Cairo University Faculty of Engineering

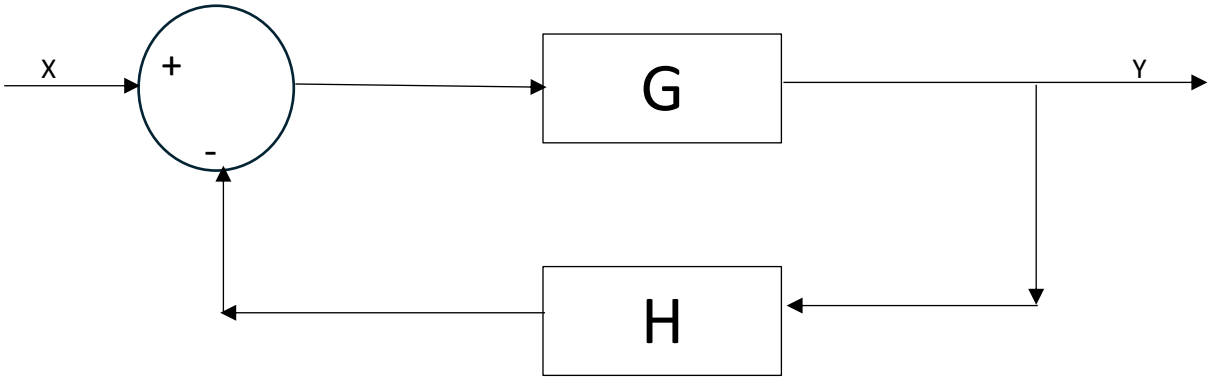
Dept. of Electronics and Electrical Communications Engineering

Control lab

Code	Sec.	BN.	Name
9210695	3	6	علياء عصام الدين نجيب محمد
9210776	3	16	عمرو عبد المتجلي احمد متولي

1. Bode plot:

- Loop:



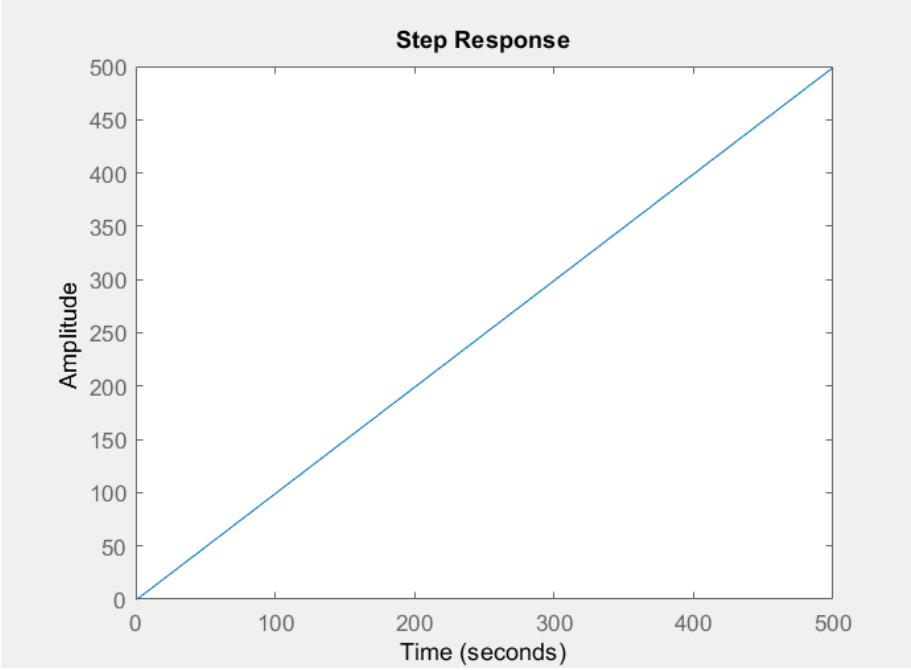
- Where: $G(s) = \frac{1}{s(s+1)}$, H(s) = 1

1)

```
task_1_code.m | Project33.m | p3.m | Code.m
1 —   G = tf([1],[1 1 0])
2 —   H = tf([1],[1])

Command Window
>> Code

G =

     1
  -------
  s^2 + s

Continuous-time transfer function.


H =

  1

Static gain.
```

2)

```
3 —   figure(1);
4 —   step(G);
```



The output of the block G for a unit step input

- No, the output of the block G(S) due to a unity step input is not stable,

$$Y(s) = \frac{1}{S} * \frac{1}{S(S+1)} = \frac{1}{S^2} + \frac{1}{S+1} - \frac{1}{S} \qquad Y(t) = t + e^{-t} - 1$$

-This is expected as the output tends to ∞ as time tends to ∞.

3)

```
7 -    Closed_loop_transfer_function = feedback(G,H)
```

```
Command Window
  >> Code

  Closed_loop_transfer_function =

          1
    -----------
    s^2 + s + 1

  Continuous-time transfer function.

fx >>
```

