Part 1

Q1)

Code

```
%--------Q1--------
% Define the open-loop transfer function G(s)
num_G = 1;
den_G = [1 1 0];     % s(s+1) = s^2 + s
G_S = tf(num_G, den_G)

% Define the feedback transfer function H(s)
num_H = [1];
den_H = [1];         % Unity Feedback
H_S = tf(num_H, den_H)
```

Output:

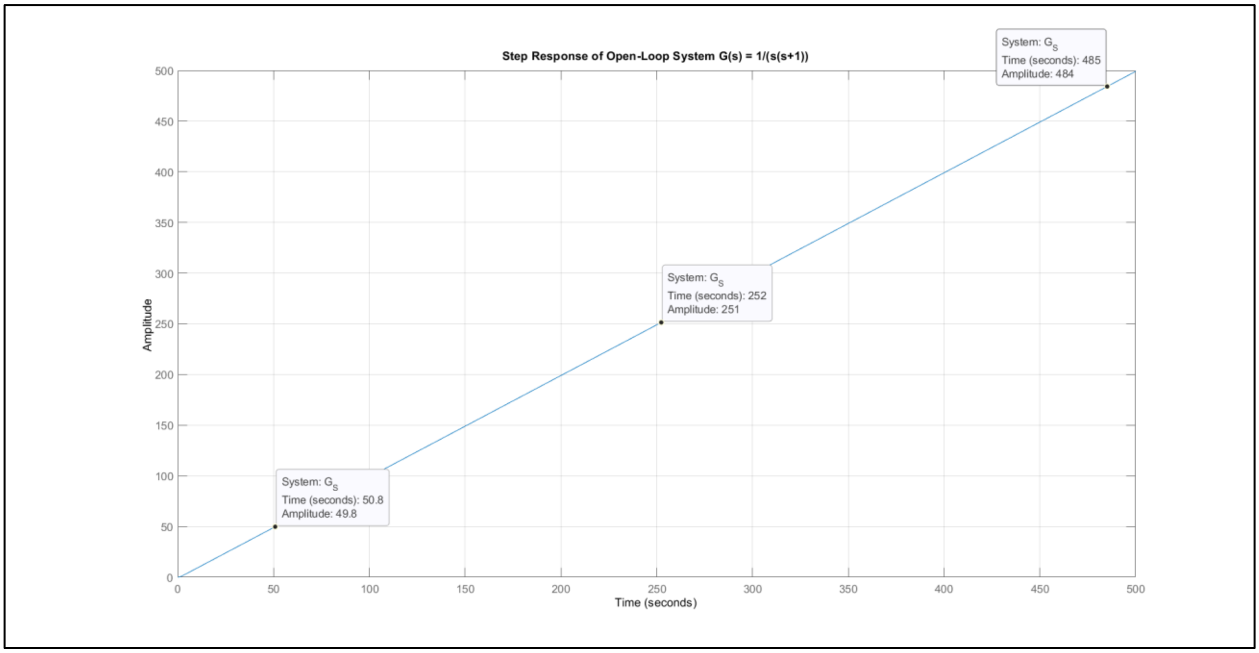
```
G_S =

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

Continuous-time transfer function.


H_S =

  1

Static gain.
```

Q2) the $step()$ command to plot the output of $G(S)$

Code: We made a function that plots step time response and checks stability

```
%--------Q2-------- Step Response of G(s) (Open-Loop)
% Plot step response of G(s)
draw_step(G_S, 'Open-Loop System G(s)');


function draw_step(sys, sys_name)
% Create figure
    figure;

    % Plot step response
    step(sys);
    title(['Step Response of ', sys_name]);
    grid on;

    % Display poles
    poles = pole(sys);
    disp(['Poles of ', sys_name, ':']);
    disp(poles);

    % Check stability
    if all(real(poles) < 0)
        disp('System is stable (all poles in LHP)');
    elseif any(real(poles) > 0)
        disp('System is unstable (at least one pole in RHP)');
    else
        disp('System is marginally stable (poles on imaginary axis)');
    end
end
```

Output:




```
Poles of Open-Loop System G(s):
     0
    -1

System is marginally stable (poles on imaginary axis)
```

Q3)

Code:

```
%--------Q3-------- Closed-Loop Analysis
disp('Closed-Loop TF using feedback():');
T_feedback = feedback(G_S, H_S)

disp('Closed-Loop TF using manual formula (G/(1+GH)):');
T_manual =   (1 / (1 + G_S * H_S)) * G_S;  % Equivalent to T(s) = G/(1+GH)
T_manual = minreal(T_manual)               % Cancel common terms
```

