Exercise 1 protocol

Alessandro Rotondi - Matr.Nr.
Frederic Heil - Matr.Nr.
Nithin Badri - Matr.Nr.
Sudeesh Suragari - Matr.Nr.

November 9, 2025

Exercise 1a): Plotting trigonometric functions

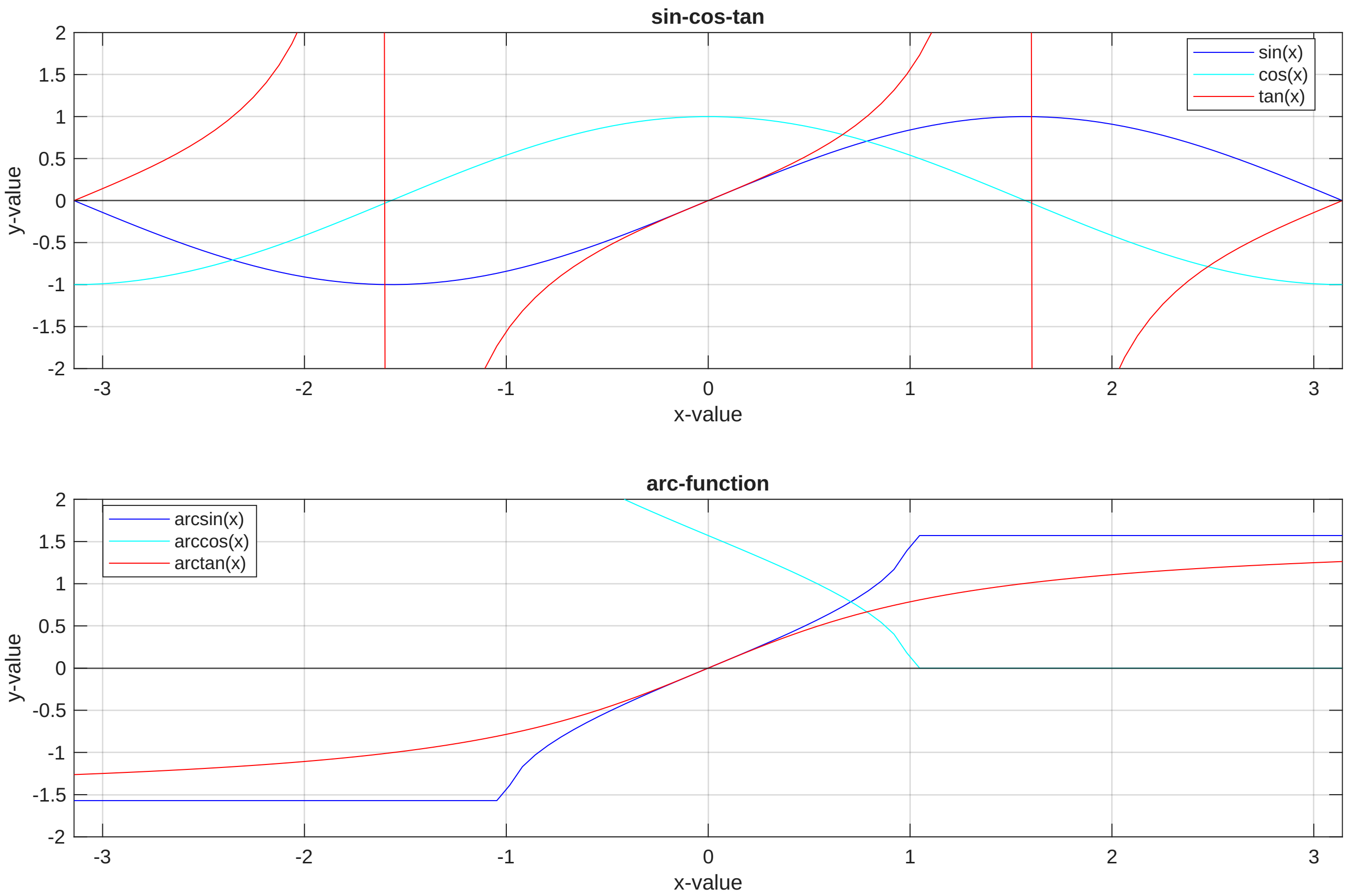

Figure 1: Plots of trigonometric functions.