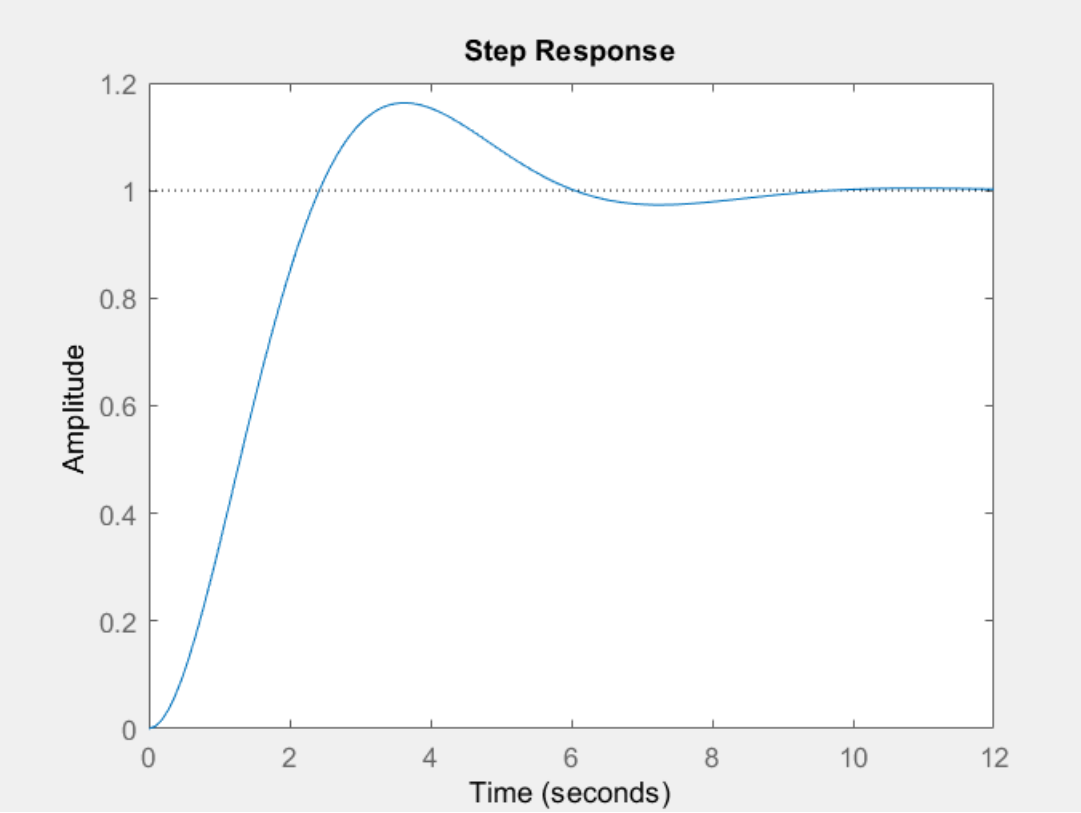
The closed loop transfer function from feedback command.

Closed loop T.F.: H(S) = $\frac{G}{1+GH} = \frac{1}{S(S+1)+1} = \frac{1}{S^2+S+1}$

- The closed loop transfer function we got from hand analysis is the same we got from feedback command.

4)

```
5 -    figure(2);
6 -    step(feedback(G,H));
```



The output of the closed loop

5) H(S) = $\frac{1}{S^2+S+1}$ ⟹ $\omega_n = 1$ & $\xi = 0.5$

ξ<1 → The closed loop system is stable.

& there are some damped oscillations.

This was expected as we saw from the graph also it is expected because negative feedback closed loop in most cases stabilize the system.

Project3.m | task_1_code.m | Code.m

```
1 -    G = tf([1],[1 1 0]);
2 -    H = tf([1],[1]);
3 -    figure(1);
4 -    step(G);
5 -    figure(2);
6 -    step(feedback(G,H));
7 -    Closed_loop_transfer_function = feedback(G,H);
8
9 -    poles = pzmap(feedback(G,H))
```

Command Window

```
>> Code

poles =

  -0.5000 + 0.8660i
  -0.5000 - 0.8660i
```

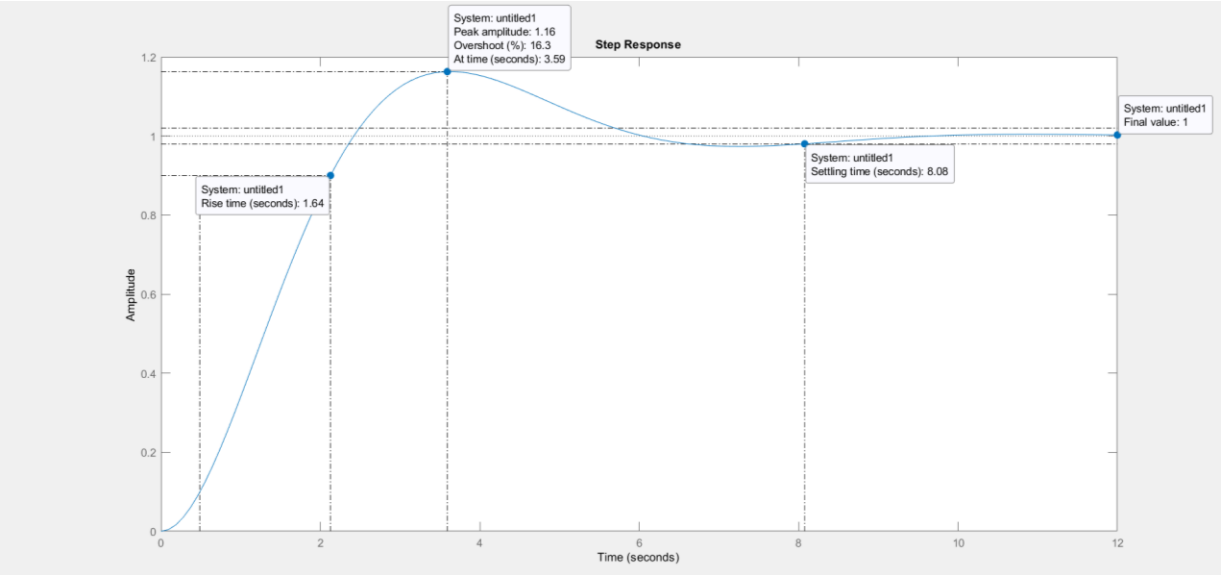
Locations of poles in closed loop transfer function from pzmap() command

- Locations of pole using hand analysis: -0.5 $\pm$ 0.866i
- The locations of poles we got from the simulation are the ones we got from the hand analysis.

This agree with the result from 4(previous step) also the locations of closed loop poles agree with your answer to that there are some

damped oscillations as the poles lies in the LHP, so this system is a stable underdamped system.

6)



step response curve showing peak response, settling time, steady state.