Output:

```
Closed-Loop TF using feedback():

T_feedback =

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

Continuous-time transfer function.

Closed-Loop TF using manual formula (G/(1+GH)):

T_manual =

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

Continuous-time transfer function.
```

Q4)

Code:

```
%--------Q4-------- Step Response of T(s) (Closed-Loop)
% Plot step response of T(s)
draw_step(T_feedback, 'Closed-Loop System T(s)');
```

Output:

	Before $G_c(s)$	**with $G_c(s)$**
$e_{s.s}\big\vert_{due\ to\ dist.}$	½	zero
ζ	0.4	0.69
ω_n	14.8	58
M_p	25%	5%
t_s	0.67	0.1

Ouput:

System: Closed-Loop System T(s)

Poles: -0.5-0.86603i -0.5+0.86603i

Stability: stable (all poles in LHP)

Over shoot MP: 16.2929% at t = 3.592 sec

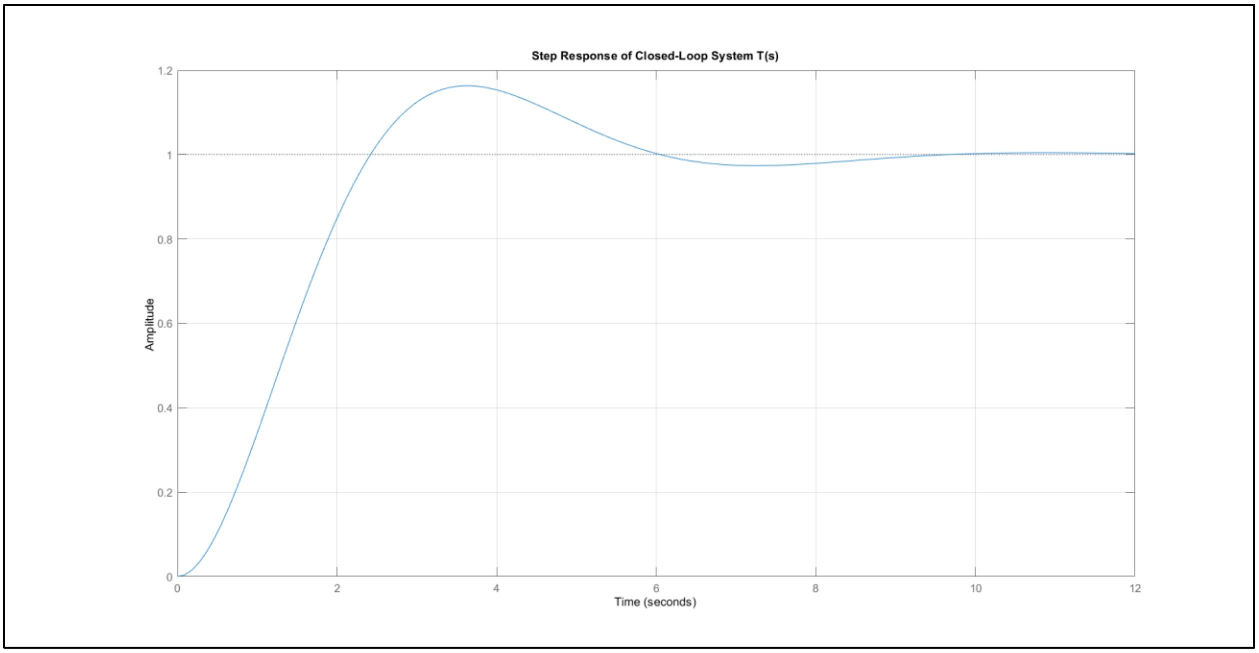
Damping ratio (ζ): 0.500

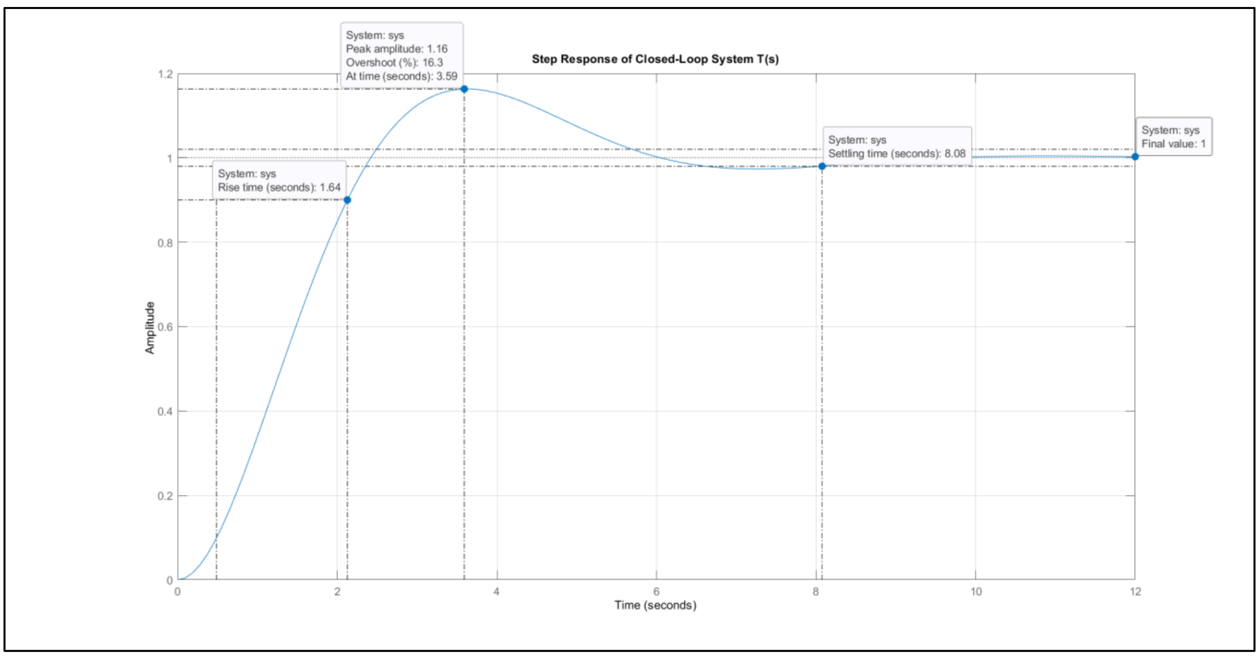
Natural frequency (ωn): 1.000 rad/s

Settling time (2%): 8.1051 sec

Rise time (10-90%): 1.6579 sec

Steady-state value: 1.0014


Step Response of Closed-Loop System T(s)
Amplitude
Time (seconds)

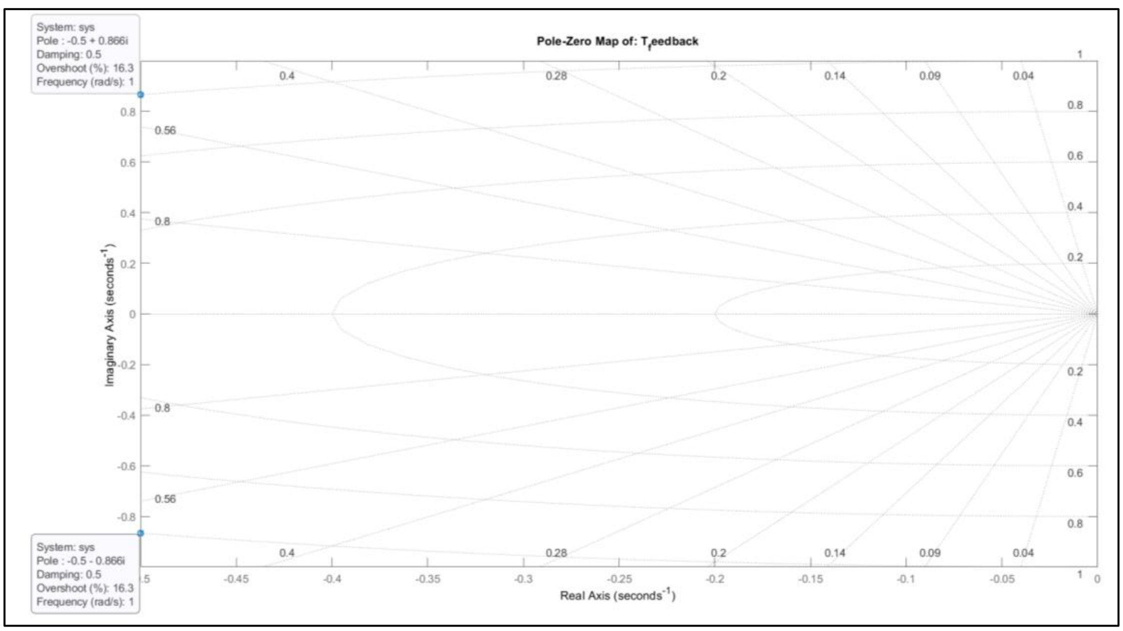

Step Response of Closed-Loop System T(s)
System: sys
Peak amplitude: 1.16
Overshoot (%): 16.3
At time (seconds): 3.59
System: sys
Rise time (seconds): 1.64
System: sys
Settling time (seconds): 8.08
System: sys
Final value: 1
Amplitude
Time (seconds)