As shown in Figure 1 above, several trigonometric functions have been plotted into two different figures, one for *sin*, *cos*, *tan* and one for their corresponding inverse functions. We chose a sufficient linspace X ranging from $-\pi$ to π with 100 steps.

Matlab Code:

```
clear; clf; clc;

xmax = pi;

x = linspace(-xmax,xmax,100);   % Creation array of X value

y.sin = sin(x);     % Calcolous of sin function
y.cos = cos(x);     % Calcolous of cos function
y.tan = tan(x);     % Calcolous of tan function
y.asin = asin(x);   % Calcolous of arcsin function
y.acos = acos(x);   % Calcolous of arccos function
y.atan = atan(x);   % Calcolous of arctan function


figure(1);                     % Creation of first figure (tab)
subplot(2,1,1)                 % definition of the subplot (2 rows, 1 colums)
plot(x,y.sin,'b',x,y.cos,'c',x,y.tan,'r'); %plot of the function sin, cos, tan
hold on;                       % Maintains the previous graphs in the plot
yline(0);                      % Plot of the X axis
axis([-xmax xmax -2 2]);       % Settings of the value in the axes
grid;                          % Enable the grid in the plot
title("sin-cos-tan");          % Write the title of the plot
xlabel("x-value");             % Write the label of X axis
ylabel("y-value");             % Write the label of Y axis
legend("sin(x)","cos(x)","tan(x)",'Location','best'); % Add the legend of the graphs

subplot(2,1,2)
plot(x,y.asin,'b',x,y.acos,'c',x,y.atan,'r');    %plot of the function arcsin, arccos, arctan
hold on;                       % Maintains the previous graphs in the plot
yline(0);                      % Plot of the X axis
axis([-xmax xmax -2 2]);       % Settings of the value in the axes
grid;                          % Enable the grid in the plot
title("arc-function");         % Write the title of the plot
xlabel("x-value");             % Write the label of X axis
ylabel("y-value");             % Write the label of Y axis
legend("arcsin(x)","arccos(x)","arctan(x)",'location','best');  % Add the legend of the graphs
```

