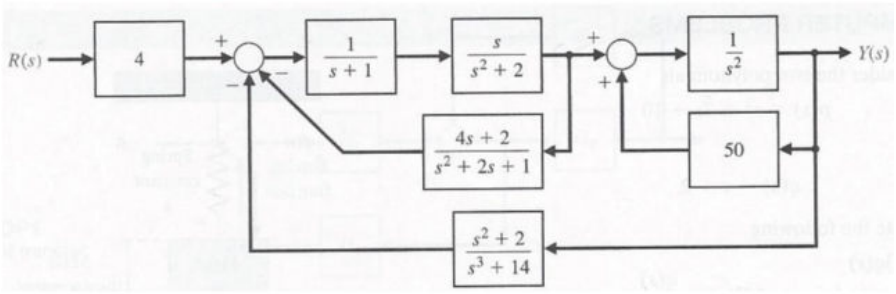
```
parallel(G1,G2)
```

Sistemas realimentados:

```
feedback(G,H,realimentación) => realimentación positiva = 1
```

Ejemplos importantes:

1. Utiliza los comandos de reducción de diagramas de bloques para calcular la función de transferencia en lazo cerrado del siguiente sistema de control:




```
syms s
% Definición de bloques
G1 = 4;
G2 = tf([1],[1 1]);
G3 = tf([1 0],[1 0 2]);
G4 = tf([1],[1 0 0]);
G5 = tf([4 2],[1 2 1]);
G6 = 50;
G7 = tf([1 0 2],[1 0 0 14]);
% Aplicación del álgebra de bloques
G23s = series(G2,G3);
G235f = feedback(G23s,G5);
G46f = feedback(G4,G6,1);
G23456s = series(G235f,G46f);
G234567f = feedback(G23456s,G7);
TFlc = series(G1,G234567f);
```

Práctica 3: Polos y ceros de una función de transferencia, respuesta transitoria de un sistema de control con MATLAB y LTIViewer, precisión y error en estado estacionario

Polos y ceros de una función de transferencia:

```
polos = pole(G);      % Polos
ceros = zero(G);      % Ceros
pzmap(G);             % Mapa de polos y ceros
```

Respuesta temporal de sistemas de primer orden:

```
figure; hold on
impulse/step(G,valor al que tiende)
legend('G')
hold off
```

ltiview(G): Para ver los parámetros característicos y las gráficas con más detalle.

Respuesta temporal de sistemas de segundo orden:

```
G = tf([K*Wn^2],[1 2*chi*Wn Wn^2]);
```

Extracción de parámetros de la respuesta temporal:

```
G = zpk([ceros],[polos],K)
A = stepinfo(G)                % Parámetros característicos
tr = A.RiseTime                % Tiempo de subida
ts = A.SettlingTime            % Tiempo de asentamiento
```

Respuesta de sistemas de primer orden a entrada rampa y aceleración:

```
G = zpk([ceros],[polos],K);
t = linspace(0,10,100);
rampa/aceleracion = entrada;
figure; hold on
lsim(G,entrada,t)
legend('G'); hold off
```

Respuesta de sistemas de segundo orden a entrada rampa y aceleración:

```
K = 1; Wn = 1; chi = 1;
G = tf([K*Wn^2],[1 2*chi*Wn Wn^2]);
t = linspace(0,10,100);
rampa/aceleracion = entrada;
figure; hold on
lsim(G,entrada,t)
legend('G'); hold off
```

Precisión y error de posición en estado estacionario:

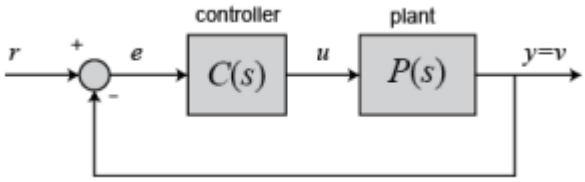
```
Glc = feedback(series(C,P),1);
Amplitud = 2;
figure; step(Amplitud * Glc)
[y,t] = step(Amplitud * Glc)
ess_pos_abs = Amplitud - y(end)
ess_pos_rel = (Amplitud - y(end)) * 100 / Amplitud
```

Precisión y error de velocidad en estado estacionario:

```
Glc = feedback(series(C,P),1);
t = linspace(0,10,100);
rampa = t;
figure; lsim(Glc,rampa,t)
[y,t] = lsim(Glc,rampa,t)
ess_vel_abs = rampa(end) - y(end)
ess_vel_rel = (rampa(end) - y(end)) * 100 / rampa(end)
Kv = 1 / ess_vel_abs
```

Ejemplos importantes:

1. Realiza ahora una única gráfica de los polos y ceros de la función de transferencia en lazo cerrado GLC(s) del sistema de la figura, con la misma P(s) y con C(s) = K, para tres valores de la constante K, K=1, K=3 y K=10.




```
P = zpk(-1,[1 -2 -4],1);
K = [1 3 10];
figure; hold on
for t=1:length(K)
    pzmap(feedback(K(t)*P,1));
end
legend('K=1','K=3','K=10');
```

2. Extrae los valores numéricos del tiempo de subida y del tiempo de asentamiento para diferentes valores de chi (segundo orden).

```
k = 1; wn = 2;
chi = [0 0.25 1 5];
for t=1:4
    A = stepinfo(tf([k*wn^2],[1 2*chi(t)*wn wn^2]));
    ts(t) = A.SettlingTime;
    tr(t) = A.RiseTime;
end
ts
tr
```