Q5)

Code:

```
function [poles] = draw_poles(sys)

    % Create figure
    figure;

    % Plot pole-zero map
    pzmap(sys);
    title(['Pole-Zero Map of: ' inputname(1)]);
    grid on;

    % Get poles
    poles = pole(sys);

    % Display poles
    disp(['Poles of ' inputname(1) ':']);
    disp(poles);


    % Damping characteristics (for complex poles)
    if ~isreal(poles)
        [wn, zeta] = damp(sys);
        fprintf('Damping ratio (?): %.3f\n', zeta(1));
        fprintf('Natural frequency (?n): %.3f rad/s\n', wn(1));
    end

end
```

Output:



```
Poles of T_feedback:
  -0.5000 + 0.8660i
  -0.5000 - 0.8660i

Damping ratio (ζ): 0.500
Natural frequency (ωn): 1.000 rad/s
```

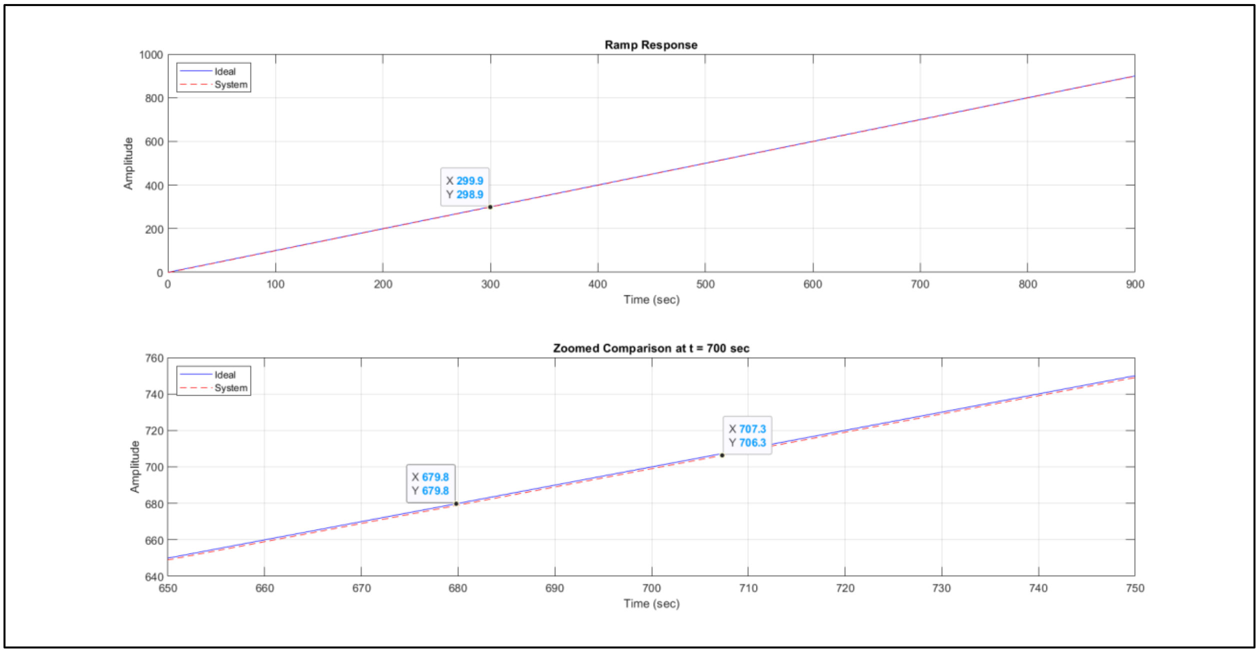
Q6,Q7 is done

Q8

Code:

```
function [ess, t_out, y_out] = draw_ramp(sys, t_end, zoom_time)
{
    % Set defaults if not provided
    if nargin < 2
        t_end = 100;
    end
    if nargin < 3
        zoom_time = 700;
    end

    % Create time vector
    t = 0:0.1:t_end;

    %getting the ramp
    ramp = tf(1,[1 0]);

    % Get response data
    [y_sys, t_sys] = step(sys.*ramp, t);
    [y_ideal, t_ideal] = step(ramp, t);

    % Create figure with three subplots
    figure;

    % Subplot 1: Ideal ramp input
    subplot(2,1,1);
    plot(t_ideal, y_ideal, 'b');
    hold on;
    plot(t_sys, y_sys, 'r--');
    title('Ramp Response');
    xlabel('Time (sec)');
    ylabel('Amplitude');
    legend('Ideal', 'System', 'Location', 'northwest');
    grid on;
    hold off;

    % Subplot 2: Zoomed comparison
    subplot(2,1,2);
    plot(t_ideal, y_ideal, 'b');
    hold on;
    plot(t_sys, y_sys, 'r--');
    xlim([zoom_time-50 zoom_time+50]);
    title(['Zoomed Comparison at t = ', num2str(zoom_time), ' sec']);
    xlabel('Time (sec)');
    ylabel('Amplitude');
    legend('Ideal', 'System', 'Location', 'northwest');
    grid on;
    hold off;

end
```