ω_n = 1, ξ = 0.5, $T_S = \frac{4}{\xi\omega_n}$, $T_p = \frac{\pi}{\omega_n\sqrt{1-\xi^2}}$, $\mathrm{OS} = e^{-\frac{\pi\xi}{\sqrt{1-\xi^2}}} * 100\%$

T_S = 8 sec. (same as that from simulation)

T_p = 3.63 sec. (same as that from simulation)

OS = 16.3% (same as that from simulation)

-Hand analysis results are the same as simulated results as expected as the system is stable underdamped as mentioned before.

7)

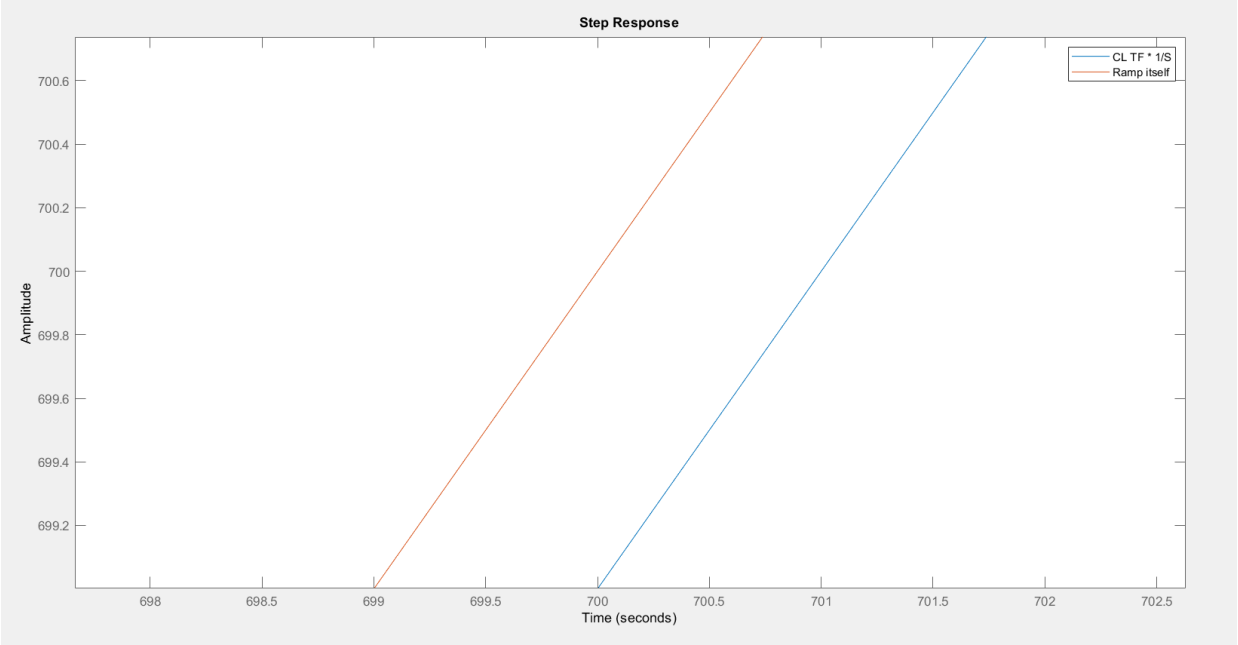
$e_{ss}|_{due\ to\ unit\ step\ input} = \lim_{S\to 0} \frac{S}{S}\frac{1}{S^2+S+1}$ = 1 (same as that from simulation)

This agrees with the type of the system as this system is type 0 &

In type 0: e_{ss} doesn't tend to infinity it tends to a finite value.

8)

```
figure(3);
ramp = tf([1],[1 0]);
N = tf([1],[1 1 1 0]);
step(N);%close loop * 1/s
hold on;
step(ramp);
legend('CL TF * 1/S','Ramp itself');
```



Closed loop & ramp itself on the same graph

We can see from the graph that the delay between the two lines at steady state (@ 700 Sec.) is 1 Sec.

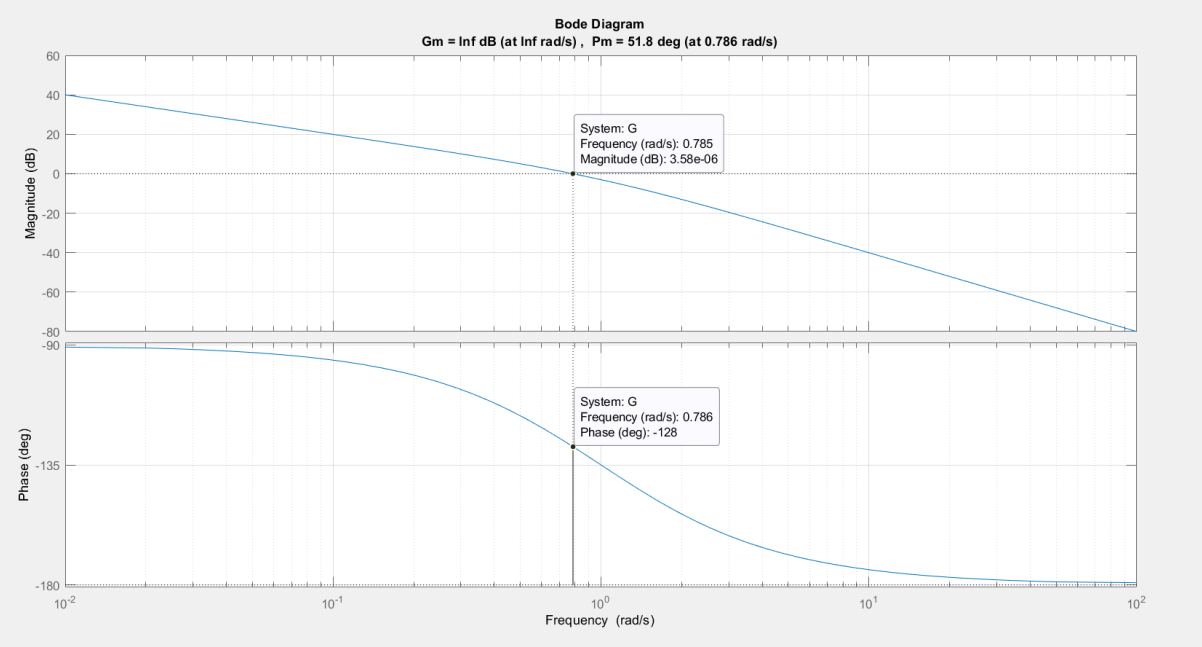
$e_{ss}|_{due\ to\ unit\ step\ input} = \frac{1}{\lim_{S\to 0} \frac{S}{S(S+1)}} = 1$ (same as that from simulation)