3. Diseño de sistemas de control

3.1 Valor del polo de un sistema de primer orden para que tr = 0.001s.

```
p=-1; tr=1;
while tr > 0.001
    T = stepinfo(zpk([],[p],1));
    tr=T.RiseTime;
    p=p-1;
end
p
tr
step(zpk([],[p],1))
```

3.2 Valor de Wn de un sistema de segundo orden para que ts = 0.01s.

```
wn=1; ts=1;
while ts > 0.01
    [num,den] = ord2(wn,chi);
    T = stepinfo(tf(num,den));
    ts=T.SettlingTime;
    wn=wn+1;
end
```

```
wn
ts
step(tf(num,den))
```

Práctica 4: Análisis y diseño de sistemas de control mediante el lugar de las raíces con MATLAB

Dibujo básico del lugar de las raíces con MATLAB:

LDR:

```
G = tf([num],[den]);
H = tf([num],[den]);
rlocus(G*H)
```

LDR con un valor determinado de K:

```
K = 20:0.01:35
rlocus(G*H,K)
```

Valor de K para un polo determinado:

```
[K,PolosLC] = rlocfind(G*H,polo)
```

Diseño de sistemas mediante el lugar de las raíces:

Sgrid para chi/Wn constantes:

```
G = tf([num],[den]);
H = tf([num],[den]);
rlocus(G*H)
sgrid(chi*wn)
K_chi/K_wn =rlocfind(G*H)
step(feedback(K_chi/K_wn*G,H))
```

Diseño de sistemas de orden superior mediante el LDR:

```
Gla = Actuador * Motor;
step(Gla);
K = 1;
Glc = feedback(K*Gla,1);
hold on;
step(Glc);
hold off;
rlocus(Gla);
K_critica = rlocfind(Gla);
```

Práctica 5: Diseño de controladores PID con MATLAB y RLTOOL

Posiciones de los polos y los ceros de los controladores PID:

Proporcional (P): cuenta sólo con una ganancia positiva, K.

Proporcional-Integral (PI): ganancia K, polo en el origen y cero en posición configurable.

```
close all;
clear
clc
for Ti = 0.001:0.1:1
    G = tf([Ti 1],[Ti 0]);
    pzmap(G);
    hold on;
end
```

Proporcional-Derivativo (PD): cuenta con una ganancia K y un cero cuya posición es configurable.

```
close all;
clear
clc
for Td = 0.001:0.1:1
    G = tf([1 1/Td],Td);
    pzmap(G);
    hold on;
end
```

Proporcional-Integral-Derivativo (PID): ganancia K, polo en el origen y dos ceros en posiciones configurables.

```
close all;
clear
clc
Td = 0.005
% for Td = 0.1:1:10
    for Ti = 0.1:0.2:1
        G = tf([Ti*Td Ti 1],[Ti 0]);
        pzmap(G);
        hold on;
    end
% end
```

RLTOOL/SISOTOOL:

rltool(G): Siendo G la función de transferencia de la planta, creada con zpk o tf.

rltool(G,C): Siendo C el controlador para el sistema en bucle cerrado con realimentación negativa.

```
close all;
clear
G = tf(num,den);
sisotool(G)
```

Ejemplos importantes:

a.- Obtener la respuesta temporal c(t) del sistema ante una entrada r(t) escalón unitario considerando que no hay perturbación (p(t) = 0), comparando las respuestas del sistema sin controlador y con los tres controladores

en la misma gráfica. Obtener en cada caso la posición de los polos en lazo cerrado del sistema y justificar el resultado obtenido.

```
% Caso 1: Controlador P (K = 2.35)
% Caso 2: Controlador PI (K = 0.235, Ti = 0.1)
% Caso 3: Controlador PID (K = 2.55, Ti = 1.28, Td = 0.09)
close all;
clear
clc
G = tf(0.5,conv([1 1],[0.5 1]));
Ti = 0.1;
Ti2 = 1.28;
Td = 0.09;
P = 2.35;
P2 = 0.235;
P3 = 2.55
PI = tf(P2*[Ti 1],[Ti 0]);
PID = tf(P3*[Ti2*Td Ti2 1],[Ti2 0]);
% Respuesta ante entrada escalón
step(G)
hold on
step(feedback(P*G,1),'r')
step(feedback(PI*G,1),'k')
step(feedback(PID*G,1),'m')
legend('original','P','PI','PID')
% Mapa de polos y ceros
figure;
pzmap(G,'b');hold on
pzmap(feedback(P*G,1),'r')
pzmap(feedback(PI*G,1),'k')
pzmap(feedback(PID*G,1),'m')
legend('original','P','PI','PID')
```

b.- Considerando que la entrada r(t) permanece a cero (p(t) = 0), obtener cómo evoluciona la salida cuando se añade una perturbación p(t) de tipo escalón unitario con los tres controladores anteriores. ¿Con qué controlador se obtiene una mejor respuesta frente a la perturbación?

```
% Respuesta ante entrada escalón
figure
step(G)
hold on
step(G/(1+G*P),'r');
step(G/(1+G*PI),'k');
step(G/(1+G*PID),'m');
legend('original','P','PI','PID')
```