Exercise 1b): Calculating minimum distance between lines in 3D space

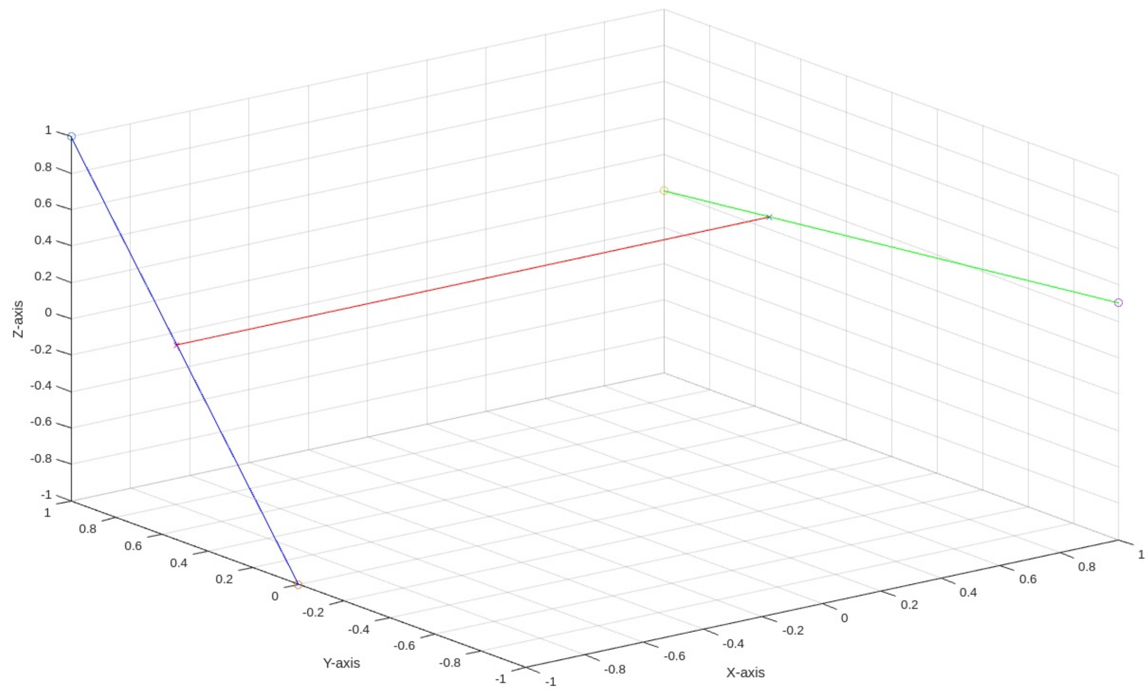

Figure 2: Calculated minimum distance between two lines in three dimensional space.

Figure 2 shows the minimum distance between the two lined defined by the four points A, B, C and D. At first we can calculate the directional vector for the two straight lines AB and CD as follows:

$$\begin{aligned} V_1 &= B - A \\ V_2 &= D - C \end{aligned} \tag{1}$$

With this we can define the two line equations $L_1(t)$ and $L_2(u)$ as follows:

$$\begin{aligned} L_1(t) &= A + t * V_1 \\ L_2(u) &= C + u * V_2 \end{aligned} \tag{2}$$

Next up, we define the difference between support vectors A and C as P_1P_2 as:

$$P_1P_2 = C - A \tag{3}$$

A single solution for the closest approach between two lines exists if they are not parallel. To check this requirement, we have to check if the cross product $n = V_1 \times V_2$ between the directional vectors is a vector of nonzero magnitude. Specifically we check if this value is greater than a threshold ϵ, which is a value close to zero because of floating point arithmetic.

$$|n| < \epsilon \tag{4}$$

Hereby we chose the value $\epsilon = 10^{-6}$. If the condition 4 is fulfilled, we can continue on to calculate the values t_0 and u_0 which correspond to the closest approach of L_1 and L_2 as follows:

$$\begin{aligned} t_0 &= \frac{(V_2 \times n) \cdot P_1P_2}{n \cdot n} \\ u_0 &= \frac{(V_1 \times n) \cdot P_1P_2}{n \cdot n} \end{aligned} \tag{5}$$

Inserting the values t_0 into $L_1(t)$ and u_0 into $L_2(u)$ yields the two points which correspond to the closest approach on each line.

Matlab Code:

```
clear; clf; clc;

% segments point
A = [-1;1;1];
B = [-1;0;-1];
C = [1;1;0];
D = [1;-1;0.3];

figure()       %Creation of the second figure (tab)
[C1,C2,dist_min] = plotdistance(A,B,C,D,1)  % Call to the function


% function to get the minimum distance between two segment
% segments shall be defined as l1: A + v1*t, l2: C + v2*u
function [C1,C2,dist_min] = plotdistance(A,B,C,D,plotVar)

    % 1. calc direction vector
    v1 = B - A;
    v2 = D - C;
    % difference between support vectors
    p1p2 = C - A;

    % cross product perpendicular to both segments
    n = cross(v1, v2)

    % chech if segments are parallel
    if norm(n) < 1e-6
        disp('The segments are parallel');
        return
    else
        % Resolution for t and u
        t0 = (dot(cross(v2, n), p1p2))/dot(n,n);
        u0 = (dot(cross(v1, n), p1p2))/dot(n,n);

        % point C1 and C2
        C1 = A + t0 * v1;
        C2 = C + u0 * v2;

        % minimun distance
        dist_min = norm(C1 - C2);
    end

    % if the plot variable is HIGH print plot the results
    if plotVar == 1          % HIGH ==> plot on
        hold on;
        grid on;
        % Plot point A,B,C,D
        plot3(A(1),A(2),A(3),'o',B(1),B(2),B(3),'o',C(1),C(2),C(3),'o',D(1),D(2),D(3),'o');
        % Plot segment AB and CD
        plot3([A(1),B(1)],[A(2),B(2)],[A(3),B(3)],'b');
        plot3([C(1),D(1)],[C(2),D(2)],[C(3),D(3)],'g');
        % plot C1 and C2 position
        plot3(C1(1),C1(2),C1(3),'x');
        plot3(C2(1),C2(2),C2(3),'x');
        % plot minimum lenght segment
        plot3([C1(1),C2(1)],[C1(2),C2(2)],[C1(3),C2(3)],'r');
        % NAme of the axes
```

```
        xlabel('X-axis');
        ylabel('Y-axis');
        zlabel('Z-axis');
        % Possibility to set the view position, view(3) makes plot three
        % dimensional
        %view([-39.69 35.28])
        view(3)
    end
end
```