9)

```
16 -    figure(4);
17 -    margin(G);
18 -    grid on;
```

-Simulated analysis:

ω_{gc} = 0.786 rad/s , $\omega_{pc} = \infty$, $\mathrm{GM} = \infty$, $\mathrm{PM} = 51.8°$



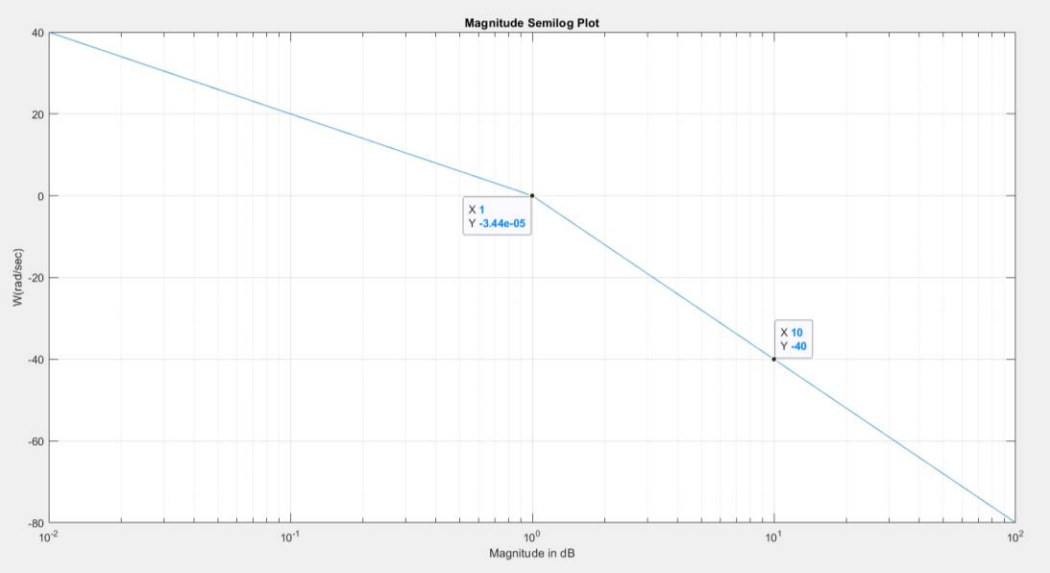
Bode diagram using margin () command

-Hand analysis: $\frac{1}{S(S+1)}$ has two poles: @ 0 rad/s & @ 1 rad/s

On the log scale, the curve is decreasing from negative infinity with slope = -20 dB/dec (effect of pole @ 0 rad/s)

The curve decreasing slope changes from -20 dB/dec to -40 dB/dec @ $\omega = 1$ rad/s (effect of another pole @ 1 rads/s)

So, the hand analysis (approximated) semi log magnitude plot will be:

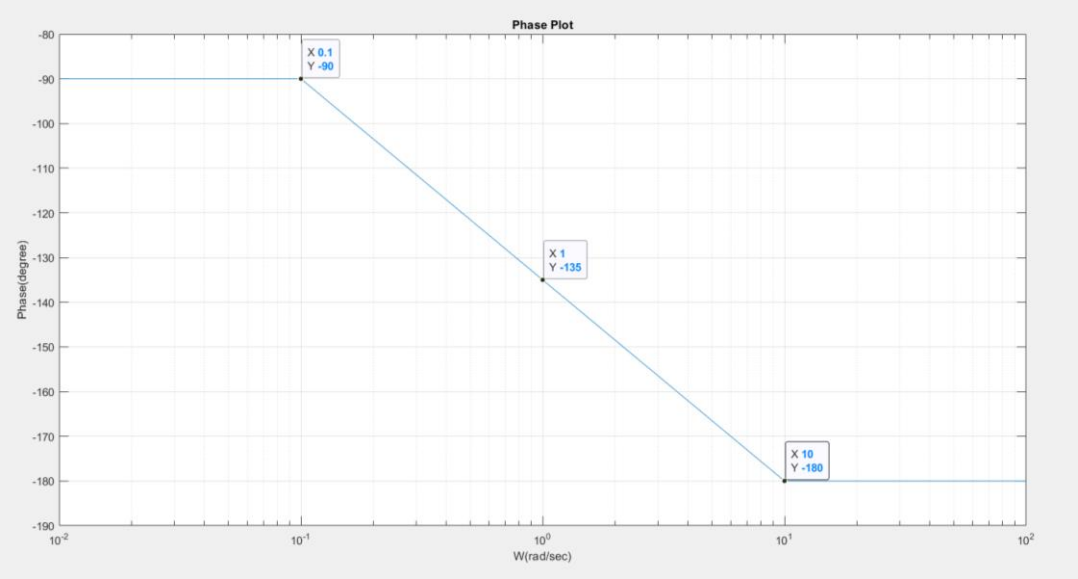


Approximated magnitude plot

On the log scale, the phase is -90 from negative infinity (effect of pole @ 0 rad/s) till 0.1 rad/s, the curve decreases with slope -45 deg/dec

Till 10 rad/s (effect of pole @ 1 rad/s)

So, the hand analysis (approximated) semi log phase plot will be:



Approximated phase plot

ω_{gc} = 1 rad/s , $\omega_{pc} = \infty$, $\mathrm{GM} = \infty$, $\mathrm{PM} = 180°\text{-}135° = 45°$

-Hand analysis results are nearly the same as the simulated results

$\omega_{pc} > \omega_{gc}$, GM > 0 & PM > 0

So, the system is stable.