Output:



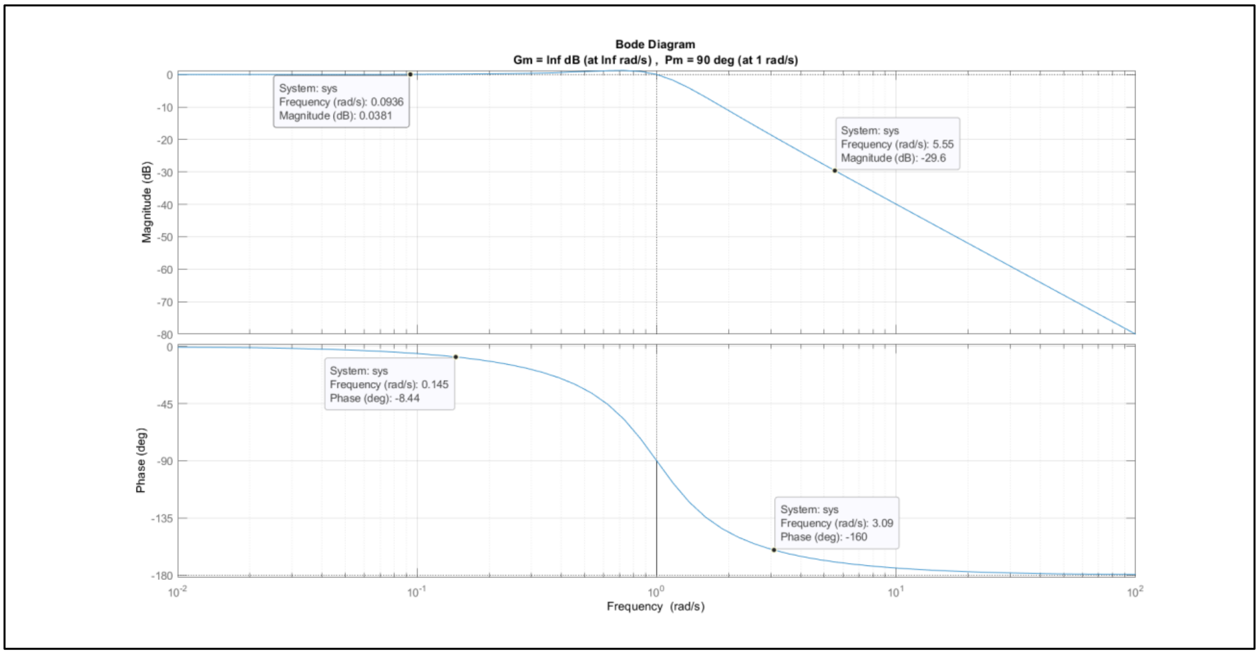
Steady-state error (ess): 1

Q9)

Code:

```
function [Gm, Pm, Wgc, Wpc] = draw_Bode_Plot(sys)
% BODE_PLOT Analyzes system stability margins and compares margin()
%   Bode_Plot(sys)
%
%   Input:
%       sys - Transfer function (tf object or state-space model)
%   Outputs:
%       Gm - Gain margin (dB)
%       Pm - Phase margin (degrees)
%       Wgc - Gain crossover frequency (rad/sec)
%       Wpc - Phase crossover frequency (rad/sec)

    % Create margin plot
    figure;
    margin(sys);
    grid on;

    % Get stability margins
    [Gm, Pm, Wgc, Wpc] = margin(sys);

    % Display results
    disp(['=== Stability Margins for ' inputname(1) ' ===']);
    disp(['Gain Margin: ', num2str(Gm), ' dB at ', num2str(Wgc), ' rad/s']);
    disp(['Phase Margin: ', num2str(Pm), '° at ', num2str(Wpc), ' rad/s']);

end
```

Output:



=== Stability Margins for T_feedback ===

Gain Margin: Inf dB at Inf rad/s

Phase Margin: 90° at 1 rad/s