Exercise 1c): Bode and step plots for two functions

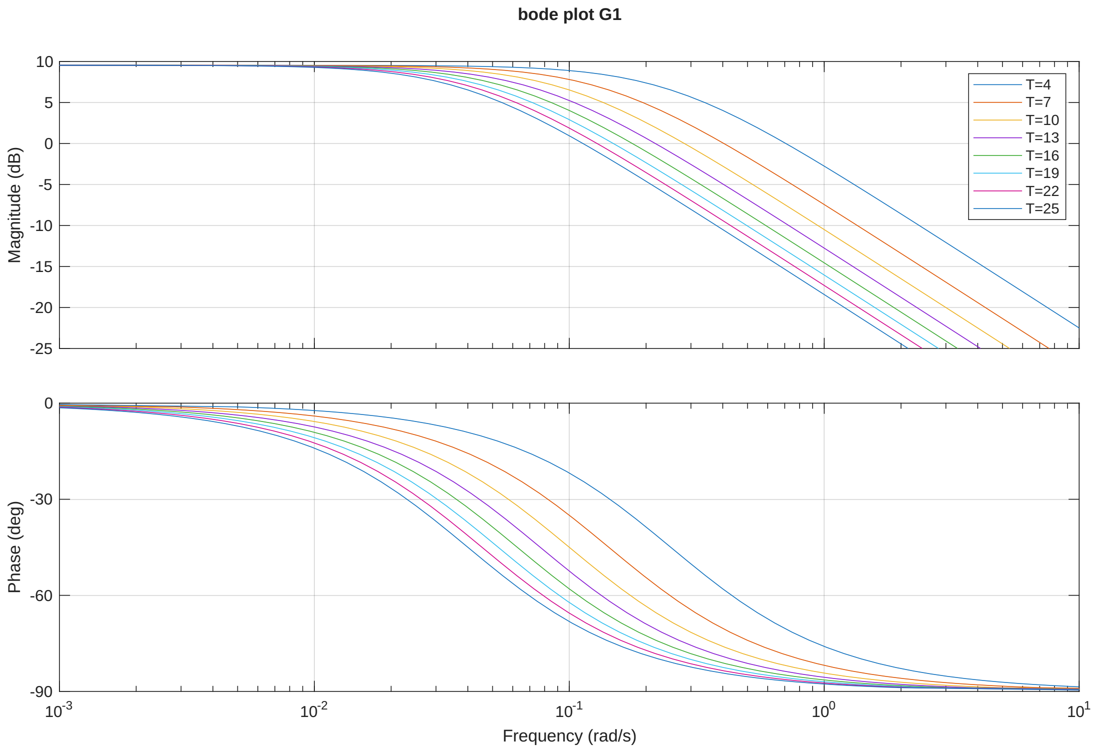

Figure 3: Bode plot for G_{PT1}

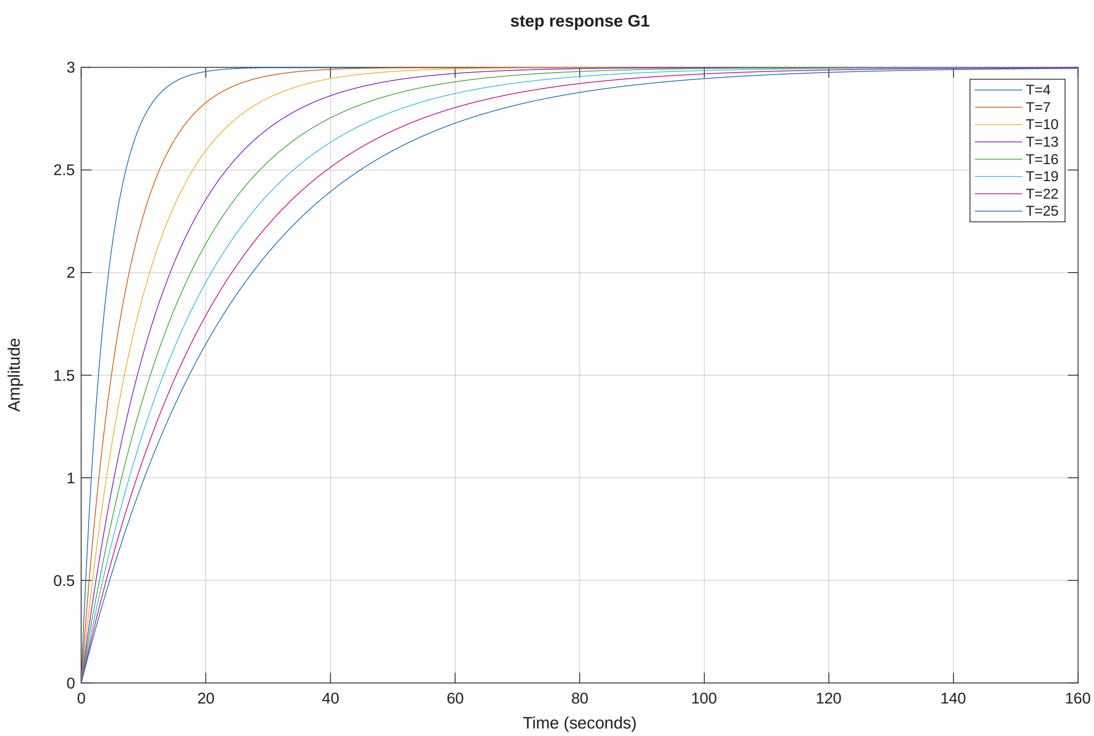

Figure 4: Step plot for G_{PT1}

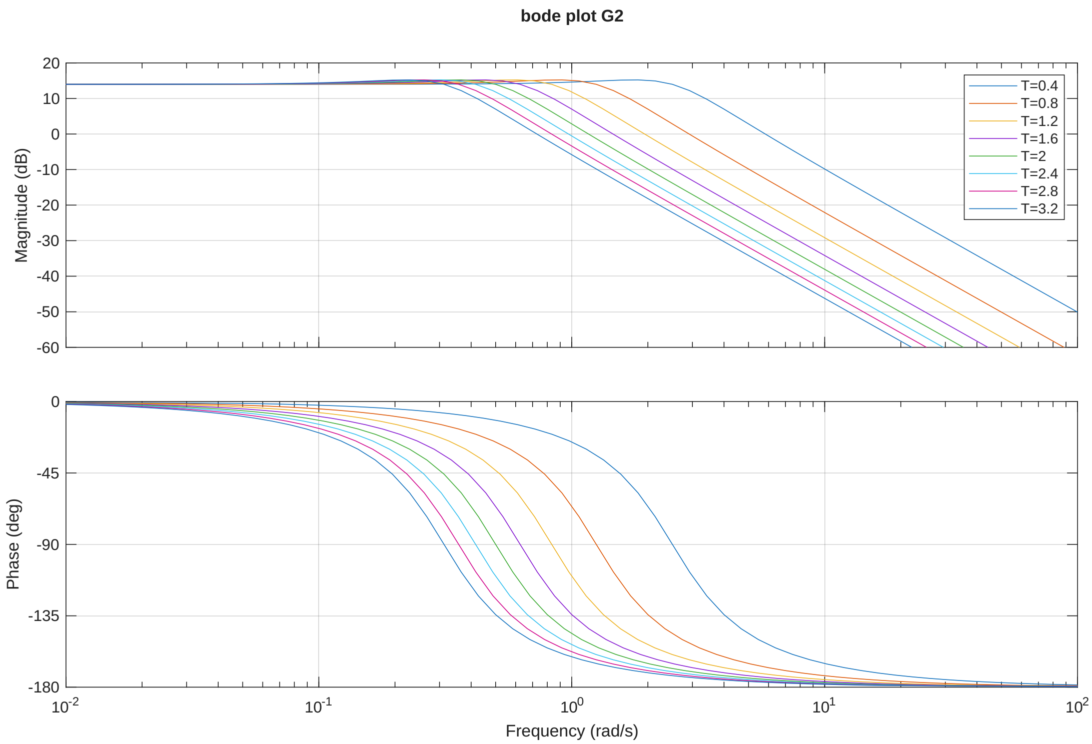

Figure 5: Bode plot for G_{PT2}

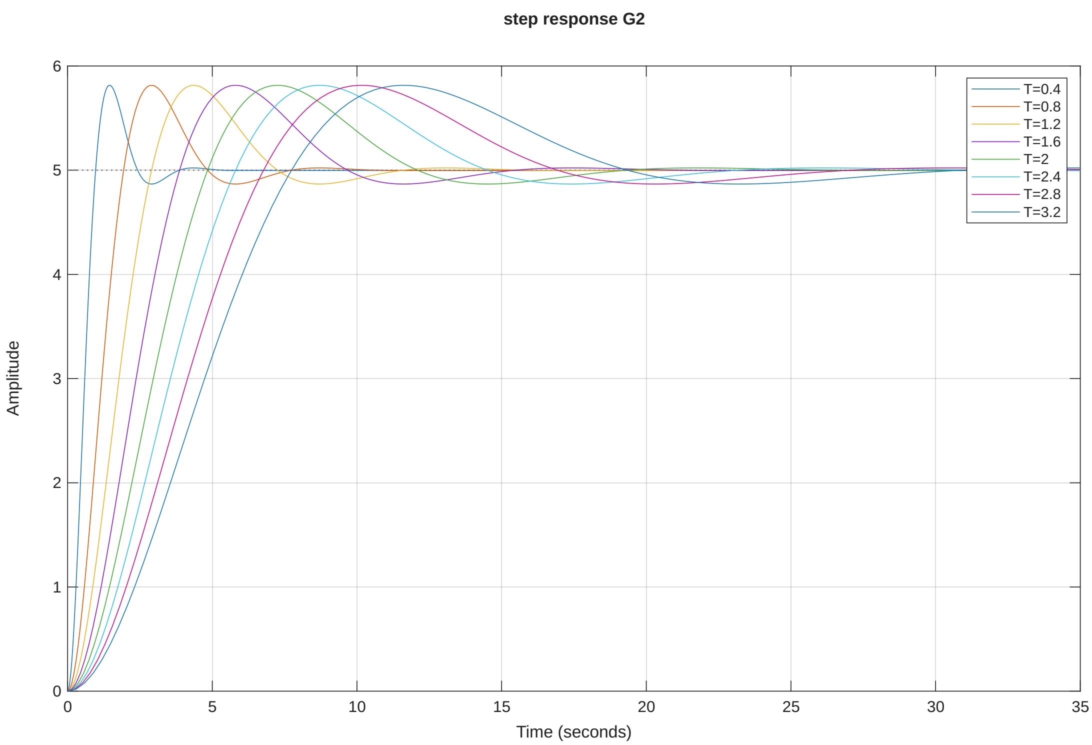

Figure 6: Step plot for G_{PT2}

```
clear; clf; clc;

s = tf('s');     % Definition of the s variable

```

```
% Paramenters
kg1 = 3;
Tg1 = 10;

kg2 = 5;
Tg2 = 0.4;
dg2 = 0.5;

legend_string_pt1 = 'legend('; % String definition for the legend

for i = 1:8               % Cicle to plot the graph with different T value
    Ti = Tg1-9+3*i;       % Modify the T value
    G1 = kg1/(1+Ti*s);    % Definition of the Transfer function with the new parameters

    figure(1);
    bode(G1);             % Plot in Figure(1) the bode diagram
    hold on;              % Maintains the previous graphs in the plot

    figure(2);
    step(G1);             % Plot in Figure(2) the step response diagram
    hold on               % Maintains the previous graphs in the plot

    legend_string_pt1 = [legend_string_pt1 '''T=' num2str(Ti) ''',']; % Add to the legend
        string the new graph parameters
end
legend_string_pt1(end) = ')'; % complete legend command

% Display title and legend for figure (1) and (2)
figure(1)
title("bode plot G1")
eval(legend_string_pt1);
grid on;
figure(2)
title("step response G1");
eval(legend_string_pt1);
grid on;



legend_string_pt2 = 'legend('; % String definition for the legend

for i = 1:8           % Cicle to plot the graph with different T value
    Ti = Tg2*i;       % Modify the T value
G2 = kg2/(1+2*Ti*dg2*s+(Ti^2)*(s^2));    % Definition of the Transfer function with the new
     parameters

    figure(3);
    bode(G2);         % Plot in Figure(3) the bode diagram
    hold on;          % Maintains the previous graphs in the plot

    figure(4);
    step(G2);         % Plot in Figure(4) the step response diagram
    hold on           % Maintains the previous graphs in the plot

    legend_string_pt2 = [legend_string_pt2 '''T=' num2str(Ti) ''',']; % Add to the legend
        string the new graph parameters
end
legend_string_pt2(end) = ')'; % complete legend command

% Display title and legend for figure (3) and (4)
figure(3)
title("bode plot G2")
eval(legend_string_pt2);
```

```
grid on;
figure(4)
title("step response G2");
eval(legend_string_pt2);
grid on;
```

Exercise 1e)

As there is no d) exercise, we continue on to exercise e).

```
clear; clf; clc;

s = tf('s');
% Parameters
k = 3;
Tt = 0.0;
T = 0.4;
d = 0.5
w = linspace(0.001,100,100000); % Freq values
% Transfer function
G = k/(1+2*T*d*s+(T^2)*(s^2))*exp(-Tt*s);
num = cell2mat(G.Numerator);
den = cell2mat(G.Denominator);

% Call to function
[mag, phase]=mybode3(num,den,Tt,w);

% Plot of the results
figure(1)
subplot(2,1,1);
semilogx(w,mag);
title("Mag");
grid on;
xlim([w(1) w(length(w))]);
ylim([min(mag)-5 max(mag)+5]);
xlabel("Frequency [rad/s]");
ylabel("MAgnitude [dB]");

subplot(2,1,2);
semilogx(w,phase)
title("phase");
grid on;
xlim([w(1) w(length(w))]);
ylim([min(phase)-10 max(phase)+10]);
xlabel("Frequency [rad/s]");
ylabel("Phase [deg]");

figure(2)
bode(G)
grid on;


% Function that get the magnitude and the phase of the G transfer function
% INPUT:
% G = num_coeff/den_coeff:   the numerator and denominator are provided as vectors
% Tt: delay of the transfer function G(s)*exp(-Tt*s)
% w: array of frequency value
% OUTPUT:
% mag_db: array of the magnitude of the transfer function for the frequency
%          values in [dB]
% phase_deg: array of the phase of the transfer function for the frequency
```

```matlab
%             values in [deg]
function [mag_db, phase_deg] = mybode3(num_coeff, den_coeff, Tt, w)

    jw = 1i * w;    % s array, s = jw for each frequency

    Num_jw = polyval(num_coeff, jw);    % Calc numerator for each Frequency
    Den_jw = polyval(den_coeff, jw);    % Calc denominator for each Frequency

    G_jw = (Num_jw ./ Den_jw);          % Calc of G for each frequency


    mag_lin = abs(G_jw);                % Calc of magnitude
    mag_db = 20*log10(mag_lin);         % Conversioni magnitude in [dB]

    phase_rad = angle(G_jw)-w*Tt;       % Calc Phase
    phase_deg = phase_rad * (180 / pi); % Convesione Phase in degree
end
```