MATLAB code for plotting the approximated bode plot:

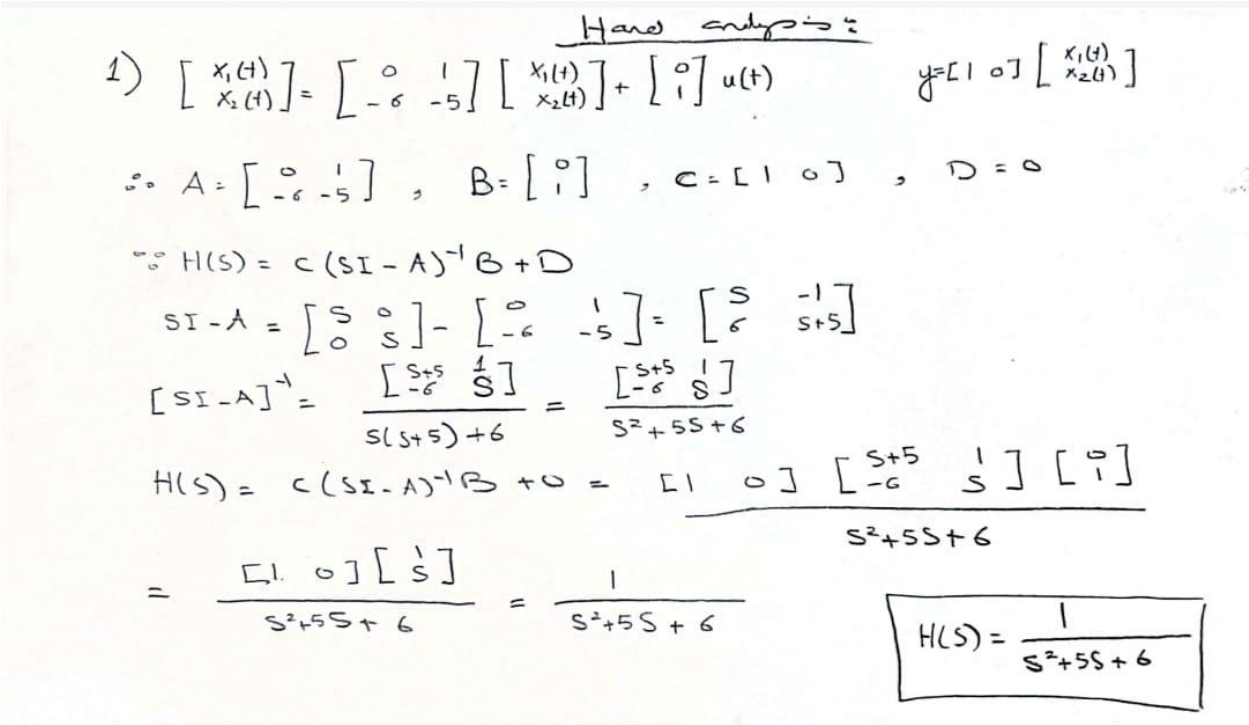
task_1_code.m | Project33.m | p3.m | Code.m

```
%approximated phase plot
%Generate x values for the entire range
x = linspace(0.01, 100, 10000);

% Generate y values for the straight line part
y = zeros(size(x));

% Define the starting y value and the slope for the decreasing straight line
starting_y = -90;
slope = -45;

% Set the constant y value for the range 0.01 to 0.1
y(x < 0.1) = starting_y;

% Calculate the y values for the decreasing straight line
y(x >= 0.1) = starting_y + slope * log10(x(x > 0.1) / 0.1);

% For the range x >= 10, y = -180
y(x > 10) = -180;

% Plotting
figure(5);
semilogx(x, y);  % Plot straight line part
title('Approximated Phase Plot');
xlabel('W(rad/sec)');
ylabel('Phase(degree)');
ylim([-190, -80]);  % Set y-axis limits
grid on;

%approximated magnitude plot
% Generate x values for the entire range
x = linspace(0.01, 100, 1000000);

% Generate y values for the straight line part
y = zeros(size(x));

% Define the slope for the first decreasing straight line
slope1 = -20;

% Calculate the y values for the first decreasing straight line
y(x <= 1) = slope1 * log10(x(x <= 1));

% Define the slope for the second decreasing straight line starting from where the first slope ended
slope2 = -40;

% Find the y value where the first slope ends at x = 1
y_at_x1 = slope1 * log10(1);

% Calculate the y values for the second decreasing straight line
y(x > 1) = slope2 * log10(x(x > 1)) + y_at_x1;

% Adjust the y value at x = 1 to be exactly 0
y(x == 1) = 0;% Plotting
figure(6);
semilogx(x, y);  % Plot straight line part
title('Approximated Magnitude Semilog Plot');
xlabel('Magnitude in dB');
ylabel('W(rad/sec)');
grid on;
```

2. State space representation:

1) The Transfer Function using hand analysis:

- SS for the hand analysis:

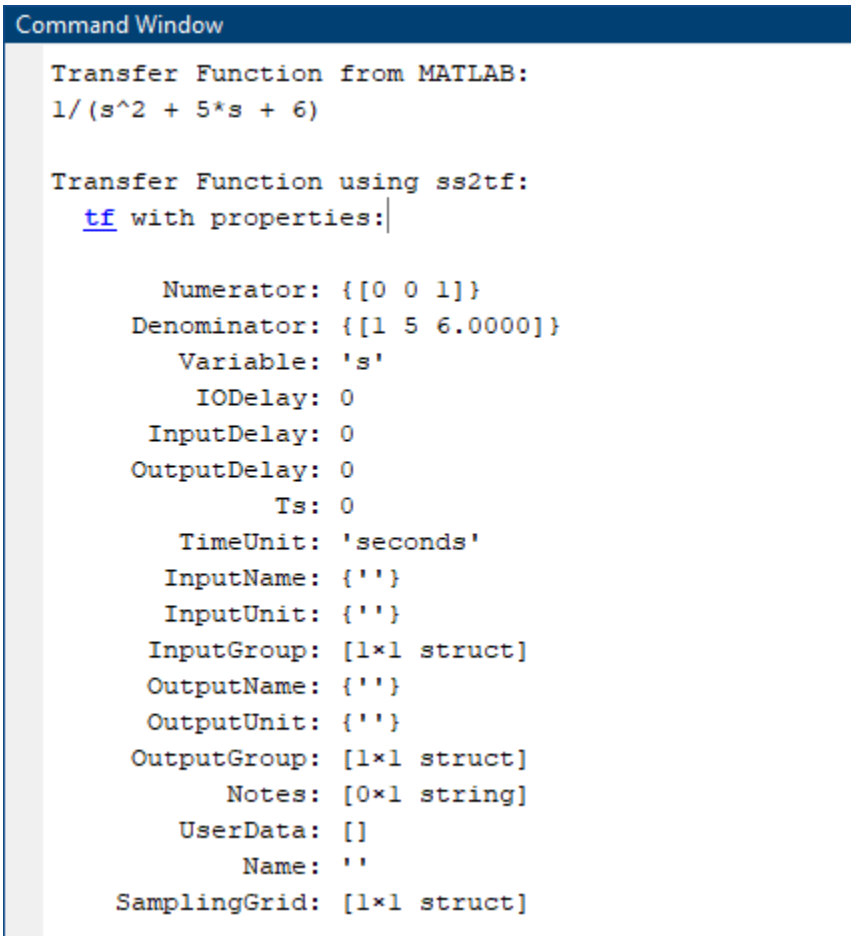


2) The Transfer Function using MATLAB code:

- SS for the code:

```
close ALL;
clear All;
clc;
syms s t;  % Define s as a variable

% Define A, B, C, D matrices
A = [0, 1; -6, -5];
B = [0; 1];
C = [1, 0];
D = 0;

% Calculate Transfer Function from State-Space representation
TF_matlab = simplify(C * inv(s*eye(size(A)) - A) * B);

% Convert State-Space to Transfer Function using ss2tf
[num, den] = ss2tf(A, B, C, D);
TF_ss2tf = tf(num, den);

% Display results
disp("Transfer Function from MATLAB:");
disp(TF_matlab);
disp("Transfer Function using ss2tf:");
disp(TF_ss2tf);

```

- SS for the output of the code:



- **By comparing the result (Transfer function) from the MATLAB code with the result (Transfer function) driven by hand analysis we notice they are the same.**

3) MATLAB code for calculating $\emptyset(t)$ and verifying $\emptyset(0) = I$:

- SS of the code for calculating $\emptyset(t)$ verifying $\emptyset(0) = I$:

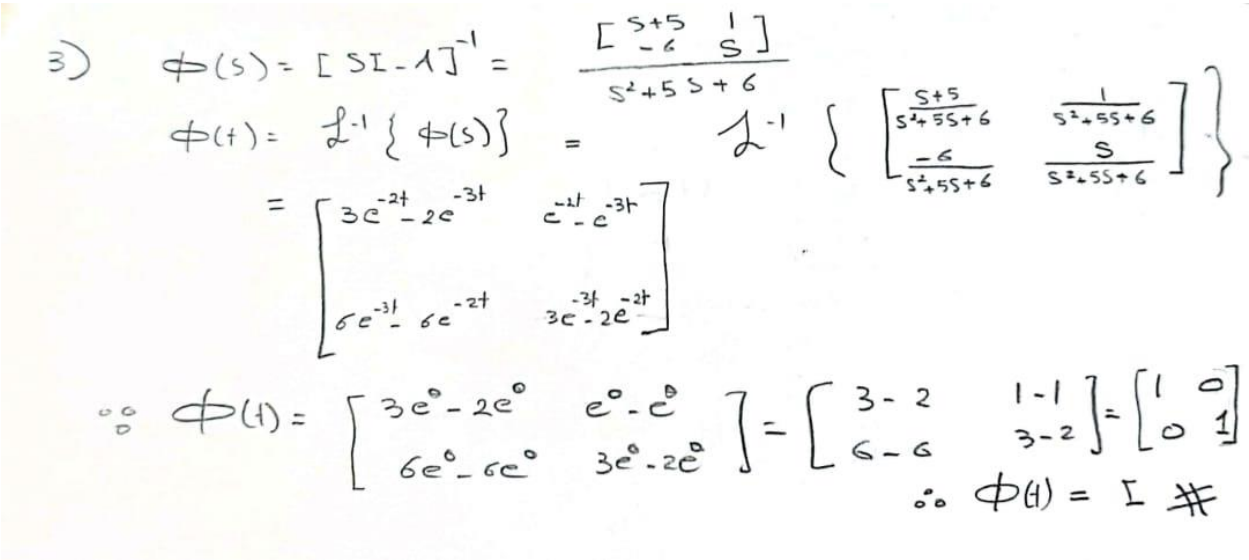
```
% Calculate State Transition Matrix
state_transition_matrix_in_Sdomain = inv(s*eye(size(A)) - A);
state_transition_matrix_in_Tdomain = ilaplace(state_transition_matrix_in_Sdomain);
state_transition_matrix_in_Tdomain = [state_transition_matrix_in_Tdomain(1,1), state_transition_matrix_in_Tdomain(1,2)
                                      state_transition_matrix_in_Tdomain(2,1), state_transition_matrix_in_Tdomain(2,2)];

% Verify state transition matrix at time = zero = I
state_transition_matrix_at_zero = subs(state_transition_matrix_in_Tdomain, t, 0);
I = eye(size(A));
disp("The state transition matrix in S-domain : ");
disp(state_transition_matrix_in_Sdomain);
disp("The state transition matrix : ");
disp(state_transition_matrix_in_Tdomain);
disp("The state transition matrix at t = 0 : ");
disp(state_transition_matrix_at_zero);
disp("The Idenitiy  matrix I : ");
disp(I);
```

