Simulation of Mechatronic Systems

Exercise 3 protocol

Alessandro Rotondi - 27445613
Frederic Heil - 28742953
Nithin Badri - 17546426
Sudeesh Suragari - 14646468

December 19, 2025

Exercise 3a): Simulating the step response of a transfer function

In the first sub task of this exercise, we want to simulate the step response of the transfer-function **PT1** which can be defined as follows:

$$G(s) = \frac{K}{1 + T \cdot s} \tag{1}$$

Hereby, variables K and T are to be defined as $K = 5$ and $T = 4$. Furthermore, the differential equation of **PT1** is given as follows:

$$\dot{x} = \frac{1}{T}(K \cdot u - x) \tag{2}$$

Now that this has been established, we can build both equations 1 and 2 within Simulink which yields the following models:

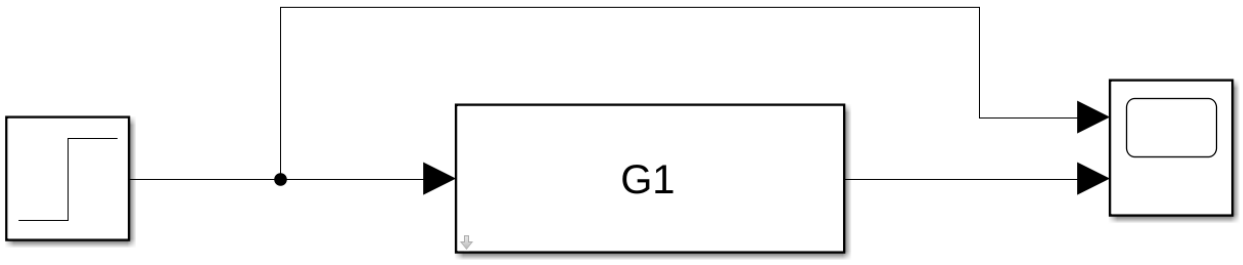

Figure 1: Step equation of **PT1** conveyed as Simulink model.

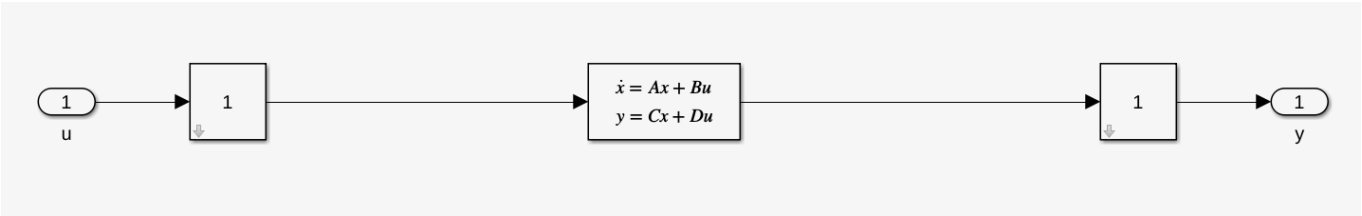

Figure 2: Inner workings of the G1 block of the step equation Simulink model as shown in figure 1.

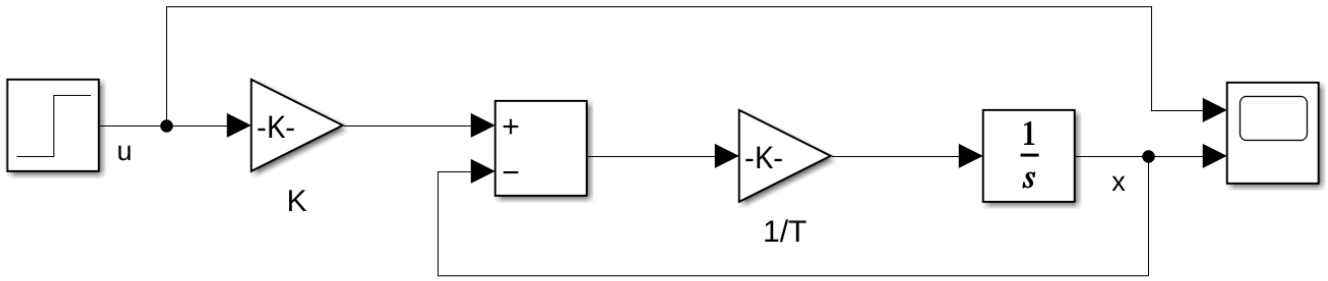

Figure 3: Differential equation of **PT1** conveyed as Simulink model.

As shown in figure 3 above, the value of u gets simulated by a step block. Both simulated values of u and x get forwarded to the scope block, which lets us examine their values within a Matlab script. Something that is not directly apparent from the graphic is the simulation run time, which has been set to 50 seconds.
Within our Matlab code, we can set and access the Simulink simulation as demonstrated in the code snippet below:

Matlab Code Snippet:

```
clear; clf; clc;
% PARAMETERS EXERCISE 1
par1.K = 5;              % Gain
par1.T = 4;              % Time constant
par1.u = 1;              % step value
par1.t.start = 0;        % start simulation time
par1.t.stop = 50;        % stop simulation time

out1 = sim('exercise3_SIM');                              % simulate the system with
    simulink and get the scope datas

figure()
plot(out1.plotG1_Tf.time,out1.plotG1_Tf.signals(1).values,...        % Plot the result of
    the simulation in simulink (transfer function)
     out1.plotG1_Tf.time,out1.plotG1_Tf.signals(2).values)
grid on, xlabel('Time [sec]');legend('step','out'); title('step response transfer function
    SIMULINK');

figure()
plot(out1.plotG1_De.time,out1.plotG1_De.signals(1).values,...        % Plot the result of
    the simulation in simulink (differential equation)
     out1.plotG1_De.time,out1.plotG1_De.signals(2).values)
grid on, xlabel('Time [sec]');legend('step','out'); title('step response differential
    equation SIMULINK');
```

Hereby, we can access concrete scope variables using `plotG1_Tf` and `plotG1_De`, a logging value which has been defined within our Simulink models and is not directly apparent from any graphic.
The plot code of the step response yields the following graphics 4 and 5:

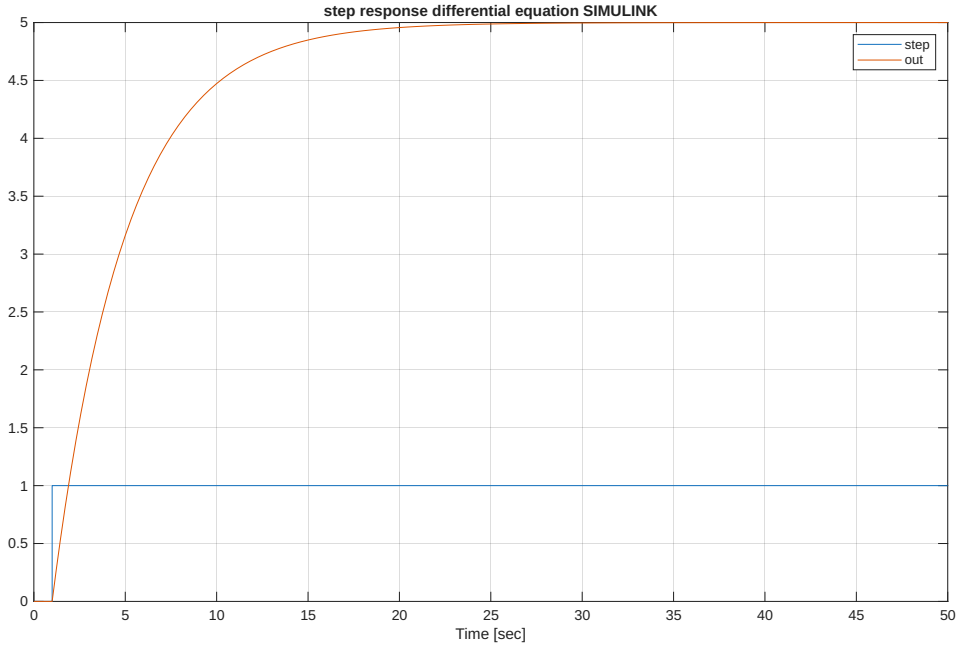

Figure 4: Step response of our Simulink model plotted within Matlab.

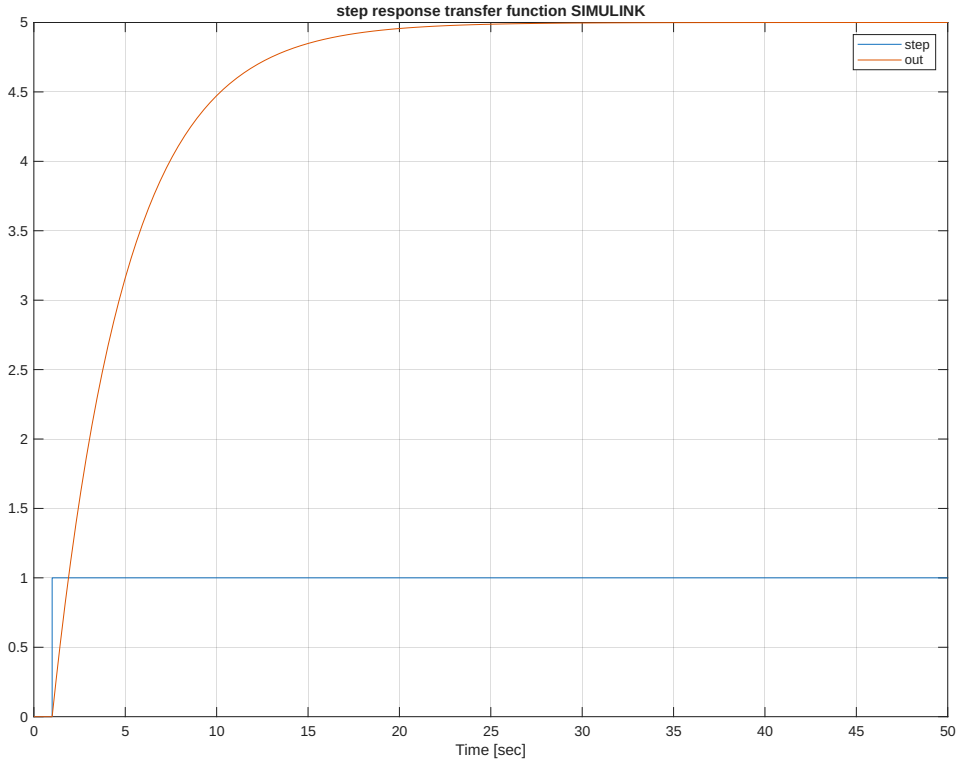

Figure 5: Step response of our Simulink model plotted within Matlab.

As expected, the output `out` of the step function **PT1** converges at a value of 5, which corresponds to the value of parameter K. Also, the output of both models is identical which also corresponds to the expected result.

Exercise 3b): Building a one-mass system in Simulink

Next, we want to simulate a one mass spring system and investigate the influence of the physical parameters on the simulation result. The physical system is illustrated with the following graphic below:

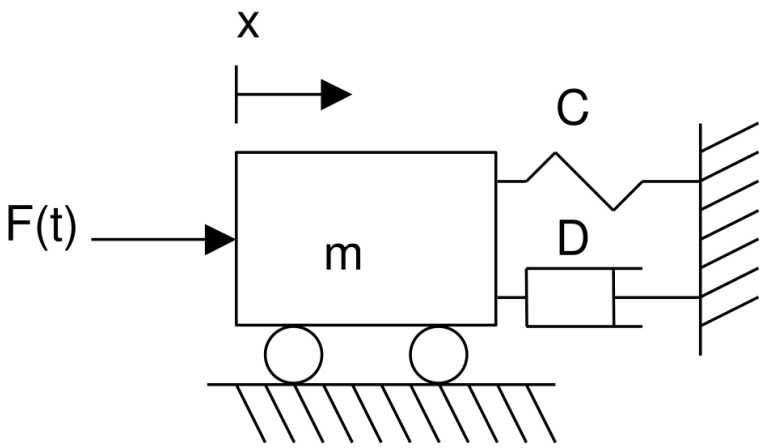

Figure 6: One-mass spring/damper system with some physical characteristics.

Hereby the movement can be described with the following differential equation which is already given by the exercise description as follow:

$$m \cdot \ddot{x} = -C \cdot x - D \cdot \dot{x} + F(t) \tag{3}$$

The initial values are defined as $m = 10kg$, $C = 100\frac{N}{m}$ and $D = 3\frac{Ns}{m}$. Using equation 3 we can build our model in Simulink which looks like this:

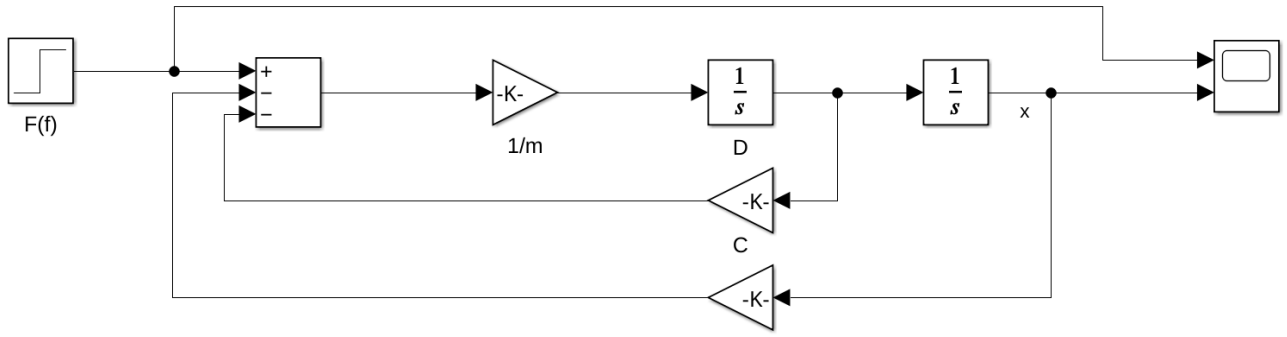

Figure 7: Differential equation of one-mass spring/damper system conveyed as Simulink model.

This model uses two integrator blocks to depict signals of $\ddot{x}$, $\dot{x}$ and x whose outputs get multiplied by their corresponding gain blocks. Furthermore, the initial force $F(t)$ is simulated by a step block.
To investigate the influence of physical parameters on the simulation result, we are at first going to run the simulation with its initial values. Then, for each physical property we are going to execute the simulation twice, once with a decreased and once with an increased value, so that only one property changes at a time. For m, D and C this yields the following plots:

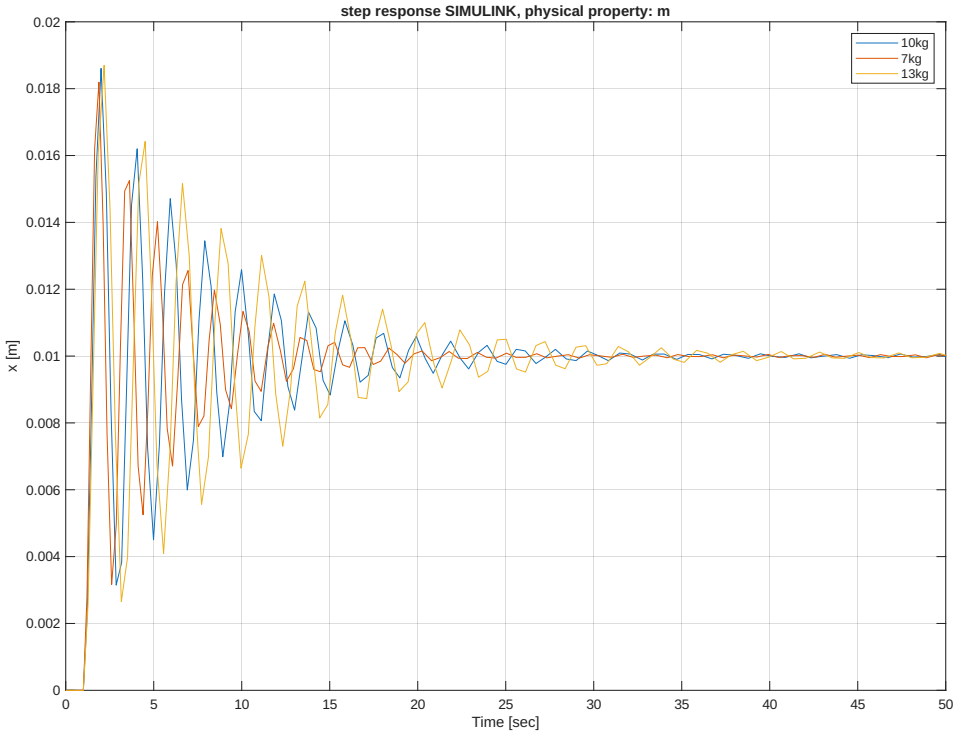

Figure 8: Step response with different values for mass.

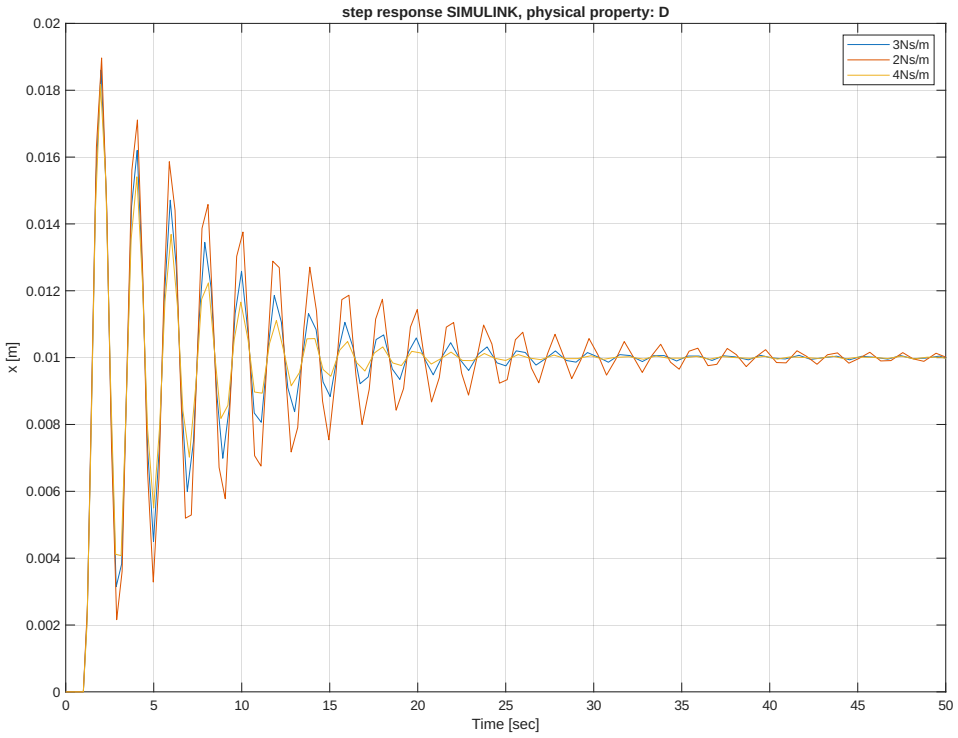

Figure 9: Step response with different values for the damper constant.

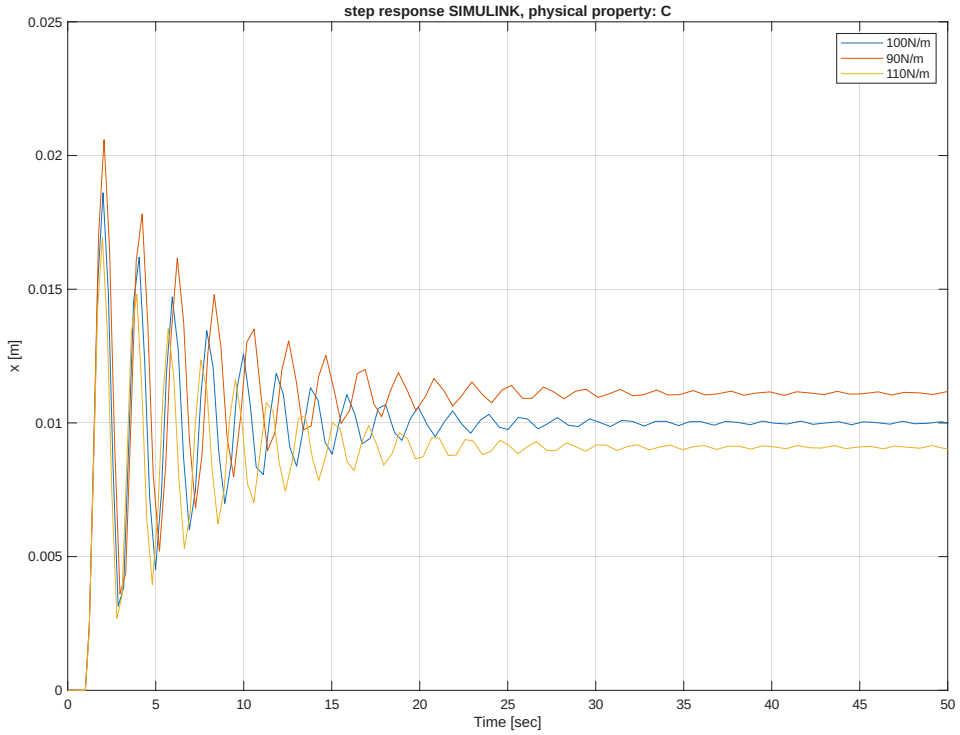

Figure 10: Step response with different values for the spring constant.

Influence of mass (m): As shown in figure 8, changing the mass influences oscillation amplitude and interval. A reduced mass leads to shorter interval and smaller amplitude, while a mass increase leads to longer interval and larger amplitude.
Influence of damper constant (D): As shown in figure 9, changing the damper constant influences only the amplitude, more specifically the rate of amplitude decrease over time. A reduced damper constant leads to a slower decrease of amplitude over time, while an increased damper constant leads to a faster decrease of amplitude over time. The oscillation interval is not being influenced by the damper value.
Influence of spring constant (C): As shown in figure 10, changing the spring constant influences the value to which x converges towards. A reduced spring constant leads to a larger convergent value of x, while an increased spring constant leads to a smaller convergent value of x.

The Matlab code for generating these plots and running the simulation is provided in the snippet below.
Matlab code snippet:

```
% EXERCISE B ------------------------------------------------------------
out2 = sim("exercise3_SIM.slx");      % step response with initial parameters
timeInitial = out2.plotE2.time;
stepResponseInitial = out2.plotE2.signals(2).values;

par2.m = 7;     % reduced mass m
out2 = sim("exercise3_SIM.slx");
timeReducedM = out2.plotE2.time;
stepResponseReducedM = out2.plotE2.signals(2).values;

par2.m = 13;    % increased mass m
out2 = sim("exercise3_SIM.slx");
timeIncreasedM = out2.plotE2.time;
stepResponseIncreasedM = out2.plotE2.signals(2).values;
par2.m = 10; % revert m to initial

par2.D = 2;    % reduced damper constant D
out2 = sim("exercise3_SIM.slx");
timeReducedD = out2.plotE2.time;
```

```
stepResponseReducedD = out2.plotE2.signals(2).values;

par2.D = 4;    % increased damper constant D
out2 = sim("exercise3_SIM.slx");
timeIncreasedD = out2.plotE2.time;
stepResponseIncreasedD = out2.plotE2.signals(2).values;
par2.D = 3; % revert D to initial

par2.C = 90;    % reduced spring constant C
out2 = sim("exercise3_SIM.slx");
timeReducedC = out2.plotE2.time;
stepResponseReducedC = out2.plotE2.signals(2).values;

par2.C = 110;    % increased spring constant C
out2 = sim("exercise3_SIM.slx");
timeIncreasedC = out2.plotE2.time;
stepResponseIncreasedC = out2.plotE2.signals(2).values;
par2.C = 100; % revert C to initial

% plot comparing different masses
figure(21);
plot(timeInitial, stepResponseInitial, ...
    timeReducedM, stepResponseReducedM, ...
    timeIncreasedM, stepResponseIncreasedM)
grid on, xlabel('Time [sec]'), ylabel('x [m]');
legend('10kg', '7kg', '13kg');
title('step response SIMULINK, physical property: m');

% plot comparing different damper constants
figure(22);
plot(timeInitial, stepResponseInitial, ...
    timeReducedD, stepResponseReducedD, ...
    timeIncreasedD, stepResponseIncreasedD)
grid on, xlabel('Time [sec]'), ylabel('x [m]');
legend('3Ns/m', '2Ns/m', '4Ns/m');
title('step response SIMULINK, physical property: D');

% plot comparing different spring constants
figure(23);
plot(timeInitial, stepResponseInitial, ...
    timeReducedC, stepResponseReducedC, ...
    timeIncreasedC, stepResponseIncreasedC)
grid on, xlabel('Time [sec]'), ylabel('x [m]');
legend('100N/m', '90N/m', '110N/m');
title('step response SIMULINK, physical property: C');
```

Exercise 3c): Building and optimizing a two-mass system in Simulink

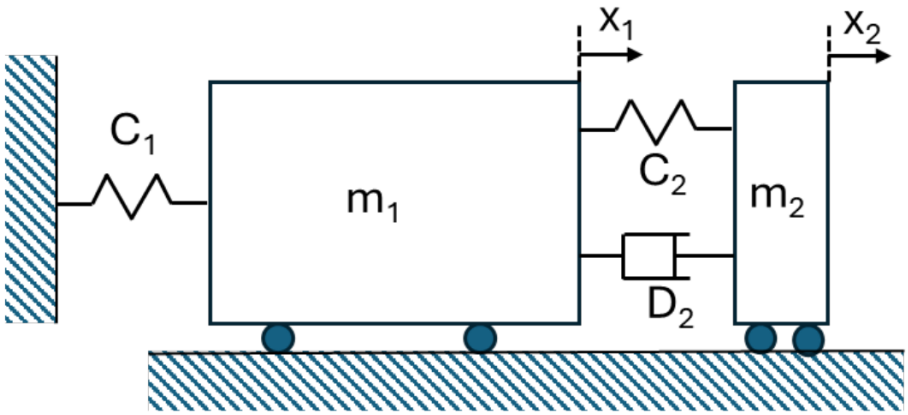

Figure 11: Two-mass spring/damper system with some physical characteristics.

The model to be simulated is illustrated in figure 11 above. This system possesses five physical properties, of which some are already specified. The specified values are $C_1 = 15\frac{N}{m}$, $m_1 = 100kg$ and $m_2 = 10kg$. The unspecified values are the spring constant C_2 and the damper constant D_2. These two values need to be optimized as such, that the oscillation of the entire system, more specifically the oscillation of m_1 gets minimized.
From this visual model, we can derive the differential movement equations for m_1 and m_2 as follows:

$$m_1\ddot{x}_1 = -C_1x_1 + C_2(x_2 - x_1) + D_2(\dot{x}_2 - \dot{x}_1) \tag{4}$$

$$m_2\ddot{x}_2 = C_2(x_1 - x_2) + D_2(\dot{x}_1 - \dot{x}_2) \tag{5}$$

Using equation 4 and 5 we can build our model in Simulink which looks like this:

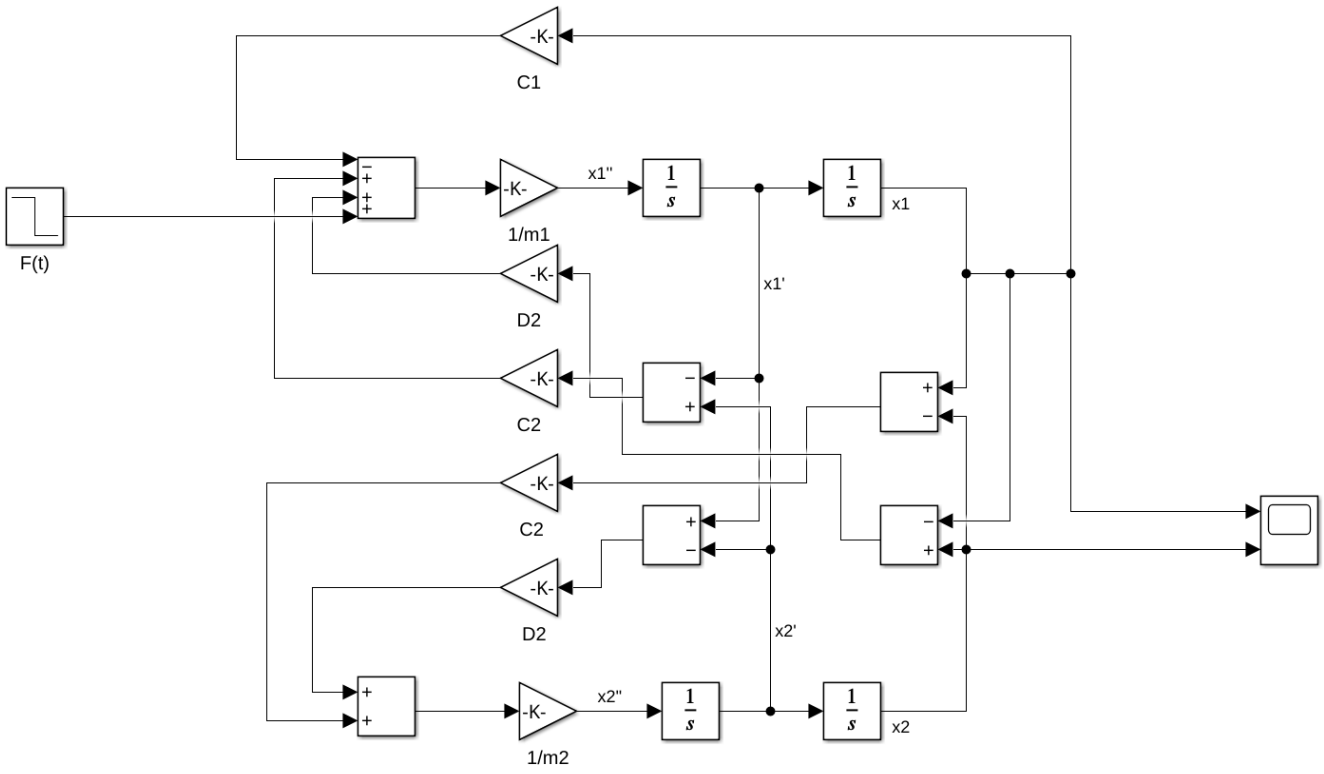

Figure 12: Differential equation of two-mass spring/damper system conveyed as Simulink model.

This model uses four integrator blocks to depict signals of $\ddot{x}_1$, $\dot{x}_1$, x_1, $\ddot{x}_2$, $\dot{x}_2$ and x_2 whose outputs get multiplied by their corresponding gain blocks and added together to form the sums of acting forces on m_1 and m_2. Furthermore, an initial force $F(t)$, simulated by a step block, gets added to the sum of forces acting upon m_1. This force is not part of our visual model, however we are using it to get our system into motion.
To elaborate, how optimal values for C_2 and D_2 are to be determined, we are going to take a closer look at the corresponding Matlab code.

Matlab code snippet:

```
clear; clf; clc;

% initial parameters exercise 3c
par3.m1 = 100;          % mass 1 [kg]
par3.m2 = 10;           % mass 2 [kg]
par3.C1 = 15;           % spring value [N/m]
par3.C2 = 0;            % spring value [N/m]
par3.D2 = 0;            % damper value [N*s/m]
par3.t.start = 0;       % start simulation time [s]
par3.t.stop = 300;     % stop simulation time [s]

% define damper and spring constant ranges
damperRange = 0.1:0.1:1.5; % [N*s/m]
springRange = 0.1:0.1:1.5; % [N/m]
areaUnderCurve = zeros(size(damperRange,2), size(springRange,2)); % area under squared
    curve matrix for different values
for damperConst = damperRange % iterate over damper value
    for springConst = springRange % iterate over spring value
        fprintf('damper: %g N*s/m, spring: %g N/m\n', damperConst, springConst); % log
            current values
        par3.D2 = damperConst; % update damper value
        par3.C2 = springConst;  % update spring value
        out3c = sim('ex3c_simulink.slx'); % simulate the system
        % extract values from simulation
        time = out3c.x1x2.time; % simulation timeframe
        x1 = out3c.x1x2.signals(1).values; % x1 oscillation curve
        x2 = out3c.x1x2.signals(2).values; % x2 oscillation curve
        area = trapz(time, x1.^2); % area under x1 squared curve
        areaUnderCurve(int32(damperConst*10), int32(springConst*10)) = area; % store area
            in matrix
    end
end

% plot for area under curve for different damper/spring constants
disp(areaUnderCurve);
figure(1);
surf(damperRange, springRange, areaUnderCurve);
xlabel('D2 damper value [N*s/m]')
ylabel('C2 spring value [N/m]')
zlabel('Area under squared x1 curve')
title('Optimization graphic for different damper/spring constants')
colorbar;

[minVal, linIdx] = min(areaUnderCurve(:)); % extract minimum value
[row, col] = ind2sub(size(areaUnderCurve), linIdx); % optimal row, col of matrix
% run simulation again with otimized values for plotting purposes
fprintf('optimal row: %g, optimal col: %g\n', row, col);
% actual optimal values derived from optial row, col
optimalDamper = row*0.1;
optimalSpring = col*0.1;
fprintf('optimal damper: %g N*s/m, optimal spring: %g N/m\n', optimalDamper, optimalSpring
    );
par3.D2 = optimalDamper; % update damper value
```

```
par3.C2 = optimalSpring;  % update spring value
out3c = sim('ex3c_simulink.slx'); % simulate the system
time = out3c.x1x2.time; % simulation timeframe
x1 = out3c.x1x2.signals(1).values; % x1 oscillation curve
x2 = out3c.x1x2.signals(2).values; % x2 oscillation curve

% plot for movement with optimized damper/spring constants
figure(2);
plot(time, x1, time, x2);
xlabel('simulation duration [s]');
ylabel('mass oscillation [m]');
title('Two mass system with optimized spring/damper constants');
legend('x1 oscillation','x2 oscillation')
grid on;
```

1. First of all, we initialize the known physical properties with their values. The unknown properties get assigned a zero value.
2. Next, we define range vectors for the damper and spring constants which we want to test. Both values should range from 0.1 to 1.5 with a step size of 0.1.
3. After that we define an initially filled with zeros 15 by 15 matrix `areaUnderCurve`, which shall store the results of each test value combination.
4. Now we test each value combination by iterating over both range vectors using nested for-loops. For each combination, the following steps are performed:
   (a) Assign the current values to the simulation parameters and run the simulation with them.
   (b) Extract the simulation results, specifically the run time-frame and result vectors for x_1 and x_2 positions.
   (c) Calculate and store the area under the squared x_1 curve. For this we use the inbuilt `trapz` function.
5. Now that that all value combinations have been tested, we can generate a surface plot of `areaUnderCurve`. Furthermore, we extract the optimal row, col where the value is minimal.
6. From this, we can determine optimal values for C_2 and D_2 and generate a plot sowing the optimal result.

From this we can determine **the optimal values, which are $C_2 = 1.3\frac{N}{m}$ and $D_2 = 1.1\frac{Ns}{m}$.** The above mentioned plots are illustrated below in figures 13 and 14:

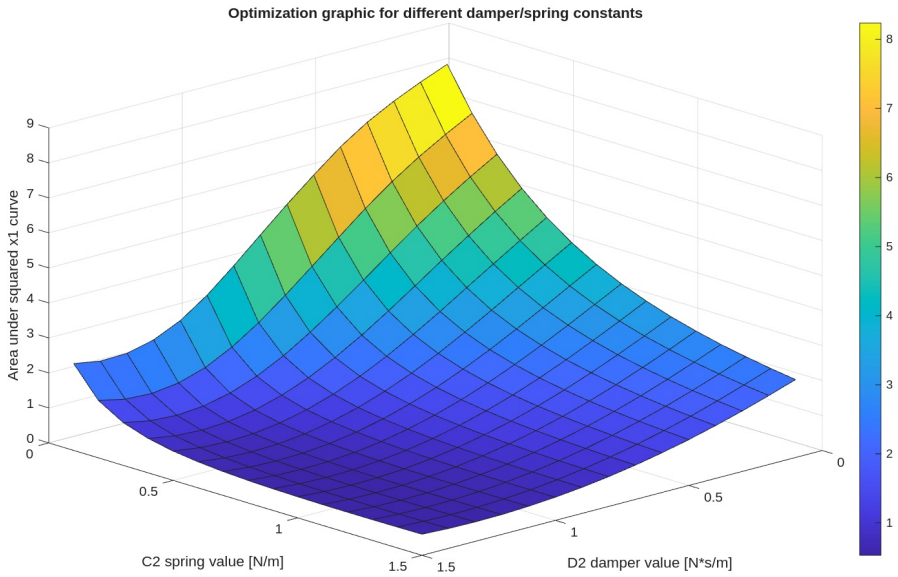

Figure 13: Area under squared x_1-curve for all tested combinations of C_2 and D_2.

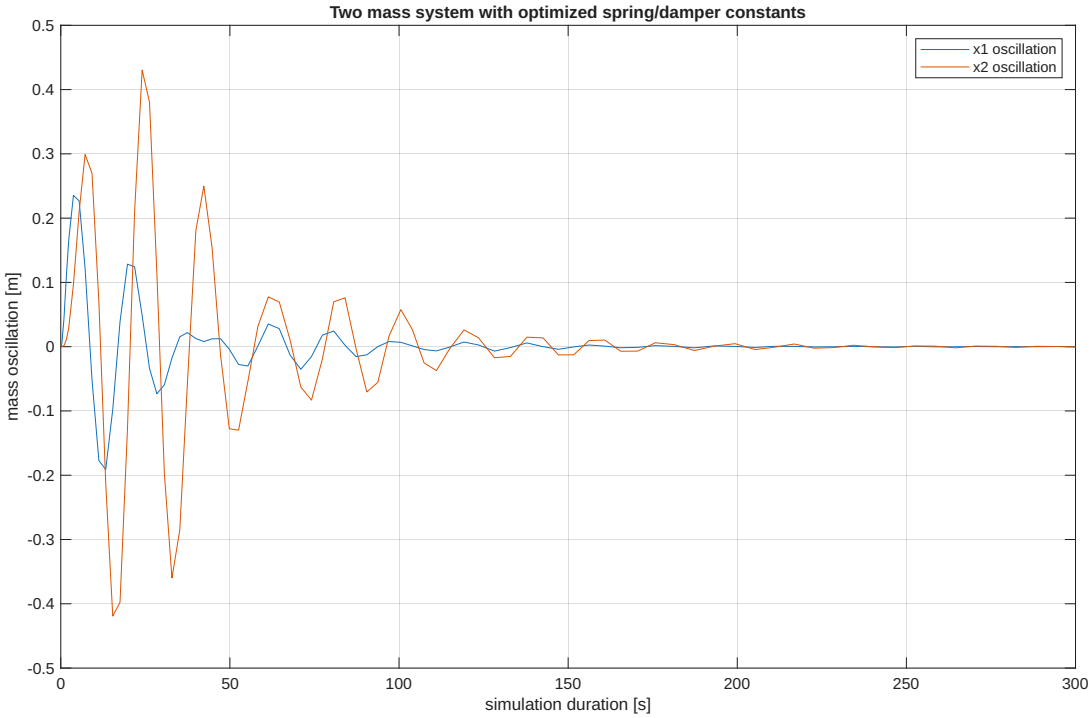

Figure 14: Oscillations of m_1 and m_2 with optimal parameters C_2 and D_2 over a time frame of 300 seconds.

Hereby we assume that the minimum area under the curve corresponds to the optimally damped model, because a smaller area corresponds to a faster converging x_1-curve. The absolute area value itself does not posses any relevant meaning for our purposes.