- SS for the output of the code:

Command Window

```
            Name: ''
    SamplingGrid: [1×1 struct]

The state transition matrix in S-domain :
[(s + 5)/(s^2 + 5*s + 6), 1/(s^2 + 5*s + 6)]
[     -6/(s^2 + 5*s + 6), s/(s^2 + 5*s + 6)]

The state transition matrix :
[3*exp(-2*t) - 2*exp(-3*t),     exp(-2*t) - exp(-3*t)]
[6*exp(-3*t) - 6*exp(-2*t), 3*exp(-3*t) - 2*exp(-2*t)]

The state transition matrix at t = 0 :
[1, 0]
[0, 1]

The Idenitiy  matrix I :
     1     0
     0     1
```

- SS for the hand analysis:



- **We notice the state transition matrix equals to identity matrix so the verification is valid.**

4) MATLAB code to verify that : $\dot{\emptyset}(t) = A\emptyset(t)$

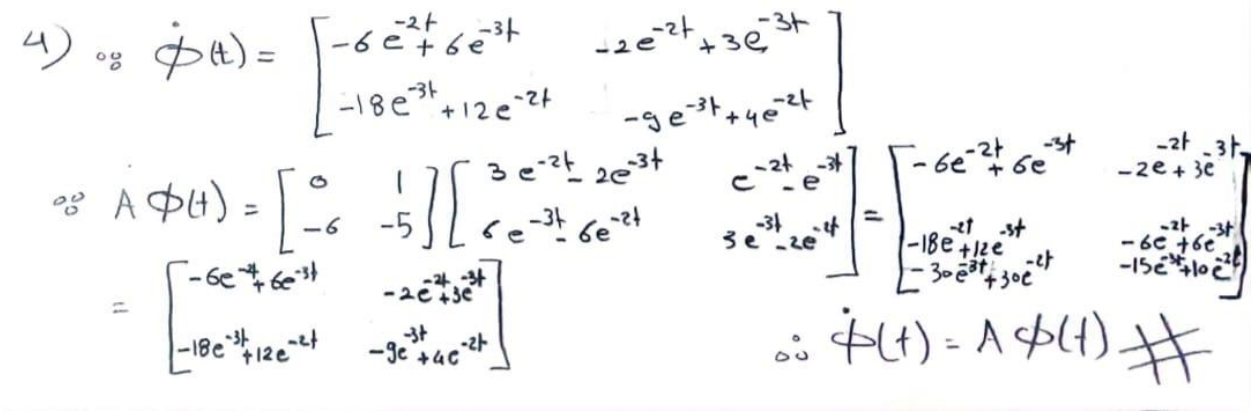
- SS of the code for verify that : $\dot{\emptyset}(t) = A\emptyset(t)$

```
41      % Calculate the derivative of state transition matrix and verify that the
42      % derivative of state transition matrix equals = A * state transition matrix
43 -    diff_state_transition_matrix = diff(state_transition_matrix_in_Tdomain,t);
44 -    state_transition_matrix_multipy_A_matrix = A*state_transition_matrix_in_Tdomain;
45 -    disp("The derivative of state transition matrix : ");
46 -    disp(diff_state_transition_matrix);
47 -    disp("The state transition matrix multiplied by A matrix: ");
48 -    disp(state_transition_matrix_multipy_A_matrix);
49
```

- SS for the output of the code:

```
Command Window

The derivative of state transition matrix :
[  6*exp(-3*t) - 6*exp(-2*t), 3*exp(-3*t) - 2*exp(-2*t)]
[12*exp(-2*t) - 18*exp(-3*t), 4*exp(-2*t) - 9*exp(-3*t)]

The state transition matrix multiplied by A matrix:
[  6*exp(-3*t) - 6*exp(-2*t), 3*exp(-3*t) - 2*exp(-2*t)]
[12*exp(-2*t) - 18*exp(-3*t), 4*exp(-2*t) - 9*exp(-3*t)]
```

- SS for the hand analysis:



- **We notice the derivative of the state transition matrix equals to the state transition matrix multiplied A-matrix so the verification is valid.**

5) MATLAB code to Check Controllability and Observability:

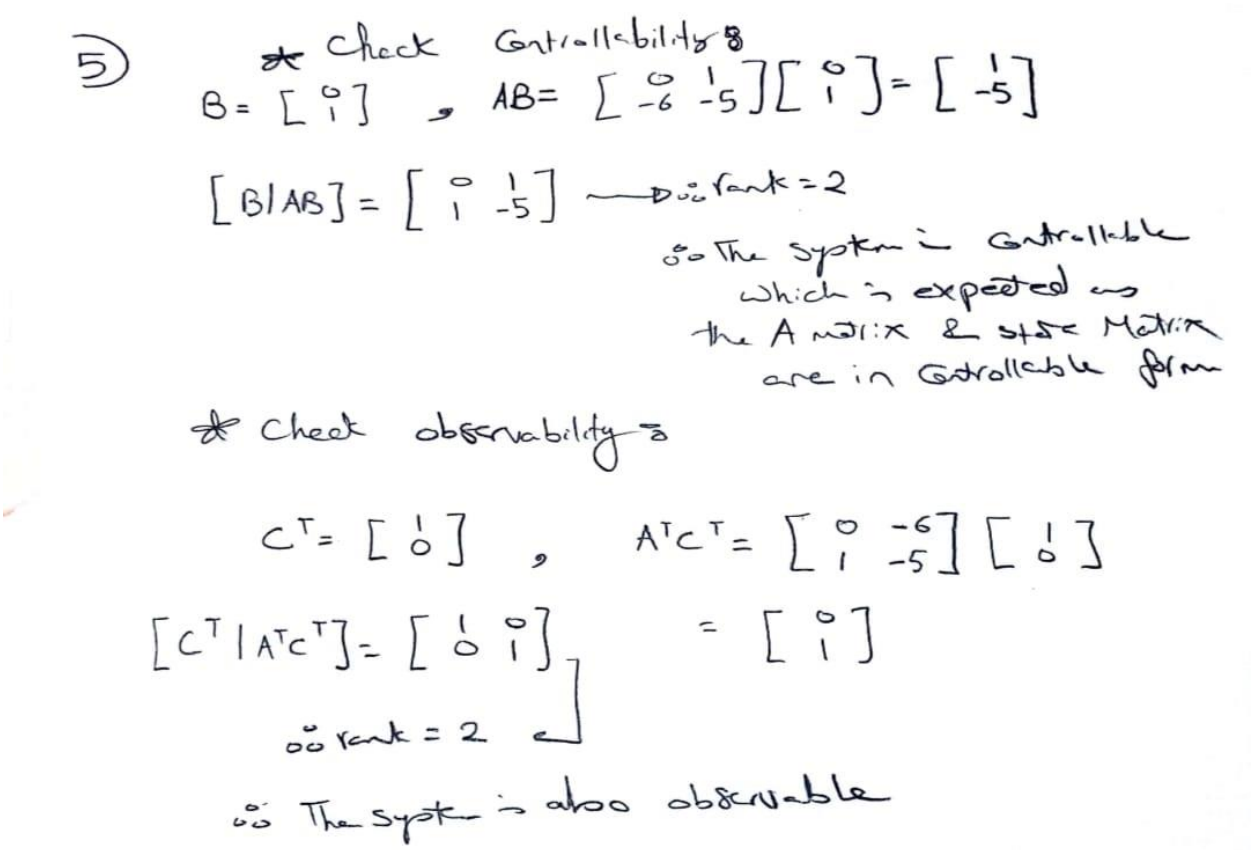
- SS of the code for checking controllability and observability:

```
50      % Check Controllability
51 -    is_controllable = rank(ctrb(A, B)) == size(A, 1);
52 -    disp("Is the system controllable? " + string(is_controllable));
53
54      % Check Observability
55 -    is_observable = rank(obsv(A, C)) == size(A, 1);
56 -    disp("Is the system observable? " + string(is_observable));
57
```

- SS for the output of the code:

```
Command Window
  Is the system controllable? true
  Is the system observable? true
```

- SS for the hand analysis:



- **Yes, the results are expected as the state space is represented in the controllable canonical form so we are sure that the system is controllable.**

6) MATLAB code for calculating unforced (Homogeneous) solution of the states and unforced response y(t):

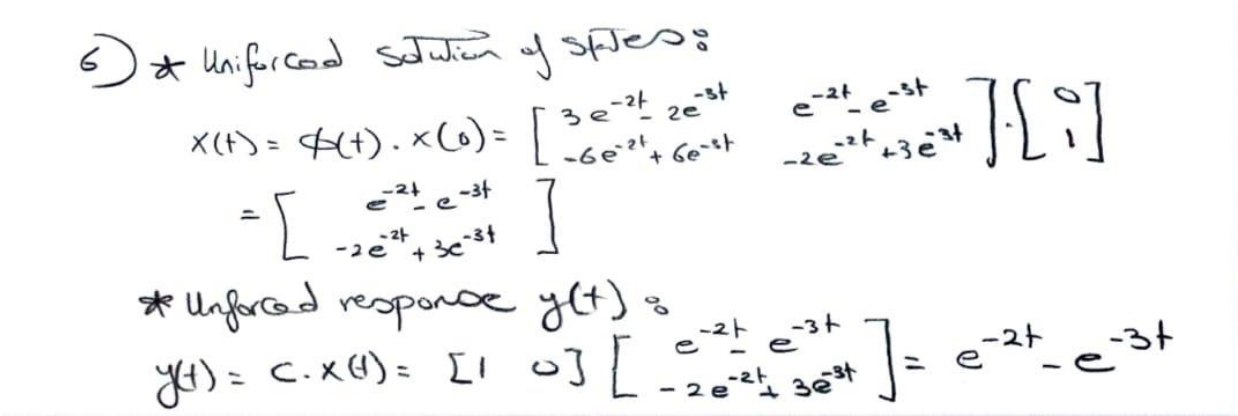
- SS of the code for unforced (Homogeneous) solution of the states and unforced response y (t):

```
%%
states_initial_conditions = [0;1];
x_unforced = state_transition_matrix_in_Tdomain * states_initial_conditions;
disp("Unforced (Homogeneous) Solution of the states : x(t)= " );
disp(x_unforced);
y_unforced = C * x_unforced;
disp("Unforced response : y(t)= " );
disp(y_unforced);
```

- SS for the output of the code:

```
Command Window
Unforced (Homogeneous) Solution of the states : x(t)=
    exp(-2*t) - exp(-3*t)
3*exp(-3*t) - 2*exp(-2*t)

Unforced response : y(t)=
exp(-2*t) - exp(-3*t)
```

- SS for the hand analysis:



7) MATLAB code for the states solution and the response to unit step input:

- SS of the code for forced (Non-homogeneous) solution of the states and forced response y (t):

```
66
67 -    x_forced = x_unforced + ilaplace(inv(s*eye(size(A)) - A) * B * (1/s));
68 -    disp("forced (Non_homogeneous) Solution of the states : x(t)= " );
69 -    disp(x_forced);
70 -    y_forced = C * x_forced;
71 -    disp("forced response : y(t)= " );
72 -    disp(y_forced);
```

- SS for the output of the code:

```
Command Window
forced (Non_homogeneous) Solution of the states : x(t)=
exp(-2*t)/2 - (2*exp(-3*t))/3 + 1/6
            2*exp(-3*t) - exp(-2*t)

forced response : y(t)=
exp(-2*t)/2 - (2*exp(-3*t))/3 + 1/6

fx >>
```

- SS for the hand analysis: