Linear Control Design 2 - Fall 2018 - Exam Part I - Solution Manual

Introduction

The Part I of the Exam in Linear Control Design 2 (E18) consists of numerical exercises testing the acquisition of competences in the areas of analysis and design of control systems using modern control theory based on state space representation of system dynamics.

The scoring of each problem is clearly stated.

It is the sole responsability of the student to guarantee that the solution delivered for evaluation can be run by the examiner without the need of contacting the student. All dependencies on files external to this Matlab Live Script must be checked and included in the final delivery. **If the examiner will not be able to execute the Matlab Live Script delivered as solution by the student, the Part I will be considered failed.**

```
% Fill in your information
Exam = 'LCD2 E18'
```

```
Exam =
'LCD2 E18'
```

```
Student_Name = 'Student Name'
```

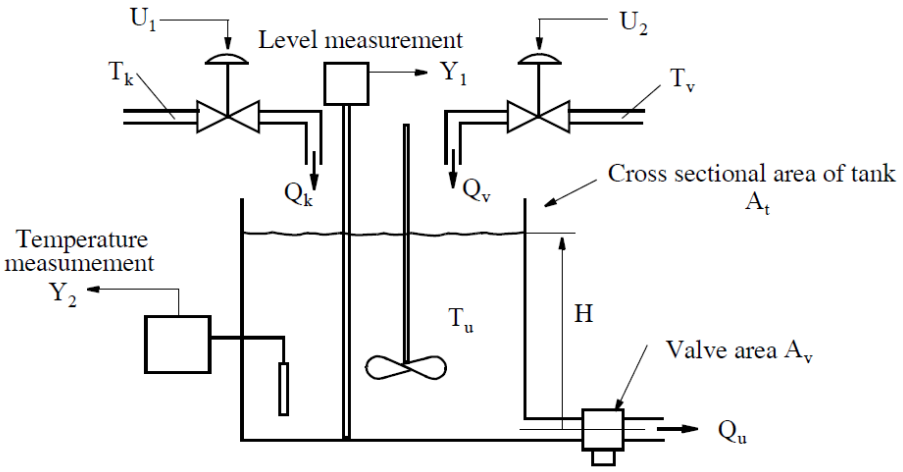
```
Student_Name =
'Student Name'
```

```
Student_Number = 'Student Number'
```

```
Student_Number =
'Student Number'
```

Output feedback control of a mixing tank

A mixing tank is a dynamical system where a liquid with temperature T_k and volume flow Q_k is mixed with another liquid with temperature T_v and volume flow Q_v. It is assumed that the mixed liquid is stirred so that its temperature T_u is approximately the same everywhere in the tank. The temperature of the released liquid at the outlet is T_u (the same as in the tank) and the volume flow is Q_u. The outlet valve has a constant but adjustable area setting A_v. The two volume flows Q_k and Q_v are determined by individual control valves which receive the input voltages U_1 and U_2. The level H of the mixed flow in the tank and the temperature T_u are measured with transducers whose outputs are the voltages Y_1 and Y_2. The following figure shows a schematic of the mixing tank.



System Dynamics

The nonlinear dynamics of the mixing tank is described by the following equations:

- Conservation of mass/volume: $A_t \frac{dH}{dt} = Q_k + Q_v - Q_u$, where A_t is the cross-sectional area of the tank
- Conservation of energy: $A_t \rho c \frac{d}{dt}(T_u H) = \rho c(Q_k T_k + Q_v T_v - Q_u T_u)$, where ρ is the mass density and c is the specific heat capacity of the liquid.
- Flow equation for both control valves: $Q = C_v U$, where C_v is the valve flow coefficient
- Temperature measurement: $Y_1 = k_t T_u$, where k_t is the transducing constant
- Level measurement: $Y_2 = k_h H$, where k_h is the transducing constant

Let the state vector be $\mathbf{x} = [H, T_u]^T$ and the output vector be $\mathbf{y} = [Y_1, Y_2]^T$. Then the nonlinear state space representation of the system is

$$\dot{x}_1 = \frac{1}{A_t}\left(C_v(U_1 + U_2) - k_v A_v \sqrt{H}\right)$$

$$\dot{x}_2 = \frac{C_v}{A_t H}((T_k - T_u)U_1 + (T_v - T_u)U_2)$$

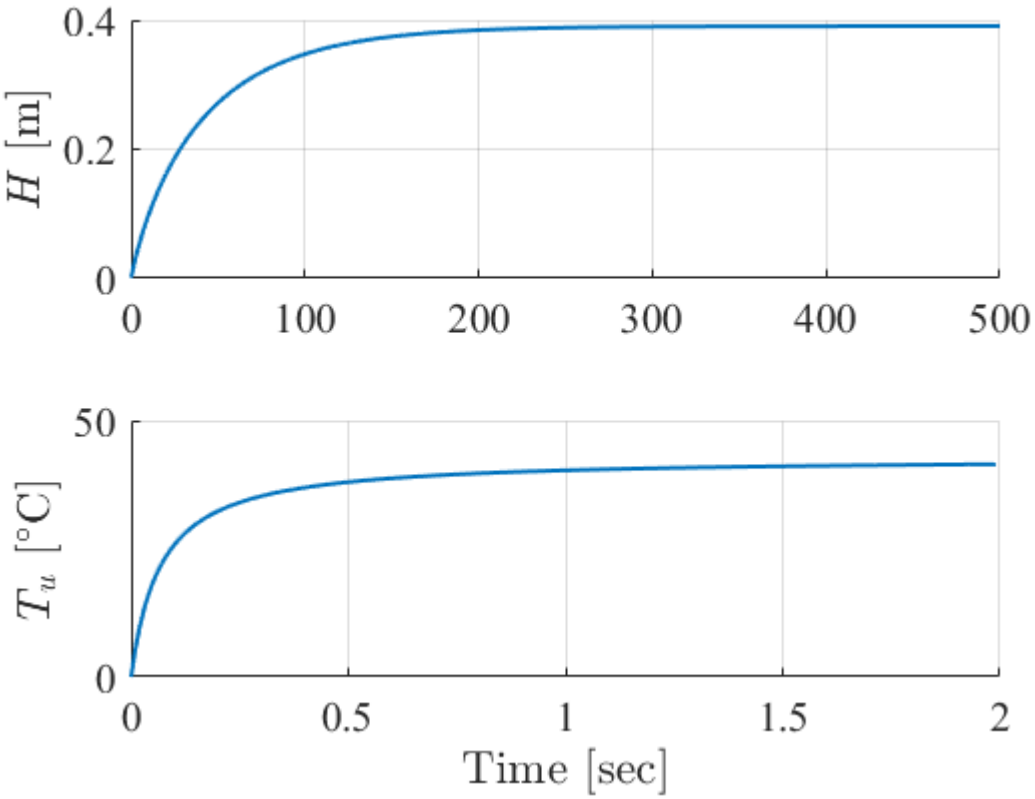
$$y_1 = k_t T_u$$

$$y_2 = k_h H$$

The nonlinear model is implemented in the Simulink file MixingTank_NonlinearModel_SimulinkYYYYC, where YYYYC refers to the Simulink version (2017b, 2017a, 2016b, etc.). The numerical values for the model parameters are provided right below.

```
% Simulator parameters (RG)
SIM_TIME = 500; % simulation time (can be changed as needed)
```

```
STEP_SIZE = 0.01; % integration step size of Simulink (can be changed as
needed)
% Change with the name of the model you are using
SIMULINK_FILENAME = 'MixingTank_NonlinearModel_Simulink2017b';

% Model paramters (RG)
% System parameters
At = 0.2; % tank cross-sectional area [m^2]
kv = 0.004; % valve flow coefficient [m^(5/2)/sec]
Av = 1.2; % valve area (dimensionless parameter)
Cv = 0.0005; % valve flow coefficient [m^3/(sec*V)]
kt = 0.1; % transducer constant [0.1 V/degC]
kh = 5; % transducer constant [V/m]
Tk = 25; % temperature [deg]
Tv = 60; % temperature [deg]

% Test the provided nonlinear model (RG)
time = (0:STEP_SIZE:SIM_TIME)';
x0 = [0.001,0]; % initial condition x0 = [H,Tu]'. The initial condition for
H is different
% from zero because in the second differential equation we divide by H and a
zero initial
% condition will give rise to an infinite rate of change.
U10 = 3; % input voltage valve 1 [V]
U20 = 3; % input voltage valve 2 [V]
U = [U10 U20].*ones(length(time),1); % controllable inputs
D = [Tk Tv Av].*ones(length(time),1); % disturbances
sim(SIMULINK_FILENAME,SIM_TIME,[],[time U D])

% Plot temporal behaviour of the state of the nonlinear model (RG)
H = logsout.getElement(1).Values.Data;
Tu = logsout.getElement(2).Values.Data;

figure, h1 = subplot(2,1,1); set(h1,'FontName','times','FontSize',16)
hold on, grid on
plot(time,H,'LineWidth',1.5)
ylabel('$H$ [m]','FontName','times','FontSize',16,'Interpreter','latex')
h2 = subplot(2,1,2); set(h2,'FontName','times','FontSize',16)
hold on, grid on
plot(time(1:200),Tu(1:200),'LineWidth',1.5)
ylabel('$T_u$
[$^\circ$C]','FontName','times','FontSize',16,'Interpreter','latex')
xlabel('Time [sec]','FontName','times','FontSize',16,'Interpreter','latex')
```



Operating point and linearization

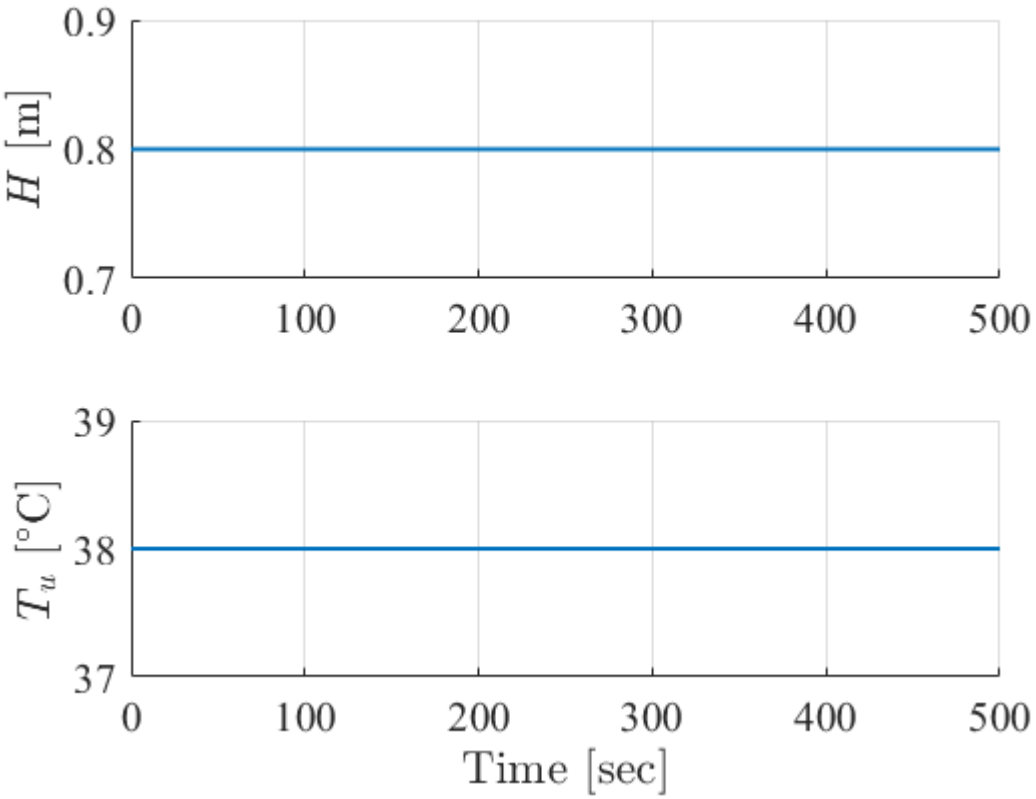
Problem 1 [2 points] Numerically determine the operating point $\mathbf{u}_0$ of the valves U_1 and U_2 associated with the stationary state $\mathbf{x}_0 = [0.8, 38]^T$ and while keeping the disturbances constant at their nominal values, i.e. $\mathbf{d}_0 = [T_k, Tv, Av]^T = [25, 60, 1.2]^T$. Linearize the system around the operatig point $(\mathbf{x}_0, \mathbf{u}_0, \mathbf{d}_0)$.

```
% The steady state value of the input u0 is found by using the function trim
x0 = [0.8,38]';
d0 = [25,60,1.2]';
y0 = [kt*x0(2) kh*x0(1)]';
[xss,uss,yss,dx]=trim(SIMULINK_FILENAME,x0,[3,3,d0']',y0,[1;2],[3;4;5],[1;2])
```

```
xss = 2×1
    0.8000
   38.0000
uss = 5×1
    5.3972
    3.1893
   25.0000
   60.0000
    1.2000
yss = 2×1
    3.8000
    4.0000
dx = 2×1
```

```
10^-12 ×
    0.0138
    0.2976
```

```
x0 = xss;
u0 = uss(1:2);
d0 = uss(3:5);

% The stationary condition is verified through a simulation of the nonlinear
system
U = [time ones(length(time),length(uss)).*uss'];
sim(SIMULINK_FILENAME,SIM_TIME,[],U)

% Plot temporal behaviour of the state of the nonlinear model
H = logsout.getElement(1).Values.Data;
Tu = logsout.getElement(2).Values.Data;

figure, h1 = subplot(2,1,1); set(h1,'FontName','times','FontSize',16)
hold on, grid on
plot(time,H,'LineWidth',1.5)
ylabel('$H$ [m]','FontName','times','FontSize',16,'Interpreter','latex')
ylim([0.7 0.9]);
h2 = subplot(2,1,2); set(h2,'FontName','times','FontSize',16)
hold on, grid on
plot(time,Tu,'LineWidth',1.5)
ylabel('$T_u$
[$^\circ$C]','FontName','times','FontSize',16,'Interpreter','latex')
xlabel('Time [sec]','FontName','times','FontSize',16,'Interpreter','latex')
```




```
% Linearization of the system is performed using linmod
[Aol,Bol,Col,Dol] = linmod(SIMULINK_FILENAME,x0,[u0',d0']');
Bv = Bol(:,3:5); % disturbance input matrix
Bol = Bol(:,1:2);
Dol = Dol(:,1:2);
sys_ol = ss(Aol,Bol,Col,Dol)
```

```
sys_ol =

  A =
              x1          x2
   x1    -0.01342           0
   x2   -3.72e-13    -0.02683

  B =
             u1         u2
   x1     0.0025     0.0025
   x2   -0.04062    0.06875

  C =
        x1   x2
   y1    0  0.1
   y2    5    0

  D =
       u1  u2
   y1   0   0
   y2   0   0

Continuous-time state-space model.
```

Stability analysis

Problem 2 [2 points] Assess the internal and external stability of the linear system.

```
% The internal stability is assessed by checking the position of the
% eigenvalues in the complex plane. The external stability is evaluated
% looking at tht position of the poles.

lambda_ol = eig(Aol); % open loop eignevalues
disp('The open loop eignevalues are')
```

```
The open loop eignevalues are
```

```
disp(lambda_ol)
```

```
   -0.0268
   -0.0134
```

```
disp(['Since the real part of the both eigenvalues is negative then the
system' ...
    ' is asymptotically stable'])
```

```
Since the real part of the both eigenvalues is negative then the system is asymptotically stable
```

```

[numG,denG] = ss2tf(sys_ol.a,sys_ol.b,sys_ol.c,sys_ol.d,1);
p_ol = roots(denG); % open loop poles
disp('The open loop poles are')
```

```
The open loop poles are
```

```
disp(p_ol)
```

```
   -0.0268
   -0.0134
```

```
disp(['Since the real part of the both poles is negative then the system' ...
    ' is BIBO stable'])
```

```
Since the real part of the both poles is negative then the system is BIBO stable
```

Control system design for regulation of level and temperature

The main control objective is to regulate the level H and the temperature T_u of the mixed liquid at the desired set-points despite the action of the disturbances T_k, T_v and A_v.

Closed-loop system requirements

1) The steady state error for both outputs must be zero in relation to step changes of the set-points and/or the disturbances.

2) For step changes of $\pm 20\%$ of the nominal value of A_v the controller regulates the fluid level H with

a) a maximum overshoot/downshoot in the range 3% - 5%

b) a settling time between 50 and 100 seconds

Problem 3 [6 points] Under the assumption that the state is fully accessible design a discrete time controller that meets the aforementioned specifications on the nonlinear system using the eigenstructure assignment method. Validate your design through simulation on the linear and nonlinear model.

```
% To meet the first closed-loop requirement the control architecture must
include
% an integral action for both state variables. Since the controller must be
designed
% in discrete time a proper sampling time need to be identified. The desired
settling
% time is then used to determine a suitable Ts.

min_sett_time = 50; % [sec]
Ts = min_sett_time/10
```

```
Ts = 5
```

```
tau_ol = 1./abs(lambda_ol); % open loop time constants
disp('The open loop time constants are')
```

```
The open loop time constants are
```

```
disp(tau_ol)
```

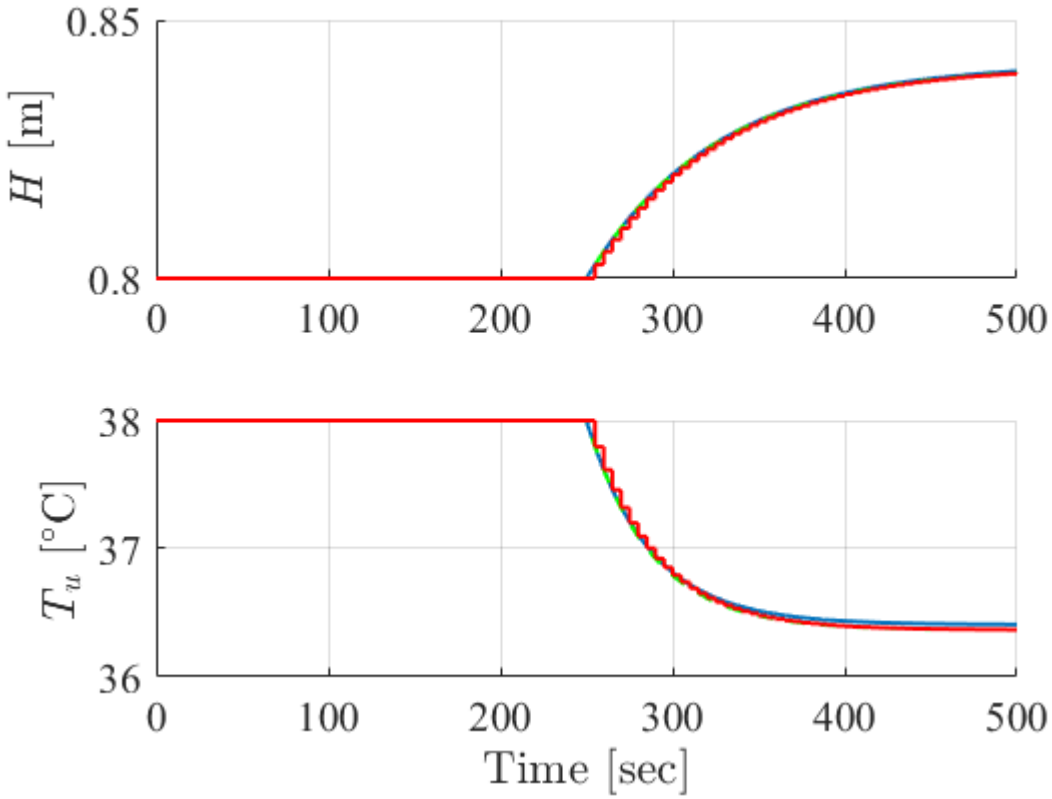
```
   37.2678
   74.5356
```

```
% The smallest open loop time constant is less than 10 times larger than the
computed
% sampling time. If we take this into consideration then we could reduce the
sampling
% time to 4 seconds. We verify the suitability of the chosen sampling time
by comparing
% the step response of the discretized system with that one of the
continuous time system

sys_ol_dt = c2d(sys_ol,Ts)
```

```
sys_ol_dt =

  A =
              x1          x2
   x1      0.9351           0
   x2  -1.682e-12      0.8744

  B =
            u1       u2
   x1  0.01209  0.01209
   x2  -0.1901   0.3217

  C =
```

```
       x1   x2
  y1    0  0.1
  y2    5    0

D =
      u1  u2
  y1   0   0
  y2   0   0

Sample time: 5 seconds
Discrete-time state-space model.
```

```
[~,Gv] = c2d(Aol,Bv,Ts); % discretization of disturbance input matrix

U = [[u0(1) u0(2)].*ones((length(time)-1)/2,1);[1.1*u0(1)
0.9*u0(2)].*ones((length(time)-1)/2+1,1)]; % controllable inputs
D = [d0(1) d0(2) d0(3)].*ones(length(time),1); % disturbances
sim(SIMULINK_FILENAME,SIM_TIME,[],[time U D])

% Plot temporal behaviour of the state of the nonlinear model
H = logsout.getElement(1).Values.Data;
Tu = logsout.getElement(2).Values.Data;

% Simulation of continuous time linearized system
U_ct = U-[u0(1) u0(2)];
[y_ct,t_ct,x_ct] = lsim(sys_ol,U_ct,time);

% Simulation of discrete time linearized system
U_dt = U(1:500:end,:)-[u0(1) u0(2)]; % sampled input
[y_dt,t_dt,x_dt] = lsim(sys_ol_dt,U_dt);

figure, h1 = subplot(2,1,1); set(h1,'FontName','times','FontSize',16)
hold on, grid on
plot(time,H,time,x_ct(:,1)+x0(1),'--g','LineWidth',1.5)
stairs(t_dt,x_dt(:,1)+x0(1),'r','LineWidth',1.5)
ylabel('$H$ [m]','FontName','times','FontSize',16,'Interpreter','latex')
h2 = subplot(2,1,2); set(h2,'FontName','times','FontSize',16)
hold on, grid on
plot(time,Tu,time,x_ct(:,2)+x0(2),'--g','LineWidth',1.5)
stairs(t_dt,x_dt(:,2)+x0(2),'r','LineWidth',1.5)
ylabel('$T_u$
[$^\circ$C]','FontName','times','FontSize',16,'Interpreter','latex')
xlabel('Time [sec]','FontName','times','FontSize',16,'Interpreter','latex')
```




```
% The simulation shows that the discretized linear model accurately follows
% the continous time linear model. Both models have the same mismatch with
% respect to the nonlinear model. Therefore we can conclude that the chosen
sampling
% time suffices to properly evaluate the dynamics of open loop system.

% Full state feedback with integral action
F = sys_ol_dt.a;
G = sys_ol_dt.b;
% Compute the reachability matrix and check if the system is reachable
Mr = ctrb(F,G);
if rank(Mr) == size(F,1)
    disp('The open loop system is reachable')
end
```

```
The open loop system is reachable
```

```
Fi = [F zeros(size(F,1),2);-Ts*Col eye(size(Col,1),2)];
Gi = [G;zeros(size(Col,1),2)];
Gr = [zeros(2);Ts*[kt 0;0 kh]]; % reference input matrix (assume that the
reference is
% in degrees Celsius and meters)
Gvi = [Gv;zeros(size(Col,1),3)];
Ci = [Col,zeros(size(Col,1),2)];
```

```
% From closed-loop requirement 2 we derive the region of the complex plane
where the
% eigenvalues should be placed.

sett_time_des = 90; % desired settling time [sec]
tau_cl = sett_time_des/4; % closed-loop time constant

dr_des = 0.7; % desired damping ratio

% To fulfill the requirement 2 we assign a complex pair alpha +/- j*beta.
Alpha determines
% the decay rate of the envelope of the oscillatory motion when the system
is initialized
% with a non-zero initial condition. Beta is related to the damping

alpha1 = -1/tau_cl;
beta1 = abs(alpha1)*sqrt((1-dr_des^2)/dr_des^2);
tau2 = 20;
dr_des2 = 0.55;
alpha2 = -1/tau2;
beta2 = abs(alpha2)*sqrt((1-dr_des2^2)/dr_des2^2);
lambda_cl_des =
[alpha1+1i*beta1,alpha1-1i*beta1,alpha2+1i*beta2,alpha2-1i*beta2];
lambda_cl_des_dt = exp(lambda_cl_des.*Ts)
```

```
lambda_cl_des_dt = 1×4 complex
   0.7802 + 0.1800i   0.7802 - 0.1800i   0.7234 + 0.2886i   0.7234 - 0.2886i
```

```
K = place(Fi,Gi,lambda_cl_des_dt)
```

```
K = 2×4
   17.4744   -0.7689    0.5316   -0.1121
   14.4355    0.8406   -0.6313   -0.1806
```

```
Kfs = K(:,1:2);
Ki = -K(:,3:4);

sys_cl_dt = ss(Fi-Gi*K,[Gr Gvi],Ci,[],Ts);
lambda_cl_dt = eig(sys_cl_dt)
```

```
lambda_cl_dt = 4×1 complex
   0.7802 + 0.1800i
   0.7802 - 0.1800i
   0.7234 + 0.2886i
   0.7234 - 0.2886i
```

```
% Design validation on the linear model
SIMULINK_FILENAME = 'MixingTank_IntegralControl_Simulink2017b';
R = [x0(2) x0(1)].*ones(length(time),1); % references
D = [[d0(1) d0(2) d0(3)].*ones((length(time)-1)/4,1);...
    [d0(1) d0(2) 1.2*d0(3)].*ones((length(time)-1)/4,1);...
```

```
    [d0(1) d0(2) d0(3)].*ones((length(time)-1)/4,1);...
    [d0(1) d0(2) 0.8*d0(3)].*ones((length(time)-1)/4+1,1)]; % disturbances
xi0 = zeros(2,1); % initial condition for the controller integrators
sim(SIMULINK_FILENAME,SIM_TIME,[],[time R D])

% Plot temporal behaviour of the state of the nonlinear model
u_fs = -logsout.getElement(1).Values.Data;
u_i = logsout.getElement(2).Values.Data;
timeDT = logsout.getElement(1).Values.Time;
H = logsout.getElement(3).Values.Data;
Tu = logsout.getElement(4).Values.Data;
u = logsout.getElement(5).Values.Data;

% Simulation of discrete time closed-loop linearized system
U = [R D];
U_dt = U(1:500:end,:)-[x0(2) x0(1) d0(1) d0(2) d0(3)]; % sampled input
[y_dt,t_dt,x_dt] = lsim(sys_cl_dt,U_dt);

bound_3pct = x0(1)*[0.97 1.03];
bound_5pct = x0(1)*[0.95 1.05];
settling_p1pct = x0(1)*1.01;
settling_m1pct = x0(1)*0.99;
downshoot = 100*(x0(1) - min(H))/x0(1);
overshoot = 100*(max(H) - x0(1))/x0(1);
disp('The overshootshoot is')
```

```
The overshootshoot is
```

```
disp(strcat(num2str(overshoot),'%'))
```

```
4.666%
```

```
disp('The downhootshoot is')
```

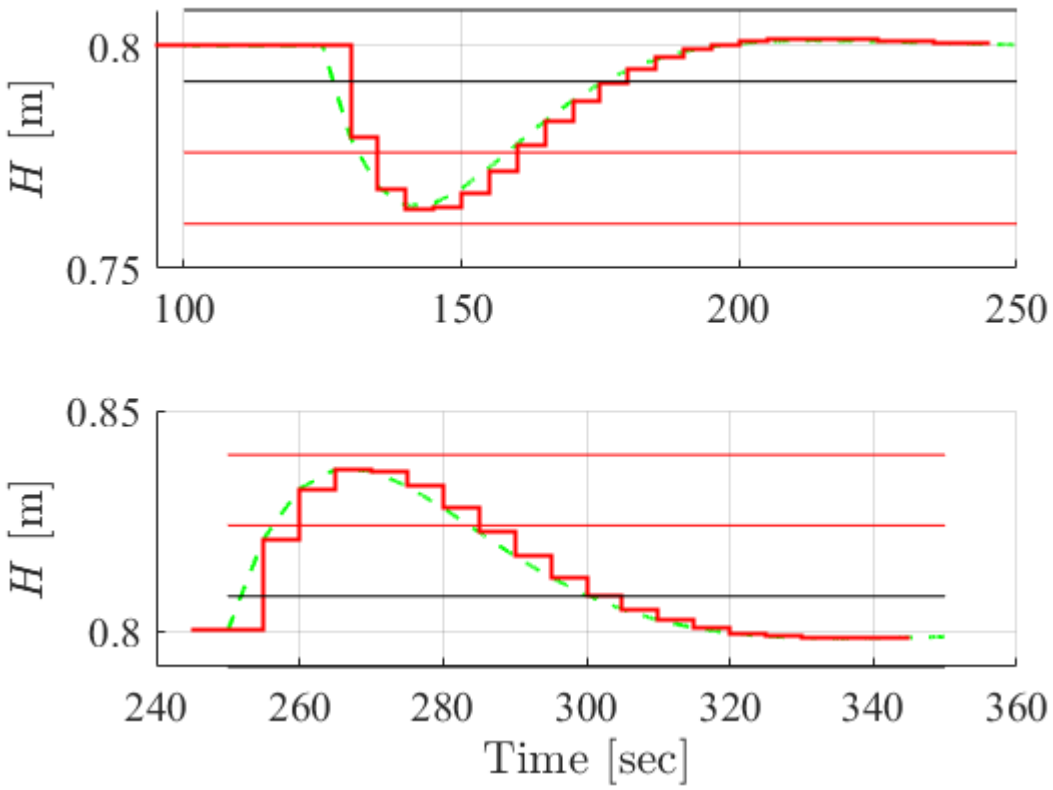
```
The downhootshoot is
```

```
disp(strcat(num2str(downshoot),'%'))
```

```
4.4906%
```

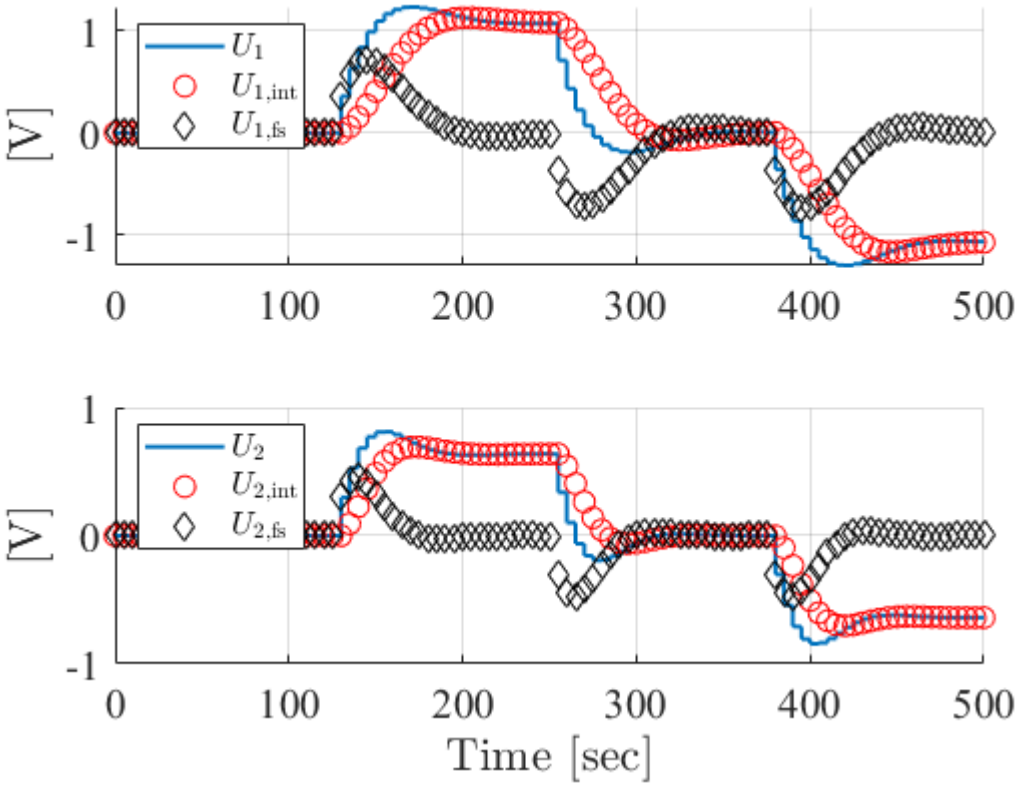
```
figure, h1 = subplot(2,1,1); set(h1,'FontName','times','FontSize',16)
hold on, grid on
plot(time(100/STEP_SIZE:250/STEP_SIZE),H(100/STEP_SIZE:250/STEP_SIZE),'--
g','LineWidth',1.5)
stairs(t_dt(100/Ts:250/Ts),x_dt(100/Ts:250/Ts,1)+x0(1),'r','LineWidth',1.5)
line([time(100/STEP_SIZE) time(250/STEP_SIZE)],[bound_3pct(1)
bound_3pct(1)],'Color','r')
line([time(100/STEP_SIZE) time(250/STEP_SIZE)],[bound_5pct(1)
bound_5pct(1)],'Color','r')
line([time(100/STEP_SIZE) time(250/STEP_SIZE)],[settling_p1pct
settling_p1pct],'Color','k')
```

```
line([time(100/STEP_SIZE) time(250/STEP_SIZE)],[settling_m1pct
settling_m1pct],'Color','k')
ylabel('$H$ [m]','FontName','times','FontSize',16,'Interpreter','latex')
h2 = subplot(2,1,2); set(h2,'FontName','times','FontSize',16)
hold on, grid on
plot(time(250/STEP_SIZE:350/STEP_SIZE),H(250/STEP_SIZE:350/STEP_SIZE),'--
g','LineWidth',1.5)
stairs(t_dt(250/Ts:350/Ts),x_dt(250/Ts:350/Ts,1)+x0(1),'r','LineWidth',1.5)
line([time(250/STEP_SIZE) time(350/STEP_SIZE)],[bound_3pct(2)
bound_3pct(2)],'Color','r')
line([time(250/STEP_SIZE) time(350/STEP_SIZE)],[bound_5pct(2)
bound_5pct(2)],'Color','r')
line([time(250/STEP_SIZE) time(350/STEP_SIZE)],[settling_p1pct
settling_p1pct],'Color','k')
line([time(250/STEP_SIZE) time(350/STEP_SIZE)],[settling_m1pct
settling_m1pct],'Color','k')
ylabel('$H$ [m]','FontName','times','FontSize',16,'Interpreter','latex')
% plot(time,Tu,'--g','LineWidth',1.5)
% stairs(t_dt,x_dt(:,2)+x0(2),'r','LineWidth',1.5)
% ylabel('$T_u$
[$^\circ$C]','FontName','times','FontSize',16,'Interpreter','latex')
xlabel('Time [sec]','FontName','times','FontSize',16,'Interpreter','latex')
```




```
figure, h1 = subplot(2,1,1); set(h1,'FontName','times','FontSize',16)
hold on, grid on
stairs(timeDT,u(:,1)-u0(1),'LineWidth',1.5)
```

```
plot(timeDT,u_i(:,1),'or',timeDT,u_fs(:,1),'dk','MarkerSize',8)
ylabel('[V]','FontName','times','FontSize',16,'Interpreter','latex')
leg1 = legend('$U_1$','$U_{1,\mathrm{int}}$','$U_{1,\mathrm{fs}}
$','Location','NorthWest');
set(leg1,'FontName','times','FontSize',12,'Interpreter','latex')
h2 = subplot(2,1,2); set(h2,'FontName','times','FontSize',16)
hold on, grid on
stairs(timeDT,u(:,2)-u0(2),'LineWidth',1.5)
plot(timeDT,u_i(:,2),'or',timeDT,u_fs(:,2),'dk','MarkerSize',8)
ylabel('[V]','FontName','times','FontSize',16,'Interpreter','latex')
xlabel('Time [sec]','FontName','times','FontSize',16,'Interpreter','latex')
leg2 = legend('$U_2$','$U_{2,\mathrm{int}}$','$U_{2,\mathrm{fs}}
$','Location','NorthWest');
set(leg2,'FontName','times','FontSize',12,'Interpreter','latex')
```



Observer design for estimation of the valve area A_v

To determine the unknown valve area A_v determining the output flow Q_u and thereby affecting the liquid level H, a state estimator in the form of a Kalman filter can be employed.

The two measured outputs are affected by uncorrelated white noise sources $[v_l, v_h]^T$ with zero mean and maximum amplitudes of ± 0.05 volts, i.e.

$$y_1 = k_t T_u + v_t$$
$$y_2 = k_h H + v_h$$

Problem 4 [4 points] Under the assumption that the valve area A_v is constant, design a discrete time Kalman filter able to reconstruct the state vector and the unknown valve area.

```
% Model for Kalman filter design including estimation of the disturbance
F_kf = [F, Gv(:,3);zeros(1,2) 1];
G_kf = [G;zeros(1,2)];
C_kf = [Col zeros(2,1)];
% It is assumed that process noise acts on all state variables of the
system.
% The continuous time noise input matrix Bv is now defined under the
assumption
% that the noise enters the system in the feedack loops of the open loop
system
% Bv_kf = diag([Aol(1,1) Aol(2,2) 1]);
Bv_kf = diag([1 1 1]);
sigma_H = 1;
sigma_Tu = 0.5;
sigma_Av = 5;
V1 = diag([sigma_H^2 sigma_Tu^2 sigma_Av^2]); % Process noise intensity
meas_noise_max_Ampl = 0.05;
sigma_y1 = meas_noise_max_Ampl/3;
sigma_y2 = meas_noise_max_Ampl/3;
V2 = diag([sigma_y1^2 sigma_y2^2]); % Measurement noise intensity
% Process noise discretization
V1d = Bv_kf*V1*Bv_kf'*Ts;
Gv_kf = eye(3);
V2d = V2/Ts;

% Check observability
Mo = obsv(F_kf,C_kf);
if rank(Mo) == size(F_kf,1)
    disp('The open loop system is observable')
end
```

```
The open loop system is observable
```

```
% Design of the discrete time closed form Kalman filter
[L_kf,P,Qe_th,lambda_kf_dt] = dlqe(F_kf,Gv_kf,C_kf,V1d,V2d)
```

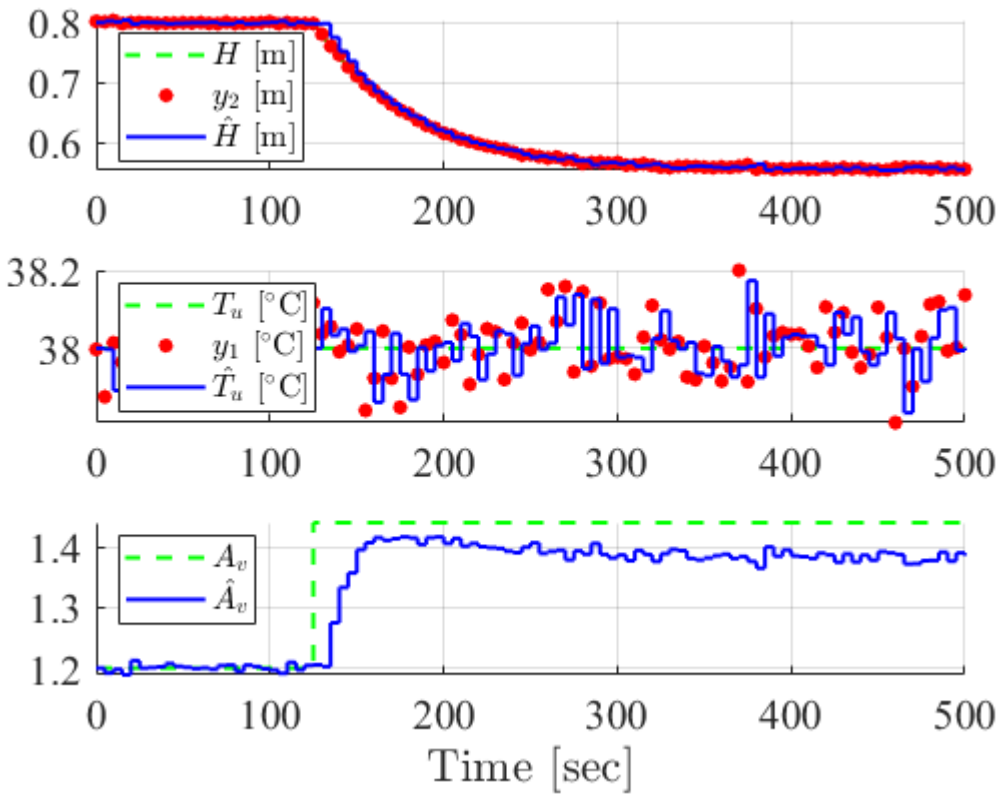
```
L_kf = 3×2
   -0.0000    0.2000
    9.9559   -0.0000
    0.0000   -0.8068
P = 3×3
    7.6805   -0.0000  -30.9848
   -0.0000    1.2542    0.0000
  -30.9848    0.0000  483.1696
```

```
Qe_th = 3×3
    0.0000   -0.0000   -0.0000
   -0.0000    0.0055    0.0000
   -0.0000    0.0000  358.1696
lambda_kf_dt = 3×1
    0.0000
    0.6510
    0.0039
```

Problem 5 [4 points] Assess the estimation performance of the Kalman filter when tested on the open loop nonlinear system for step changes of $\pm 20\%$ of the nominal value of A_v.

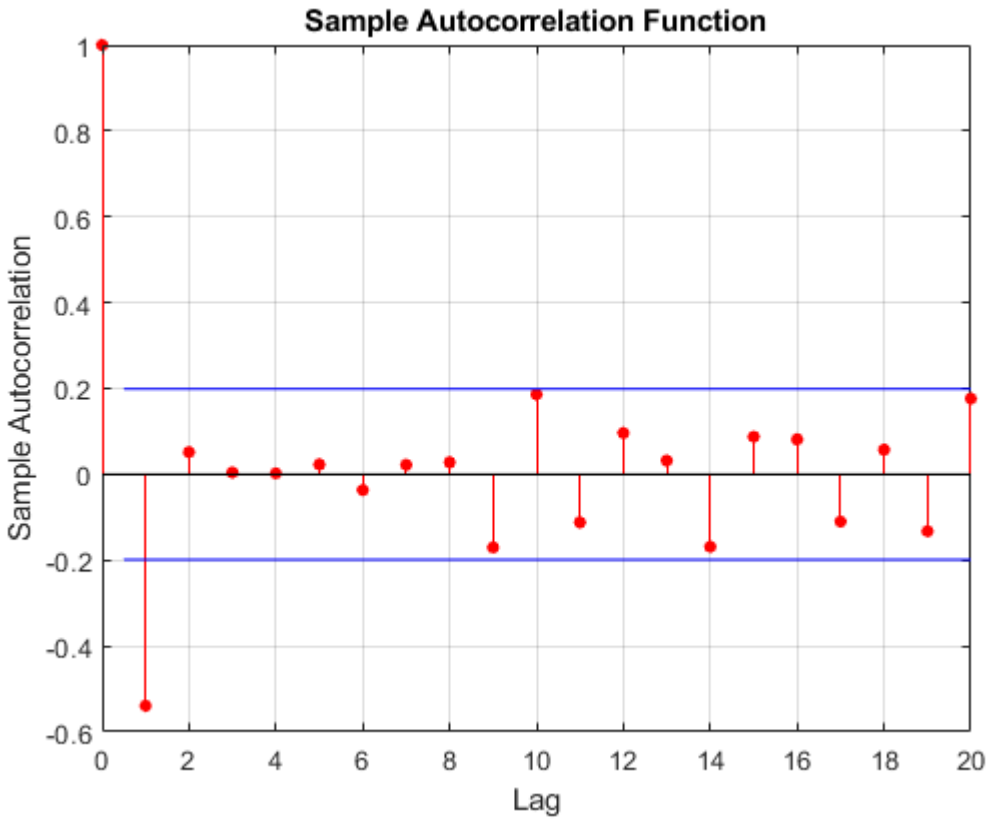
```
% Your solution goes here

% Design validation on the linear model
SIMULINK_FILENAME = 'MixingTank_KalmanFilter_Simulink2017b';
D = [[d0(1) d0(2) d0(3)].*ones((length(time)-1)/4,1);...
    [d0(1) d0(2) 1.2*d0(3)].*ones(3*(length(time)-1)/4+1,1)]; % disturbances
x0hat = zeros(3,1); % initial condition for the Kalman filter
sim(SIMULINK_FILENAME,SIM_TIME,[],[time D])

% Plot temporal behaviour of the state of the nonlinear model
xhat = logsout.getElement(1).Values.Data;
yhat = logsout.getElement(2).Values.Data;
timeDT = logsout.getElement(1).Values.Time;
H = logsout.getElement(3).Values.Data;
Tu = logsout.getElement(4).Values.Data;
y1 = 1/kt*logsout.getElement(7).Values.Data; % measurement 1 expressed in
deg Celsius
y2 = 1/kh*logsout.getElement(6).Values.Data; % measurement 1 expressed in
meters

figure, h1 = subplot(3,1,1); set(h1,'FontName','times','FontSize',16)
hold on, grid on
plot(time,H,'--
g',timeDT,y2,'or','MarkerSize',4,'MarkerFaceColor','r','LineWidth',1.5)
stairs(timeDT,xhat(:,1),'b','LineWidth',1.5)
leg1 = legend('$H$ [m]','$y_2$ [m]','$\hat{H}$ [m]','Location','NorthWest');
set(leg1,'FontName','times','FontSize',12,'Interpreter','latex')

h2 = subplot(3,1,2); set(h2,'FontName','times','FontSize',16)
hold on, grid on
plot(time,Tu,'--
g',timeDT,y1,'or','MarkerSize',4,'MarkerFaceColor','r','LineWidth',1.5)
stairs(timeDT,xhat(:,2),'b','LineWidth',1.5)
leg2 = legend('$T_u$ [$^\circ$C]','$y_1$ [$^\circ$C]','$\hat{T_u}$
[$^\circ$C]','Location','NorthWest');
set(leg2,'FontName','times','FontSize',12,'Interpreter','latex')
```

```
h3 = subplot(3,1,3); set(h3,'FontName','times','FontSize',16)
hold on, grid on
plot(time,D(:,3),'--g','LineWidth',1.5)
stairs(timeDT,xhat(:,3),'b','LineWidth',1.5)
leg3 = legend('$A_v$','$\hat{A_v}$','Location','NorthWest');
set(leg3,'FontName','times','FontSize',12,'Interpreter','latex')
xlabel('Time [sec]','FontName','times','FontSize',16,'Interpreter','latex')
```

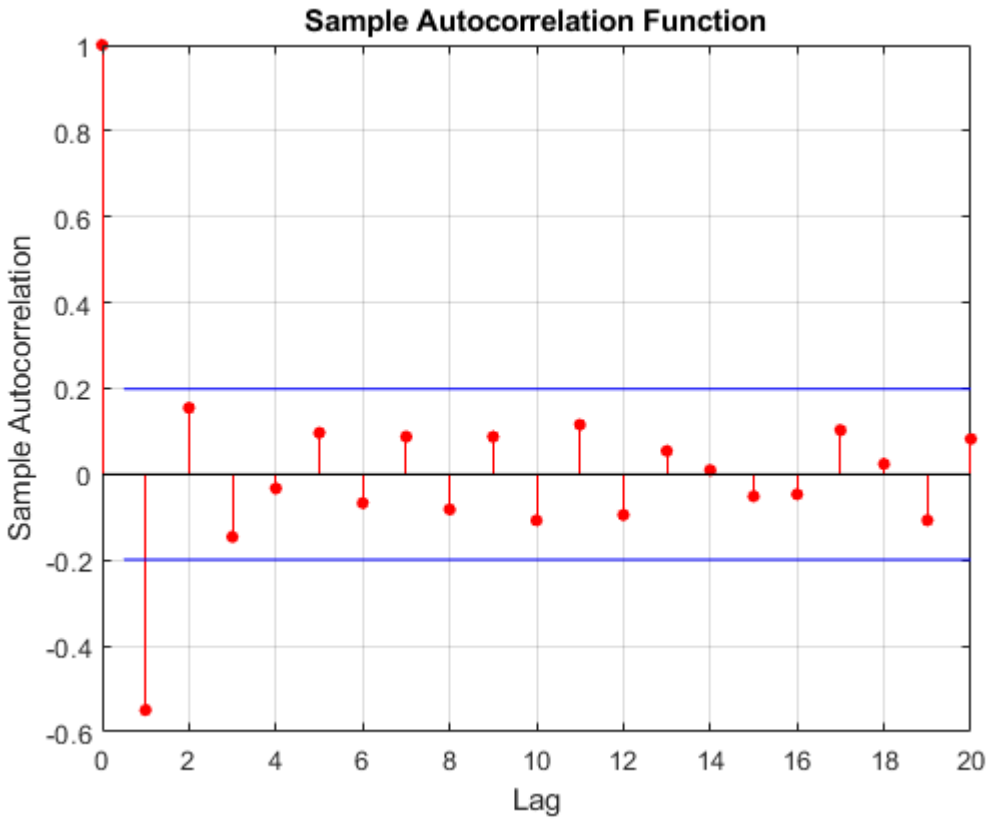



```
% Kalman filter performance analysis in stationary conditions
SIMULINK_FILENAME = 'MixingTank_KalmanFilter_Simulink2017b';
D = [[d0(1) d0(2) d0(3)].*ones((length(time)),1)]; % disturbances
x0hat = zeros(3,1); % initial condition for the Kalman filter
sim(SIMULINK_FILENAME,SIM_TIME,[],[time D])

% Plot temporal behaviour of the state of the nonlinear model
xhat = logsout.getElement(1).Values.Data;
yhat = logsout.getElement(2).Values.Data;
timeDT = logsout.getElement(1).Values.Time;
H = logsout.getElement(3).Values.Data;
Tu = logsout.getElement(4).Values.Data;
y1 = logsout.getElement(7).Values.Data; % measurement 1 expressed in deg
Celsius
y2 = logsout.getElement(6).Values.Data; % measurement 1 expressed in meters

inno = [y1 y2] - yhat; % innovation process
```

```
figure, autocorr(inno(:,1))
```




```
figure, autocorr(inno(:,2))
```




```
x = [H(1:Ts/STEP_SIZE:end) Tu(1:Ts/STEP_SIZE:end) D(1:Ts/STEP_SIZE:end,3)];
est_err = x - xhat;
Qe_sim = cov(est_err)
```

```
Qe_sim = 3×3
    0.0000   -0.0000   -0.0000
   -0.0000    0.0043    0.0000
   -0.0000    0.0000    0.0000
```

```
disp('Qe_th =')
```

```
Qe_th =
```

```
disp(Qe_th)
```

```
    0.0000   -0.0000   -0.0000
   -0.0000    0.0055    0.0000
   -0.0000    0.0000  358.1696
```

Linear Control Design 2 - Fall 2019 - Exam Part I

Introduction

The Part I of the Exam in Linear Control Design 2 (E19) consists of numerical exercises testing the acquisition of competences in the areas of analysis and design of control systems using modern control theory based on state space representation of system dynamics.

The scoring of each problem is clearly stated.

It is the sole responsability of the student to guarantee that the solution delivered for evaluation can be run by the examiner without the need of contacting the student. All dependencies on files external to this Matlab Live Script must be checked and included in the final delivery. **If the examiner will not be able to execute the Matlab Live Script delivered as solution by the student, the Part I will be considered failed.**

```
% Fill in your information
Exam = 'LCD2 E19'
```

```
Exam =
'LCD2 E19'
```

```
Student_Name = 'Student Name'
```

```
Student_Name =
'Student Name'
```

```
Student_Number = 'Student Number'
```

```
Student_Number =
'Student Number'
```

Optimal control of an active suspension system

The suspension system is a key component of the vehicle directly affecting the passenger comfort and the ability of the vehicle itself to hold the road. In active suspension system an actuator is positioned between the sprung mass of the vehicle and the unsprung mass (wheel, tyre, brake, etc.) parallel to the suspension elements with the objective of pulling/pushing the vehicle body in order to suppress the vibrations due to road irregularities.

Quarter-car suspension system

The system dynamics of a suspension system for a single wheel is descibed by

$$\begin{aligned} m_s\ddot{z}_s &= -b_s(\dot{z}_s - \dot{z}_u) - k_s(z_s - z_u) + f_a \\ m_u\ddot{z}_u &= b_s(\dot{z}_s - \dot{z}_u) + k_s(z_s - z_u) - f_a + b_t(\dot{z}_r - \dot{z}_u) + k_t(z_r - z_u) \end{aligned} \tag{1}$$

where

- m_s, b_s, k_s are the mass, damping and stiffness of the sprung element (quarter-car)
- m_u is the mass of the unsprung element (wheel-tyre-brake)

- b_t, k_t are the damping and stiffness of the tyre
- z_s is the displacement of the sprung element
- z_u is the displacement of the unsprung element
- z_r is the displacement of the road
- f_a is the force exerted by an hydraulic actuator placed between the sprung and unsprung elements

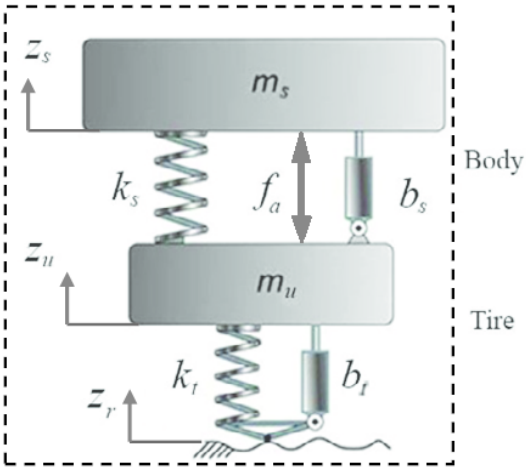
The applied force on the tyre

$$f_d = b_t\dot{z}_r + k_t z_r \tag{2}$$

is considered as an unknown disturbance in the sytem, which should be properly handled by the active suspension. Based on this, the system dynamics in Eq. (1) can be rewritten as

$$\begin{aligned} m_s\ddot{z}_s &= -b_s(\dot{z}_s - \dot{z}_u) - k_s(z_s - z_u) + f_a \\ m_u\ddot{z}_u &= b_s(\dot{z}_s - \dot{z}_u) + k_s(z_s - z_u) - f_a + f_d \end{aligned} \tag{3}$$

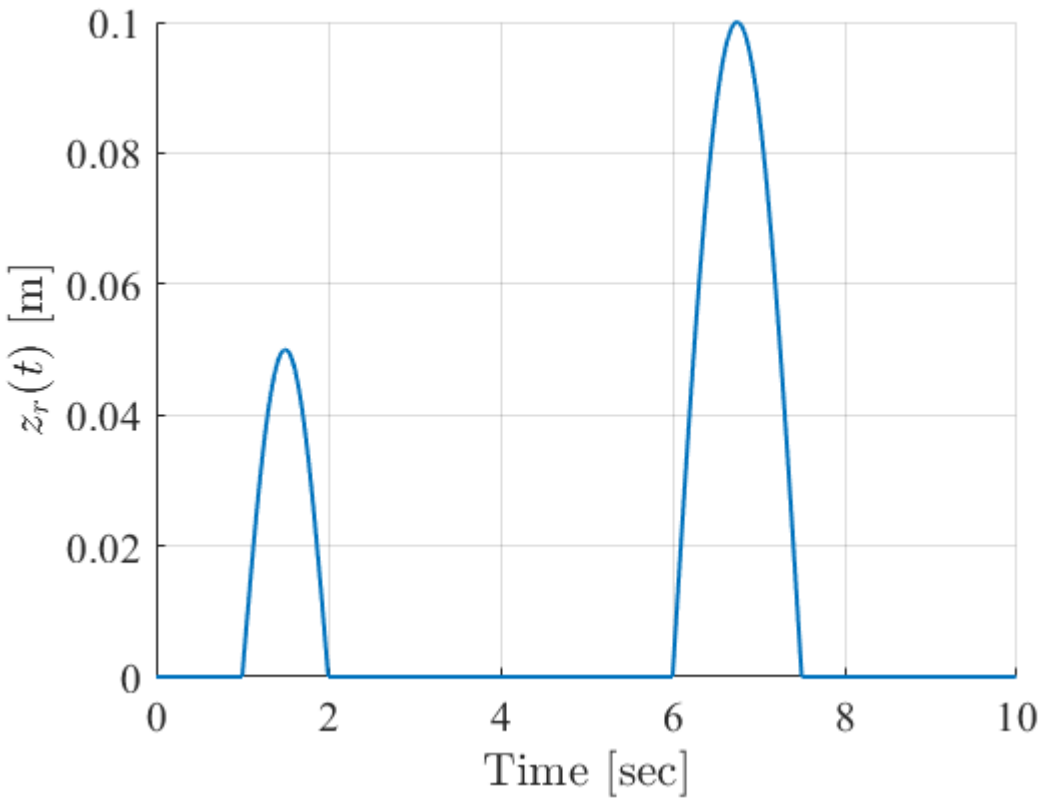
The following figure details the schematic of the system with all the relevant quantities.



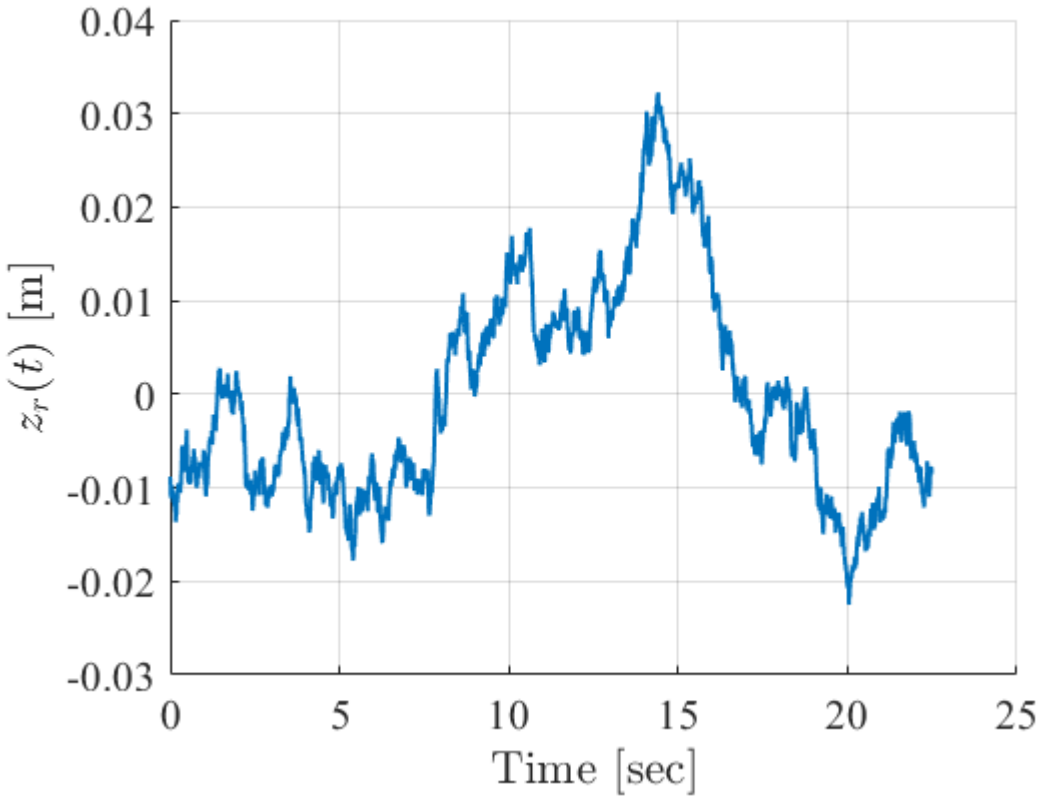
The linear model in Eq. (1) is implemented in the Simulink file QuarterCar_ActiveSusp_SimulinkYYYYC, where YYYYC refers to the Simulink version (2018b, 2018a, 2017b, 2017a, 2016b, 2016a). The numerical values for the model parameters are provided right below.

```
% Model paramters (RG)
% System parameters
m_s = 243; % sprung mass [kg]
m_u = 40; % unsprung mass [kg]
b_s = 370; % damping coefficient [N*s/m]
b_t = 414; % tyre damping coefficient [N*s/m]
k_s = 14761; % stiffness coefficient [N/m]
k_t = 124660; % tyre stiffness coefficient [N/m]

% Road profiles for open and closed-loop simulations (RG)
load road_profile.mat
```

```
time_det = time_det'; % column vector
time_stc = time_stc'; % column vector
z_r_det = z_r_det'; % column vector
z_r_stc = z_r_stc'; % column vector
% two road profiles are included:
% [t_det,z_r_det,z_r_dot_det] - deterministic road profile where t_det is the
% travelling time, z_r_det is the vertical road displacement and z_r_dot_det
% is the rate of change of the road profile.
% [t_stc,z_r_stc,z_r_dot_stc] - stochastic road profile where t_stc is the
% travelling time, z_r_stc is the vertical road displacement and z_r_dot_stc
% is the rate of change of the road profile.
% Both profiles have been generated for a car traveling with a forward
% speed of 40 km/h

% Plot road profiles (RG)
figure, h1 = axes; set(h1,'FontName','times','FontSize',16)
hold on, grid on
plot(time_det,z_r_det,'LineWidth',1.5)
xlabel('Time [sec]','FontName','times','FontSize',16,'Interpreter','latex')
ylabel('$z_r(t)$ [m]','FontName','times','FontSize',16,'Interpreter','latex')
```

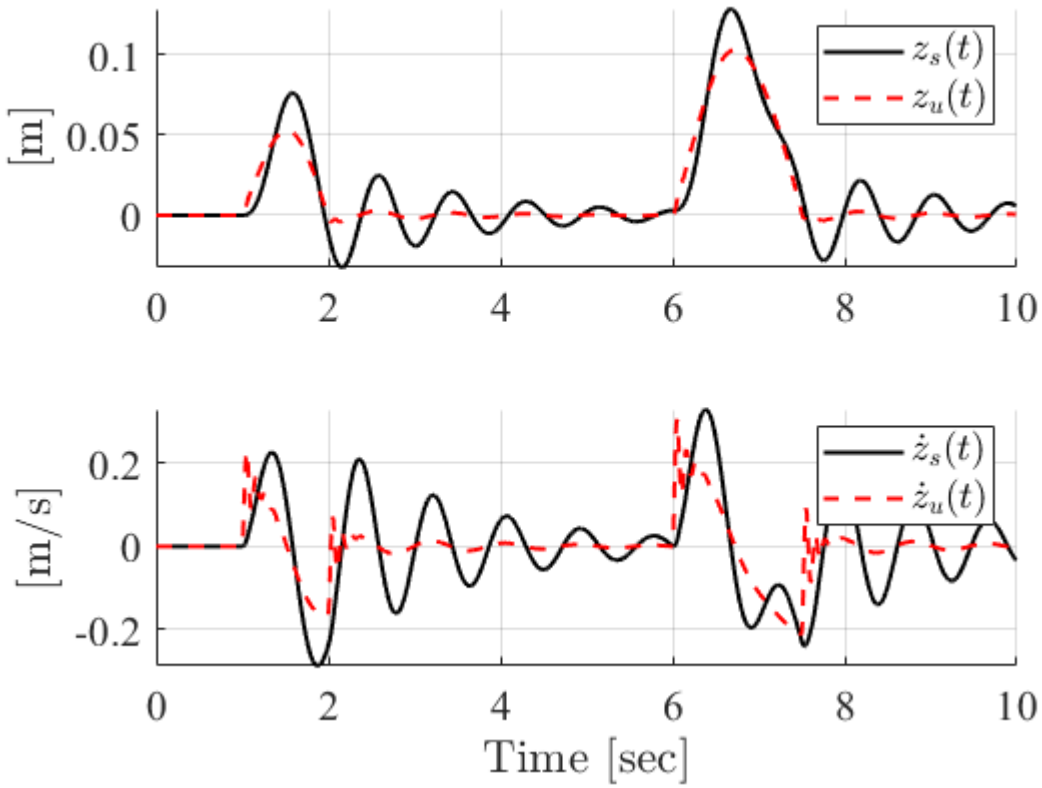



```
figure, h2 = axes; set(h2,'FontName','times','FontSize',16)
hold on, grid on
plot(time_stc,z_r_stc,'LineWidth',1.5)
xlabel('Time [sec]','FontName','times','FontSize',16,'Interpreter','latex')
ylabel('$z_r(t)$ [m]','FontName','times','FontSize',16,'Interpreter','latex')
```




```
% Test the provided active suspension model on deterministic road profile
(RG)
SIM_TIME = time_det(end); % simulation time (can be changed as needed)
STEP_SIZE = 0.01; % integration step size of Simulink (can be changed as
needed)
% Change with the name of the model you are using
SIMULINK_FILENAME = 'QuarterCar_ActiveSusp_Simulink2018b';
f_a = zeros(length(time_det),1);
sim(SIMULINK_FILENAME,SIM_TIME,[],[time_det f_a z_r_det z_r_dot_det])

% Plot the temporal behaviour of the state variables
z_s = logsout.getElement('z_s').Values.Data;
z_s_dot = logsout.getElement('z_s_dot').Values.Data;
z_u = logsout.getElement('z_u').Values.Data;
z_u_dot = logsout.getElement('z_u_dot').Values.Data;

figure, h5 = subplot(2,1,1); set(h5,'FontName','times','FontSize',16)
hold on, grid on
plot(time_det,z_s,'k',time_det,z_u,'--r','LineWidth',1.5)
ylabel('[m]','FontName','times','FontSize',16,'Interpreter','latex')
leg1 = legend('$z_s(t)$','$z_u(t)$');
set(leg1,'FontName','times','FontSize',14,'Interpreter','latex')
h6 = subplot(2,1,2); set(h6,'FontName','times','FontSize',16)
hold on, grid on
plot(time_det,z_s_dot,'k',time_det,z_u_dot,'--r','LineWidth',1.5)
ylabel('[m/s]','FontName','times','FontSize',16,'Interpreter','latex')
```

```
xlabel('Time [sec]','FontName','times','FontSize',16,'Interpreter','latex')
leg2 = legend('$\dot{z}_s(t)$','$\dot{z}_u(t)$');
set(leg2,'FontName','times','FontSize',14,'Interpreter','latex')
```



Open-loop system analysis

Problem 1 [2 points] Based on the system dynamics described in Eq. (3), set-up the linear state space model of the system.

```
% Your solution goes here
A = [0 1 0 0; -k_s/m_s -b_s/m_s k_s/m_s b_s/m_s; ...
     0 0 0 1; k_s/m_u b_s/m_u -(k_s+k_t)/m_u -(b_s+b_t)/m_u];
B = [0 1/m_s 0 -1/m_u]';
Bv = [0 0 0 1/m_u]';
C = [-k_s/m_s -b_s/m_s k_s/m_s b_s/m_s];
D =  1/m_s;
Dv = 0;

act_susp = ss(A,[B Bv],C,[D Dv]);
disp('The linear model describing the active suspension is described by')
```

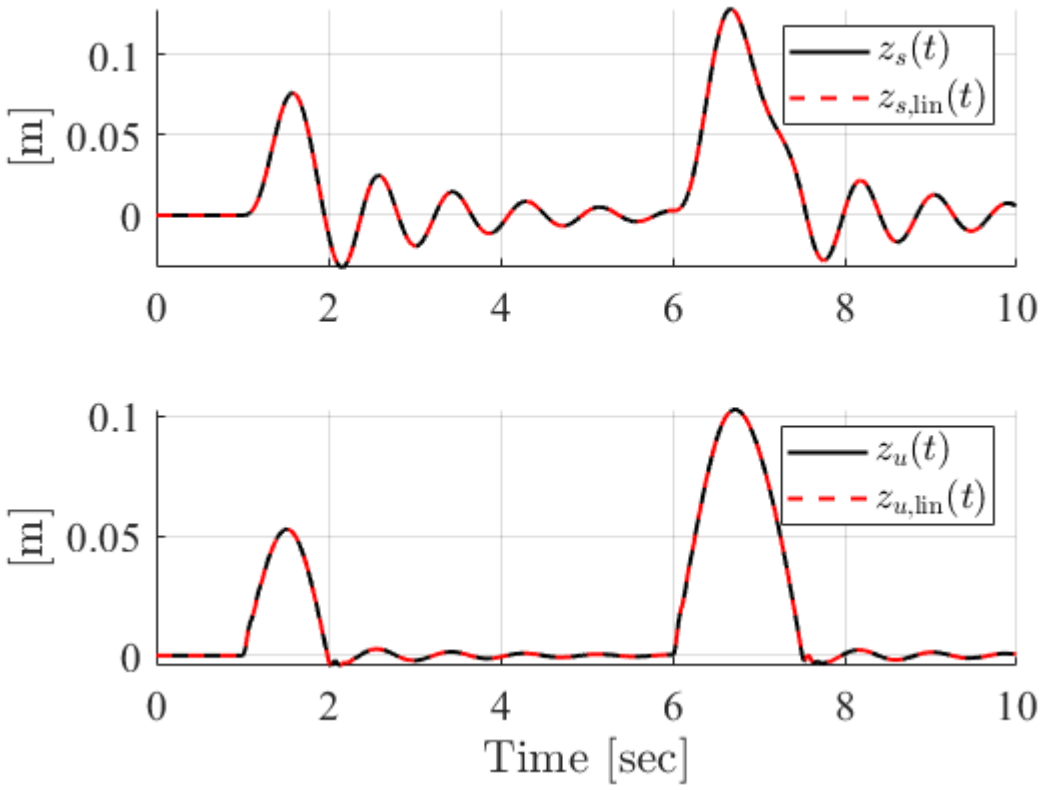
```
The linear model describing the active suspension is described by
```

```
act_susp
```

```
act_susp =
```

```
A =
         x1      x2      x3      x4
 x1       0       1       0       0
 x2  -60.74  -1.523   60.74   1.523
 x3       0       0       0       1
 x4     369    9.25   -3486   -19.6

B =
           u1        u2
 x1         0         0
 x2  0.004115         0
 x3         0         0
 x4    -0.025     0.025

C =
         x1      x2      x3      x4
 y1  -60.74  -1.523   60.74   1.523

D =
           u1        u2
 y1  0.004115         0

Continuous-time state-space model.
```

```
f_d = k_t*z_r_det + b_t*z_r_dot_det;
u = [zeros(length(time_det),1) f_d];
x0 = zeros(4,1);
[y,t,x] = lsim(act_susp,u,time_det,x0);

figure, h3 = subplot(2,1,1); set(h3,'FontName','times','FontSize',16)
hold on, grid on
plot(time_det,z_s,'k',time_det,x(:,1),'--r','LineWidth',1.5)
ylabel('[m]','FontName','times','FontSize',16,'Interpreter','latex')
leg1 = legend('$z_s(t)$','$z_{s,\mathrm{lin}}(t)$');
set(leg1,'FontName','times','FontSize',14,'Interpreter','latex')
h4 = subplot(2,1,2); set(h4,'FontName','times','FontSize',16)
hold on, grid on
plot(time_det,z_u,'k',time_det,x(:,3),'--r','LineWidth',1.5)
ylabel('[m]','FontName','times','FontSize',16,'Interpreter','latex')
xlabel('Time [sec]','FontName','times','FontSize',16,'Interpreter','latex')
leg2 = legend('$z_u(t)$','$z_{u,\mathrm{lin}}(t)$');
set(leg2,'FontName','times','FontSize',14,'Interpreter','latex')
```



Problem 2 [2 points] Evaluate the internal and the external stability of system.

```
% Your solution goes here
lambda_A_ol = eig(A);
disp(lambda_A_ol)
```

```
  -9.9451 +58.1744i
  -9.9451 -58.1744i
  -0.6162 + 7.3465i
  -0.6162 - 7.3465i
```

```
disp('The real part of all four eigenvalues is negative, hence')
```

```
The real part of all four eigenvalues is negative, hence
```

```
disp('the system is internally asymptotically stable. Due to internal')
```

```
the system is internally asymptotically stable. Due to internal
```

```
disp('stability the system is also externally stable.')
```

```
stability the system is also externally stable.
```

Control system design for vibration damping

The main control objectives are to improve the passenger comfort and vehicle handling in the presence of road profile variations by actively damping the introduced vibrations. The passenger comfort is evaluated in terms of

vertical acceleration $\ddot{z}_s$ experienced by the sprung mass; whereas the vehicle handling is evaluated in terms of tire deflection $(z_u - z_r)$ and suspension deflection $(z_s - z_u)$.

Closed-loop system requirements

When subject to the deterministic road profile $(z_r(t), \dot{z}_r(t))_{\text{det}}$ the closed-loop system is expected to fulfil the following requirements

1. The peak of the vertical acceleration $\ddot{z}_s$ of the sprung mass is reduced by 20-40% in comparison to the peak displayed in open-loop (passive suspension)
2. The vertical acceleration $\ddot{z}_s$ of the sprung mass settles to zero within 2-4 seconds since the time a road bump hits the tire
3. The control effor f_a is kept within $\pm 1\ kN$

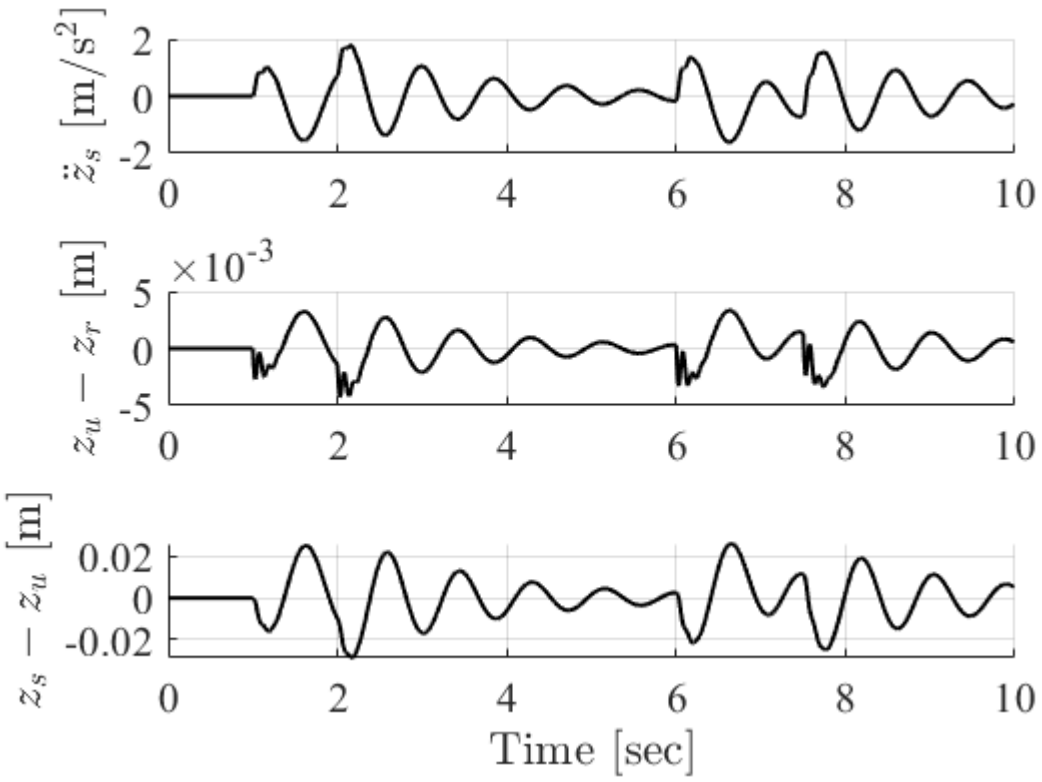
Problem 3 [2 points] Assess the open-loop characteristics of the

- vertical acceleration $\ddot{z}_s$
- tire deflection $(z_u - z_r)$
- suspension deflection $(z_s - z_u)$

when the system is subject to the deterministic road profile $(z_r(t), \dot{z}_r(t))_{\text{det}}$ in terms of largest deviation from the rest position, decay rate and settling time.

```
% Your solution goes here

% Plot the temporal behaviour of
%   *   vertical accelaration
%   *   tire deflection
%   *   suspension deflection

z_s = logsout.getElement('z_s').Values.Data;
z_s_dot = logsout.getElement('z_s_dot').Values.Data;
z_s_ddot = logsout.getElement('z_s_ddot').Values.Data;
z_u = logsout.getElement('z_u').Values.Data;
z_u_dot = logsout.getElement('z_u_dot').Values.Data;

tire_defl = z_u - z_r_det;
susp_defl = z_s - z_u;

figure, h6 = subplot(3,1,1); set(h6,'FontName','times','FontSize',16)
hold on, grid on
plot(time_det,z_s_ddot,'k','LineWidth',1.5)
ylabel('$\ddot{z}_s$ [m/
s$^2$]','FontName','times','FontSize',16,'Interpreter','latex')
h7 = subplot(3,1,2); set(h7,'FontName','times','FontSize',16)
hold on, grid on
```

```
plot(time_det,tire_defl,'k','LineWidth',1.5)
ylabel('$z_u -z_r$
[m]','FontName','times','FontSize',16,'Interpreter','latex')
h8 = subplot(3,1,3); set(h8,'FontName','times','FontSize',16)
hold on, grid on
plot(time_det,susp_defl,'k','LineWidth',1.5)
ylabel('$z_s-z_u$
[m]','FontName','times','FontSize',16,'Interpreter','latex')
xlabel('Time [sec]','FontName','times','FontSize',16,'Interpreter','latex')
```




```
max_z_s_ddot = max(abs(z_s_ddot));
disp('The maximum peak of the vertical acceleration in open loop is [m/
s^2]:')
```

```
The maximum peak of the vertical acceleration in open loop is [m/s^2]:
```

```
disp(max_z_s_ddot)
```

```
    1.7917
```

```
max_tire_defl = max(abs(tire_defl));
disp('The maximum peak of the tire deflection in open loop is [m]:')
```

```
The maximum peak of the tire deflection in open loop is [m]:
```

```
disp(max_tire_defl)
```

```
    0.0043
```

```
max_susp_defl = max(abs(susp_defl));
disp('The maximum peak of the suspension deflection in open loop is [m]:')
```

```
The maximum peak of the suspension deflection in open loop is [m]:
```

```
disp(max_susp_defl)
```

```
    0.0289
```

An estimate of the decay rate and settling time can be obtained through the analysis of the system eigenvalues

```
[wn,zeta] = damp(A)
```

```
wn = 4×1
   59.0183
   59.0183
    7.3723
    7.3723
zeta = 4×1
    0.1685
    0.1685
    0.0836
    0.0836
```

```
T_osc = 2*pi./[wn(1) wn(3)] % periods of oscialltory modes
```

```
T_osc = 1×2
    0.1065    0.8523
```

Two oscillatory modes characterize the state and output response of the system: one oscillation with period of approximately 0.1s, and a second oscillation with period of approximately 0.85s. These different oscillations are clearly visible from the plots of acceleration, tire and suspension deflection. The decay rate and settling time are determined by the oscillatory mode with the larger oscillation period, which is also associated with the smallest damping ratio ($\zeta = 0.0836$).

```
% Decay rate
alpha = wn(3)*zeta(3) % this is the real part of the complex eigenvalue,
which is also appearing in the exponent of the oscillatory eigenmode
```

```
alpha = 0.6162
```

```
% Settling time
t_set = 5*1/alpha % the settling time [s] is estimated as 5 times the time
constant associated with the slow complex eigenvalue
```

```
t_set = 8.1137
```

Problem 4 [6 points] Under the assumption that the state is fully accessible design a discrete time optimal controller that meets the aforementioned specifications using the linear quadratic regulator approach.

```
% Your solution goes here
% Discretization
w_max = max(wn); % fastest dynamics
```

```
t_set_min = 2; % smallest desired settling time [s]
Ts = min([(2*pi/ceil(w_max))/10 t_set_min/10]); % sampling time (ten times
faster than the fastest dynamics)
Ts = Ts - rem(Ts,STEP_SIZE)
```

```
Ts = 0.0100
```

```
[F,G] = c2d(A,B,Ts)
```

```
F = 4×4
    0.9971    0.0099    0.0021    0.0001
   -0.5735    0.9827    0.3290    0.0157
    0.0167    0.0005    0.8413    0.0086
    3.1319    0.0952  -29.8260    0.6741
G = 4×1
10^-3 ×
    0.0002
    0.0388
   -0.0011
   -0.2122
```

```
[~,Gv] = c2d(A,Bv,Ts)
```

```
Gv = 4×1
10^-3 ×
    0.0000
    0.0020
    0.0011
    0.2141
```

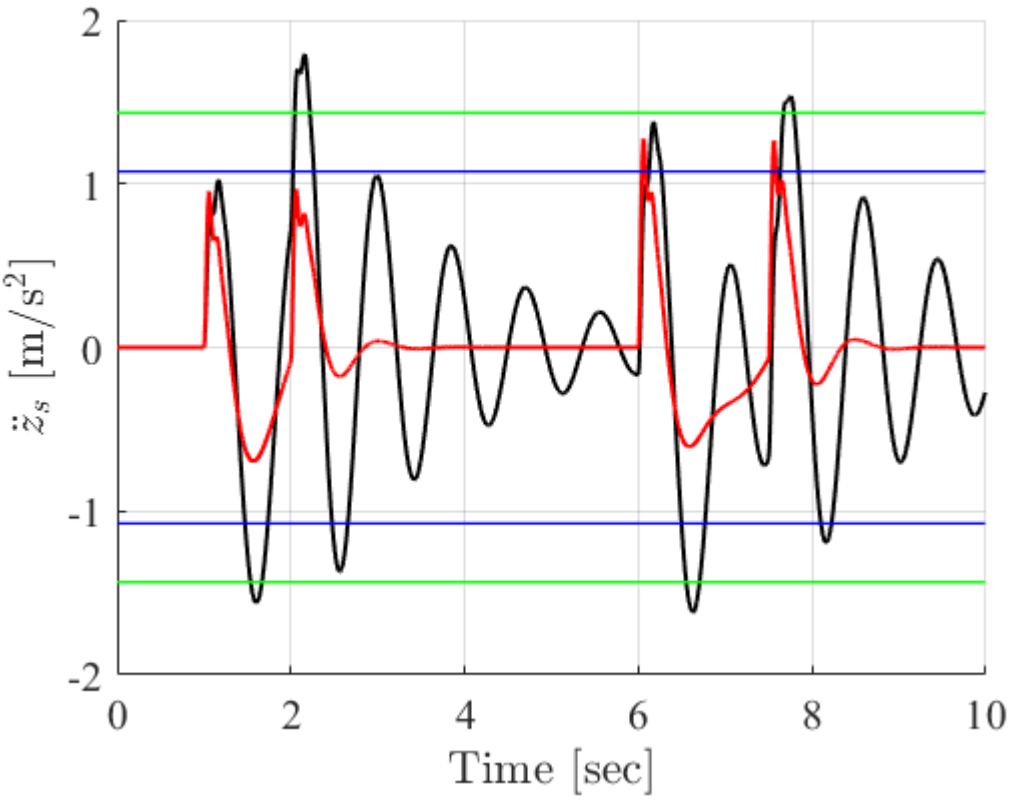
```
% Design a discrete time LQR regulator
R1 = diag([100 0.1 100 0.1]);
R2 = 1/(750)^2; % maximum force is 1kN hence the weight is chosen using 75%
of total maximum force
[K_lqr,~,lambda_lqr] = dlqr(F,G,R1*Ts,R2*Ts)
```

```
K_lqr = 1×4
10^3 ×
    1.4832    0.7352   -0.4907   -0.0254
lambda_lqr = 4×1 complex
   0.9777 + 0.0739i
   0.9777 - 0.0739i
   0.7525 + 0.4943i
   0.7525 - 0.4943i
```

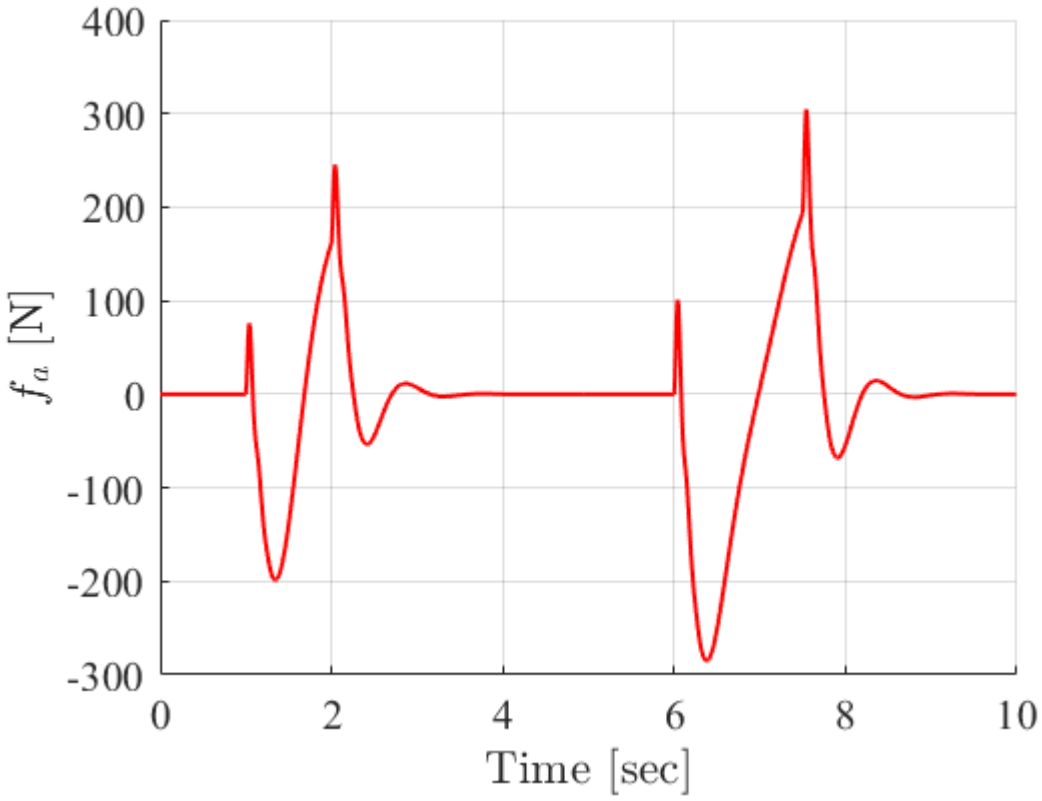
Problem 5 [2 points] Implement the discrete time optimal controller in the given Simulink diagram and evalute the closed-loop system performance against the open-loop one, when the sytem is subject to the deterministic road profile $(z_r(t), \dot{z}_r(t))_{\text{det}}$. Verify that the closed-loop requirements are fulfilled.

```
% Your solution goes here
STEP_SIZE = 0.005; % integration step size of Simulink (can be changed as
needed)
% Change with the name of the model you are using
SIMULINK_FILENAME = 'QuarterCar_ActiveSusp_DLQR_Simulink2018b';
sim(SIMULINK_FILENAME,SIM_TIME,[],[time_det z_r_det z_r_dot_det])
```

```
% Plot the temporal behaviour of the state variables
time_sim = logsout.getElement('z_s').Values.Time;
z_s_cl = logsout.getElement('z_s').Values.Data;
z_s_dot_cl = logsout.getElement('z_s_dot').Values.Data;
z_s_ddot_cl = logsout.getElement('z_s_ddot').Values.Data;
z_u_cl = logsout.getElement('z_u').Values.Data;
z_u_dot_cl = logsout.getElement('z_u_dot').Values.Data;
z_r_sim = logsout.getElement('z_r').Values.Data;
z_r_dot_sim = logsout.getElement('z_r_dot').Values.Data;
f_a = logsout.getElement('f_a').Values.Data;

tire_defl_cl = z_u_cl - z_r_sim;
susp_defl_cl = z_s_cl - z_u_cl;

max_peak_vert_acc_20 = 0.8*max_z_s_ddot;
max_peak_vert_acc_40 = 0.6*max_z_s_ddot;

figure, h6 = axes; set(h6,'FontName','times','FontSize',16)
hold on, grid on
plot(time_det,z_s_ddot,'k',time_sim,z_s_ddot_cl,'-r','LineWidth',1.5)
line([time_det(1) time_det(end)],[max_peak_vert_acc_20
max_peak_vert_acc_20],'Color','g','LineWidth',1)
line([time_det(1) time_det(end)],[max_peak_vert_acc_40
max_peak_vert_acc_40],'Color','b','LineWidth',1)
line([time_det(1) time_det(end)],-[max_peak_vert_acc_20
max_peak_vert_acc_20],'Color','g','LineWidth',1)
line([time_det(1) time_det(end)],-[max_peak_vert_acc_40
max_peak_vert_acc_40],'Color','b','LineWidth',1)
ylabel('$\ddot{z}_s$ [m/
s$^2$]','FontName','times','FontSize',16,'Interpreter','latex')
xlabel('Time [sec]','FontName','times','FontSize',16,'Interpreter','latex')
```




```
figure, h9 = axes; set(h9,'FontName','times','FontSize',16)
hold on, grid on
plot(time_det,f_a,'-r','LineWidth',1.5)
ylabel('$f_a$ [N]','FontName','times','FontSize',16,'Interpreter','latex')
xlabel('Time [sec]','FontName','times','FontSize',16,'Interpreter','latex')
```




```
max_z_s_ddot_cl = max(abs(z_s_ddot_cl));
disp('The maximum peak of the vertical acceleration in open loop is [m/
s^2]:')
```

```
The maximum peak of the vertical acceleration in open loop is [m/s^2]:
```

```
disp(max_z_s_ddot_cl)
```

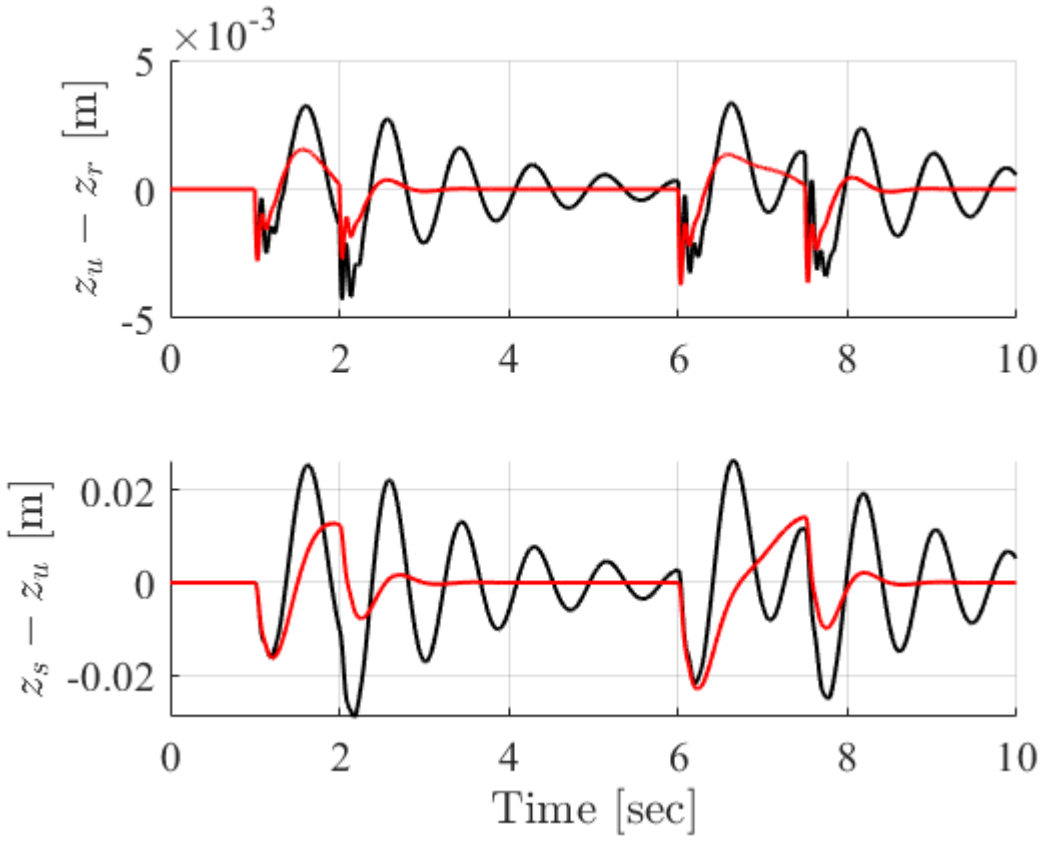
```
    1.2756
```

```
if max_z_s_ddot_cl <= max_peak_vert_acc_20 && max_z_s_ddot_cl >=
max_peak_vert_acc_40
    disp('Damping requirement is fulfilled and the peak reduction is')
    damp_red = (max_z_s_ddot - max_z_s_ddot_cl)/max_z_s_ddot*100;
    disp(strcat(num2str(damp_red),'%'))
else
    disp('Damping requirement is not fulfilled and the peak reduction is')
    damp_red = (max_z_s_ddot - max_z_s_ddot_cl)/max_z_s_ddot*100;
    disp(strcat(num2str(damp_red),'%'))
end
```

```
Damping requirement is fulfilled and the peak reduction is
28.8061%
```

```
figure, h7 = subplot(2,1,1); set(h7,'FontName','times','FontSize',16)
hold on, grid on
plot(time_det,tire_defl,'k',time_sim,tire_defl_cl,'-r','LineWidth',1.5)
```

```
ylabel('$z_u -z_r$
[m]','FontName','times','FontSize',16,'Interpreter','latex')
h8 = subplot(2,1,2); set(h8,'FontName','times','FontSize',16)
hold on, grid on
plot(time_det,susp_defl,'k',time_sim,susp_defl_cl,'-r','LineWidth',1.5)
ylabel('$z_s-z_u$
[m]','FontName','times','FontSize',16,'Interpreter','latex')
xlabel('Time [sec]','FontName','times','FontSize',16,'Interpreter','latex')
```



Observer design for estimation of road profile

In order to implement an output feedback controller for the active suspension, the state vector must be estimated based on available measurements. For the active suspension a common measurement easy to implement is the measurement of the acceleration of the sprung mass, i.e. $\ddot{z}_s$. Therefore in the last part of the exercise, we assume that the following measurement is available

$$y_1 = \ddot{z}_s + v_1 \tag{4}$$

where v_1 is white noise with zero mean and noise intensity $I_{\ddot{z}_s} = 6.25 \times 10^{-4}\, m^2/s^4$.

In the resolution of the following problems it is assumed that the system dynamics Eq. (3) is subject to the stochastic road profile $(z_r(t), \dot{z}_r(t))_{\text{stc}}$.

Problem 6 [6 points] Under the assumption that the disturbance f_d is unknown, design a discrete time Kalman filter able to reconstruct the state vector $\mathbf{x} = [z_s, \dot{z}_s, z_u, \dot{z}_u]^{\mathrm{T}}$.

```
% Your solution goes here
F_kf = F;
G_kf = G;
C_kf = C;

% Check observability
Mo = obsv(F_kf,C_kf);
if rank(Mo) == size(F_kf,1)
    disp('The system is fully observable')
else
    disp('The system is not fully observable')
end
```

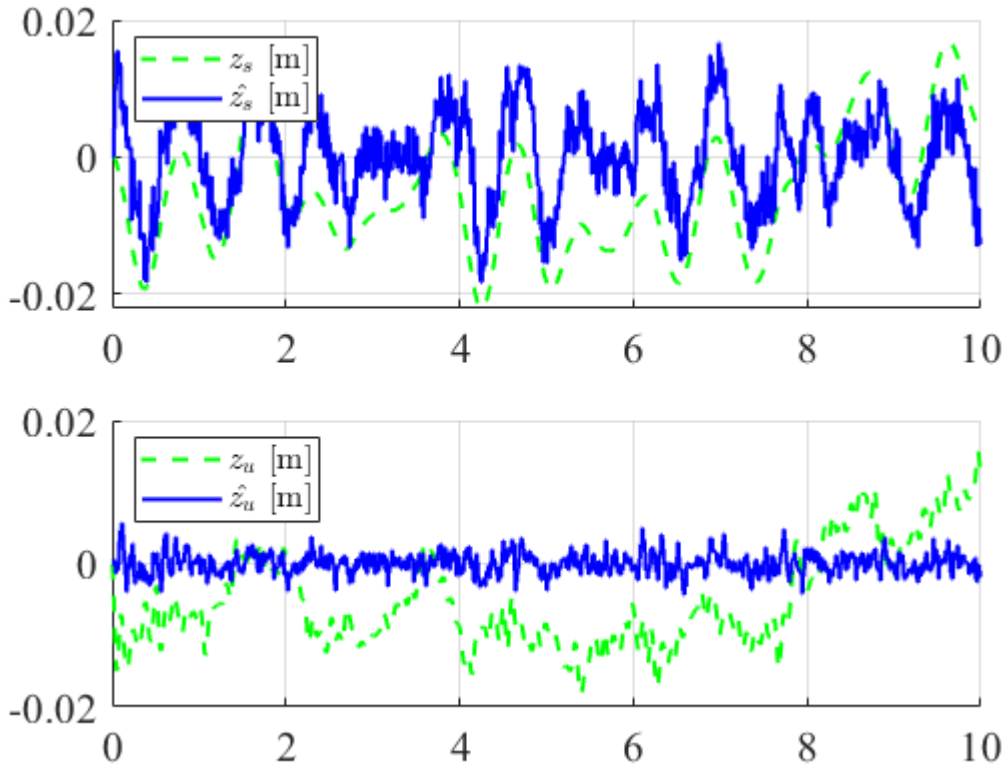
```
The system is fully observable
```

```
% Process noise
V1 = diag([0.0001 0.005 0.0001 5e-6]); % intensity of the process noise
% (other tunings of the process noise intensities could provide better
% estimation of the state variables)
Bn = [1 0 0 0;0 k_s/m_s 0 0;0 0 1 0;0 0 0 k_t/m_u];
V1d = Bn*V1*Bn'*Ts;
Gn = eye(4);

% Measurement noise
I_zs_ddot = 6.25*10^-4; % [m^2/s^4]
V2 = I_zs_ddot;
V2d = V2/Ts;

% Design the Kalman filter
[L_kf,P_kf,Qe_th,lambda_e] = lqed(A,Bn,C_kf,V1,V2,Ts)
```

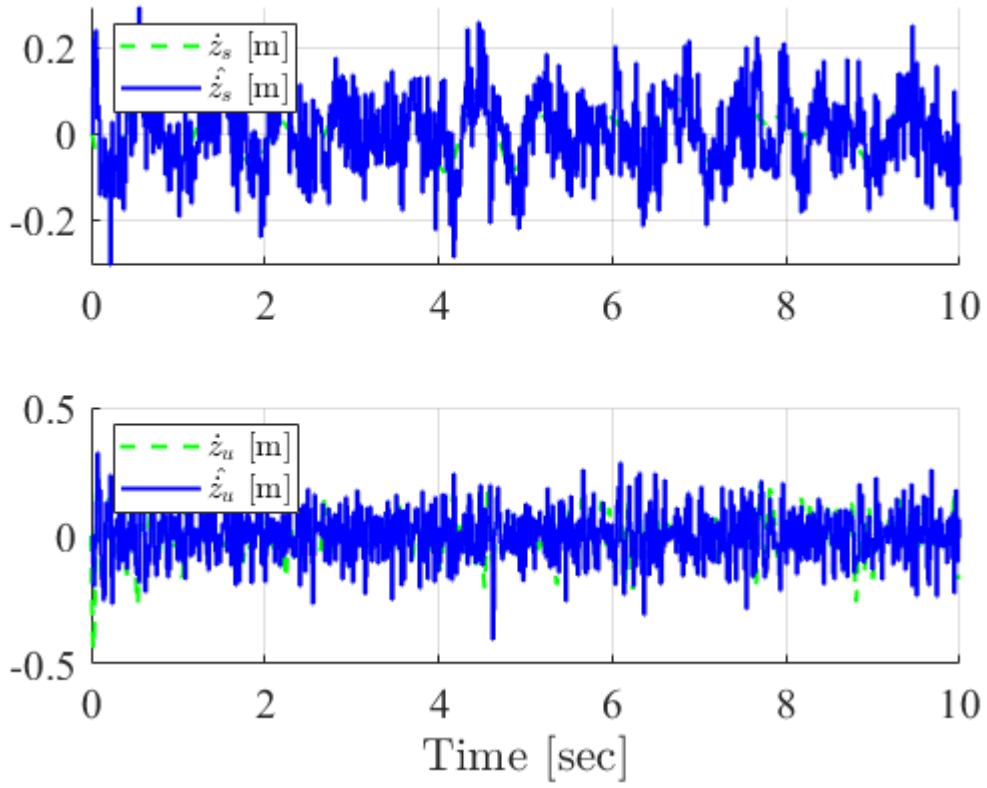
```
L_kf = 4×1
   -0.0069
   -0.2560
   -0.0041
    0.2728
P_kf = 4×4
    0.0004    0.0090    0.0003   -0.0010
    0.0090    0.5763    0.0071    0.1934
    0.0003    0.0071    0.0003    0.0008
   -0.0010    0.1934    0.0008    0.6060
Qe_th = 4×4
    0.0002    0.0042    0.0002    0.0041
    0.0042    0.3985    0.0043    0.3829
    0.0002    0.0043    0.0002    0.0039
    0.0041    0.3829    0.0039    0.4042
lambda_e = 4×1 complex
   0.7659 + 0.2805i
   0.7659 - 0.2805i
   0.6620 + 0.0000i
   0.0424 + 0.0000i
```

Problem 7 [4 points] Implement the discrete time Kalman filter in Simulink diagram of the open-loop system and assess its estimation performance when the open-loop system is subject to the stochastic road profile $(z_r(t), \dot{z}_r(t))_{\text{stc}}$.

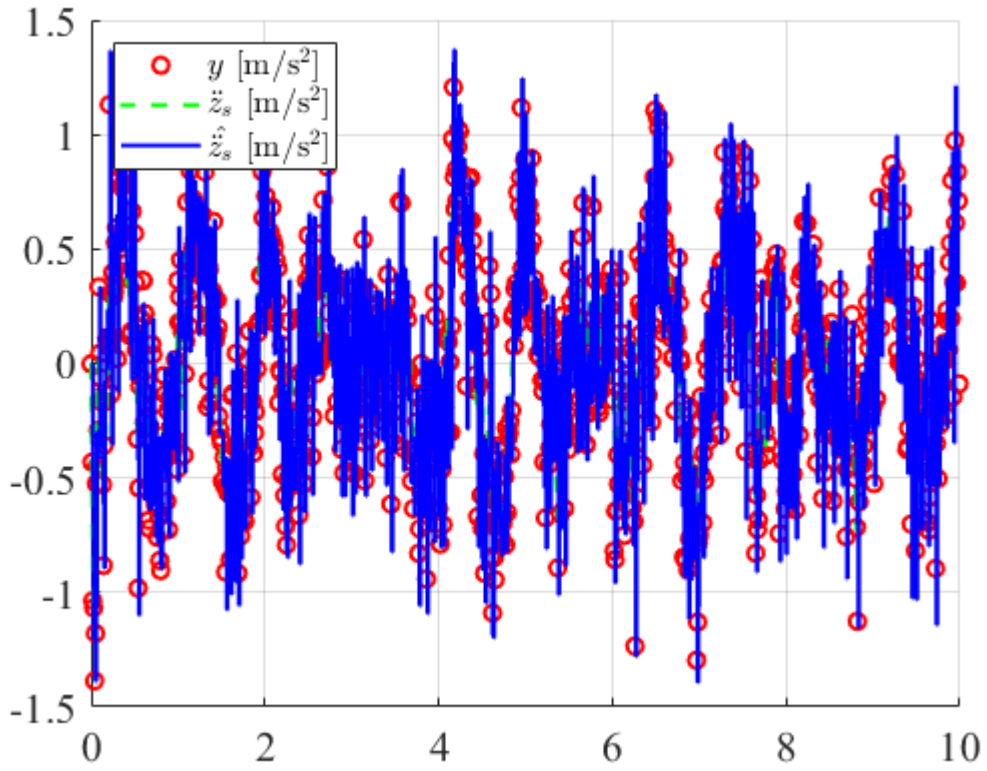
```
% Your solution goes here
SIMULINK_FILENAME = 'QuarterCar_ActiveSusp_KF_Simulink2018b';
x0hat = zeros(4,1);
A_f = 0; % amplitude of the force [N]
w_f = 2*pi*2; % frequency of the force [rad/s]
f_a = A_f*sin(w_f*time_stc);
sim(SIMULINK_FILENAME,SIM_TIME,[],[time_stc z_r_stc z_r_dot_stc f_a])

% Plot the temporal behaviour of the state variables
time_sim = logsout.getElement('z_s').Values.Time;
z_s = logsout.getElement('z_s').Values.Data;
z_s_dot = logsout.getElement('z_s_dot').Values.Data;
z_s_ddot = logsout.getElement('z_s_ddot').Values.Data;
z_u = logsout.getElement('z_u').Values.Data;
z_u_dot = logsout.getElement('z_u_dot').Values.Data;
xhat = logsout.getElement('xhat').Values.Data;
yhat = logsout.getElement('yhat').Values.Data;
y = logsout.getElement('y').Values.Data;
f_a = logsout.getElement('f_a').Values.Data;
timeDT = logsout.getElement('xhat').Values.Time;

figure, h1 = subplot(2,1,1); set(h1,'FontName','times','FontSize',16)
hold on, grid on
plot(time_sim,z_s,'--g','MarkerSize',4,'MarkerFaceColor','r','LineWidth',1.5)
stairs(timeDT,xhat(:,1),'b','LineWidth',1.5)
leg1 = legend('$z_s$ [m]','$\hat{z_s}$ [m]','Location','NorthWest');
set(leg1,'FontName','times','FontSize',12,'Interpreter','latex')

h2 = subplot(2,1,2); set(h2,'FontName','times','FontSize',16)
hold on, grid on
plot(time_sim,z_u,'--g','MarkerSize',4,'MarkerFaceColor','r','LineWidth',1.5)
stairs(timeDT,xhat(:,3),'b','LineWidth',1.5)
leg2 = legend('$z_u$ [m]','$\hat{z_u}$ [m]','Location','NorthWest');
set(leg2,'FontName','times','FontSize',12,'Interpreter','latex')
```



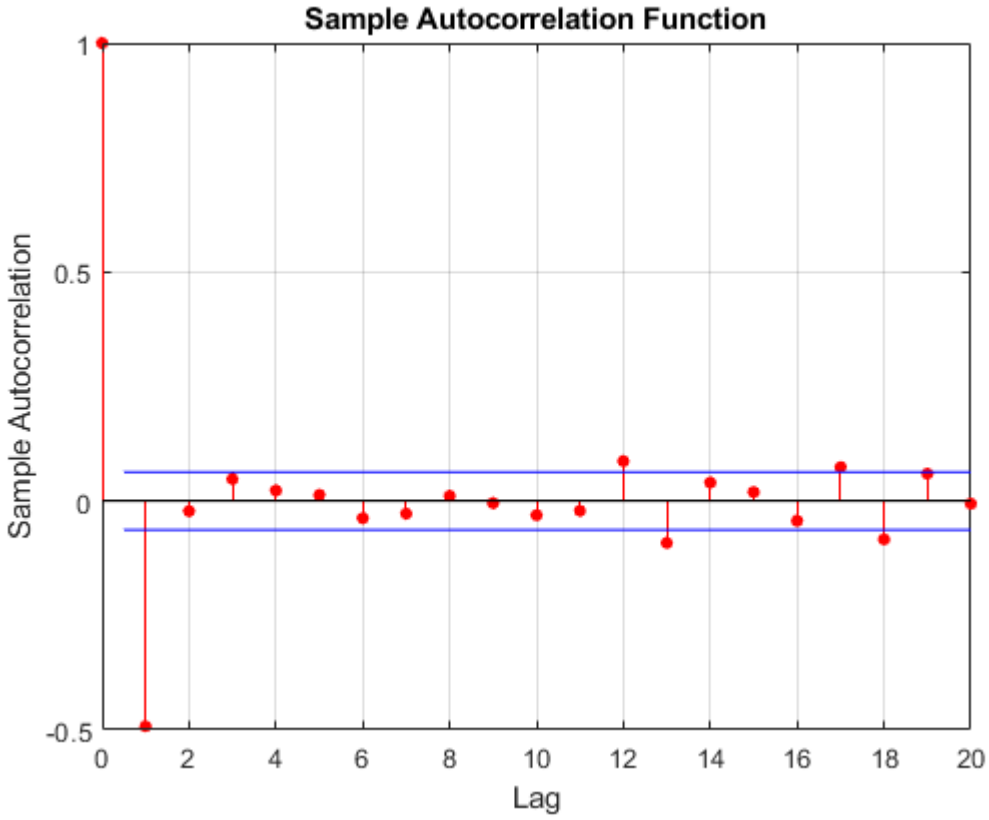

```
figure, h3 = subplot(2,1,1); set(h3,'FontName','times','FontSize',16)
hold on, grid on
plot(time_sim,z_s_dot,'--g','LineWidth',1.5)
stairs(timeDT,xhat(:,2),'b','LineWidth',1.5)
leg3 = legend('$\dot{z}_s$ [m]','$\hat{\dot{z}_s}$
[m]','Location','NorthWest');
set(leg3,'FontName','times','FontSize',12,'Interpreter','latex')

h4 = subplot(2,1,2); set(h4,'FontName','times','FontSize',16)
hold on, grid on
plot(time_sim,z_u_dot,'--g','LineWidth',1.5)
stairs(timeDT,xhat(:,4),'b','LineWidth',1.5)
leg4 = legend('$\dot{z}_u$ [m]','$\hat{\dot{z}_u}$
[m]','Location','NorthWest');
set(leg4,'FontName','times','FontSize',12,'Interpreter','latex')
xlabel('Time [sec]','FontName','times','FontSize',16,'Interpreter','latex')
```




```
z_s_ddot_hat = f_a/m_s -k_s/m_s*xhat(:,1) -b_s/m_s*xhat(:,2) + k_s/
m_s*xhat(:,3) + b_s/m_s*xhat(:,4);

figure, h5 = axes; set(h5,'FontName','times','FontSize',16)
hold on, grid on
plot(timeDT,y,'or',time_sim,z_s_ddot,'--g','LineWidth',1.5)
stairs(timeDT,z_s_ddot_hat,'b','LineWidth',1.5)
leg5 = legend('$y$ [m/s$^2$]','$\ddot{z}_s$ [m/s$^2$]','$\hat{\ddot{z}_s}$
[m/s$^2$]','Location','NorthWest');
set(leg5,'FontName','times','FontSize',12,'Interpreter','latex')
```




```
inno = y - yhat; % innovation process
figure, autocorr(inno)
```



```
x = [z_s z_s_ddot z_u z_u_dot];
est_err = x(1:Ts/STEP_SIZE:end,:) - xhat;
Qe_sim = cov(est_err)
```

```
Qe_sim = 4×4
    0.0001    0.0003    0.0000   -0.0001
    0.0003    0.1592    0.0002    0.0061
    0.0000    0.0002    0.0000   -0.0000
   -0.0001    0.0061   -0.0000    0.0093
```

```
disp('Qe_th =')
```

```
Qe_th =
```

```
disp(Qe_th)
```

```
    0.0002    0.0042    0.0002    0.0041
    0.0042    0.3985    0.0043    0.3829
    0.0002    0.0043    0.0002    0.0039
    0.0041    0.3829    0.0039    0.4042
```

Linear Control Design 2 - Spring 2020 - Exam Part I - Solution Manual

Introduction

The Part I of the Exam in Linear Control Design 2 (F20) consists of numerical exercises testing the acquisition of competences in the areas of analysis and design of control systems using modern control theory based on state space representation of system dynamics.

The scoring of each problem is clearly stated.

It is the sole responsability of the student to guarantee that the solution delivered for evaluation can be run by the examiner without the need of contacting the student. All dependencies on files external to this Matlab Live Script must be checked and included in the final delivery. **If the examiner will not be able to execute the Matlab Live Script delivered as solution by the student, the Part I will be considered failed.**

```
% Fill in your information
Exam = 'LCD2 F20'
```

```
Exam =
'LCD2 F20'
```

```
Student_Name = 'Student Name'
```

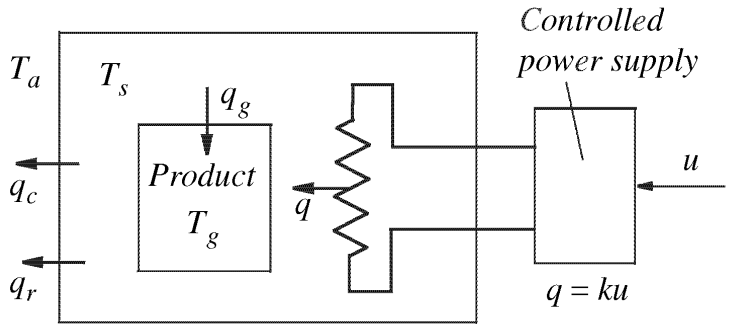
```
Student_Name =
'Student Name'
```

```
Student_Number = 'Student Number'
```

```
Student_Number =
'Student Number'
```

Optimal control of a high temperature oven

A high temperature oven with a product to be heat treated is shown in the figure below.



The outer surface temperature T_s of the oven is the same as the temperature in the inner oven space. The ambient temperature is T_a and the product temperature is T_g. All the temperatures are assumed to be uniform. The variables denoted q, q_c, q_r, q_g are the heat powers exchanged between the different parts of the system.

The oven surface temperature is so high that the heat loss to the surroundings is not only due to convection but also to radiation. Assume that the convection power is proportional to the temperature difference, i. e.,

$$q_c = k_c(T_s - T_a)$$
$$q_g = k_g(T_s - T_g)$$

whereas the radiation heat loss follows the Stefan-Boltzmann law,

$$q_r = k_r(T_s^4 - T_a^4)$$

Note that the temperatures here are absolute temperatures, i.e. measured in Kelvin. The output is the temperature T_g. The total heat capacities of the oven air space and the product are C_s and C_g.

The heating element of the oven is connected to a power supply, which is regulated through the voltage input u ranging between 0 and 10 volts. The heat power injected into the oven is given by

$$q = ku$$

Nonlinear model of the high temperature oven

The nonlinear model describing the dynamical variations of the high temperature oven is dervied by writing out the heat balance equations for the oven surface and the product, i.e.

$$\begin{aligned} C_s \frac{\mathrm{d}T_s}{\mathrm{d}t} &= q - q_c - q_g - q_r \\ &= ku - k_c(T_s - T_a) - k_g(T_s - T_g) - k_r(T_s^4 - T_a^4) \\ C_g \frac{\mathrm{d}T_g}{\mathrm{d}t} &= q_g \\ &= k_g(T_s - T_g) \end{aligned} \tag{1}$$

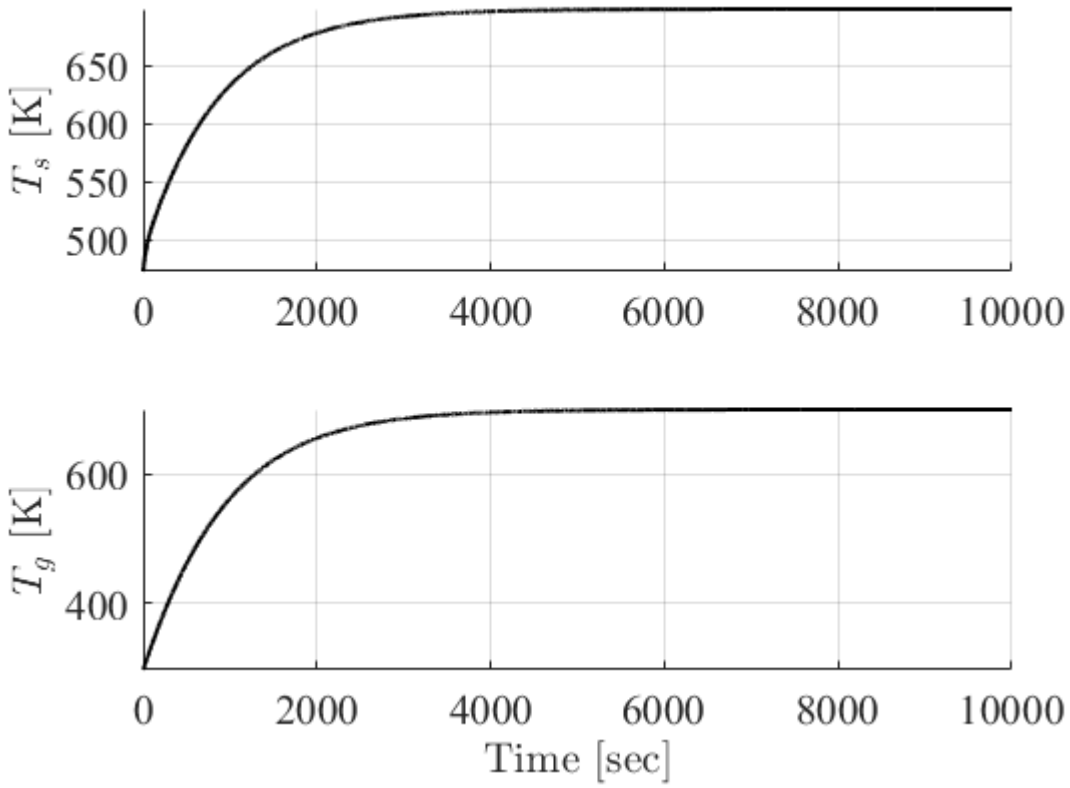
where

- C_s is the heat capacity of the oven
- C_g is the heat capacity of the product
- k_c is the convection constant of the oven
- k_r is the radiation constant of the oven
- k_g is the convection constant of the product
- k is the power constant of the power supply

The nonlinear model in Eq. (1) is implemented in the Simulink file HighTemperatureOven_SimulinkYYYYC, where YYYYC refers to the Simulink version (2019b, 2019a, 2018b, 2018a, 2017b, 2017a, 2016b, 2016a). The numerical values for the model parameters are provided right below.

```
% Model paramters (RG)
% System parameters
kc = 0.8;      % convection constant of the oven [W/K]
```

```
kg = 1.5;      % convection constant of the product [W/K]
kr = 7.8e-10;  % radiation constant of the oven [W/K^4]
Cs = 85;      % heat capacity of the oven [J/K]
Cg = 700;      % heat capacity of the product [J/K]
k  = 100;      % power constant [W/V]
C2K = 273.15;  % conversion from degree Celsius to Kelvin (e.g. 20 degree
Celsius = 20 + C2K Kelvin)

% Initial conditions for testing the Simulink model (RG)
Ta0 = 25; % ambient temperature [degree Celsius]
Ta0K = Ta0 + C2K; % ambient temperature [K]
Ts0 = 200; % oven surface temperature [degree Celsius]
Ts0K = Ts0 + C2K; % oven surface temperature [K]
Tg0 = 25; % product temperature [degree Celsius]
Tg0K = Tg0 + C2K; % product temperature [K]
u0 = 5; % voltage to power supply [V]
umin = 0; % minimum voltage [V]
umax = 10; % maximum voltage [V]

% Test the provided high temperature oven (RG)
SIM_TIME = 10000; % simulation time (can be changed as needed)
STEP_SIZE = 1; % integration step size of Simulink (can be changed as needed)
% Change with the name of the model you are using
SIMULINK_FILENAME = 'HighTemperatureOven_Simulink2019b';
time = (0:STEP_SIZE:SIM_TIME)';
u = u0*ones(length(time),1);
Ta = Ta0K*ones(length(time),1);
sim(SIMULINK_FILENAME,SIM_TIME,[],[time u Ta])

% Plot the temporal behaviour of the oven and product temperatures (RG)
Ts = logsout.getElement('Ts').Values.Data;
Tg = logsout.getElement('Tg').Values.Data;

figure, h1 = subplot(2,1,1); set(h1,'FontName','times','FontSize',16)
hold on, grid on
plot(time,Ts,'k','LineWidth',1.5)
ylabel('$T_s$ [K]','FontName','times','FontSize',16,'Interpreter','latex')
h2 = subplot(2,1,2); set(h2,'FontName','times','FontSize',16)
hold on, grid on
plot(time,Tg,'k','LineWidth',1.5)
ylabel('$T_g$ [K]','FontName','times','FontSize',16,'Interpreter','latex')
xlabel('Time [sec]','FontName','times','FontSize',16,'Interpreter','latex')
```



Open-loop system analysis

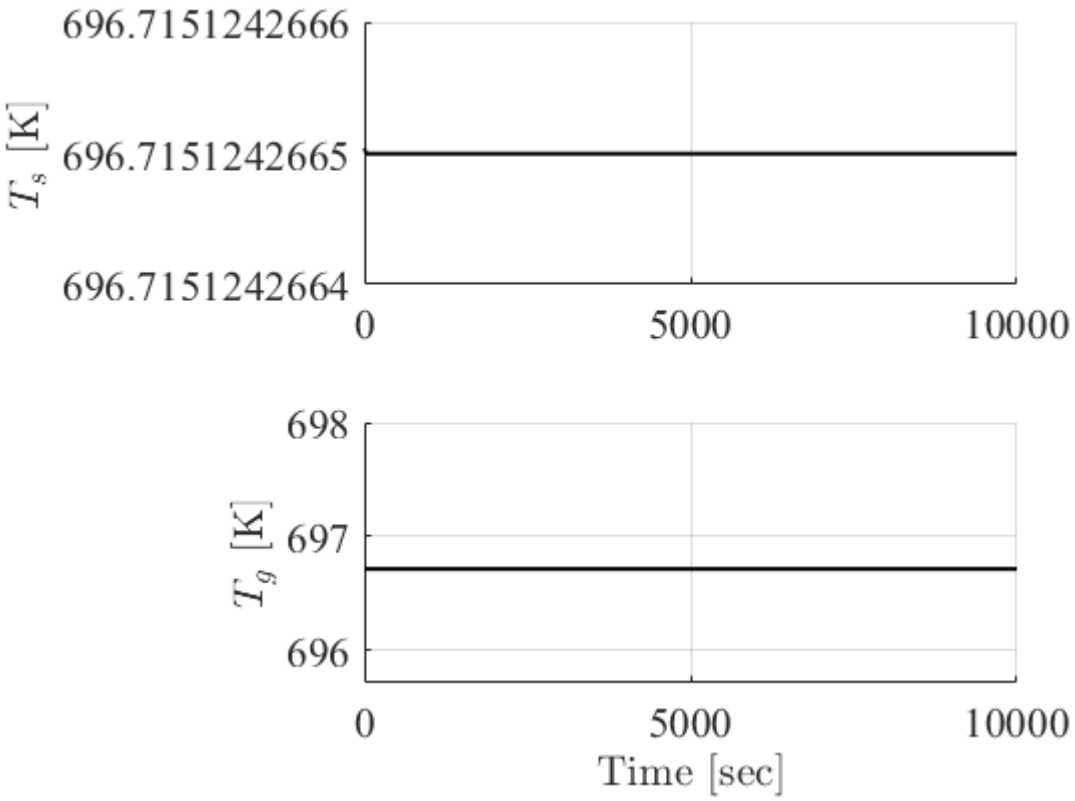
Problem 1 [2 points] Based on the system dynamics described in Eq. (1), determine the system state vector, input, disturbance and output. Exploiting the Simulink model HighTemperatureOven_SimulinkYYYYC compute the stationary state associated to the following initial conditions: $T_{a,0} = 21^{\circ}C$, $T_{g,0} = 10^{\circ}C$, $T_{s,0} = 220^{\circ}C$, $u_0 = 5V$.

```
% Your solution goes here
% Initial conditions for testing the Simulink model
Ta0 = 21; % ambient temperature [degree Celsius]
Ta0K = Ta0 + C2K; % ambient temperature [K]
Ts0 = 220; % oven surface temperature [degree Celsius]
Ts0K = Ts0 + C2K; % oven surface temperature [K]
Tg0 = 10; % product temperature [degree Celsius]
Tg0K = Tg0 + C2K; % product temperature [K]
u0 = 5; % voltage to power supply [V]

% Steady state calculation
x0 = [Ts0K Tg0K]';
U = [u0 Ta0K]';
y0 = x0(2);
[xss,Uss,yss,dx] = trim('HighTemperatureOven_Simulink2019b',x0,U,y0,[],[1 2],
[])
```

```
xss = 2×1
  696.7151
```

```
  696.7151
Uss = 2×1
    5.0000
  294.1500
yss = 696.7151
dx = 2×1
10^-12 ×
   -0.2274
         0
```

```
% Test to verify stationarity of the high temperature oven
SIM_TIME = 10000; % simulation time (can be changed as needed)
STEP_SIZE = 1; % integration step size of Simulink (can be changed as needed)
% Change with the name of the model you are using
SIMULINK_FILENAME = 'HighTemperatureOven_Simulink2019b';
time = (0:STEP_SIZE:SIM_TIME)';
u = u0*ones(length(time),1);
Ta = Ta0K*ones(length(time),1);
% Change the initial conditions of the integrators
Ts0K = xss(1);
Tg0K = xss(2);
% ------------
sim(SIMULINK_FILENAME,SIM_TIME,[],[time u Ta])

% Plot the temporal behaviour of the oven and product temperatures (RG)
Ts = logsout.getElement('Ts').Values.Data;
Tg = logsout.getElement('Tg').Values.Data;

figure, h1 = subplot(2,1,1); set(h1,'FontName','times','FontSize',16)
hold on, grid on
plot(time,Ts,'k','LineWidth',1.5)
ylabel('$T_s$ [K]','FontName','times','FontSize',16,'Interpreter','latex')
h2 = subplot(2,1,2); set(h2,'FontName','times','FontSize',16)
hold on, grid on
plot(time,Tg,'k','LineWidth',1.5)
ylabel('$T_g$ [K]','FontName','times','FontSize',16,'Interpreter','latex')
xlabel('Time [sec]','FontName','times','FontSize',16,'Interpreter','latex')
```



Then, linearize the nonlinear model around the computed stationary state.

```
% Your solution goes here
[A,B,C,D] = linmod('HighTemperatureOven_Simulink2019b',xss,Uss,yss);
Bv = B(:,2);
B = B(:,1);
Dv = D(:,2);
D = D(:,1);

disp('The open loop linearized system is given by')
```

```
The open loop linearized system is given by
```

```
A
```

```
A = 2×2
   -0.0395    0.0176
    0.0021   -0.0021
```

```
B
```

```
B = 2×1
    1.1765
         0
```

```
Bv
```

```
Bv = 2×1
    0.0103
```

```
          0
```

```
C
```

```
C = 1×2
     0     1
```

```
D
```

```
D = 0
```

```
Dv
```

```
Dv = 0
```

Problem 2 [2 points] Evaluate the internal and the external stability of system.

```
% Your solution goes here
% Eigenvalues of the open loop system
lambda_ol = eig(A)
```

```
lambda_ol = 2×1
   -0.0405
   -0.0012
```

```
disp('Both eigenvalues have negative real part, then the system is
internally asymptotically stable')
```

```
Both eigenvalues have negative real part, then the system is internally asymptotically stable
```

```
disp('Since the system is internally asymptotically stable, then the system
is also BIBO stable')
```

```
Since the system is internally asymptotically stable, then the system is also BIBO stable
```

```
hto_sys_ol = ss(A,B,C,D);
p_ol = pole(hto_sys_ol)
```

```
p_ol = 2×1
   -0.0405
   -0.0012
```

Problem 3 [2 points] Compute the system time constants, and determine the eigenmodes associated with the state response.

```
% Your solution goes here
% Time constants
tau_ol = -1./lambda_ol
```

```
tau_ol = 2×1
   24.7161
  865.0935
```

Control system design for temperature control

The main control objective is to regulate the temperature of the product at the desired reference temperature despite fluctuations in the environment temperature.

Closed-loop system requirements

Let $T_{g,ref}(t)$ be the desired reference for the temperature of the product. Then the following closed-loop requirements should be fulfilled

1. The product temperature T_g is regulated to its desired reference value $T_{g,ref}(t)$ with zero steady state error for step changes in the reference value as well as in the ambient temperature T_a;
2. For a step change of 25 degrees Celsius in $T_{g,ref}(t)$, the product temperature is regulated to the new reference value with a maximum overshoot of 2% and a settling time between 450 and 650 seconds;
3. The control effor u is kept in the range $[0, 10]\ V$

Problem 4 [6 points] Under the assumption that the state is fully accessible design a discrete time controller that meets the aforementioned specifications.

```
% Your solution goes here
% Check controllability
Mc = ctrb(A,B);
if rank(Mc) == size(A,1)
    disp('The open loop system is fully controllable')
end
```

```
The open loop system is fully controllable
```

```
% Discretize the continuous time system
tau_min = min(tau_ol); % fastest dynamics
t_set_min = 180; % smallest desired settling time [s]
Tsampl = min([tau_min/10 t_set_min/10]); % sampling time (ten times faster
than the fastest dynamics)
Tsampl = Tsampl - rem(Tsampl,STEP_SIZE)
```

```
Tsampl = 2
```

```
[F,G] = c2d(A,B,Tsampl)
```

```
F = 2×2
    0.9242    0.0339
    0.0041    0.9958
G = 2×1
    2.2625
    0.0049
```

```
[~,Gv] = c2d(A,Bv,Tsampl)
```

```
Gv = 2×1
    0.0199
    0.0000
```

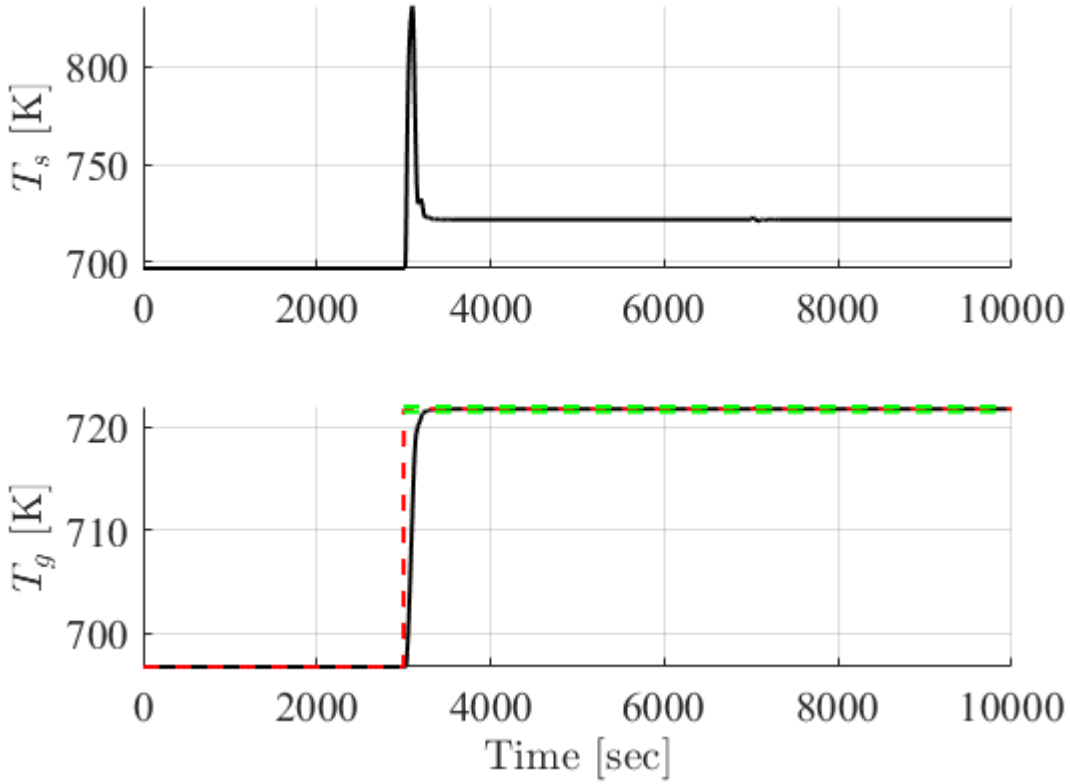
```
% Design desired eignevalues
t_settl_des = 180; % desired settling time
```

```
tau_des = t_settl_des/5;
lambda_des_ct = [lambda_ol(1) -1/tau_des -0.03];
lambda_des_dt = exp(lambda_des_ct*Tsampl);

% Add integral action to ensure perfect reference tracking and disturbance
% rejection
Fi = [F zeros(2,1);-Tsampl*C 1];
Gi = [G' 0]';
Gr = [zeros(2,1);Tsampl];

K = place(Fi,Gi,lambda_des_dt);
Kfs = K(1:2);
Ki = -K(3);
xi0 = 0;
```

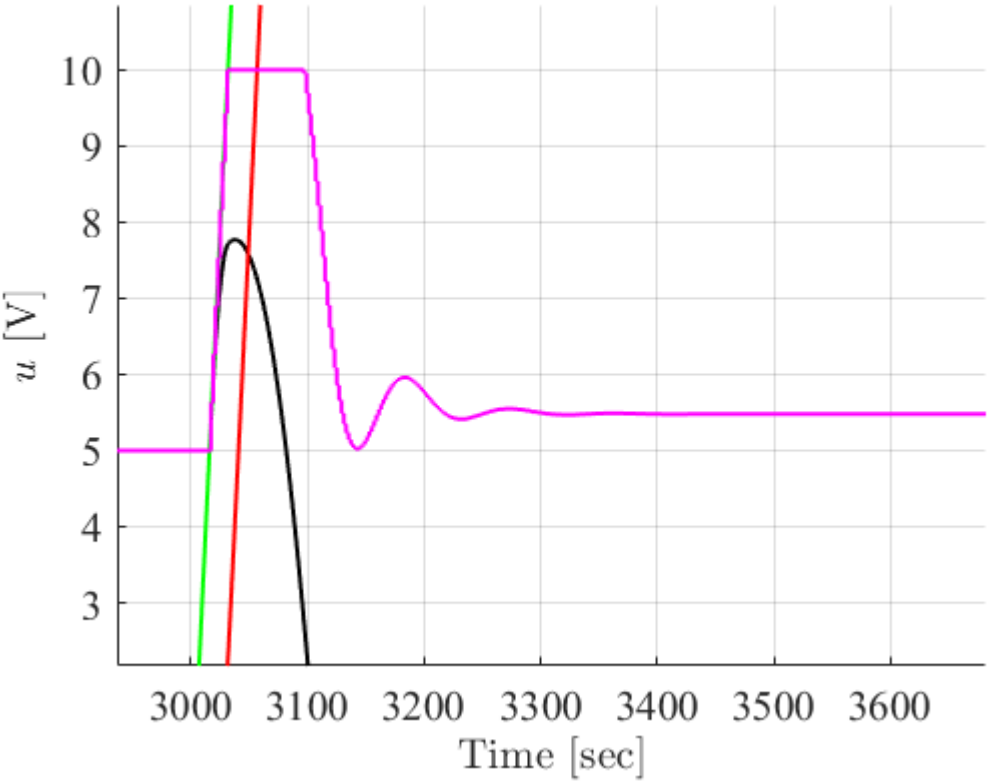
Problem 5 [2 points] Implement the discrete time controller in the Simulink model of the nonlinear system, and evalute the closed-loop system performance against the given requirements, when the sytem is subject to first a step change in reference temperature ($T_{g,ref}$ changes from $T_{g,ss}$ to $T_{g,ss} + 25^{\circ}C$, where $T_{g,ss}$ is the steady state product temperature computed in Problem 1) and then to a step change in ambient temperature (T_a changes from $T_{a,0}$ to $T_{a,0} + 10^{\circ}C$).

```
% Your solution goes here
SIM_TIME = 10000; % simulation time (can be changed as needed)
STEP_SIZE = 1; % integration step size of Simulink (can be changed as needed)
% Change with the name of the model you are using
SIMULINK_FILENAME = 'HighTemperatureOven_FullStateFeedback_Simulink2019b';
time = (0:STEP_SIZE:SIM_TIME)';
t_step_Tg = 3000;
step_Tg = 25;
TgfK = (xss(2) + step_Tg);
t_step_Ta = 7000;
TafK = (Ta0 + 10) + C2K;
% Change the initial conditions of the integrators
Ts0K = xss(1);
Tg0K = xss(2);
% ------------
sim(SIMULINK_FILENAME,SIM_TIME)

% Plot the temporal behaviour of the oven and product temperatures (RG)
Ts = logsout.getElement('Ts').Values.Data;
Tg = logsout.getElement('Tg').Values.Data;
timeDT = logsout.getElement('uc').Values.Time;
uc = logsout.getElement('uc').Values.Data;
ue = logsout.getElement('ue').Values.Data;
ui = logsout.getElement('u_i').Values.Data;
ufs = logsout.getElement('u_fs').Values.Data;
Tgref = logsout.getElement('Tg_ref').Values.Data;
```

```
figure, h1 = subplot(2,1,1); set(h1,'FontName','times','FontSize',16)
hold on, grid on
plot(time,Ts,'k','LineWidth',1.5)
ylabel('$T_s$ [K]','FontName','times','FontSize',16,'Interpreter','latex')
h2 = subplot(2,1,2); set(h2,'FontName','times','FontSize',16)
hold on, grid on
plot(time,Tg,'k',timeDT,Tgref,'--r','LineWidth',1.5)
line([time(t_step_Tg) time(end)],[Tgref(end)-0.01*step_Tg
Tgref(end)-0.01*step_Tg],'Color','g','LineStyle','--','LineWidth',1.5)
line([time(t_step_Tg) time(end)],[Tgref(end)+0.01*step_Tg Tgref(end)
+0.01*step_Tg],'Color','g','LineStyle','--','LineWidth',1.5)
ylabel('$T_g$ [K]','FontName','times','FontSize',16,'Interpreter','latex')
xlabel('Time [sec]','FontName','times','FontSize',16,'Interpreter','latex')
```




```
figure, h3 = axes; set(h3,'FontName','times','FontSize',16)
hold on, grid on
plot(timeDT,uc,'k',timeDT,ufs,'r',timeDT,ui,'g','LineWidth',1.5)
stairs(timeDT,ue,'LineWidth',1.5,'Color','m')
ylabel('$u$ [V]','FontName','times','FontSize',16,'Interpreter','latex')
xlabel('Time [sec]','FontName','times','FontSize',16,'Interpreter','latex')
```



State estimation for output feedback control

In order to implement an output feedback controller for the temperature control, the state vector must be estimated based on the available measurement of product temperature T_g. In the last part of the exercise, we assume the following measurement model

$$y_1 = T_g + v_1 \qquad (2)$$

where v_1 is zero mean white noise. The noise amplitude is $\pm 1.5^{\circ}C$.

Further, we assume that the ambient temperature T_a is given by the sum of two components

$$T_a = \bar{T}_a + T_{a,f}$$

where $\bar{T}_a$ is the mean ambient temperature and $T_{a,f}$ is the ambient temperature fluctuation, which is modeled as band-limited noise characterized by the following autocorrelation function

$$R_{T_a}(\tau) = \sigma_a^2 \, e^{-\beta_a |\tau|}$$

with $\sigma_a = 0.5^{\circ}C$ and $\beta_a = 0.008$.

Problem 6 [6 points] Under the assumption that the mean ambient temperature $\bar{T}_a$ is unknown, design a discrete time Kalman filter able to estimate the oven surface temperature T_s, the product temperature T_g, and the mean ambient temperature $\bar{T}_a$.

```
% Your solution goes here
beta_a = 0.008;
sigma_a = 0.5;
A_gm = -beta_a;
B_gm = sqrt(2*beta_a)*sigma_a;
A_kf = [A Bv Bv;zeros(1,4);zeros(1,3) A_gm];
B_kf = [B' 0 0]';
Bn_kf = [0 0;0 0;1 0;0 B_gm];
C_kf = [C 0 0];

[F_kf,G_kf] = c2d(A_kf,B_kf,Tsampl);

% Check observability
Mo = obsv(F_kf,C_kf);
if rank(Mo) == size(F_kf,1)
    disp('The system is fully observable')
else
    disp('The system is not fully observable')
end
```

```
The system is fully observable
```

```
% Process noise
sigma_nu = 0.5; % white noise driving the estimate of the mean ambient
temperature
sigma_e = 1; % white noise driving the Gauss-Markov process driving T_{a,f}
V1 = diag([sigma_nu^2 sigma_e^2]);
V1d = Bn_kf*V1*Bn_kf'*Tsampl;
Gn_kf = eye(4);
% Measurement noise
sigma_g = 1.5/3; % standard deviation of measurement noise
V2 = sigma_g^2;
V2d = V2/Tsampl;

% Design of the Kalman filter
[L_kf,P_kf,Qe_th,lambda_e] = dlqe(F_kf,Gn_kf,C_kf,V1d,V2d)
```

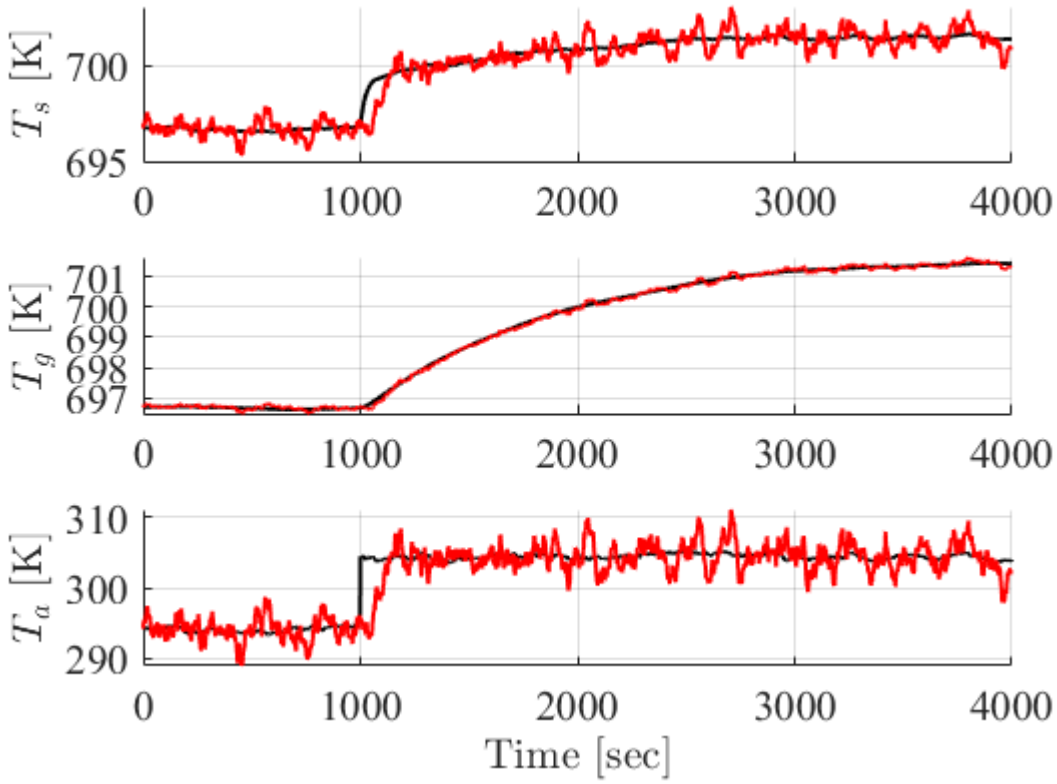
```
L_kf = 4×1
    0.4860
    0.0595
    1.9395
    0.0082
P_kf = 4×4
    0.8546    0.0646    3.8940    0.0205
    0.0646    0.0079    0.2578    0.0011
    3.8940    0.2578   20.8820   -0.1318
    0.0205    0.0011   -0.1318    0.2537
```

```
Qe_th = 4×4
    0.8232    0.0608    3.7687    0.0200
    0.0608    0.0074    0.2424    0.0010
    3.7687    0.2424   20.3820   -0.1339
    0.0200    0.0010   -0.1339    0.2537
lambda_e = 4×1 complex
   0.9843 + 0.0000i
   0.9698 + 0.0337i
   0.9698 - 0.0337i
   0.9188 + 0.0000i
```

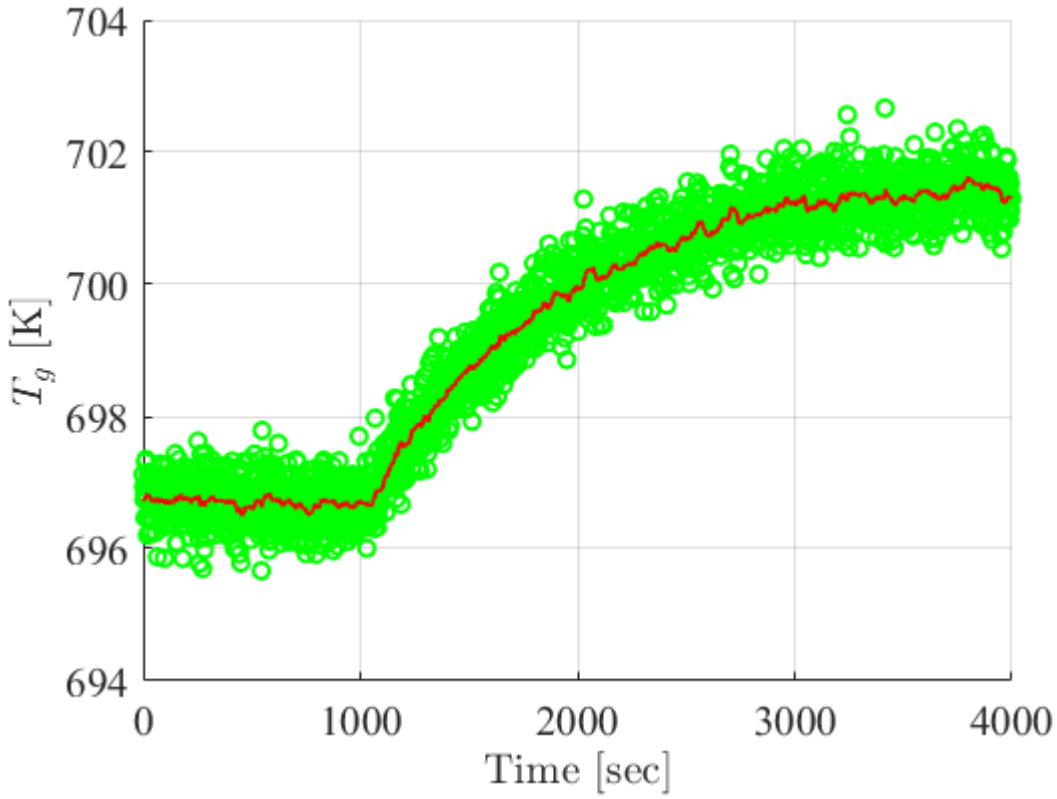
Problem 7 [4 points] Implement the discrete time Kalman filter in the Simulink diagram of the open-loop system and assess its estimation performance when the open-loop system is subject to a step change in mean ambient temperature ($\bar{T}_a$ changes from $\bar{T}_a = T_{a,0}$ to $\bar{T}_a = T_{a,0} + 10^{\circ}C$)

```
% Your solution goes here
x0hat = zeros(4,1);

SIMULINK_FILENAME = 'HighTemperatureOven_KalmanFilter_Simulink2019b';
SIM_TIME = 4000;
time = (0:STEP_SIZE:SIM_TIME)';
u = u0*ones(length(time),1);
t_step_Ta = 1000;
TafK = (Ta0 + 10) + C2K;
% Change the initial conditions of the integrators
Ts0K = xss(1);
Tg0K = xss(2);
% ------------
sim(SIMULINK_FILENAME,SIM_TIME,[],[time u])

% Plot the temporal behaviour of the oven and product temperatures (RG)
Ts = logsout.getElement('Ts').Values.Data;
Tg = logsout.getElement('Tg').Values.Data;
timeDT = logsout.getElement('xhat').Values.Time;
Tshat = logsout.getElement('xhat').Values.Data(:,1);
Tghat = logsout.getElement('xhat').Values.Data(:,2);
Tahat = logsout.getElement('xhat').Values.Data(:,3);
yhat = logsout.getElement('yhat').Values.Data;
Tgmeas = logsout.getElement('Tg_meas').Values.Data;
Ta = logsout.getElement('Ta').Values.Data;

figure, h1 = subplot(3,1,1); set(h1,'FontName','times','FontSize',16)
hold on, grid on
plot(time,Ts,'k',timeDT,Tshat+xss(1),'r','LineWidth',1.5)
ylabel('$T_s$ [K]','FontName','times','FontSize',16,'Interpreter','latex')
h2 = subplot(3,1,2); set(h2,'FontName','times','FontSize',16)
hold on, grid on
plot(time,Tg,'k',timeDT,Tghat+xss(2),'r','LineWidth',1.5)
ylabel('$T_g$ [K]','FontName','times','FontSize',16,'Interpreter','latex')
h3 = subplot(3,1,3); set(h3,'FontName','times','FontSize',16)
hold on, grid on
```

```
plot(time,Ta,'k',timeDT,Tahat+Ta0K,'r','LineWidth',1.5)
ylabel('$T_a$ [K]','FontName','times','FontSize',16,'Interpreter','latex')
xlabel('Time [sec]','FontName','times','FontSize',16,'Interpreter','latex')
```




```
figure, h4 = axes; set(h4,'FontName','times','FontSize',16)
hold on, grid on
plot(time,Tg,'k',timeDT,Tgmeas,'og',timeDT,yhat+yss,'r','LineWidth',1.5)
ylabel('$T_g$ [K]','FontName','times','FontSize',16,'Interpreter','latex')
xlabel('Time [sec]','FontName','times','FontSize',16,'Interpreter','latex')
```


704
702
700
698
696
694
T_g [K]
0
1000
2000
3000
4000
Time [sec]

Linear Control Design 2 - Fall 2021 - Exam Part I

Introduction

The Part I of the Exam in Linear Control Design 2 (E21) consists of numerical exercises testing the acquisition of competences in the areas of analysis and design of control systems using modern control theory based on state space representation of system dynamics.

All exercises are equally weighted towards the overall assessment of the examination.

It is the sole responsability of the student to guarantee that the solution delivered for evaluation can be run by the examiner without the need of contacting the student. All dependencies on files external to this Matlab Live Script must be checked and included in the final delivery. **If the examiner will not be able to execute the Matlab Live Script delivered as solution by the student, the Part I will be considered failed.**

```
% Fill in your information
Exam = 'LCD2 E21'
```

```
Exam =
'LCD2 E21'
```

```
Student_Name = 'Student Name'
```

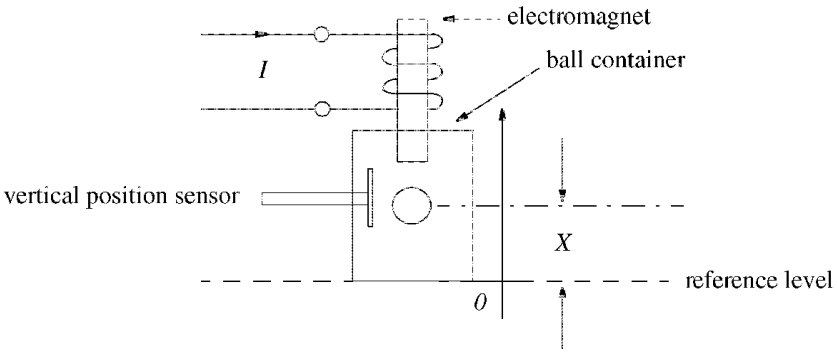
```
Student_Name =
'Student Name'
```

```
Student_Number = 'Student Number'
```

```
Student_Number =
'Student Number'
```

Output feedback control of a test mass accelerometer

A test mass accelerometer is a measuring system utilized to measured vertical accelerations in e.g. vehicles. The figure below shows a simplified schematic of such device.



The test mass is a steel ball with a mass $m = 10\,\mathrm{g}$. The ball is held floating in a magnetic field by the electromagnet above it. The vertical position of the ball is measured by a position sensor.

The electromagnetic force that keeps the ball floating is given by

$$F(X,I) = 1.805 \times 10^3 X^2 + 14.44\,XI - 6.498\,X + 0.02888\,I^2 + 0.374\,I - 0.1742$$

where X is the ball position (in meters) in relation to the reference point and I is the current (in Amperes).

The ball is also influenced by the air resistance. The force that air resistance exerts on the ball is

$$F(V) = -V(c_1 + c_2|V|)$$

where $c_1 = 1.55 \times 10^{-6}\mathrm{N}(m/s)$, $c_2 = 2.2 \times 10^{-4}\mathrm{N(m/s)}^2$ and the ball's velocity $V$is messured in m/s.

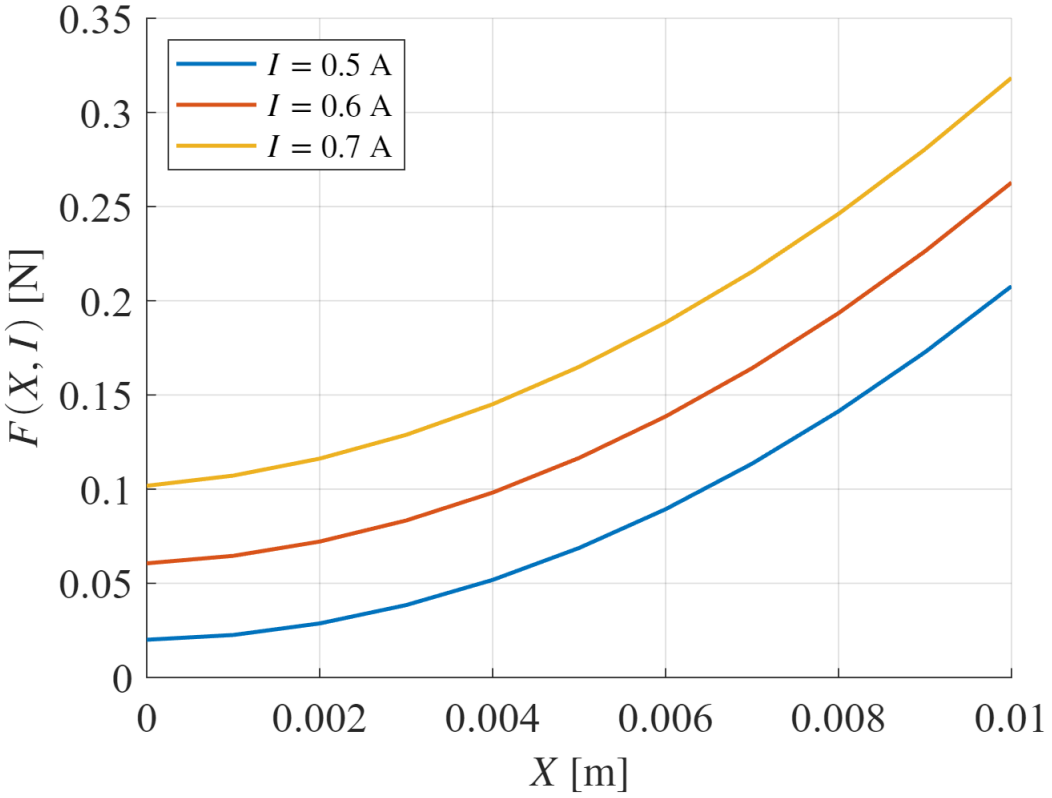
The accelerometer is under the influence of an external vertical acceleration A. The nonlinear model describing the motion of the ball reads

$$\frac{d^2X}{dt^2} = \frac{1}{m}F(V) + \frac{1}{m}F(X,I) + A - g$$

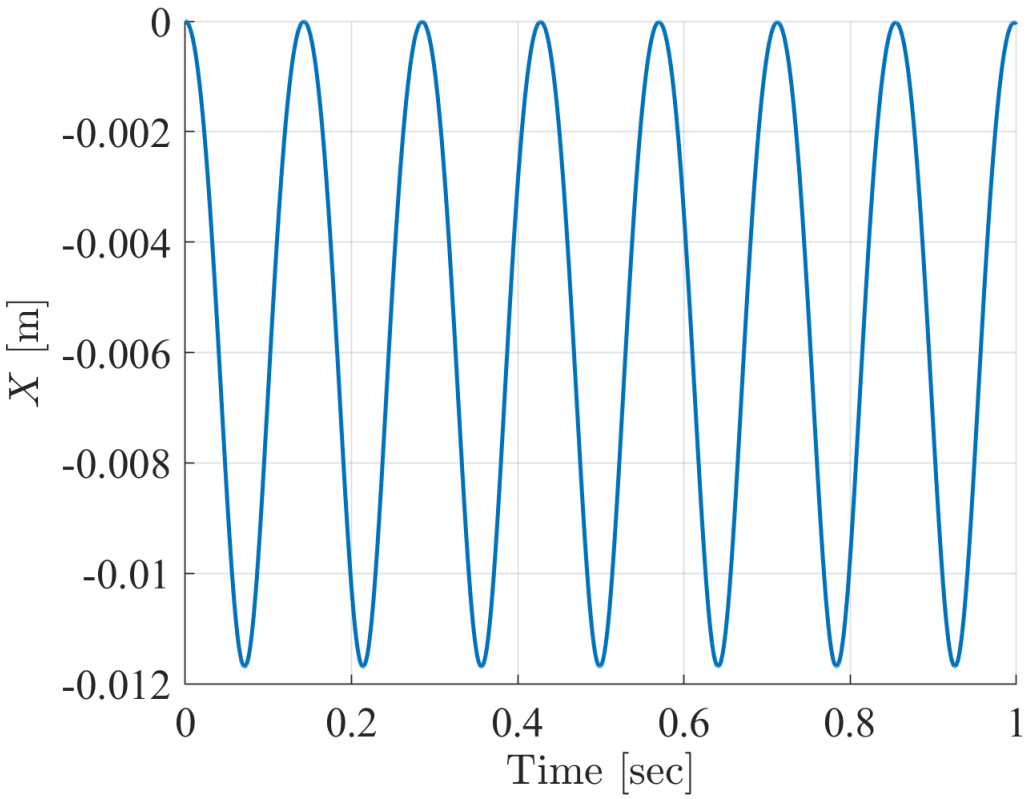
where $g = 9.81\,\mathrm{m/s}^2$ is the acceleration of gravity.

```
% Simulator parameters (RG)
SIM_TIME = 1;
STEP_SIZE = 0.0001;
% Change with the name of the model you are using
SIMULINK_FILENAME = 'TestMassAccelerometer_NonlinearModel_Simulink2020b';

% Model paramters (RG)
m = 0.01; % mass in kilograms
c1 = 1.55*10^-6; % N*m/s
c2 = 2.2*10^-4; % N*m^2/s^2
% Coefficients of the electromagnetic force
f1 = 1.805*10^3;
f2 = 14.44;
f3 = 6.498;
f4 = 0.02888;
f5 = 0.3740;
f6 = 0.1742;
% ------------------------------------------
g = 9.81; % m/s^2
A = 0; % m/s^2

% Electromagnetic force for different values of control current I (RG)
X = (0:0.001:0.01)'; % mass displacement range [m]
I0 = [0.5 0.6 0.7]; % control current [A]
for ii = 1:length(I0)
    F_XI(:,ii) = f1*X.^2 + f2*X.*I0(ii) - f3*X + f4*I0(ii).^2 + f5*I0(ii) -
f6;
end

figure, h1 = axes; set(h1,'FontName','times','FontSize',16)
hold on, grid on
plot(X,F_XI,'LineWidth',1.5)
```

```
ylabel('$F(X,I)$ [N]','FontName','times','FontSize',16,'Interpreter','latex')
xlabel('$X$ [m]','FontName','times','FontSize',16,'Interpreter','latex')
leg = legend('$I = 0.5$ A','$I = 0.6$ A','$I = 0.7$
A','Location','NorthWest');
set(leg,'FontName','times','FontSize',12,'Interpreter','latex')
```




```
% Test the provided nonlinear model (RG)
time = (0:STEP_SIZE:SIM_TIME)';
x0 = zeros(2,1); % initial condition x0 = [X,V]'
I0 = 0.5; % test current through the electromagnet [A]
I = I0*ones(length(time),1);
sim(SIMULINK_FILENAME,SIM_TIME,[],[time I])

% Plot simulation result
t = logsout.getElement(1).Values.Time;
X = logsout.getElement(1).Values.Data;
V = logsout.getElement(2).Values.Data;

figure, h2 = axes; set(h2,'FontName','times','FontSize',16)
hold on, grid on
plot(t,X,'LineWidth',1.5)
ylabel('$X$ [m]','FontName','times','FontSize',16,'Interpreter','latex')
xlabel('Time [sec]','FontName','times','FontSize',16,'Interpreter','latex')
```



Operating point and linearization

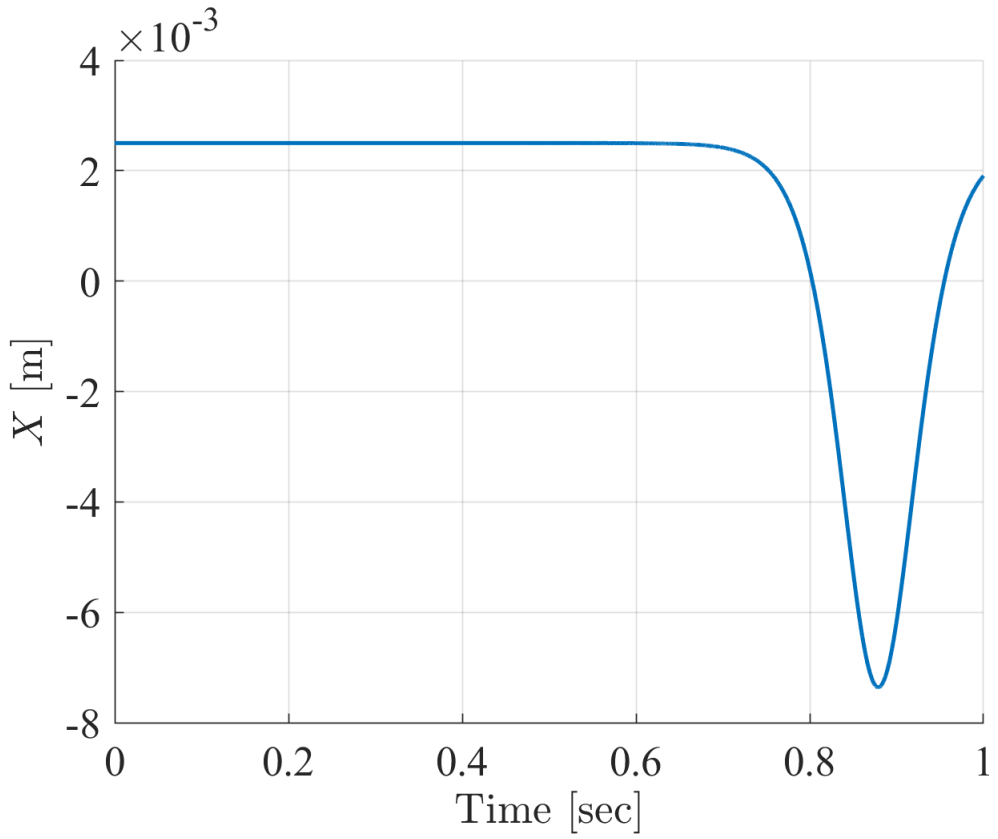
Problem 1 Determine the operating point associated with a nominal verical displacement of the mass $X_n = 2.5\text{mm}$ when the external vertical acceleration $A = 0$. Linearize the system around the operatig point.

```
% Your solution goes here
x0 = [0.0025 0]';
y0 = 0.0025;
[x,u,y,dx] = trim(SIMULINK_FILENAME,x0,[],y0,[1 1],[],1)
```

```
x = 2×1
    0.0025
   -0.0000
u = 0.6466
y = 0.0025
dx = 2×1
10^-11 ×
   -0.0000
   -0.6674
```

```
Xss = x(1);
Vss = x(2);
Iss = u;
I = Iss*ones(length(time),1);
sim(SIMULINK_FILENAME,SIM_TIME,[],[time I])
```

```
% Plot simulation result
t = logsout.getElement(1).Values.Time;
X = logsout.getElement(1).Values.Data;
V = logsout.getElement(2).Values.Data;

figure, h3 = axes; set(h3,'FontName','times','FontSize',16)
hold on, grid on
plot(t,X,'LineWidth',1.5)
ylabel('$X$ [m]','FontName','times','FontSize',16,'Interpreter','latex')
xlabel('Time [sec]','FontName','times','FontSize',16,'Interpreter','latex')
```




```
% Linearization of the nonlinear system
[Aol,Bol,Col,Dol] = linmod(SIMULINK_FILENAME,x,u)
```

```
Aol = 2×2
10^3 ×
         0    0.0010
    1.1865   -0.0000
Bol = 2×1
         0
   44.7450
Col = 1×2
     1     0
Dol = 0
```

```
Bdol = [0 1]'; % input matrix for the "disturbance" acceleration
```

Stability analysis

Problem 2 Assess the internal and external stability of the linear system.

```
% Your solution goes here

% Internal stability
lambda_ol = eig(Aol)
```

```
lambda_ol = 2×1
   34.4448
  -34.4450
```

```
disp('The open loop system is unstable because one eigenvalue is positive')
```

```
The open loop system is unstable because one eigenvalue is positive
```

```

% External stability
[numG_ix,denG_ix] = ss2tf(Aol,Bol,Col,Dol);
G_ix = tf(numG_ix,denG_ix)
```

```
G_ix =

          44.75
  ------------------------
  s^2 + 0.0001552 s - 1186

Continuous-time transfer function.
```

```
poles_ol = pole(G_ix)
```

```
poles_ol = 2×1
  -34.4450
   34.4448
```

```
disp('The open loop system is not BIBO stable because one pole is positive')
```

```
The open loop system is not BIBO stable because one pole is positive
```

Control system design for measurement of external acceleration

The test mass accelerometer in opern loop is not capable of providing a measurement of the external vertical acceleration A. In order to enable the measuring device to provide such measurement a control system need to be designed.

Closed-loop system requirements

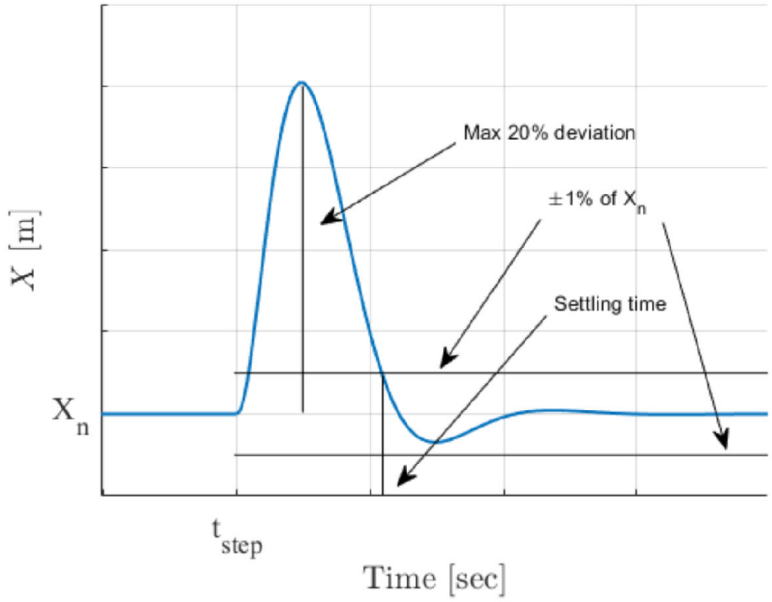
1) The position X of the test mass is regulated to its nominal value X_n despite step changes in the external vertical acceleration A.

2) For a step change in the external vertical acceleration A equal to $0.25g$ at a time $t = t_{step}$ the position X is regulated back to its nominal value X_n with

a) a maximum 20% deviation from its steady state value during the transient part of the response

b) settling time between 50 and 75 milliseconds

Please refer to the following figure for a clear definition of the quantities described in the requirements. **IMPORTANT** The response shown in the figure (blue curve) serves only to display quantities of interest for the design requirements; therefore it does not necessarily represent the expected system behaviour.



Problem 3 Under the assumption that the state is fully accessible **design a discrete time optimal controller** that meets the aforementioned specifications on the nonlinear system.

```
% Your solution goes here

% Discretize the open loop system
Ts = 0.005; % approximately 10 times smaller than the desired settling time
[Fol,Gol] = c2d(Aol,Bol,Ts)
```

```
Fol = 2×2
    1.0149    0.0050
    5.9616    1.0149
Gol = 2×1
    0.0006
    0.2248
```

```
[~,Gdol] = c2d(Aol,Bdol,Ts)
```

```
Gdol = 2×1
    0.0000
    0.0050
```

```
% Compute the reachability matrix and check if the system is reachable
Mr = ctrb(Fol,Gol);
if rank(Mr) == size(Fol,1)
    disp('The open loop system is reachable')
end
```

```
The open loop system is reachable
```

```
% Augment the system with the integral state in order to achieve disturbance
rejection
% against constant disturbances (Requirement 1)
Fol_int = [Fol zeros(2,1);-Ts*Col 1];
Gol_int = [Gol' 0]';
Col_int = [Col 0];

% Design the LQR to meet Requirements 2a and 2b (other values of R1 and R2
could provide
% similar performance within the given specifications)
R1 = diag([10 0.001 400000]);
R2 = 0.005;
K_lqr = dlqr(Fol_int,Gol_int,R1*Ts,R2*Ts);
K_lqr_fs = K_lqr(1:2)
```

```
K_lqr_fs = 1×2
  231.6239    2.9628
```

```
K_lqr_int = -K_lqr(3)
```

```
K_lqr_int = 5.9797e+03
```

```
x0_int = 0;

% Closed-loop linear system
Fcl = Fol_int - Gol_int*K_lqr;
% Discrete time eigenvalues
lambda_cl_dt = eig(Fcl)
```

```
lambda_cl_dt = 3×1 complex
   0.7789 + 0.2336i
   0.7789 - 0.2336i
   0.6760 + 0.0000i
```

```
% Equivalent continuous time eigenvalues
lambda_cl_ct = 1/Ts*log(lambda_cl_dt);
disp('The fastest closed-loop dynamics is')
```

```
The fastest closed-loop dynamics is
```

```
disp(strcat(num2str(max(abs(lambda_cl_ct))),'rad/s'))
```

```
78.3252rad/s
```

```
disp('The damping ratio is')
```

```
The damping ratio is
```

```
zeta_cl = -real(lambda_cl_ct(2))/abs(lambda_cl_ct(2));
disp(num2str(zeta_cl))
```

```
0.57879
```

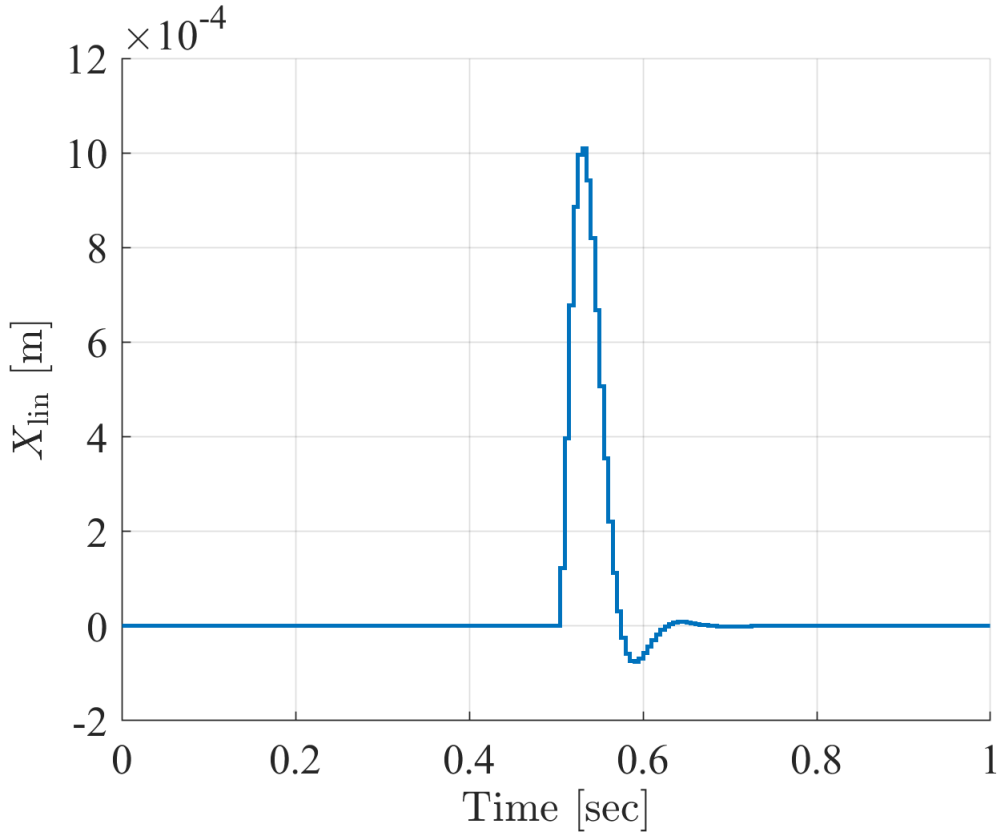
```
% Check if requirements are fulfilled on the linear system
Br = [0 0 Ts]'; % reference input matrix
Gcl = [Br [Gdol;0]];
sys_cl_dt = ss(Fcl,Gcl,Col_int,[],Ts);
tlin = (0:Ts:1)';
ulin = [zeros(length(tlin),1) [zeros((length(tlin)-1)/
2,1);g*ones((length(tlin)-1)/2+1,1)]];
x0lin = zeros(3,1);
[ylin,tlin,xlin] = lsim(sys_cl_dt,ulin,tlin,x0lin);

overshoot_lin = 100*(Xss - min(Xss+xlin(:,1)))/Xss;
disp('The overshoot of the linear response is')
```

```
The overshoot of the linear response is
```

```
disp(strcat(num2str(overshoot_lin),'%'))
```

```
3.0403%
```

```
figure, h4 = axes; set(h4,'FontName','times','FontSize',16)
hold on, grid on
stairs(tlin,ylin,'LineWidth',1.5)
ylabel('$X_{\mathrm{lin}}$
[m]','FontName','times','FontSize',16,'Interpreter','latex')
xlabel('Time [sec]','FontName','times','FontSize',16,'Interpreter','latex')
```



Problem 4 Implement the discrete time optimal controller in the Simulink model of the nonlinear system, and evalute the closed-loop system performance against the given requirements, when the sytem is subject to the step change in the external vertical acceleration A of $0.25g$.

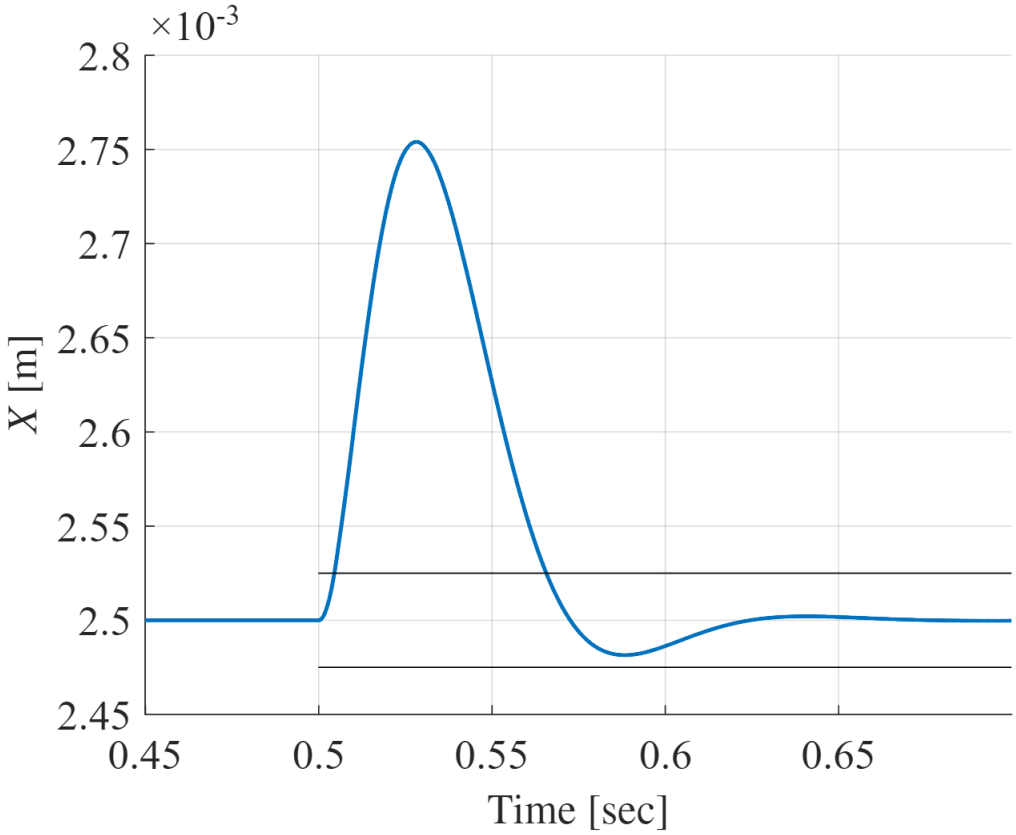
```
% Check if the requirements are fulfilled
SIMULINK_FILENAME = 'TestMassAccelerometer_NonlinearModel_LQR_Simulink2017';

X_ref = Xss;
STEP_TIME = 0.5;
A0 = 0;
A1 = 0.25*g;
SIM_TIME = 1;
STEP_SIZE = 0.0001;
sim(SIMULINK_FILENAME,SIM_TIME)

% Plot simulation result
timeCT = logsout.getElement(3).Values.Time;
X = logsout.getElement(3).Values.Data;
V = logsout.getElement(4).Values.Data;
timeDT = logsout.getElement(2).Values.Time;
int_ctrl = logsout.getElement(2).Values.Data;
fs_ctrl = -logsout.getElement(1).Values.Data;
ctrl_inp = logsout.getElement(7).Values.Data;

settling_p1pct = Xss*1.01;
settling_m1pct = Xss*0.99;
overshoot = 100*(max(X)- Xss)/Xss;
disp('The overshoot is')
```

```
The overshoot is
```

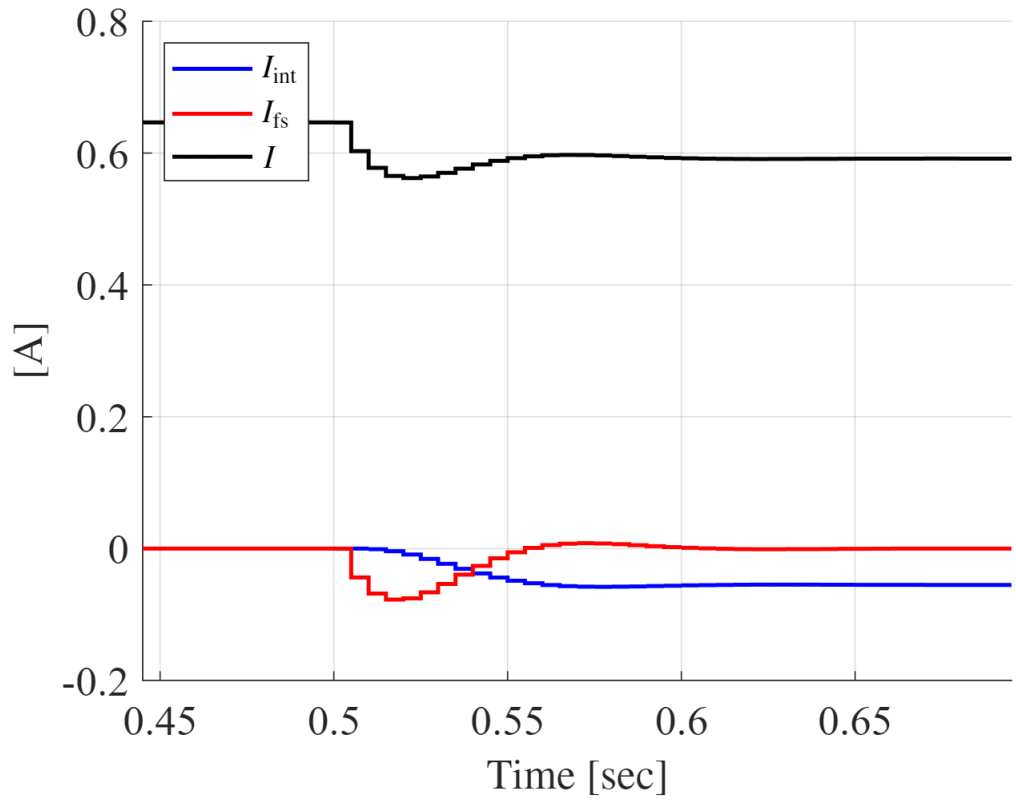
```
disp(strcat(num2str(overshoot),'%'))
```

```
10.1613%
```

```


figure, h5 = axes; set(h5,'FontName','times','FontSize',16)
hold on, grid on
plot(timeCT,X,'LineWidth',1.5)
line([timeCT(0.5/STEP_SIZE) timeCT(round(0.7/STEP_SIZE))],[settling_p1pct
settling_p1pct],'Color','k')
line([timeCT(0.5/STEP_SIZE) timeCT(round(0.7/STEP_SIZE))],[settling_m1pct
settling_m1pct],'Color','k')
xlim([timeCT(0.45/STEP_SIZE) timeCT(round(0.7/STEP_SIZE))])
ylabel('$X$ [m]','FontName','times','FontSize',16,'Interpreter','latex')
xlabel('Time [sec]','FontName','times','FontSize',16,'Interpreter','latex')
```




```
figure, h6 = axes; set(h6,'FontName','times','FontSize',16)
hold on, grid on
stairs(timeDT,int_ctrl,'b','LineWidth',1.5)
stairs(timeDT,fs_ctrl,'r','LineWidth',1.5)
stairs(timeDT,ctrl_inp,'k','LineWidth',1.5)
xlim([timeDT(0.45/Ts) timeDT(0.7/Ts)])
ylabel('[A]','FontName','times','FontSize',16,'Interpreter','latex')
xlabel('Time [sec]','FontName','times','FontSize',16,'Interpreter','latex')
leg2 = legend('$I_{\mathrm{int}}$','$I_{\mathrm{fs}}
$','$I$','Location','NorthWest');
set(leg2,'FontName','times','FontSize',12,'Interpreter','latex')
```



Observer design for estimation of the external acceleration

In order to determine the unknown acceleration A acting on the measuring device a state estimator in the form of a Kalman filter can be employed.

The position measurement is affected by white noise w with zero mean and noise intensity $V_2 = 1 \times 10^{-10}$, i.e.

$$y = x + w$$

Problem 5 Under the assumption that the exteral vertical acceleration is constant, **design a discrete time Kalman filter** able to reconstruct the state vector $x = [X, V]^{\mathrm{T}}$ and the unknown constant acceleration A.

```
% Your solution goes here

% Model for the design of the Kalman filter where the system model is
augmented with an
% additional state modeling the constant disturbance A
F_kf = [Fol Gdol;0 0 1];
G_kf = [Gol' 0]';
Bv_kf = [0 0 1]'; % continuous time process noise input matrix
Gv_kf = eye(3); % discrete time process noise input matrix
C_kf = [Col 0];
V1 = 50; % intensity of the process noise driving the equation for the
estimate of A
V1d = Bv_kf*V1*Bv_kf'*Ts;
```

```
V2 = 1*10^(-10); % intensity of the measurement noise
V2d = V2/Ts;

% Check observability
Mo = obsv(F_kf,C_kf);
if rank(Mo) == size(F_kf,1)
    disp('The open loop system is observable')
end
```

```
The open loop system is observable
```

```
% Design of the discrete time closed form Kalman filter
[L_kf,P,Qe_th,lambda_kf_dt] = dlqe(F_kf,Gv_kf,C_kf,V1d,V2d)
```

```
L_kf = 3×1
10^3 ×
    0.0006
    0.0559
    2.2150
P = 3×3
    0.0000    0.0000    0.0001
    0.0000    0.0003    0.0169
    0.0001    0.0169    1.4226
Qe_th = 3×3
    0.0000    0.0000    0.0000
    0.0000    0.0002    0.0106
    0.0000    0.0106    1.1726
lambda_kf_dt = 3×1 complex
   0.7389 + 0.2837i
   0.7389 - 0.2837i
   0.6265 + 0.0000i
```

```
% Equivalent continuous time eigenvalues of the estimation error dynamics
lambda_kf_ct = 1/Ts*log(lambda_kf_dt);
disp('The slowest estimation error dynamics is')
```

```
The slowest estimation error dynamics is
```

```
disp(strcat(num2str(min(abs(lambda_kf_ct))),'rad/s'))
```

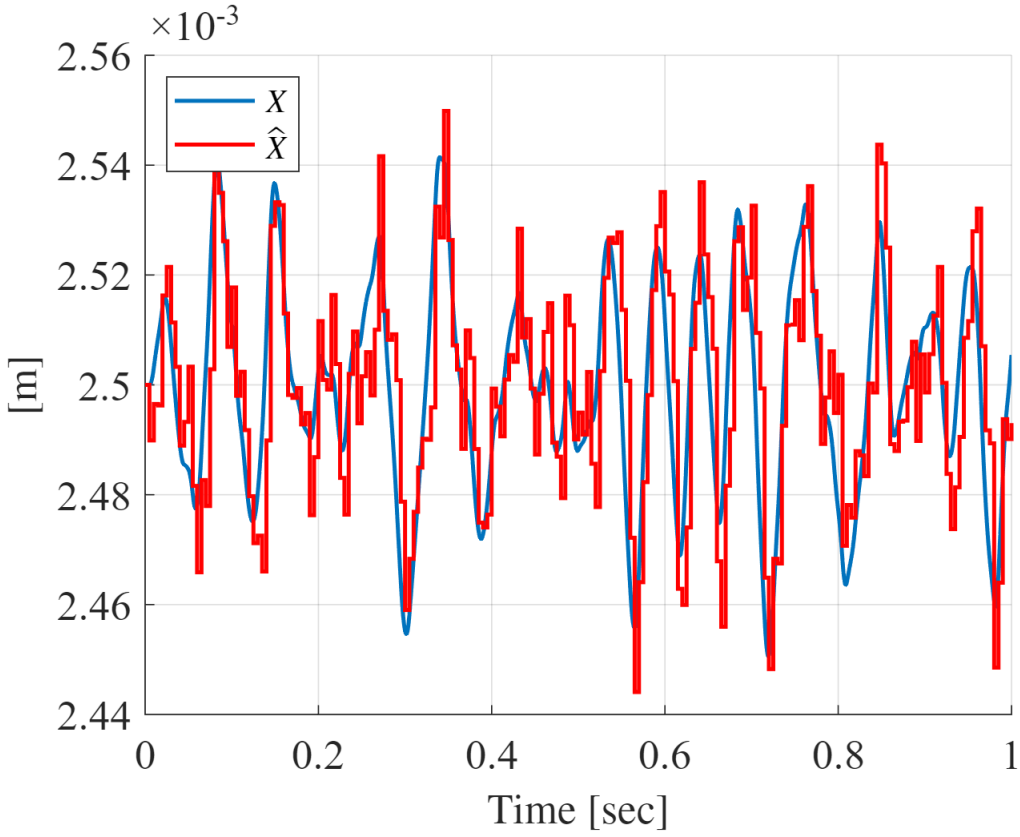
```
86.9539rad/s
```

```
if min(abs(lambda_kf_ct)) > max(abs(lambda_cl_ct))
    disp('Observer dynamics faster than controller dynamics --> OK!')
else
    disp('Observer dynamics slower than controller dynamics --> NOT OK!')
end
```

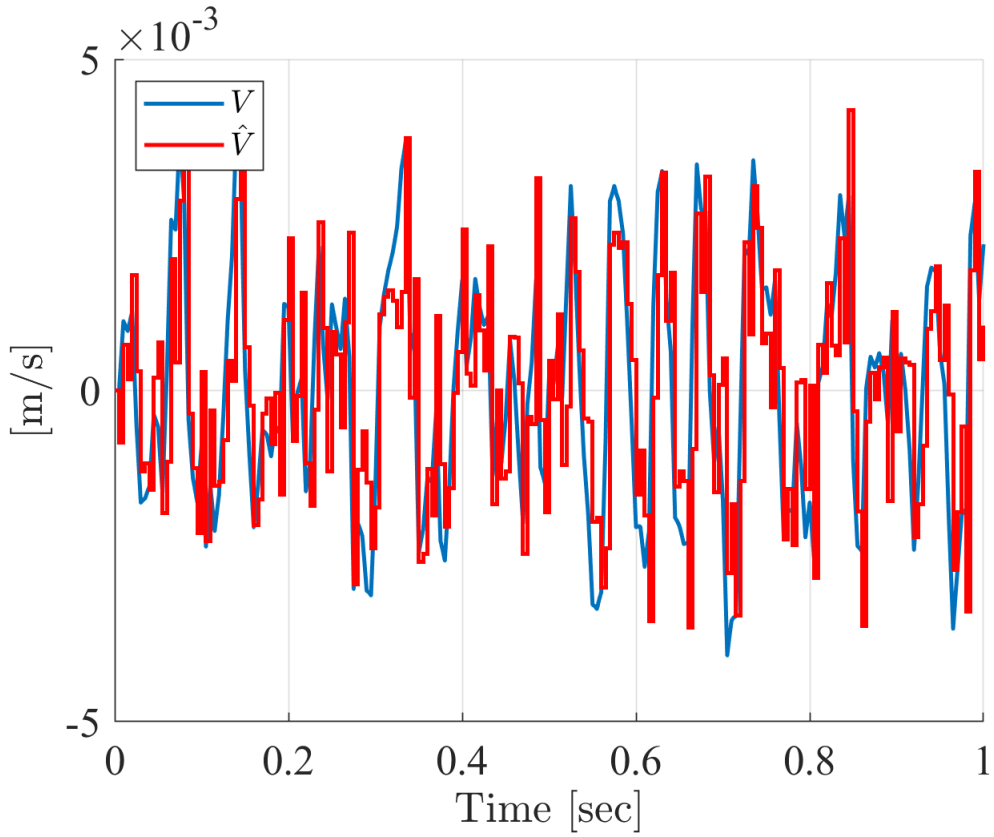
```
Observer dynamics faster than controller dynamics --> OK!
```

Problem 6 Implement the discrete time Kalman filter in the Simulink diagram of the closed-loop system, and assess the estimation performance of the Kalman filter when tested on the nonlinear system together with the control system designed in Problem 3, both in stationary conditions and in the presence of the step change in the external vertical acceleration A of $0.25g$.

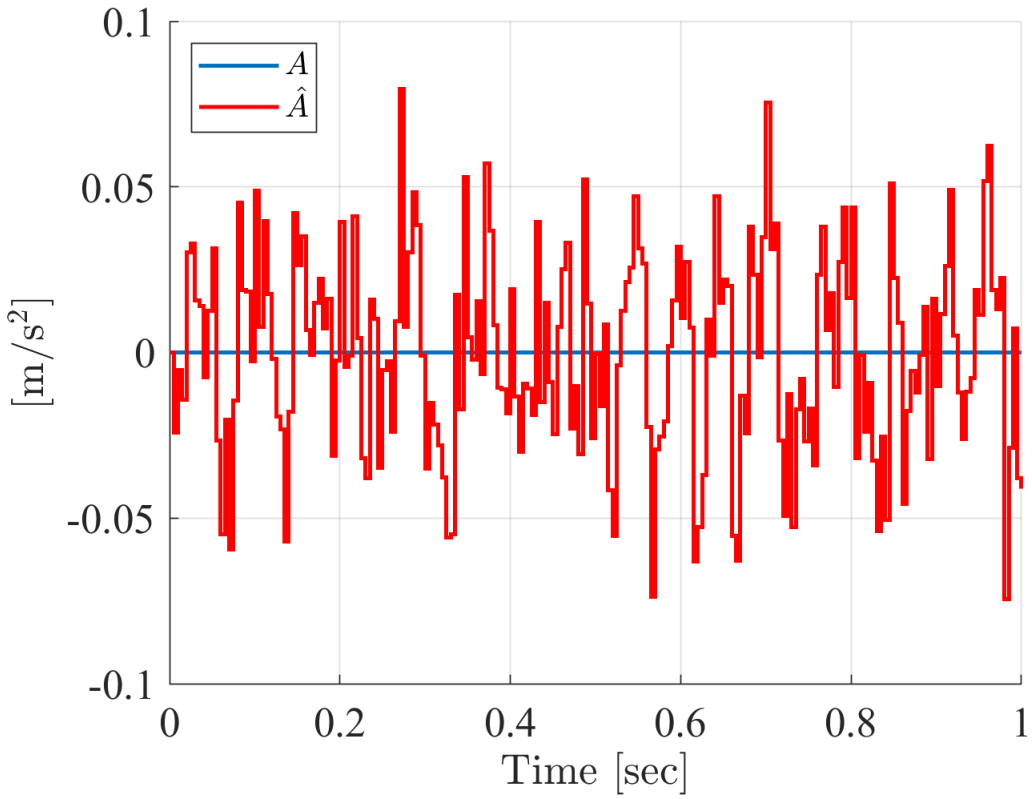
```
% Your solution goes here

SIMULINK_FILENAME =
'TestMassAccelerometer_NonlinearModel_LQR_KF_Simulink2017';

% Simulate steady state
X_ref = Xss;
x0_hat = zeros(3,1);
STEP_TIME = 0.5;
A0 = 0;
A1 = 0;
SIM_TIME = 1;
STEP_SIZE = 0.001;
sim(SIMULINK_FILENAME,SIM_TIME)

% Plot simulation result
timeCT = logsout.getElement(3).Values.Time;
X = logsout.getElement(3).Values.Data;
V = logsout.getElement(4).Values.Data;
A = logsout.getElement(10).Values.Data;
timeDT = logsout.getElement(2).Values.Time;
int_ctrl = logsout.getElement(2).Values.Data;
fs_ctrl = -logsout.getElement(1).Values.Data;
ctrl_inp = logsout.getElement(9).Values.Data;
X_hat = logsout.getElement(8).Values.Data(:,1)+Xss;
V_hat = logsout.getElement(8).Values.Data(:,2);
A_hat = logsout.getElement(8).Values.Data(:,3);
inno = logsout.getElement(6).Values.Data;

figure, h7 = axes; set(h7,'FontName','times','FontSize',16)
hold on, grid on
plot(timeCT,X,'LineWidth',1.5)
stairs(timeDT,X_hat,'r','LineWidth',1.5)
ylabel('[m]','FontName','times','FontSize',16,'Interpreter','latex')
xlabel('Time [sec]','FontName','times','FontSize',16,'Interpreter','latex')
leg3 = legend('$X$','$\hat{X}$','Location','NorthWest');
set(leg3,'FontName','times','FontSize',12,'Interpreter','latex')
```



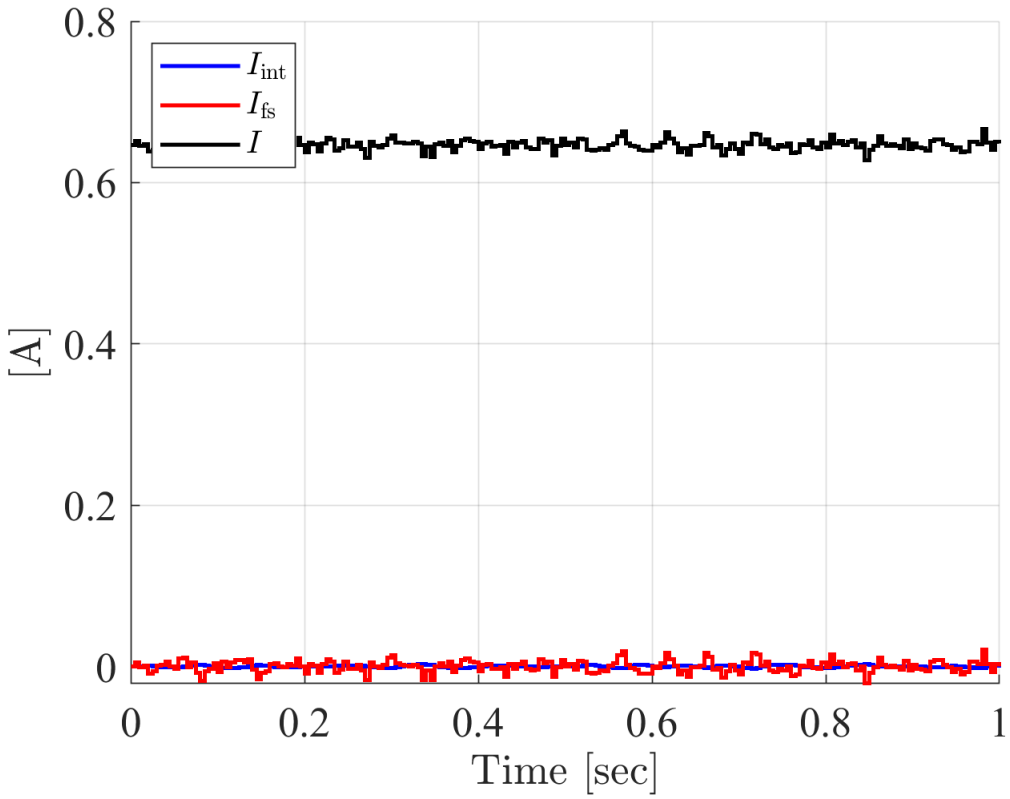

```
figure, h8 = axes; set(h8,'FontName','times','FontSize',16)
hold on, grid on
plot(timeCT,V,'LineWidth',1.5)
stairs(timeDT,V_hat,'r','LineWidth',1.5)
ylabel('[m/s]','FontName','times','FontSize',16,'Interpreter','latex')
xlabel('Time [sec]','FontName','times','FontSize',16,'Interpreter','latex')
leg4 = legend('$V$','$\hat{V}$','Location','NorthWest');
set(leg4,'FontName','times','FontSize',12,'Interpreter','latex')
```



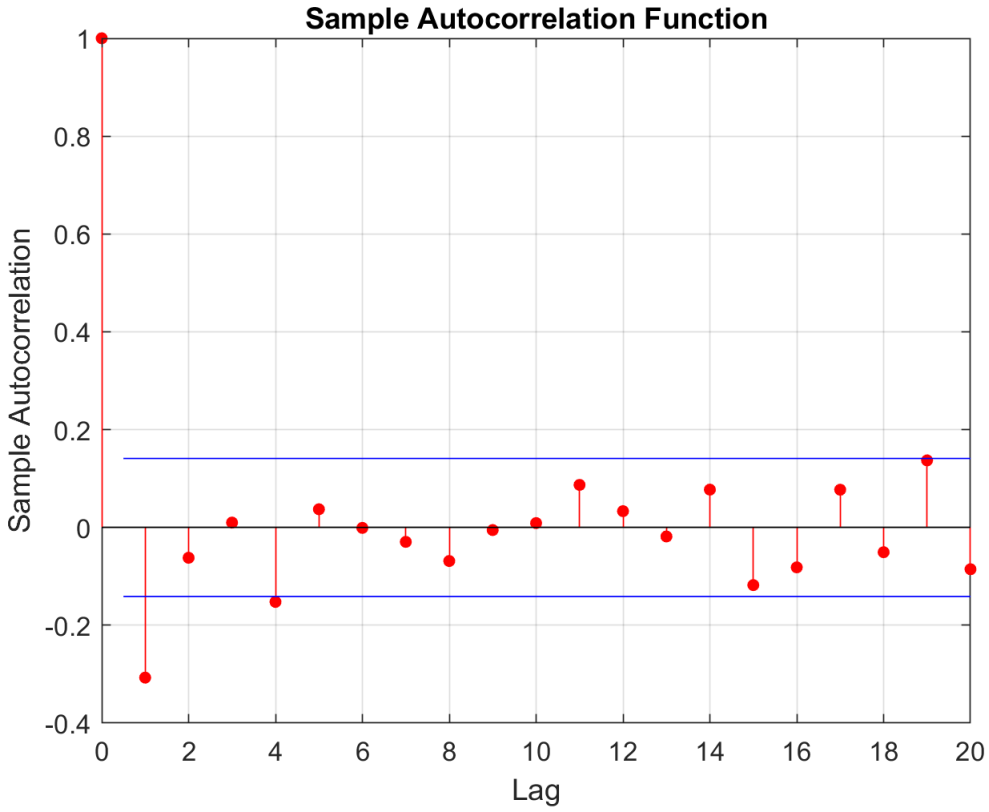

```
figure, h9 = axes; set(h9,'FontName','times','FontSize',16)
hold on, grid on
plot(timeCT,A,'LineWidth',1.5)
stairs(timeDT,A_hat,'r','LineWidth',1.5)
ylabel('[m/s$^2$]','FontName','times','FontSize',16,'Interpreter','latex')
xlabel('Time [sec]','FontName','times','FontSize',16,'Interpreter','latex')
leg5 = legend('$A$','$\hat{A}$','Location','NorthWest');
set(leg5,'FontName','times','FontSize',12,'Interpreter','latex')
```




```
figure, h10 = axes; set(h10,'FontName','times','FontSize',16)
hold on, grid on
stairs(timeDT,int_ctrl,'b','LineWidth',1.5)
stairs(timeDT,fs_ctrl,'r','LineWidth',1.5)
stairs(timeDT,ctrl_inp,'k','LineWidth',1.5)
ylabel('[A]','FontName','times','FontSize',16,'Interpreter','latex')
xlabel('Time [sec]','FontName','times','FontSize',16,'Interpreter','latex')
leg10 = legend('$I_{\mathrm{int}}$','$I_{\mathrm{fs}}
$','$I$','Location','NorthWest');
set(leg10,'FontName','times','FontSize',12,'Interpreter','latex')
```




```
% Check the innovation
figure, autocorr(inno)
```




```
var_inno = var(inno);
x = [X(1:Ts/STEP_SIZE:end) V(1:Ts/STEP_SIZE:end) A(1:Ts/STEP_SIZE:end)];
xhat = [X_hat V_hat A_hat];
est_err = x - xhat;
Qe_sim = cov(est_err)
```

```
Qe_sim = 3×3
10^-3 ×
    0.0000    0.0000    0.0002
    0.0000    0.0007    0.0244
    0.0002    0.0244    0.9251
```

```
disp('Qe_th =')
```

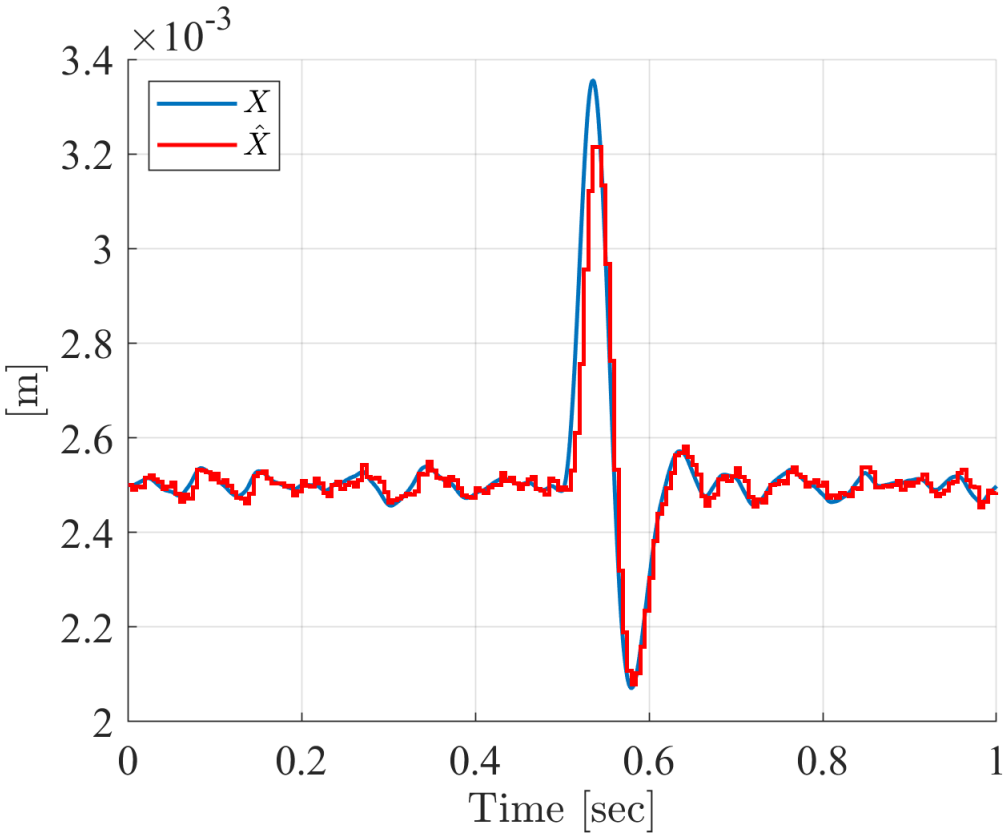
```
Qe_th =
```

```
disp(Qe_th)
```

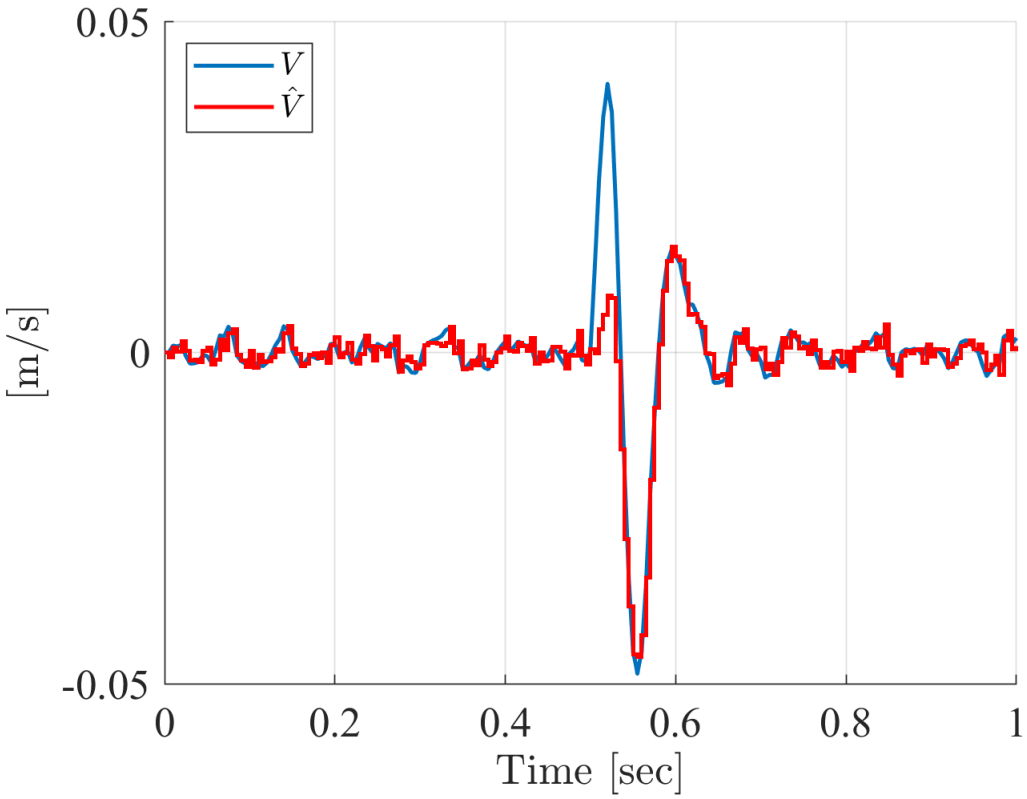
```
    0.0000    0.0000    0.0000
    0.0000    0.0002    0.0106
    0.0000    0.0106    1.1726
```

```
% Simulate step change of 0.25g
X_ref = Xss;
x0_hat = zeros(3,1);
STEP_TIME = 0.5;
A0 = 0;
A1 = 0.25*g;
```

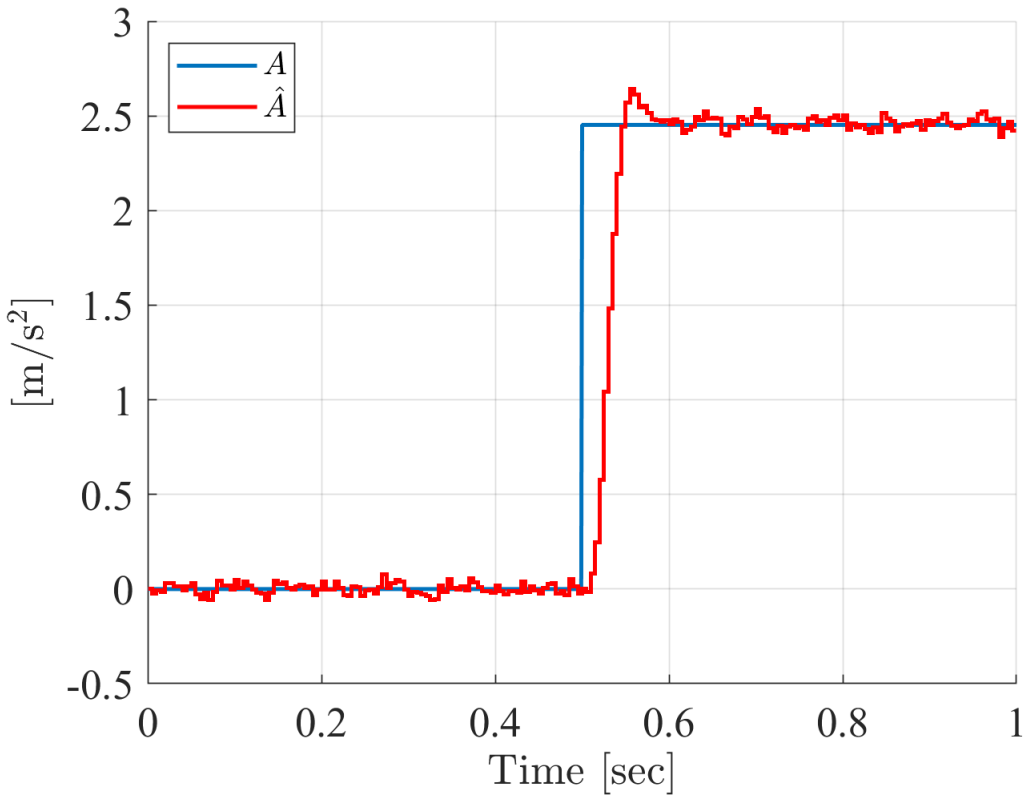
```
SIM_TIME = 1;
STEP_SIZE = 0.001;
sim(SIMULINK_FILENAME,SIM_TIME)

% Plot simulation result
timeCT = logsout.getElement(3).Values.Time;
X = logsout.getElement(3).Values.Data;
V = logsout.getElement(4).Values.Data;
A = logsout.getElement(10).Values.Data;
timeDT = logsout.getElement(2).Values.Time;
int_ctrl = logsout.getElement(2).Values.Data;
fs_ctrl = -logsout.getElement(1).Values.Data;
ctrl_inp = logsout.getElement(9).Values.Data;
X_hat = logsout.getElement(8).Values.Data(:,1)+Xss;
V_hat = logsout.getElement(8).Values.Data(:,2);
A_hat = logsout.getElement(8).Values.Data(:,3);
inno = logsout.getElement(6).Values.Data;

figure, h7 = axes; set(h7,'FontName','times','FontSize',16)
hold on, grid on
plot(timeCT,X,'LineWidth',1.5)
stairs(timeDT,X_hat,'r','LineWidth',1.5)
ylabel('[m]','FontName','times','FontSize',16,'Interpreter','latex')
xlabel('Time [sec]','FontName','times','FontSize',16,'Interpreter','latex')
leg3 = legend('$X$','$\hat{X}$','Location','NorthWest');
set(leg3,'FontName','times','FontSize',12,'Interpreter','latex')
```



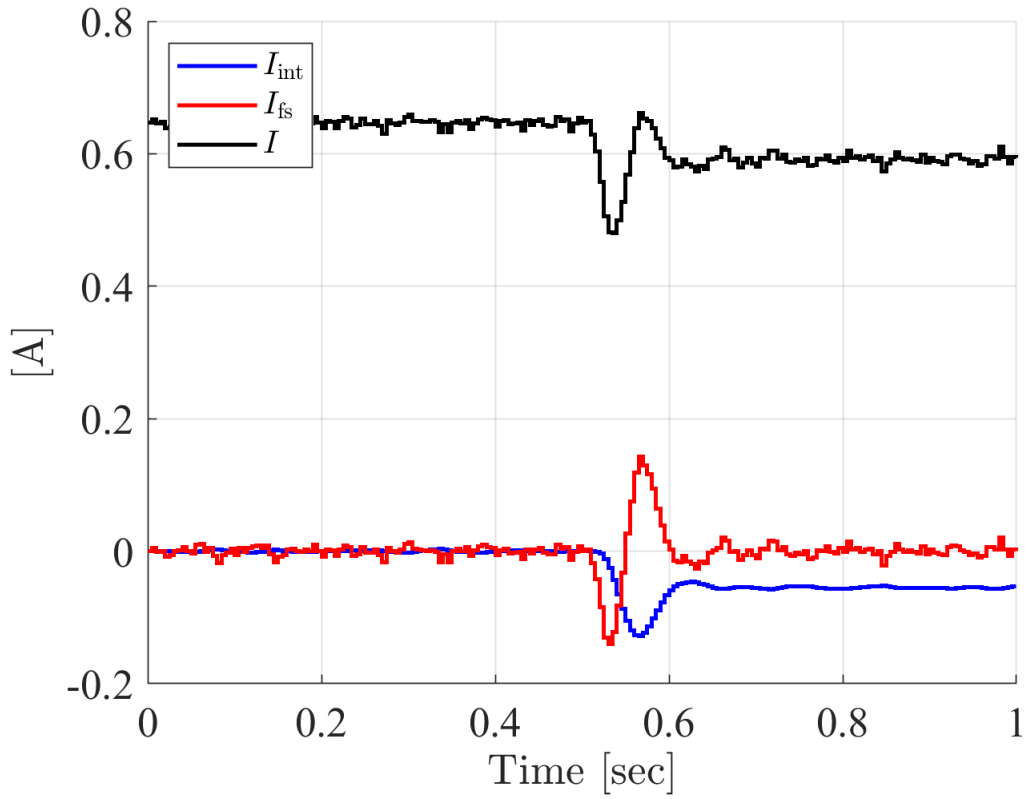
```
figure, h8 = axes; set(h8,'FontName','times','FontSize',16)
hold on, grid on
plot(timeCT,V,'LineWidth',1.5)
stairs(timeDT,V_hat,'r','LineWidth',1.5)
ylabel('[m/s]','FontName','times','FontSize',16,'Interpreter','latex')
xlabel('Time [sec]','FontName','times','FontSize',16,'Interpreter','latex')
leg4 = legend('$V$','$\hat{V}$','Location','NorthWest');
set(leg4,'FontName','times','FontSize',12,'Interpreter','latex')
```




```
figure, h9 = axes; set(h9,'FontName','times','FontSize',16)
hold on, grid on
plot(timeCT,A,'LineWidth',1.5)
stairs(timeDT,A_hat,'r','LineWidth',1.5)
ylabel('[m/s$^2$]','FontName','times','FontSize',16,'Interpreter','latex')
xlabel('Time [sec]','FontName','times','FontSize',16,'Interpreter','latex')
leg5 = legend('$A$','$\hat{A}$','Location','NorthWest');
set(leg5,'FontName','times','FontSize',12,'Interpreter','latex')
```




```
figure, h10 = axes; set(h10,'FontName','times','FontSize',16)
hold on, grid on
stairs(timeDT,int_ctrl,'b','LineWidth',1.5)
stairs(timeDT,fs_ctrl,'r','LineWidth',1.5)
stairs(timeDT,ctrl_inp,'k','LineWidth',1.5)
ylabel('[A]','FontName','times','FontSize',16,'Interpreter','latex')
xlabel('Time [sec]','FontName','times','FontSize',16,'Interpreter','latex')
leg10 = legend('$I_{\mathrm{int}}$','$I_{\mathrm{fs}}
$','$I$','Location','NorthWest');
set(leg10,'FontName','times','FontSize',12,'Interpreter','latex')
```


0.8
0.6
0.4
0.2
0
-0.2
[A]
I_{int}
I_{fs}
I
0
0.2
0.4
0.6
0.8
1
Time [sec]

Linear Control Design 2 - Fall 2022 - Exam Part I - Solution Manual

Introduction

The Part I of the Exam in Linear Control Design 2 (E22) consists of numerical exercises testing the acquisition of competences in the areas of analysis and design of control systems using modern control theory based on state space representation of system dynamics.

The scoring of each problem is clearly stated.

It is the sole responsability of the student to guarantee that the solution delivered for evaluation can be run by the examiner without the need of contacting the student. All dependencies on files external to this Matlab Live Script must be checked and included in the final delivery. **If the examiner will not be able to execute the Matlab Live Script delivered as solution by the student, the Part I will be considered failed.**

```
% Fill in your information
Exam = 'LCD2 E22'
```

```
Exam =
'LCD2 E22'
```

```
Student_Name = 'Student Name'
```

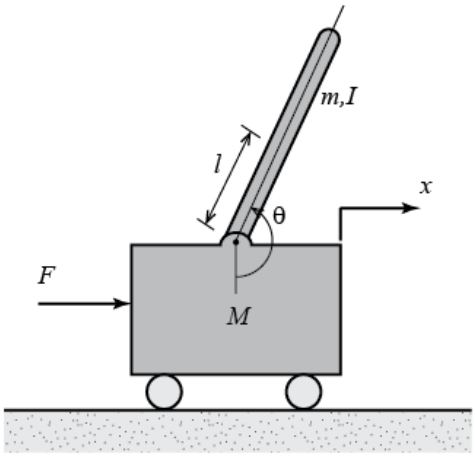
```
Student_Name =
'Student Name'
```

```
Student_Number = 'Student Number'
```

```
Student_Number =
'Student Number'
```

Optimal output feedback control of a balance system

A balance system is a mechanical system in which the center of mass is balanced above a pivot point. A popular example of a balance system is the Segway Personal Transporter (shown in the figure below) where a motorized platform is used to stabilize the person standing on top of it. Balance systems are a generalization of a mass-spring-damper system to multi-degree-of-freedom, and the inverted pendulum on a cart (also shown in the figure below) is a good mechanical approximation of such systems.



For this system the control input is the force F that moves the cart in the longitudinal direction, and the measured output is the longitudinal position x of the cart. The control objective is to keep the pendulum in its uprigth position ($\theta = \pi$ rad) while moving the cart along the longitudinal direction. This is equivalent to the Segway Personal Transporter keeping its upright position while the rider leans forward to move along the ground.

Nonlinear model of the inverted pendulum on a cart

The nonlinear model describing the dynamical variations of the angular position of the pendulum θ and the linear position of the cart x is described by the following nonlinear coupled second order differential equations

$$\begin{aligned}(I + ml^2)\ddot{\theta} + c\dot{\theta} + mgl \sin\theta &= -ml\ddot{x}\cos\theta \\ (M + m)\ddot{x} + b\dot{x} + ml\ddot{\theta}\cos\theta - ml\dot{\theta}^2 \sin\theta &= F\end{aligned} \tag{1}$$

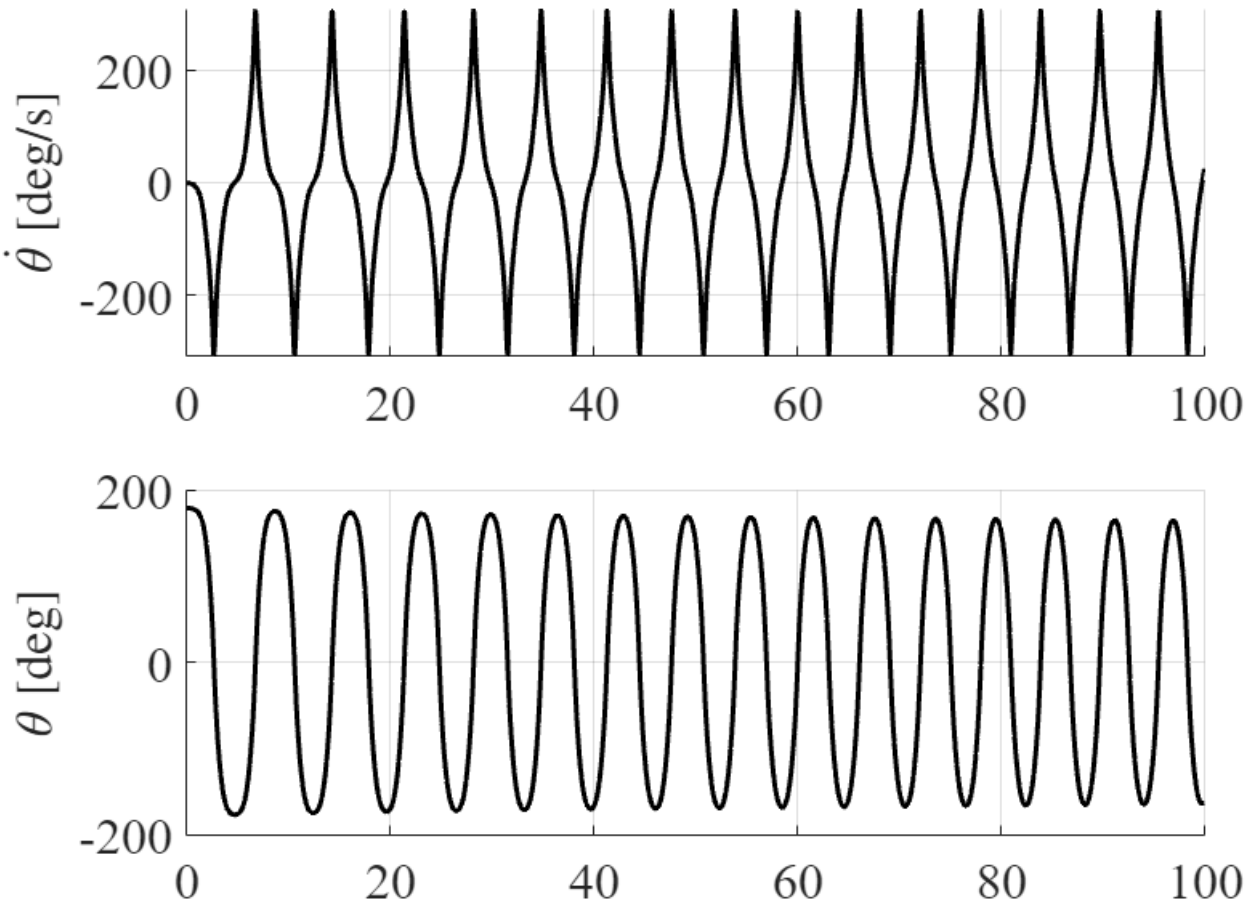
where

- M is the mass of the cart
- m is the mass of the pendulum
- I is the inertia of the pendulum
- b is the viscous friction coefficient of the cart
- c is the viscous friction coefficient of the pendulum
- l is the lenght of the pendulum
- g is the gravitational constant

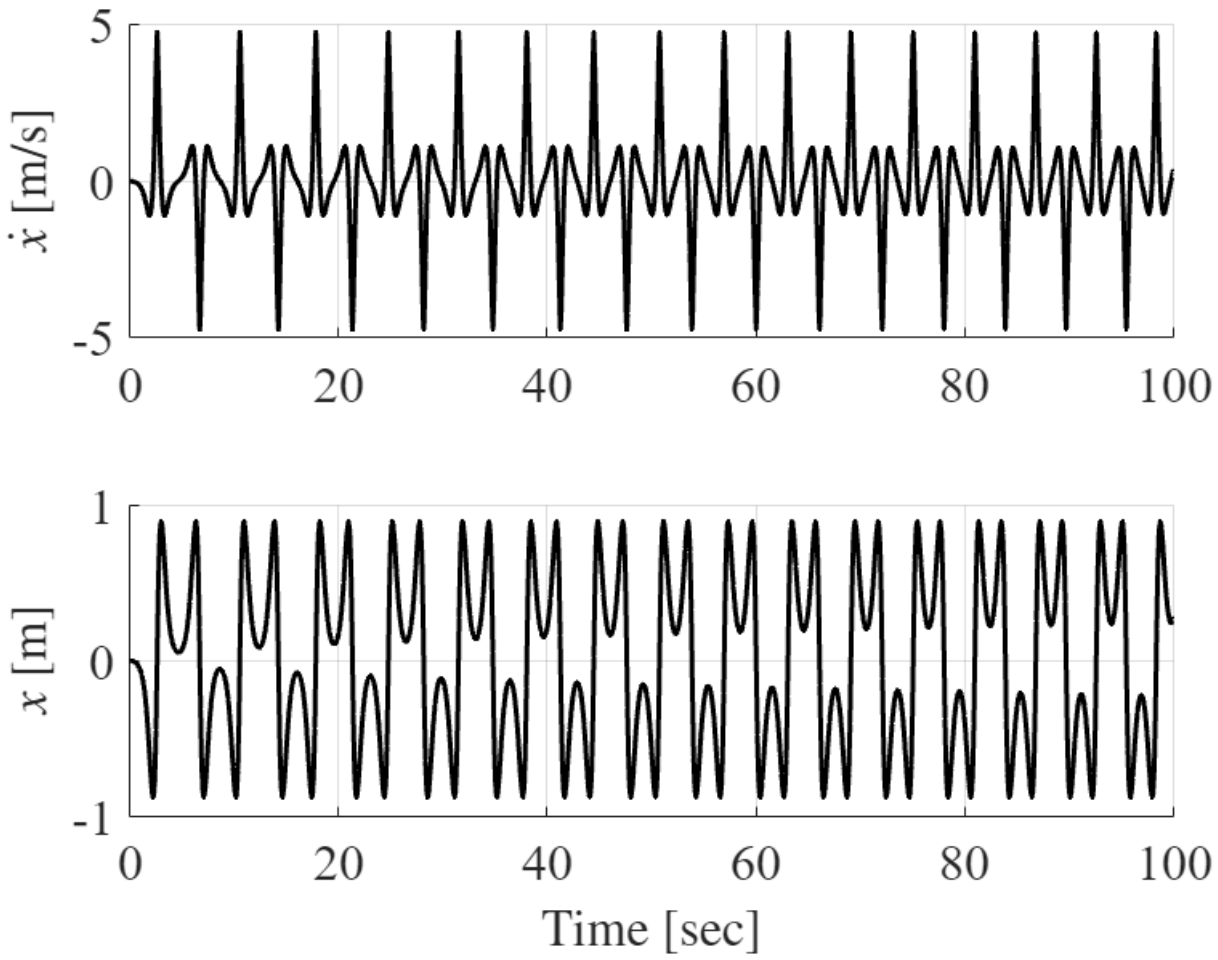
The nonlinear model in Eq. (1) is implemented in the Simulink file InvertedPendulum_SimulinkYYYYC, where YYYYC refers to the Simulink version (2021b, 2021a, 2020b, 2020a, 2019b, 2019a, 2018b, 2018a, 2017b, 2017a). The numerical values for the model parameters are provided right below.

```
% Model paramters (RG)
% Prepare the workspace
```

```
clear all
clc
% System parameters
M = 10; % mass of trolley [kg]
m = 80; % mass of pendulum bob [kg]
b = 0.1; % friction coefficient trolley [kg/s]
c = 0.01; % friction coefficient pendulum [kg*m^2/s]
I = 100; % inertia of pendulum [kg*m^2]
g = 9.8; % gravity constant [m/s^2]
l = 1; % lenght of pendulum rod [m]

% Initial conditions for testing the Simulink model (RG)
theta_dot0 = 0; % pendulum angular velocity [rad/s]
theta0 = pi-0.01; % pendulum angular position [rad]
x_dot0 = 0; % trolley linear velocity [m/s]
x0 = 0; % trolley linear position [m]
F0 = 0; % linear force applied to the trolley [N]

% Test the provided inverted pendulum model (RG)
SIM_TIME = 100; % simulation time (can be changed as needed - if simulated
for longer time, the pendulum returns to the downward position)
STEP_SIZE = 0.001; % integration step size of Simulink (can be changed as
needed)
% Change with the name of the model you are using
SIMULINK_FILENAME = 'InvertedPendulum_Simulink2021b';
time = (0:STEP_SIZE:SIM_TIME)';
F_ctrl = F0*ones(length(time),1);
sim(SIMULINK_FILENAME,SIM_TIME,[],[time F_ctrl])

% Plot the temporal behaviour of the quantities of interest (RG)
theta_dot = rad2deg(logsout.getElement('theta_dot').Values.Data);
theta = wrapTo180(rad2deg(logsout.getElement('theta').Values.Data)); % the
pendulum angular position is wrapped between -180 and 180 degress
x_dot = logsout.getElement('x_dot').Values.Data;
x = logsout.getElement('x').Values.Data;

figure, h1 = subplot(2,1,1); set(h1,'FontName','times','FontSize',16)
hold on, grid on
plot(time,theta_dot,'k','LineWidth',1.5)
ylabel('$\dot{\theta}$ [deg/
s]','FontName','times','FontSize',16,'Interpreter','latex')
h2 = subplot(2,1,2); set(h2,'FontName','times','FontSize',16)
hold on, grid on
plot(time,theta,'k','LineWidth',1.5)
ylabel('$\theta$
[deg]','FontName','times','FontSize',16,'Interpreter','latex')
```




```
figure, h1 = subplot(2,1,1); set(h1,'FontName','times','FontSize',16)
hold on, grid on
plot(time,x_dot,'k','LineWidth',1.5)
ylabel('$\dot{x}$ [m/
s]','FontName','times','FontSize',16,'Interpreter','latex')
h2 = subplot(2,1,2); set(h2,'FontName','times','FontSize',16)
hold on, grid on
plot(time,x,'k','LineWidth',1.5)
ylabel('$x$ [m]','FontName','times','FontSize',16,'Interpreter','latex')
xlabel('Time [sec]','FontName','times','FontSize',16,'Interpreter','latex')
```



Open-loop system analysis

Problem 1 [2 points] Based on the system dynamics described in Eq. (1), determine the system state vector, input, disturbance and output. Exploiting the Simulink model InvertedPendulum_SimulinkYYYYC linearize the system about the following operating point: $\theta = \pi$ rad; $\dot{\theta} = 0$ rad/s; $x = 0$ m; $\dot{x} = 0$ m/s.

```
% Your solution goes here

xss = [0;pi;0;0]; % stationary state
yss = 0;
uss = 0;
[A,B,C,D] = linmod(SIMULINK_FILENAME,xss,uss);
B = B(:,1);
D = D(:,1);

disp('Linearized model @ xss = [0;pi;0;0]')
```

```
Linearized model @ xss = [0;pi;0;0]
```

```
disp('System dynamical matrix A')
```

```
System dynamical matrix A
```

```
display(A)
```

```
A = 4×4
   -0.0001    7.2000   -0.0008         0
    1.0000         0         0         0
   -0.0001    6.4000   -0.0018         0
         0         0    1.0000         0
```

```
disp('System input matrix B')
```

```
System input matrix B
```

```
display(B)
```

```
B = 4×1
    0.0082
         0
    0.0184
         0
```

```
disp('System output matrix C')
```

```
System output matrix C
```

```
display(C)
```

```
C = 1×4
     0     0     0     1
```

```
disp('System input feedthrough matrix D')
```

```
System input feedthrough matrix D
```

```
display(D)
```

```
D = 0
```

Problem 2 [2 points] Evaluate the internal and the external stability of system.

```
% Your solution goes here
lambda_ol = eig(A)
```

```
lambda_ol = 4×1
         0
    2.6829
   -2.6837
   -0.0011
```

```
disp('One eigenvalue has positive real part => System is unstable')
```

```
One eigenvalue has positive real part => System is unstable
```

```
[numG,denG] = ss2tf(A,B,C,D);

G_Fx = minreal(tf(numG,denG))
```

```
G_Fx =

     0.01837 s^2 + 1.02e-06 s - 0.08
  ------------------------------------
  s^4 + 0.001929 s^3 - 7.2 s^2 - 0.008 s

Continuous-time transfer function.
```

```
p = pole(G_Fx)
```

```
p = 4×1
         0
   -2.6837
    2.6829
   -0.0011
```

```
disp('One pole has positive real part => System is not BIBO stable')
```

```
One pole has positive real part => System is not BIBO stable
```

Problem 3 [2 points] Compute the step response of the system, and discuss the behaviour of the system state and output in the light of the stability analysis.

```
% Your solution goes here
disp('When stepping on the input force the cart start moving with constant
acceleration. Over time this will give rise to a change in position
following a parabola (position is the second integration of acceleration).')
```

```
When stepping on the input force the cart start moving with constant acceleration. Over time this will giv
```

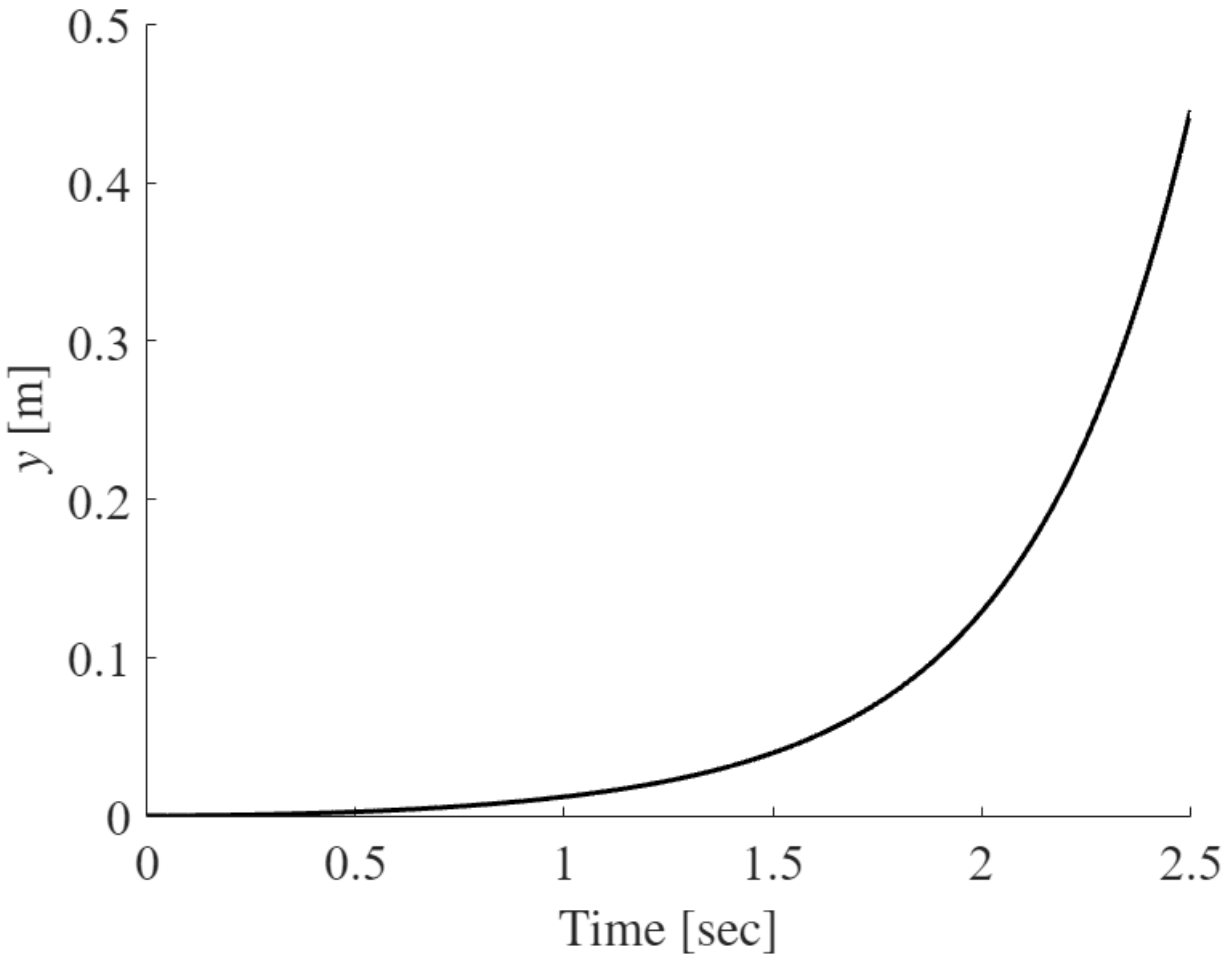
```
disp('In response to the step the pendulum position and velocity of the
linear system will diverge to infinity because of the unstable eigenvalue.
If the step is performed on the nonlinear system the response of the')
```

```
In response to the step the pendulum position and velocity of the linear system will diverge to infinity b
```

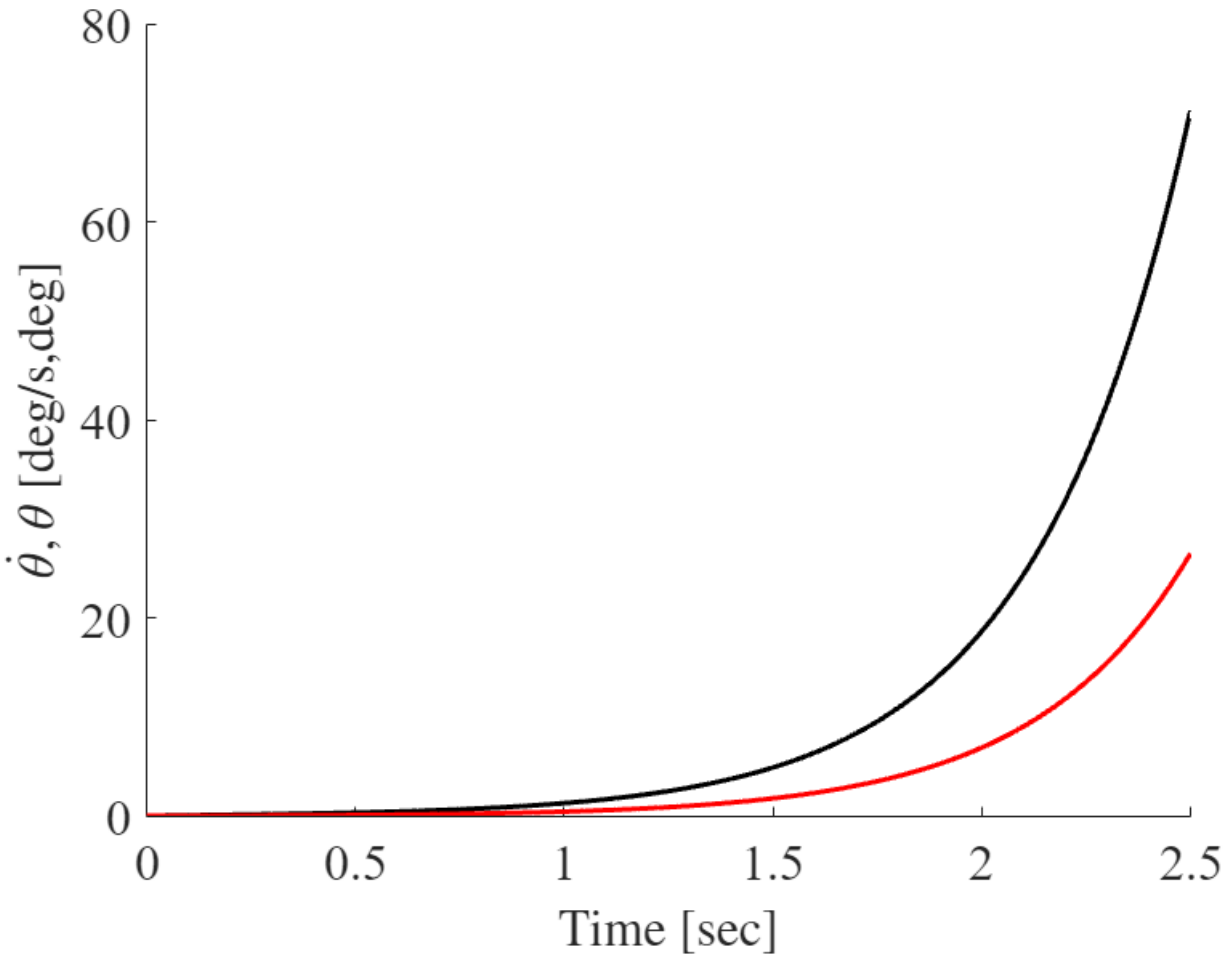
```
disp('cart position is exactly the same, however the pendulum will start
swinging around its hinging point since its position is physically bounded
between 0 and 2*pi.')
```

```
cart position is exactly the same, however the pendulum will start swinging around its hinging point since
```

```


sys_ol = ss(A,B,C,D);
tFinal = 2.5;
[y,t,x] = step(sys_ol,tFinal);
figure, h5 = axes; set(h5,'FontName','times','FontSize',16)
hold on
plot(t,y,'k','LineWidth',1.5)
ylabel('$y$ [m]','FontName','times','FontSize',16,'Interpreter','latex')
xlabel('Time [sec]','FontName','times','FontSize',16,'Interpreter','latex')
```




```
figure, h5 = axes; set(h5,'FontName','times','FontSize',16)
hold on
plot(t,rad2deg(x(:,1)),'k',t,rad2deg(x(:,2)),'r','LineWidth',1.5)
ylabel('$\dot{\theta},\theta$ [deg/
s,deg]','FontName','times','FontSize',16,'Interpreter','latex')
xlabel('Time [sec]','FontName','times','FontSize',16,'Interpreter','latex')
```



Control system design for balance system

The main control objective is to maintain the pendulum in its upright position while the cart changes its position along the longitudinal direction, e.g. following a trajectory $x_{ref}(t)$.

Closed-loop system requirements

The following closed-loop requirements should be fulfilled when the position of the cart changes of 1m

1. The 1% settling time for x is between 25 and 35 seconds;
2. The pendulum angular position θ does not deviate more than $\pi/48$ rad (≈ 4 deg) from the upright position;
3. The steady state error is less than 3% for x
4. The applied control force F is within ± 10 N
5. The pendulum angular position is kept at $\theta_{ref} = \pi$ rad.

Problem 4 [6 points] Under the assumption that the state is fully accessible design a discrete time optimal controller that meets the aforementioned requirements.

```
% Your solution goes here
```

```
% Controllability test
Mc = ctrb(A,B);

if rank(Mc) == size(A,1)
    disp('Open loop linear system is controllable')
else
    disp('Open loop linear system is not controllable')
end
```

```
Open loop linear system is controllable
```

```
% Sampling time
T_sett = 25; % selected settling time [s]
tau_sett = T_sett/5; % slowest time constant of closed loop system [s]

tau_ol = 1./abs(lambda_ol(abs(lambda_ol)>0));
Ts_max = min([tau_sett min(tau_ol)]./10) % maximum sampling time [s]
```

```
Ts_max = 0.0373
```

```
Ts = 0.01; % sampling time [s]
[F,G] = c2d(A,B,Ts);

% Discrete time optimal controller
% Q = diag([0.1 0.1 1 5]); % state weighting matrix
Q = diag([100 1/(pi/48)^2 100 1/(0.03)^2]) % based on requirements on x (3%
of error on 1m step change is 0.03m) and theta; weights on theta_dot and
x_dot are selected to have a well damped response.
```

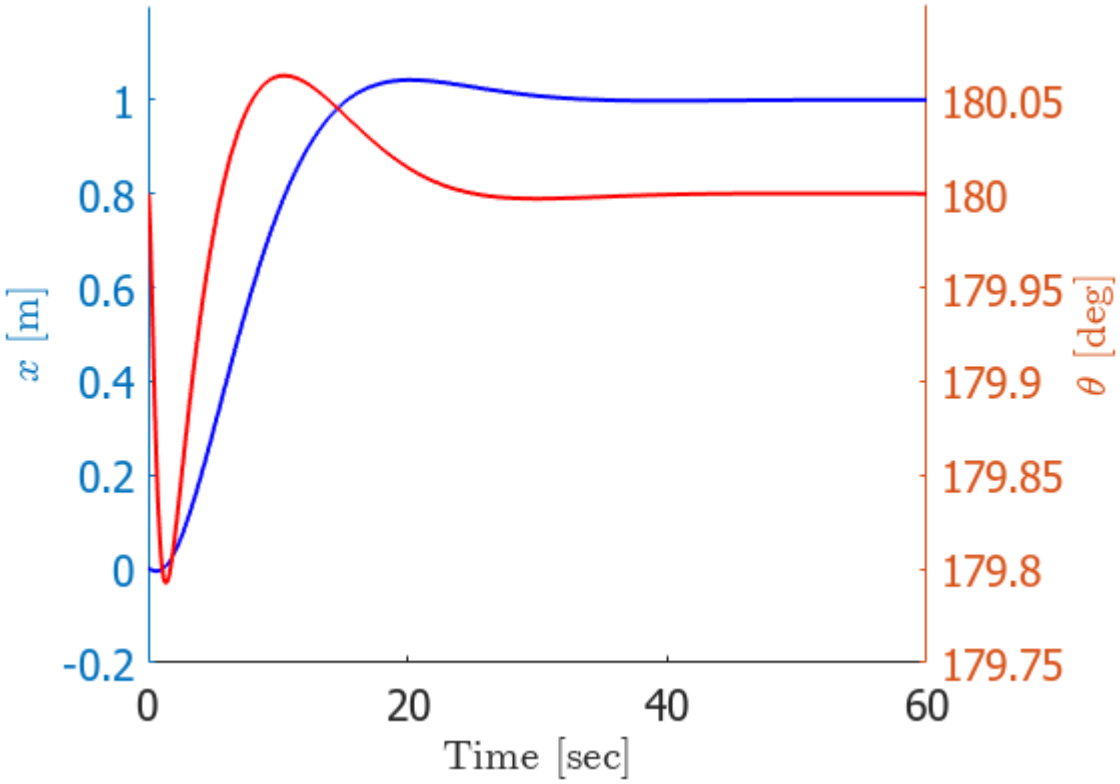
```
Q = 4×4
10^3 ×
    0.1000         0         0         0
         0    0.2334         0         0
         0         0    0.1000         0
         0         0         0    1.1111
```

```
R = 50; % input weighting matrix tuned to trade off between speed of
response and use of control signal

[K_opt,S,lambda_cl_dt] = dlqr(F,G,Q*Ts,R*Ts);
lambda_cl_ct = 1/Ts.*log(lambda_cl_dt)
```

```
lambda_cl_ct = 4×1 complex
  -0.1624 + 0.1613i
  -0.1624 - 0.1613i
  -2.6804 + 0.0000i
  -2.6862 + 0.0000i
```

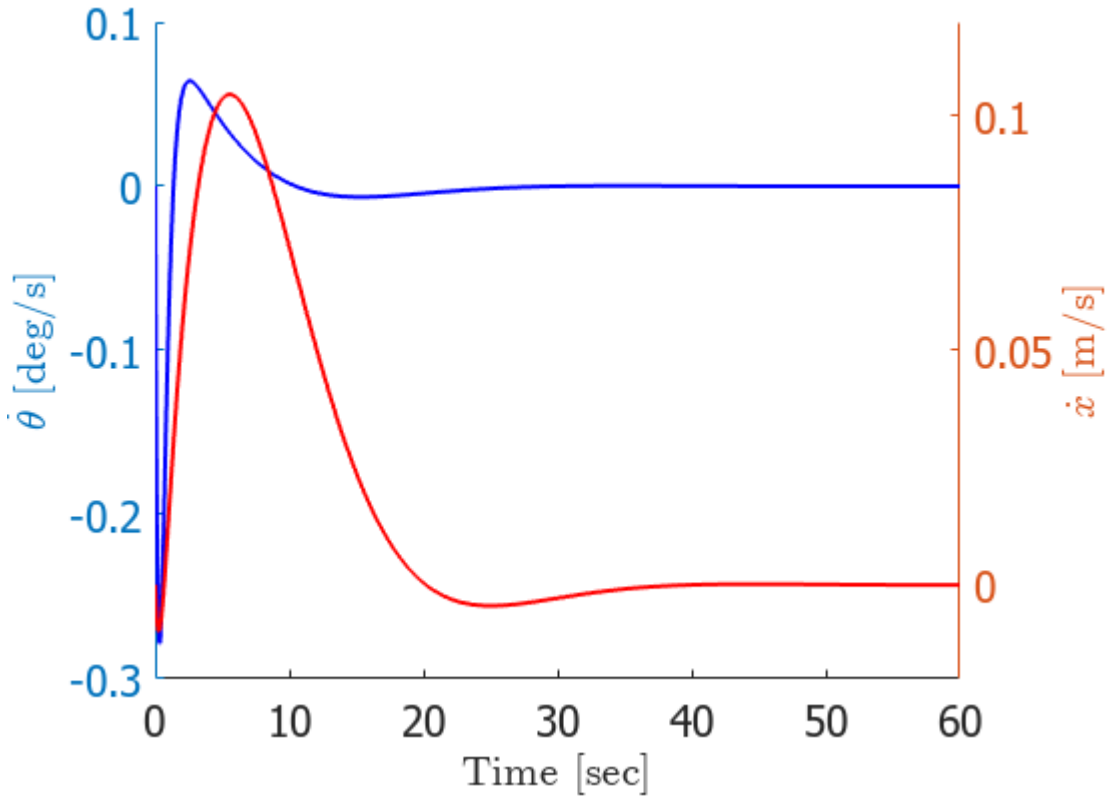
```
% Set-point control (a solution with integral action would also work here)
N = inv(C*inv(eye(4)-(F-G*K_opt))*G);
sys_cl_dt = ss(F-G*K_opt,G*N,C,D,Ts); % ver2: u = -K_LQR*x + Nr

% Simulate closed-loop step response
```

```
t_STEP = 10; % 10 seconds into the simulation
x_ref0 = 1;
x_ref = x_ref0*(ones(length(time(1:Ts/STEP_SIZE:end)),1));
ref = x_ref;
[y,t,x_lin] = lsim(sys_cl_dt,ref,time(1:Ts/STEP_SIZE:end));
% F_ctrl_lin = -K_opt*(x_lin'-ref);
F_ctrl_lin = -K_opt*x_lin' + N*x_ref';

figure, h6 = axes; set(h6,'FontName','times','FontSize',16)
hold on
yyaxis left
stairs(t,y,'b','LineWidth',1.5)
ylabel('$x$ [m]','FontName','times','FontSize',16,'Interpreter','latex')
xlabel('Time [sec]','FontName','times','FontSize',16,'Interpreter','latex')
yyaxis right
stairs(t,rad2deg(x_lin(:,2)+pi),'r','LineWidth',1.5)
ylabel('$\theta$
[deg]','FontName','times','FontSize',16,'Interpreter','latex')
xlim([0,60])
```

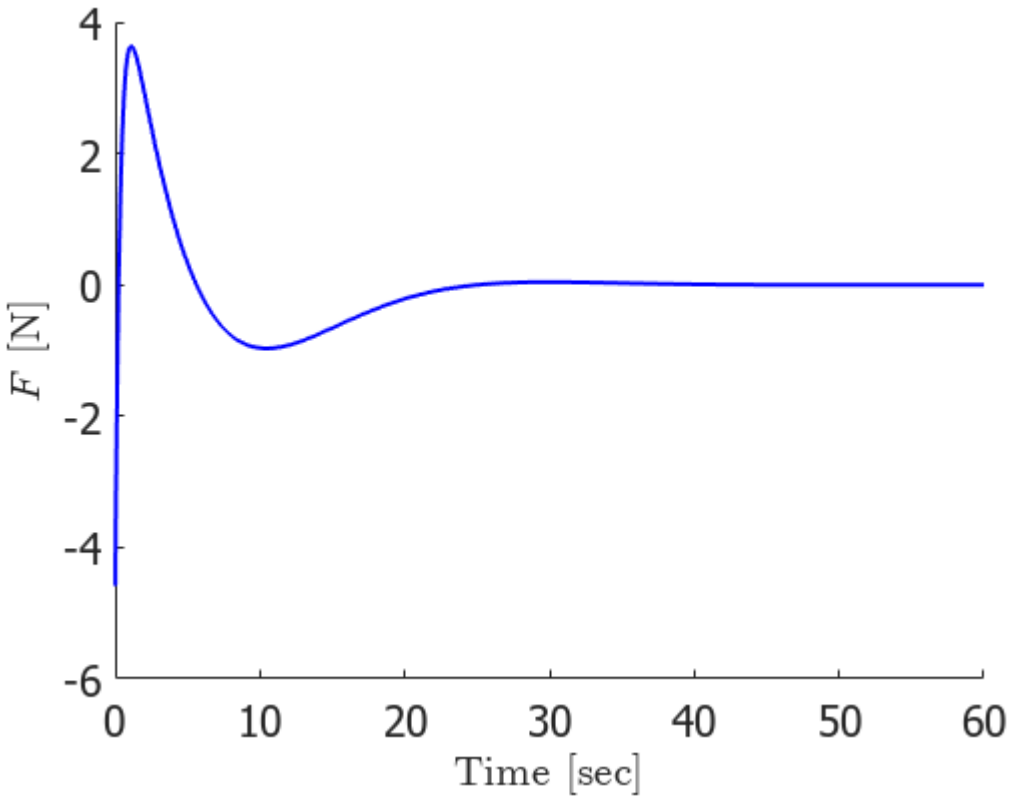



```
figure, h6 = axes; set(h6,'FontName','times','FontSize',16)
hold on
yyaxis left
stairs(t,rad2deg(x_lin(:,1)),'b','LineWidth',1.5)
ylabel('$\dot{\theta}$ [deg/
s]','FontName','times','FontSize',16,'Interpreter','latex')
xlabel('Time [sec]','FontName','times','FontSize',16,'Interpreter','latex')
```

```
yyaxis right
stairs(t,x_lin(:,3),'r','LineWidth',1.5)
ylabel('$\dot{x}$ [m/
s]','FontName','times','FontSize',16,'Interpreter','latex')
xlim([0,60])
```



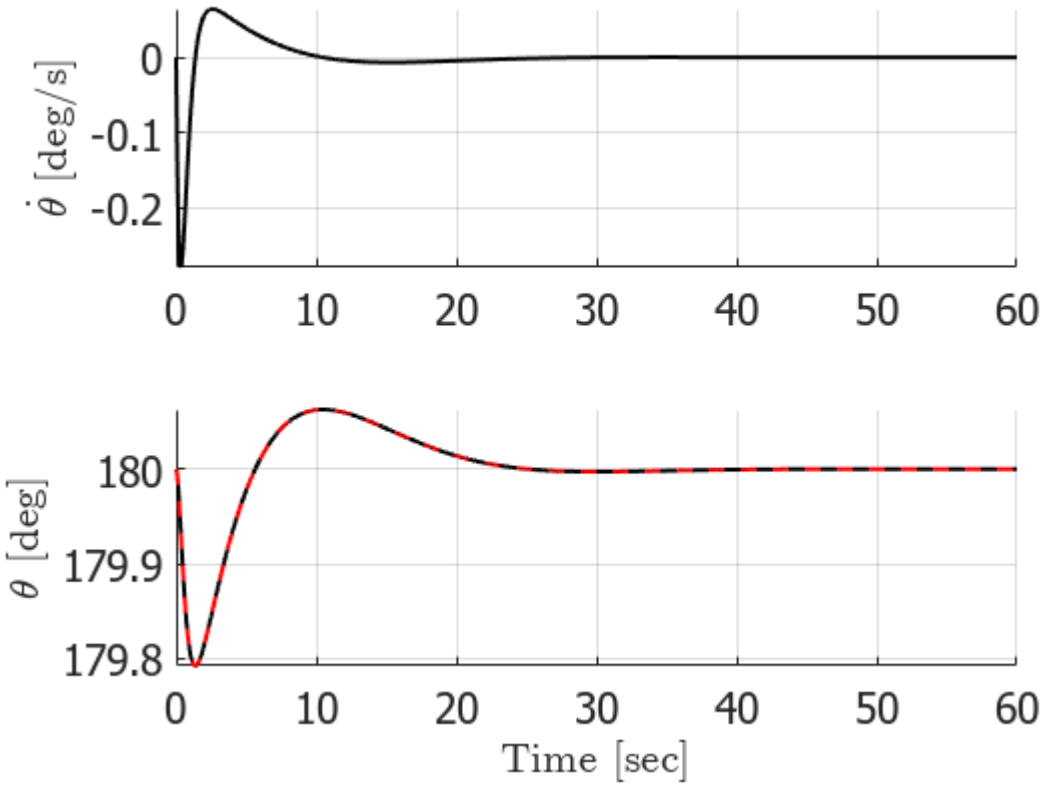

```
figure, h6 = axes; set(h6,'FontName','times','FontSize',16)
hold on
stairs(t,F_ctrl_lin,'b','LineWidth',1.5)
ylabel('$F$ [N]','FontName','times','FontSize',16,'Interpreter','latex')
xlabel('Time [sec]','FontName','times','FontSize',16,'Interpreter','latex')
```



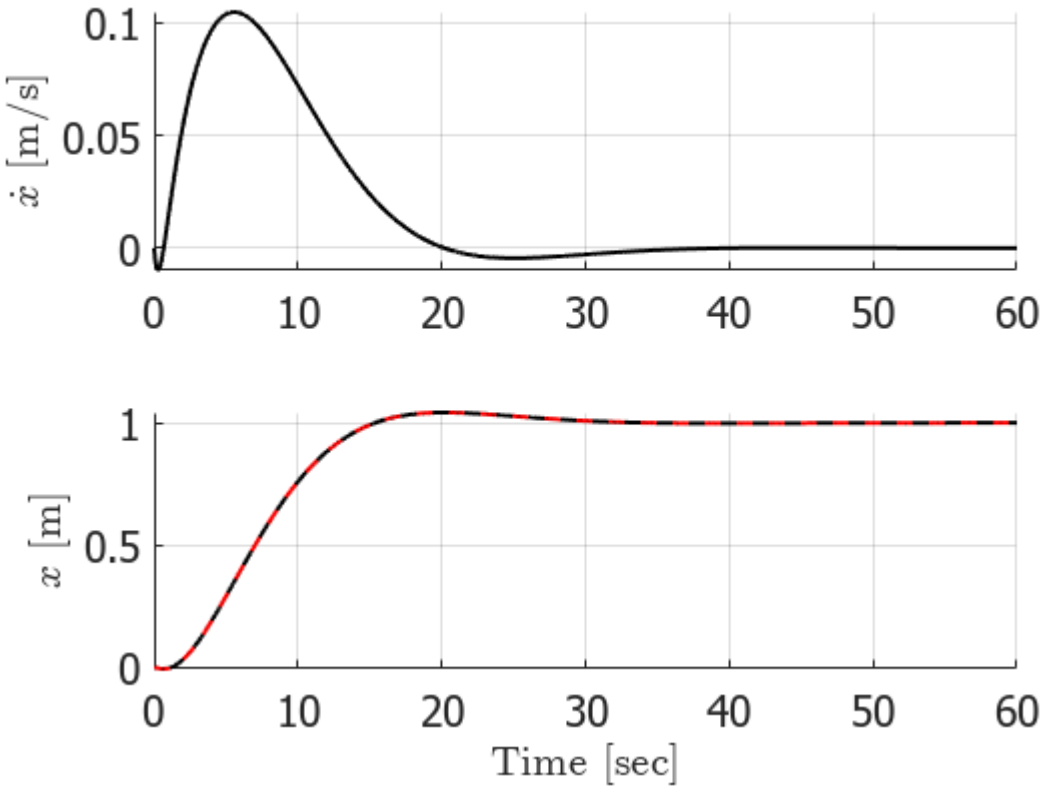
Problem 5 [2 points] Implement the discrete time optimal controller in the Simulink model of the nonlinear system, and evalute the closed-loop system performance against the given requirements, when the sytem is subject to a step change in the reference trolley position of $x_{ref} = 0.5$ m.

```
% Your solution goes here

% Test the closed-loop nonlinear system
SIM_TIME = 60; % simulation time (can be changed as needed)
STEP_SIZE = 0.001; % integration step size of Simulink (can be changed as
needed)
% Change with the name of the model you are using
theta0 = pi;
SIMULINK_FILENAME = 'InvertedPendulum_DLQR_Simulink2020b';
sim(SIMULINK_FILENAME,SIM_TIME,[])

% Plot the temporal behaviour of the quantities of interest
time = logsout.getElement('theta_dot').Values.Time;
theta_dot = rad2deg(logsout.getElement('theta_dot').Values.Data);
theta = rad2deg(logsout.getElement('theta').Values.Data); % the pendulum
angular position is wrapped between -180 and 180 degress
x_dot = logsout.getElement('x_dot').Values.Data;
x = logsout.getElement('x').Values.Data;
time_dt = logsout.getElement('F').Values.Time;
F_ctrl = logsout.getElement('F').Values.Data;
```

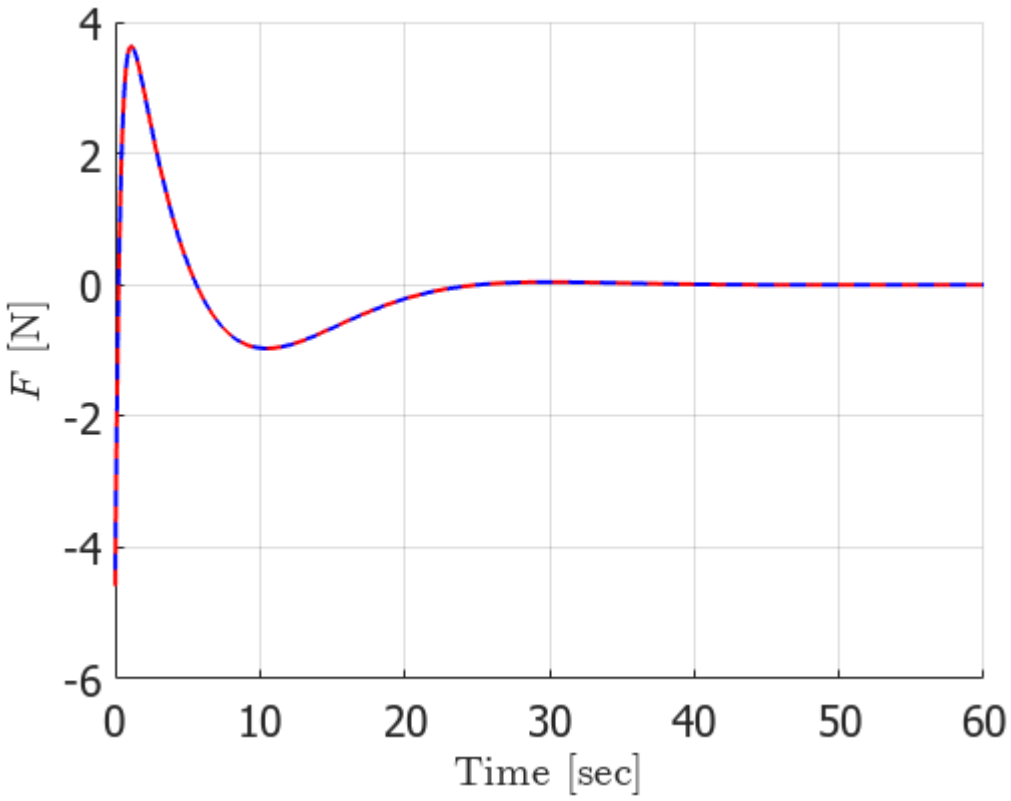
```
figure, h7 = subplot(2,1,1); set(h7,'FontName','times','FontSize',16)
hold on, grid on
plot(time,theta_dot,'k','LineWidth',1.5)
ylabel('$\dot{\theta}$ [deg/
s]','FontName','times','FontSize',16,'Interpreter','latex')
h8 = subplot(2,1,2); set(h8,'FontName','times','FontSize',16)
hold on, grid on
plot(time,theta,'k',t,rad2deg(x_lin(:,2)+pi),'--r','LineWidth',1.5)
ylabel('$\theta$
[deg]','FontName','times','FontSize',16,'Interpreter','latex')
xlabel('Time [sec]','FontName','times','FontSize',16,'Interpreter','latex')
```




```
figure, h9 = subplot(2,1,1); set(h9,'FontName','times','FontSize',16)
hold on, grid on
plot(time,x_dot,'k','LineWidth',1.5)
ylabel('$\dot{x}$ [m/
s]','FontName','times','FontSize',16,'Interpreter','latex')
h10 = subplot(2,1,2); set(h10,'FontName','times','FontSize',16)
hold on, grid on
plot(time,x,'k',t,x_lin(:,4),'--r','LineWidth',1.5)
ylabel('$x$ [m]','FontName','times','FontSize',16,'Interpreter','latex')
xlabel('Time [sec]','FontName','times','FontSize',16,'Interpreter','latex')
```




```
figure, h11 = axes; set(h11,'FontName','times','FontSize',16)
hold on, grid on
stairs(time_dt,F_ctrl,'b','LineWidth',1.5)
stairs(t,F_ctrl_lin,'--r','LineWidth',1.5)
ylabel('$F$ [N]','FontName','times','FontSize',16,'Interpreter','latex')
xlabel('Time [sec]','FontName','times','FontSize',16,'Interpreter','latex')
```



State estimation for output feedback control

In order to implement an output feedback controller, the state vector must be estimated based on the available measurement of the cart position x. In the last part of the exercise, we assume the following measurement model

$$y_1 = x + v_1 \qquad (2)$$

where v_1 is zero mean white noise. The noise amplitude is ± 0.005 m.

Problem 6 [6 points] Design a discrete time Kalman filter to estimate the state vector.

```
% Your solution goes here

% Observability test
Mo = obsv(A,C);

if rank(Mo) == size(A,1)
    disp('Open loop linear system is observable')
else
    disp('Open loop linear system is not observable')
end
```

```
Open loop linear system is observable
```

```
V1 = diag([0.00001 0.000001 0.00001 0.000001]); % process noise uncertainty
is low across all states since the linear model is a good approximation of
the nonlinear model at the stationary state
Bv = diag([A(1,1) 1 A(3,3) 1]); % process noise enters the system in the
feedback loop of the states theta_dot and x_dot
V1d = Bv*V1*Bv'*Ts;
noise_ampl = 0.005;
V2 = (noise_ampl/3)^2;
V2d = V2/Ts;

F_KF = F;
G_KF = G;
Gv_KF = eye(4);
C_KF = C;

[L_KF,P,Z,lambda_ee_dt] = dlqe(F_KF,Gv_KF,C_KF,V1d,V2d)
```

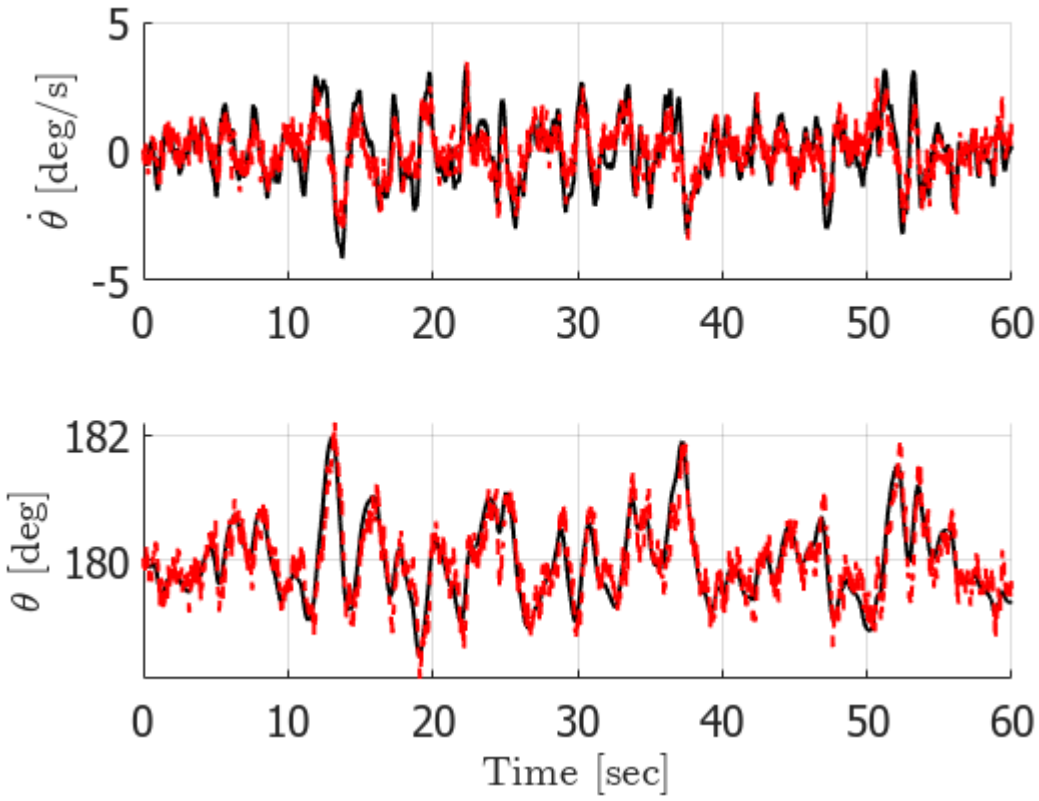
```
L_KF = 4×1
   202.1603e-003
    75.5765e-003
   179.5977e-003
    59.3151e-003
P = 4×4
   229.2948e-006    86.1087e-006   203.6734e-006    59.6966e-006
    86.1087e-006    32.5263e-006    76.4873e-006    22.3172e-006
   203.6734e-006    76.4873e-006   180.9217e-006    53.0340e-006
    59.6966e-006    22.3172e-006    53.0340e-006    17.5153e-006
Z = 4×4
   217.2265e-006    81.5971e-006   192.9520e-006    56.1556e-006
    81.5971e-006    30.8397e-006    72.4792e-006    20.9935e-006
   192.9520e-006    72.4792e-006   171.3969e-006    49.8883e-006
    56.1556e-006    20.9935e-006    49.8883e-006    16.4764e-006
lambda_ee_dt = 4×1
   999.9558e-003
   991.6088e-003
   976.8052e-003
   971.1956e-003
```

```
x0_hat = zeros(4,1);
```

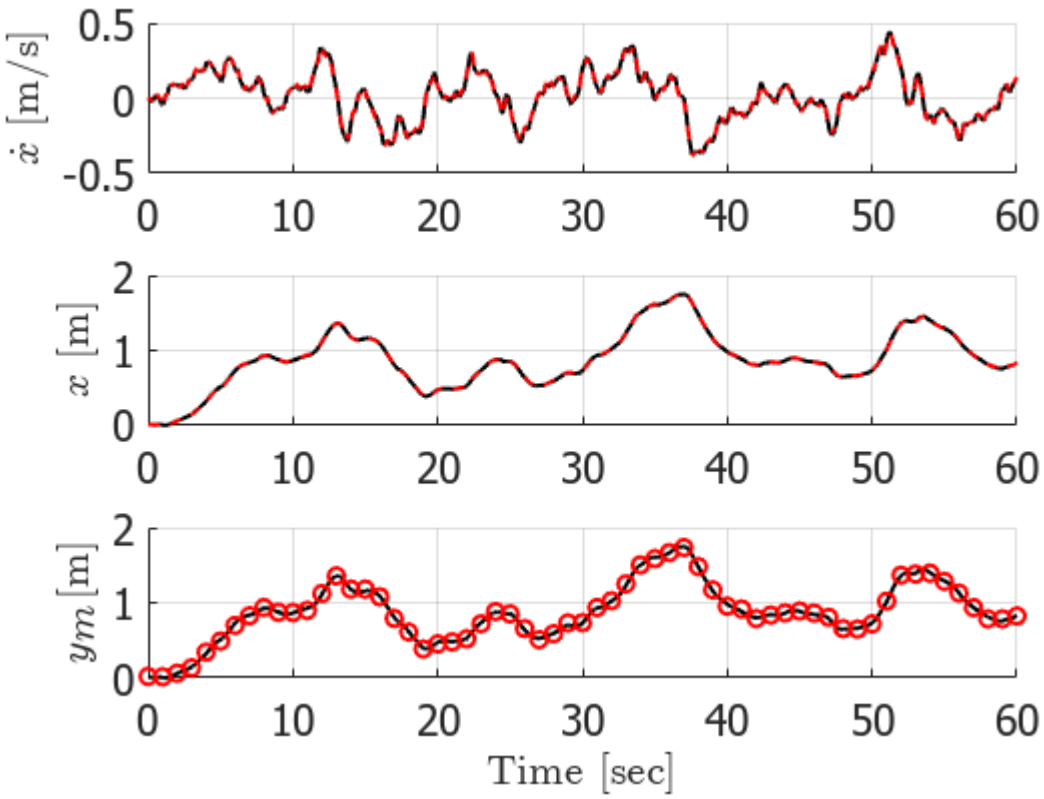
Problem 7 [4 points] Implement the discrete time Kalman filter in the Simulink diagram of the closed-loop system and assess its estimation performance when a step change in the position of the cart of $x_{ref} = 1$ m takes place.

```
% Your solution goes here

% Test the closed-loop output feedback optimal controller on the nonlinear
system
SIM_TIME = 60; % simulation time (can be changed as needed)
STEP_SIZE = 0.0001; % integration step size of Simulink (can be changed as
needed)
% Change with the name of the model you are using
SIMULINK_FILENAME = 'InvertedPendulum_DLQG_Simulink2020b';
sim(SIMULINK_FILENAME,SIM_TIME,[])
```

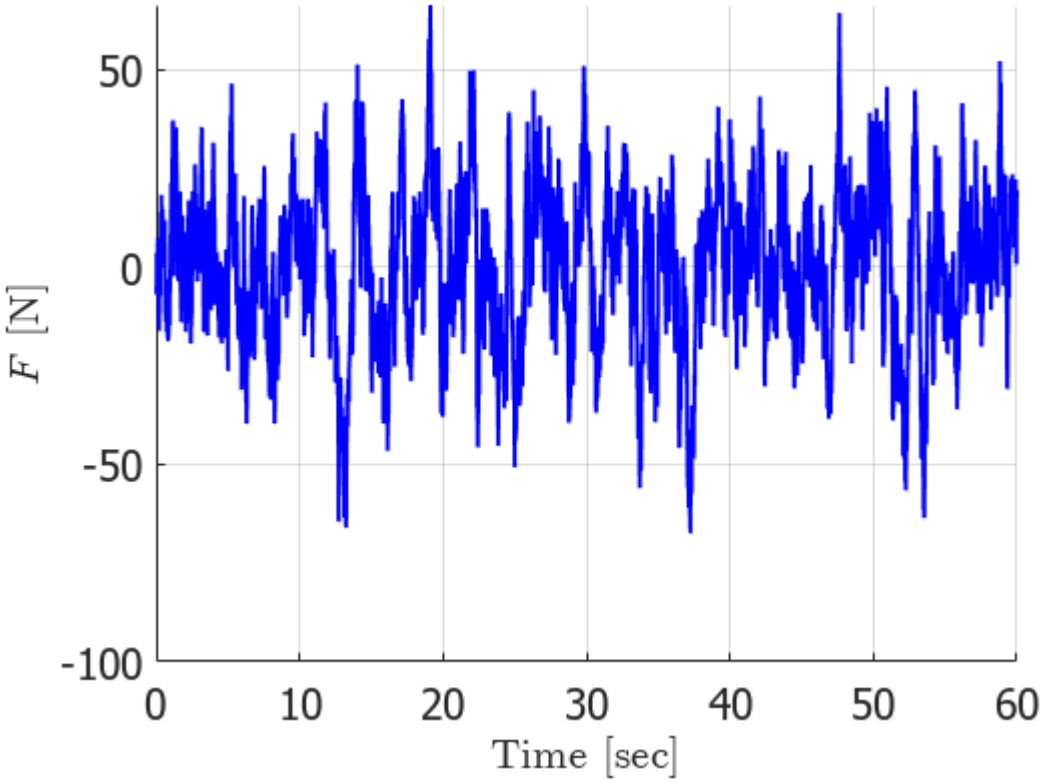
```
% Plot the temporal behaviour of the quantities of interest
time = logsout.getElement('theta_dot').Values.Time;
theta_dot = rad2deg(logsout.getElement('theta_dot').Values.Data);
theta = rad2deg(logsout.getElement('theta').Values.Data); % the pendulum
angular position is wrapped between -180 and 180 degress
x_dot = logsout.getElement('x_dot').Values.Data;
x = logsout.getElement('x').Values.Data;
time_dt = logsout.getElement('F').Values.Time;
F_ctrl = logsout.getElement('F').Values.Data;
x_hat = logsout.getElement('x_hat').Values.Data+xss';
ym = logsout.getElement('ym').Values.Data;

figure, h12 = subplot(2,1,1); set(h12,'FontName','times','FontSize',16)
hold on, grid on
plot(time,theta_dot,'k','LineWidth',1.5)
stairs(time_dt,rad2deg(x_hat(:,1)),'--r','LineWidth',1.5)
ylabel('$\dot{\theta}$ [deg/
s]','FontName','times','FontSize',16,'Interpreter','latex')
h13 = subplot(2,1,2); set(h13,'FontName','times','FontSize',16)
hold on, grid on
plot(time,theta,'k','LineWidth',1.5)
stairs(time_dt,rad2deg(x_hat(:,2)),'--r','LineWidth',1.5)
ylabel('$\theta$
[deg]','FontName','times','FontSize',16,'Interpreter','latex')
xlabel('Time [sec]','FontName','times','FontSize',16,'Interpreter','latex')
```



```
figure, h14 = subplot(3,1,1); set(h14,'FontName','times','FontSize',16)
hold on, grid on
plot(time,x_dot,'k','LineWidth',1.5)
stairs(time_dt,x_hat(:,3),'--r','LineWidth',1.5)
ylabel('$\dot{x}$ [m/
s]','FontName','times','FontSize',16,'Interpreter','latex')
h15 = subplot(3,1,2); set(h15,'FontName','times','FontSize',16)
hold on, grid on
plot(time,x,'k','LineWidth',1.5)
stairs(time_dt,x_hat(:,4),'--r','LineWidth',1.5)
ylabel('$x$ [m]','FontName','times','FontSize',16,'Interpreter','latex')
h16 = subplot(3,1,3); set(h16,'FontName','times','FontSize',16)
hold on, grid on
plot(time,x,'k','LineWidth',1.5)
plot(time_dt(1:100:end),ym(1:100:end),'or','LineWidth',1.5)
ylabel('$y_m$ [m]','FontName','times','FontSize',16,'Interpreter','latex')
xlabel('Time [sec]','FontName','times','FontSize',16,'Interpreter','latex')
```




```
figure, h16 = axes; set(h16,'FontName','times','FontSize',16)
hold on, grid on
stairs(time_dt,F_ctrl,'b','LineWidth',1.5)
ylabel('$F$ [N]','FontName','times','FontSize',16,'Interpreter','latex')
xlabel('Time [sec]','FontName','times','FontSize',16,'Interpreter','latex')
```



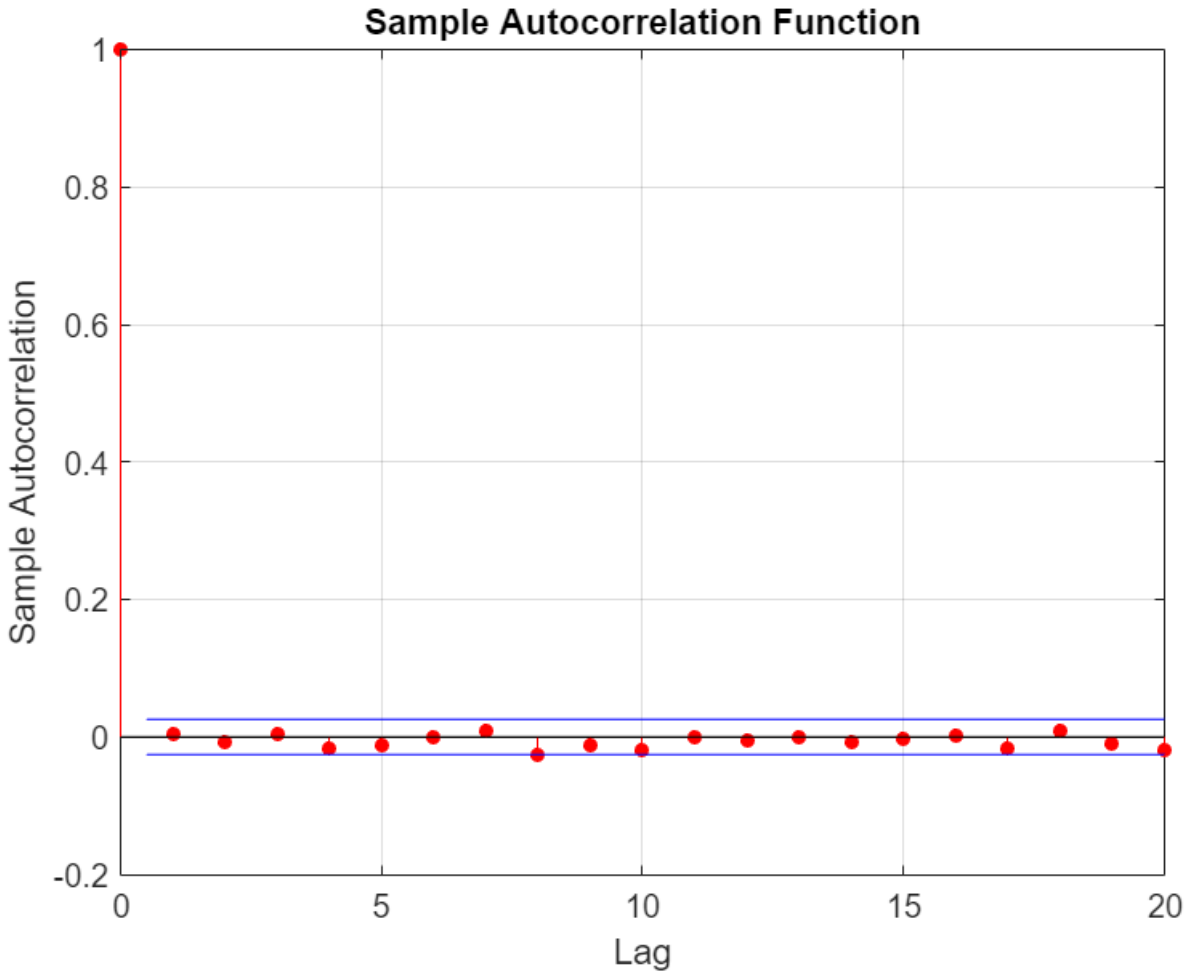

```
disp('It is worth noticing that the introduction of measurement noise
deteriorates the control performance in terms of control effort, which
clearly exceeds the 10N given in the control design requirements.')
```

```
It is worth noticing that the introduction of measurement noise deteriorates the control performance in te
```

```
disp('This may imply that give the level of sensor noise the requested
control performance is too optimistic, and therefore relaxation of some
requirements may be needed if the control signal should not exceed 10N.')
```

```
This may imply that give the level of sensor noise the requested control performance is too optimistic, an
```

```
inno = ym-(C*x_hat')';
figure, autocorr(inno)
```




```
disp('The autocorrelation of the innovation signal clearly shows that the
innovation is white noise, i.e. the Kalam filter is correctly estimating the
state of the system')
```

```
The autocorrelation of the innovation signal clearly shows that the innovation is white noise, i.e. the Ka
```

```
Q_sim = cov([deg2rad(theta_dot(1:100:end)) deg2rad(theta(1:100:end))
x_dot(1:100:end) x(1:100:end)]-x_hat) % estimation error covariance from
simulation
```

```
Q_sim = 4×4
   171.5128e-006    63.9758e-006   152.3577e-006    50.3426e-006
    63.9758e-006    23.8638e-006    56.8308e-006    18.7725e-006
   152.3577e-006    56.8308e-006   135.3419e-006    44.7237e-006
    50.3426e-006    18.7725e-006    44.7237e-006    15.7579e-006
```

```
display(Z) % estimation error covariance from Kalman filter design
```

```
Z = 4×4
   217.2265e-006    81.5971e-006   192.9520e-006    56.1556e-006
    81.5971e-006    30.8397e-006    72.4792e-006    20.9935e-006
   192.9520e-006    72.4792e-006   171.3969e-006    49.8883e-006
    56.1556e-006    20.9935e-006    49.8883e-006    16.4764e-006
```

```
disp('The comparison of the theoretical and experimental estimation error
covariance shows a very good agreement, reflecting that as long as the
controller can keep the system')
```

```
The comparison of the theoretical and experimental estimation error covariance shows a very good agreement
```

```
disp('close to the point of linearization the model utilized by the Kalman
filter is a very good approximation of the linear system enabling an
accurate estimate of the system state')
```

```
close to the point of linearization the model utilized by the Kalman filter is a very good approximation o
```

Linear Control Design 2 - Fall 2023 - Exam Part I

Introduction

The Part I of the Exam in Linear Control Design 2 (E23) consists of numerical exercises testing the acquisition of competences in the areas of analysis and design of control systems using modern control theory based on state space representation of system dynamics.

The scoring of each problem is clearly stated.

It is the sole responsability of the student to guarantee that the solution delivered for evaluation can be run by the examiner without the need of contacting the student. All dependencies on files external to this Matlab Live Script must be checked and included in the final delivery. **If the examiner will not be able to execute the Matlab Live Script delivered as solution by the student, the Part I will be considered failed.**

```
% Fill in your information
Exam = 'LCD2 F23'
```

```
Exam =
'LCD2 F23'
```

```
Student_Name = 'Student Name'
```

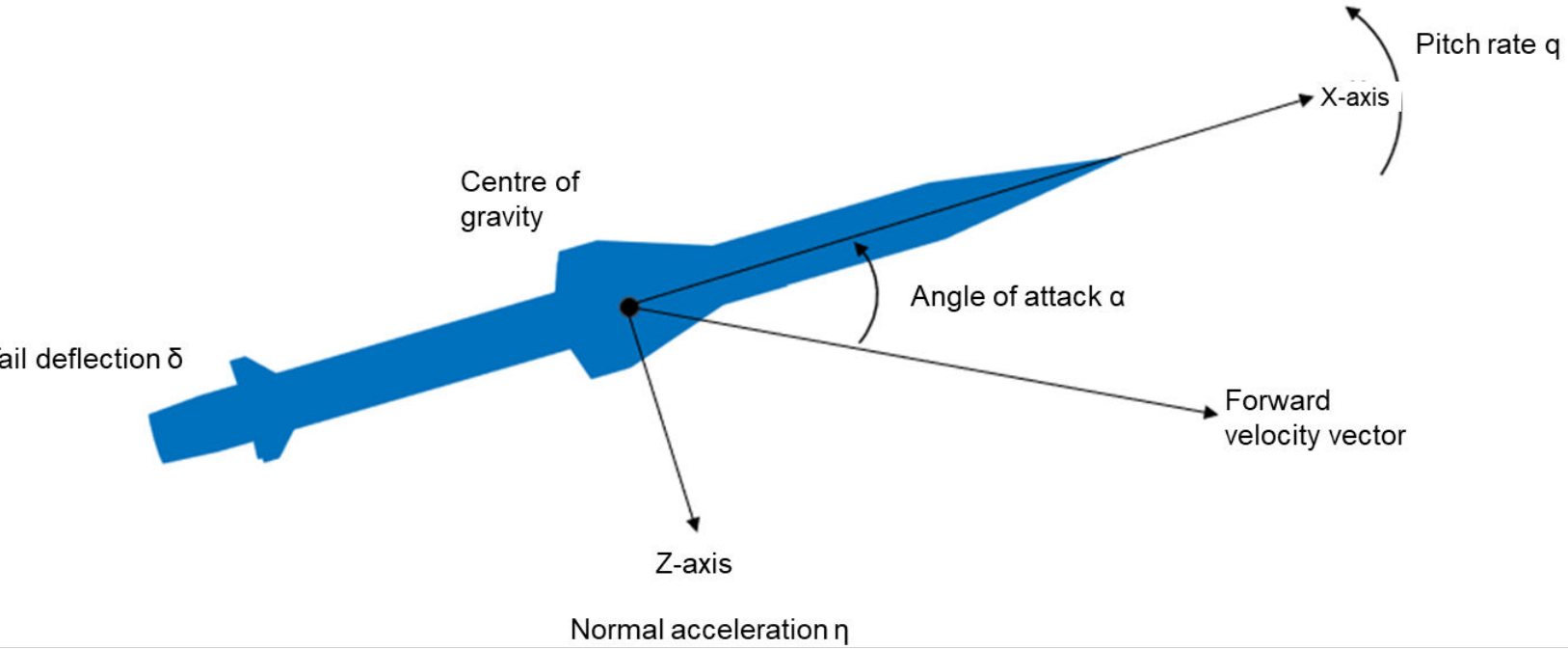
```
Student_Name =
'Student Name'
```

```
Student_Number = 'Student Number'
```

```
Student_Number =
'Student Number'
```

Missile autopilot - Problem description

Consider the missile airframe control problem illustrated in the Figure below. When the missile is flying with an angle of attack α, lift is developed. This lift may be represented as acting at a central location, called the centre of pressure. The missile is stable or unstable (without corrective tail deflections) depending on the location of the centre of presure relative to the centre of mass. The control problem requires that the autopilot generates the required tail deflection δ to produce an angle of attack corresponding to a desired manoeuvre, while stabilizing the airframe rotational motion.



Sensor measurements for feedback include missile rotation rates from rate gyros, and normal acceleration from accelerometers. Typical uncertainties to be considered in this control problem include: aerodynamic characteristics, mass and balance, wind, flexible mode dynamics, actuator and sensor nonlinearities, and sensor noise.

We wish to design an autopilot to track commanded normal acceleration manoeuvres with a steady state accuracy of 0.5% and a time constant of less than 0.2 seconds.

Missile nonlinear dynamical model

The nonlinear state equations for this control problem are given by

$$\dot{\alpha} = K_\alpha M C_n(\alpha, \delta, M) \cos\alpha + q$$
$$\dot{q} = K_q M^2 C_m(\alpha, \delta, M)$$
$$y_1 = \eta = K_z M^2 C_n(\alpha, \delta, M)$$
$$y_2 = q$$

where the aerodynamic coefficients $C_n(\alpha, \delta, M)$ and $C_m(\alpha, \delta, M)$ are given by

$$C_n(\alpha, \delta, M) = \text{sgn}(\alpha)[a_n|\alpha|^3 + b_n|\alpha|^2 + c_n(2 - M/3)|\alpha|] + d_n\delta$$
$$C_m(\alpha, \delta, M) = \text{sgn}(\alpha)[a_m|\alpha|^3 + b_m|\alpha|^2 + c_m(-7 + 8M/3)|\alpha|] + d_m\delta$$

and the function $\text{sgn}(\alpha)$ is the signum function, i.e. $\text{sgn}(\alpha) = 1$ if $\alpha > 0$; $\text{sgn}(\alpha) = -1$ if $\alpha < 0$; $\text{sgn}(\alpha) = 0$ if $\alpha = 0$. In the model above the relevant quantities are so defined

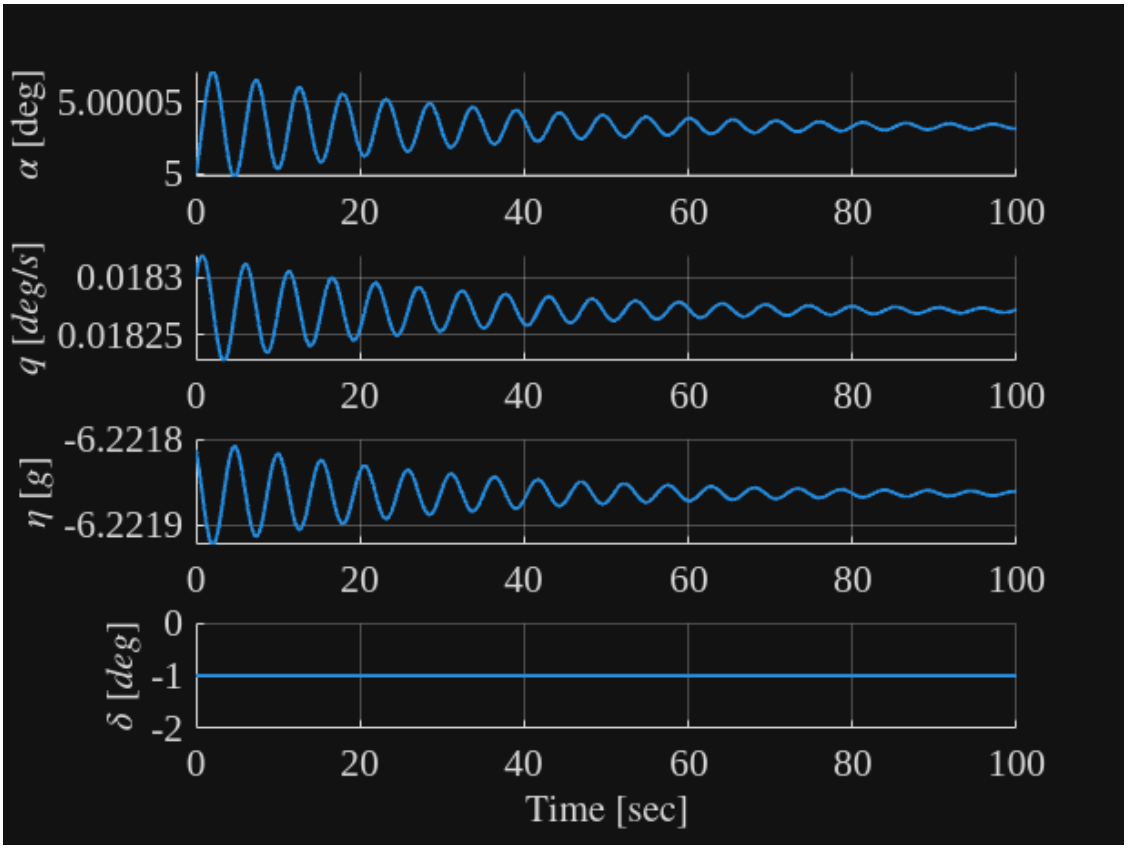
- α is the angle of attack in degrees

- q is the pitch rate in degree per second
- M is the Mach number
- δ is the tail deflection angle in degrees
- η is the normal acceleration in g's, where g is the acceleration of gravity

The nonlinear model is implemented in the Simulink file pitch_axix_missile_nonlinear_model_SimulinkYYYYC, where YYYYC refers to the Simulink version (2022b, 2022a, 2021b, etc.). The numerical values for the model parameters are provided right below.

```
% Simulator parameters (RG)
SIM_TIME = 100; % simulation time (can be changed as needed)
STEP_SIZE = 0.001; % integration step size of Simulink (can be changed as
needed)
% Change with the name of the model you are using
SIMULINK_FILENAME = 'pitch_axis_missile_nonlinear_model_Simulink2022b';

% Model paramters (RG)
P0 = 46601.86; % Static pressure at 6096 m [Pa]
S = 0.04088; % surface area [m^2]
m = 204; % mass [kg]
v_s = 315.9; % speed of sound at 6096 m [m/s]
d = 0.2286; % diameter [m]
Iy = 247.44; % pitch moment of inertia [kg*m^2]
K_alpha = 0.7*P0*S/(m*v_s); % [1/s]
K_q = 0.7*P0*S*d/Iy; % [1/s^2]
K_z = 0.7*P0*S/m; % [m/s^2]
M = 3; % Mach number
a_n = 0.000103; % [1/deg^3]
b_n = -0.00945; % [1/deg^2]
c_n = -0.1696; % [1/deg]
d_n = -0.034; % [1/deg]
a_m = 0.000215; % [1/deg^3]
b_m = -0.0195; % [1/deg^2]
c_m = 0.051; % [1/deg]
d_m = -0.206; % [1/deg]
g = 9.81; % gravity constant [m/s^2]

% Test the provided nonlinear model (RG)
time = (0:STEP_SIZE:SIM_TIME)';
x0 = [5,0.0183]'; % initial condition x0 = [alpha,q]'
delta = -0.9982; % tail deflection angle [deg]
U = delta*ones(length(time),1); % controllable input
simOut = sim(SIMULINK_FILENAME,SIM_TIME,[],[time U]);

% Plot temporal behaviour of the state of the nonlinear model (RG)
alpha = simOut.logsout.getElement(1).Values.Data;
q = simOut.logsout.getElement(2).Values.Data;
```

```
eta = simOut.logsout.getElement(3).Values.Data/g; % normalize by the gravity
constant to obtain the normal acceleration in g's

figure, h1 = subplot(4,1,1); set(h1,'FontName','times','FontSize',16)
hold on, grid on
plot(time,alpha,'LineWidth',1.5)
ylabel('$\alpha$
[deg]','FontName','times','FontSize',16,'Interpreter','latex')
h2 = subplot(4,1,2); set(h2,'FontName','times','FontSize',16)
hold on, grid on
plot(time,q,'LineWidth',1.5)
ylabel('$q$ [$deg/
s$]','FontName','times','FontSize',16,'Interpreter','latex')
h3 = subplot(4,1,3); set(h3,'FontName','times','FontSize',16)
hold on, grid on
plot(time,eta,'LineWidth',1.5)
ylabel('$\eta$ [$g$]','FontName','times','FontSize',16,'Interpreter','latex')
h4 = subplot(4,1,4); set(h4,'FontName','times','FontSize',16)
hold on, grid on
plot(time,U,'LineWidth',1.5)
ylabel('$\delta$
[$deg$]','FontName','times','FontSize',16,'Interpreter','latex')
xlabel('Time [sec]','FontName','times','FontSize',16,'Interpreter','latex')
```



Operating point and linearization

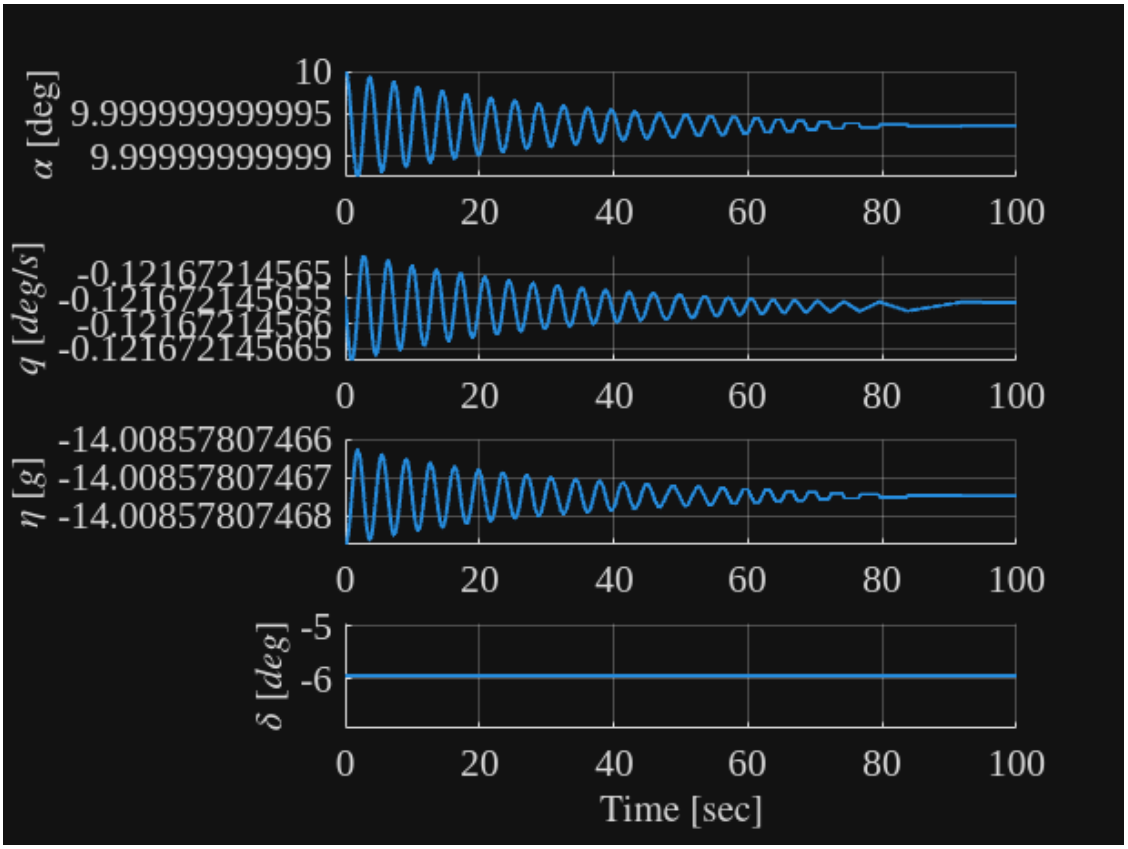
Problem 1 [2 points] Numerically determine the stationary point associated with the angle of attack $\alpha = 10$ deg , and then linearize the system around the stationary point $(\mathbf{x}_{ss}, \mathbf{u}_{ss}, \mathbf{y}_{ss},)$, where the subscript "ss" stands for stationary state.

```
% Your solution goes here

x0 = [10 0]';
delta0 = 0;
y0 = [0 0]';
[xss,uss,yss,dx]=trim(SIMULINK_FILENAME,x0,delta0,y0,[1],[],[])
```

```
xss = 2×1
   10.0000
   -0.1217
uss = -5.9466
yss = 2×1
 -137.4242
   -0.1217
dx = 2×1
10^-10 ×
    0.0170
   -0.1943
```

```
x0 = xss;
u0 = uss;

% The stationary condition is verified through a simulation of the nonlinear
system
U = ones(length(time),length(uss)).*uss';
simOut = sim(SIMULINK_FILENAME,SIM_TIME,[],[time U]);

% Plot temporal behaviour of the state of the nonlinear model
alpha = simOut.logsout.getElement(1).Values.Data;
q = simOut.logsout.getElement(2).Values.Data;
eta = simOut.logsout.getElement(3).Values.Data/g; % normalize by the gravity
constant to obtain the normal acceleration in g's

figure, h1 = subplot(4,1,1); set(h1,'FontName','times','FontSize',16)
hold on, grid on
plot(time,alpha,'LineWidth',1.5)
ylabel('$\alpha$
[deg]','FontName','times','FontSize',16,'Interpreter','latex')
h2 = subplot(4,1,2); set(h2,'FontName','times','FontSize',16)
hold on, grid on
plot(time,q,'LineWidth',1.5)
ylabel('$q$ [$deg/
s$]','FontName','times','FontSize',16,'Interpreter','latex')
h3 = subplot(4,1,3); set(h3,'FontName','times','FontSize',16)
hold on, grid on
plot(time,eta,'LineWidth',1.5)
ylabel('$\eta$ [$g$]','FontName','times','FontSize',16,'Interpreter','latex')
h4 = subplot(4,1,4); set(h4,'FontName','times','FontSize',16)
```

```
hold on, grid on
plot(time,U,'LineWidth',1.5)
ylabel('$\delta$
[$deg$]','FontName','times','FontSize',16,'Interpreter','latex')
xlabel('Time [sec]','FontName','times','FontSize',16,'Interpreter','latex')
```




```
% Linearization of the system is performed using linmod
[Aol,Bol,Col,Dol] = linmod(SIMULINK_FILENAME,x0,u0);
sys_ol = ss(Aol,Bol,Col,Dol)
```

```
sys_ol =

  A =
           x1        x2
   x1  -0.06182         1
   x2    -3.044         0

  B =
           u1
   x1  0.001771
   x2    -2.284

  C =
         x1      x2
   y1  -19.28       0
   y2       0       1

  D =
       u1
   y1  -2
   y2   0
```

```
Continuous-time state-space model.
Model Properties
```

Stability analysis

Problem 2 [2 points] Assess the internal stability of the linear system and graphically show the eigenmodes of the system.

```
% Your solution goes here

% The internal stability is assessed by checking the position of the
% eigenvalues in the complex plane. The external stability is evaluated
% looking at tht position of the poles.

lambda_ol = eig(Aol); % open loop eignevalues
disp('The open loop eignevalues are')
```

```
The open loop eignevalues are
```

```
disp(lambda_ol)
```

```
  -0.0309 + 1.7443i
  -0.0309 - 1.7443i
```

```
disp(['Since the real part of the both eigenvalues is negative then the
system' ...
    ' is asymptotically stable'])
```

```
Since the real part of the both eigenvalues is negative then the system is asymptotically stable
```

```
[numG,denG] = ss2tf(sys_ol.a,sys_ol.b,sys_ol.c,sys_ol.d,1);
p_ol = roots(denG); % open loop poles
disp('The open loop poles are')
```

```
The open loop poles are
```

```
disp(p_ol)
```

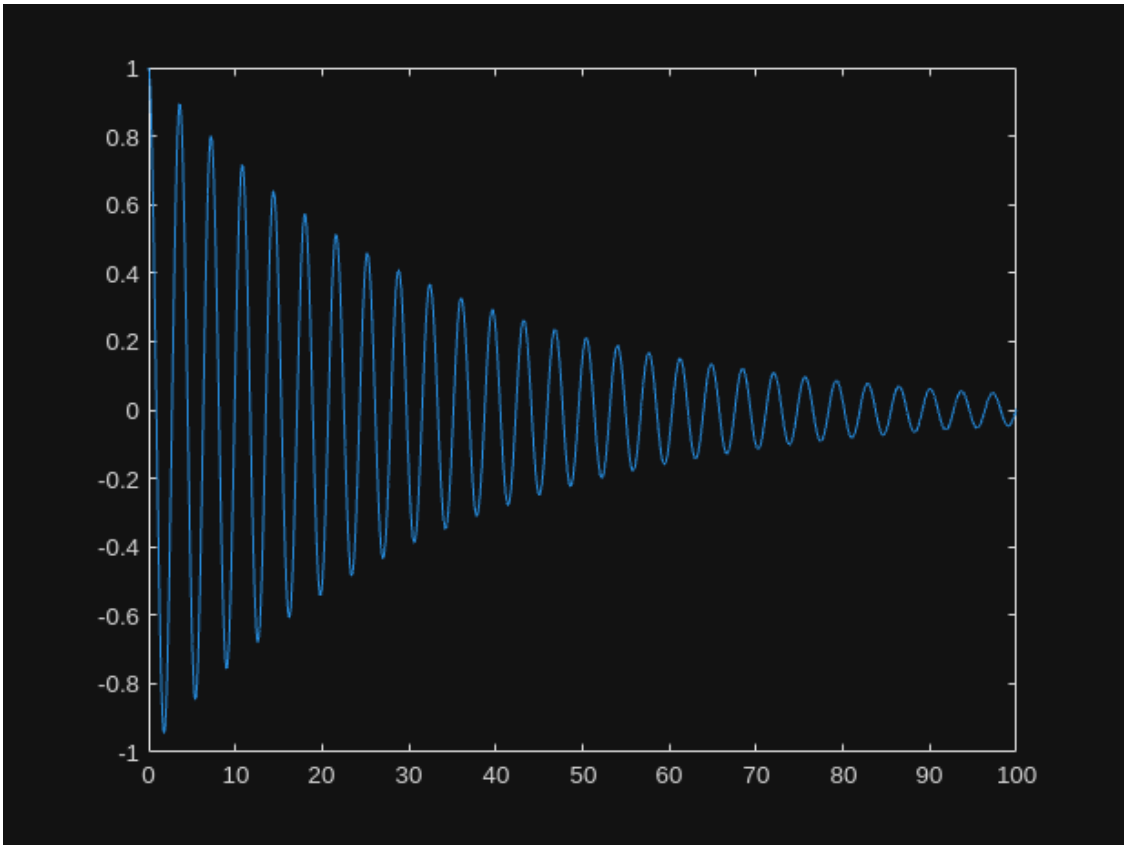
```
  -0.0309 + 1.7443i
  -0.0309 - 1.7443i
```

```
disp(['Since the real part of both poles is negative then the system' ...
    ' is BIBO stable'])
```

```
Since the real part of both poles is negative then the system is BIBO stable
```

```
eigM1 = exp(lambda_ol(1)*time);
figure, plot(time,eigM1)
```

```
Warning: Imaginary parts of complex X and/or Y arguments ignored.
```




```
damp(Aol)
```

```
        Pole              Damping       Frequency       Time Constant
                                      (rad/TimeUnit)     (TimeUnit)

  -3.09e-02 + 1.74e+00i     1.77e-02       1.74e+00          3.24e+01
  -3.09e-02 - 1.74e+00i     1.77e-02       1.74e+00          3.24e+01
```

Design of autopilot for tracking commanded normal acceleration manoeuvres

The control objective is to design an autopilot to track commanded normal acceleration manoeuvres, while stabilizing the airframe pitch motion. Given a commanded step change of normal acceleration of $\pm 1g$, the design autopilot must fulfill the following with a steady state accuracy of 0.5% and a time constant of less than 0.2 seconds.

Closed-loop system requirements

1) The steady state error $e_\eta = \eta_c - \eta$ is less than 0.5%, where η_c is the commanded normal acceleration.

2) The settling time is less than or equal to 1 second

3) The tail angular deflection does not exceed 25 degrees, i.e. $|\delta| \leq 25 \text{ deg}$

Problem 3 [4 points] Under the assumption that the state is fully accessible design a discrete time controller that meets the aforementioned specifications on the nonlinear system using the eigenstructure assignment method.

```
% Your solution goes here

% Design an integral controller to avoid steady state error on the tracked
% commanded normal acceleration
% The settling time requirement is used to determine the largest time
% constant in the closed-loop system

% Sampling time
Tsett = 0.75; % desired settling time [s]
tau_sett = Tsett/5; % if I approximate the behaviour of the closed-loop
system as first order system
tau_ol = 1/abs(real(lambda_ol(1)));
Ts = min([tau_ol tau_sett])/10
```

```
Ts = 0.0150
```

```
sys_ol_dt = c2d(sys_ol,Ts);

% Full state feedback with integral action
F = sys_ol_dt.a;
G = sys_ol_dt.b;
% Compute the reachability matrix and check if the system is reachable
Mr = ctrb(F,G);
if rank(Mr) == size(F,1)
    disp('The open loop system is reachable')
end
```

```
The open loop system is reachable
```

```
Fi = [F zeros(size(F,1),1);-Ts/g*Col(1,:) 1];
Gi = [G;-Ts/g*Dol(1,:)];
Gr = [0;0;Ts]; % reference input matrix where the reference command is
measured in g's
Ci = [Col,zeros(2,1)];
Di = Dol;

% Desired eigenvalues
zeta = 0.9;
alpha_eig = -1/tau_sett; % real part of the eigenvalue
wn = - alpha_eig/zeta;
beta_eig = wn^2 - alpha_eig^2;

lambda_des = [alpha_eig+j*beta_eig, alpha_eig-j*beta_eig, 1.1*alpha_eig];
lambda_des_dt = exp(lambda_des.*Ts)
```

```
lambda_des_dt = 1×3 complex
   0.8938 + 0.1409i   0.8938 - 0.1409i   0.8958 + 0.0000i
```

```
K = place(Fi,Gi,lambda_des_dt);
Kfs = K(:,1:2);
Ki = -K(:,3);

sys_cl_dt = ss(Fi-Gi*K,Gr,[Ci-Di*K;-K],[],Ts);
lambda_cl_dt = eig(sys_cl_dt)
```

```
lambda_cl_dt = 3×1 complex
   0.8938 + 0.1409i
   0.8938 - 0.1409i
   0.8958 + 0.0000i
```

Problem 4 [2 points] Verify your design through simulation on the linear and nonlinear model, and discuss to which degree the control requirements are fulfilled.

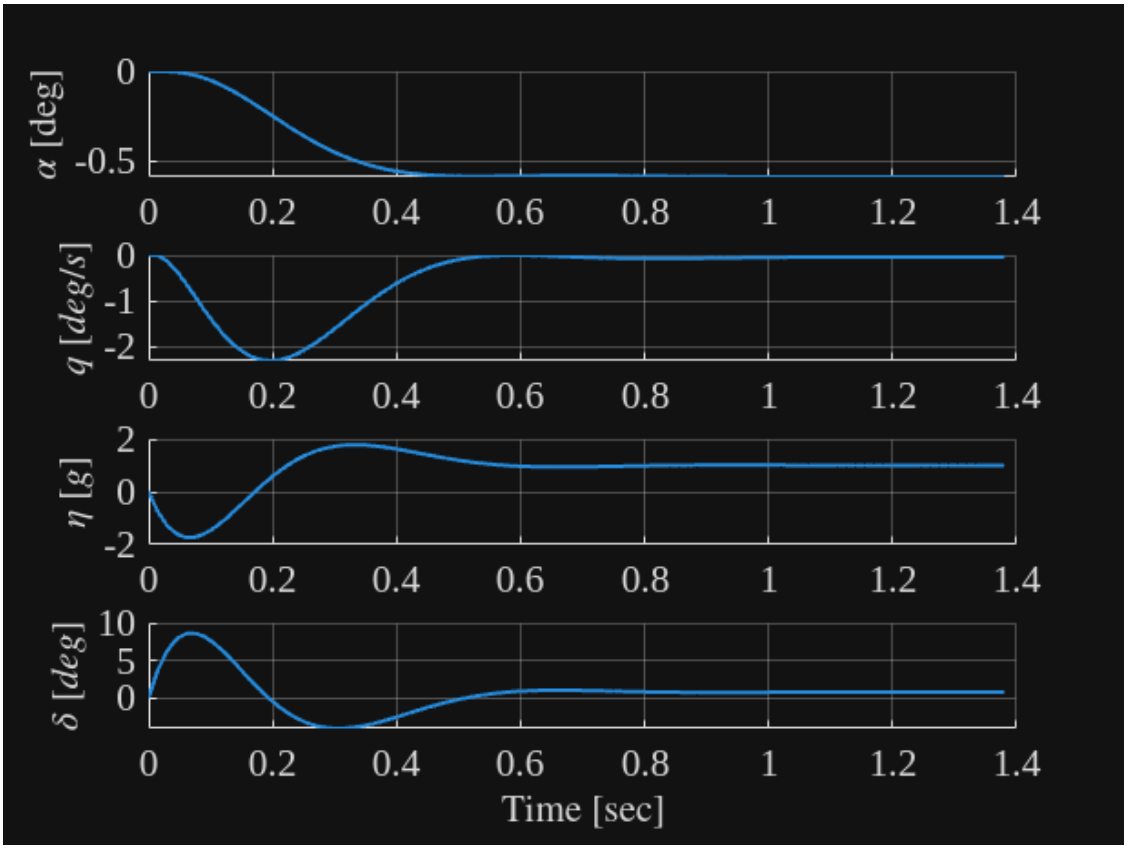
```
% Verification on the linearized model
StepAmplitude = 1;
opt = stepDataOptions('StepAmplitude',StepAmplitude);
[y,t,x] = step(sys_cl_dt,opt);
e_eta = StepAmplitude - y(end,1)/g
```

```
e_eta = 4.2114e-04
```

```
max_delta = max(y(:,3))
```

```
max_delta = 8.6263
```

```


figure, h1 = subplot(4,1,1); set(h1,'FontName','times','FontSize',16)
hold on, grid on
plot(t,x(:,1),'LineWidth',1.5)
ylabel('$\alpha$
[deg]','FontName','times','FontSize',16,'Interpreter','latex')
h2 = subplot(4,1,2); set(h2,'FontName','times','FontSize',16)
hold on, grid on
plot(t,x(:,2),'LineWidth',1.5)
ylabel('$q$ [$deg/
s$]','FontName','times','FontSize',16,'Interpreter','latex')
h3 = subplot(4,1,3); set(h3,'FontName','times','FontSize',16)
hold on, grid on
plot(t,y(:,1)/g,'LineWidth',1.5)
ylabel('$\eta$ [$g$]','FontName','times','FontSize',16,'Interpreter','latex')
h4 = subplot(4,1,4); set(h4,'FontName','times','FontSize',16)
hold on, grid on
plot(t,y(:,3),'LineWidth',1.5)
ylabel('$\delta$
[$deg$]','FontName','times','FontSize',16,'Interpreter','latex')
xlabel('Time [sec]','FontName','times','FontSize',16,'Interpreter','latex')
```



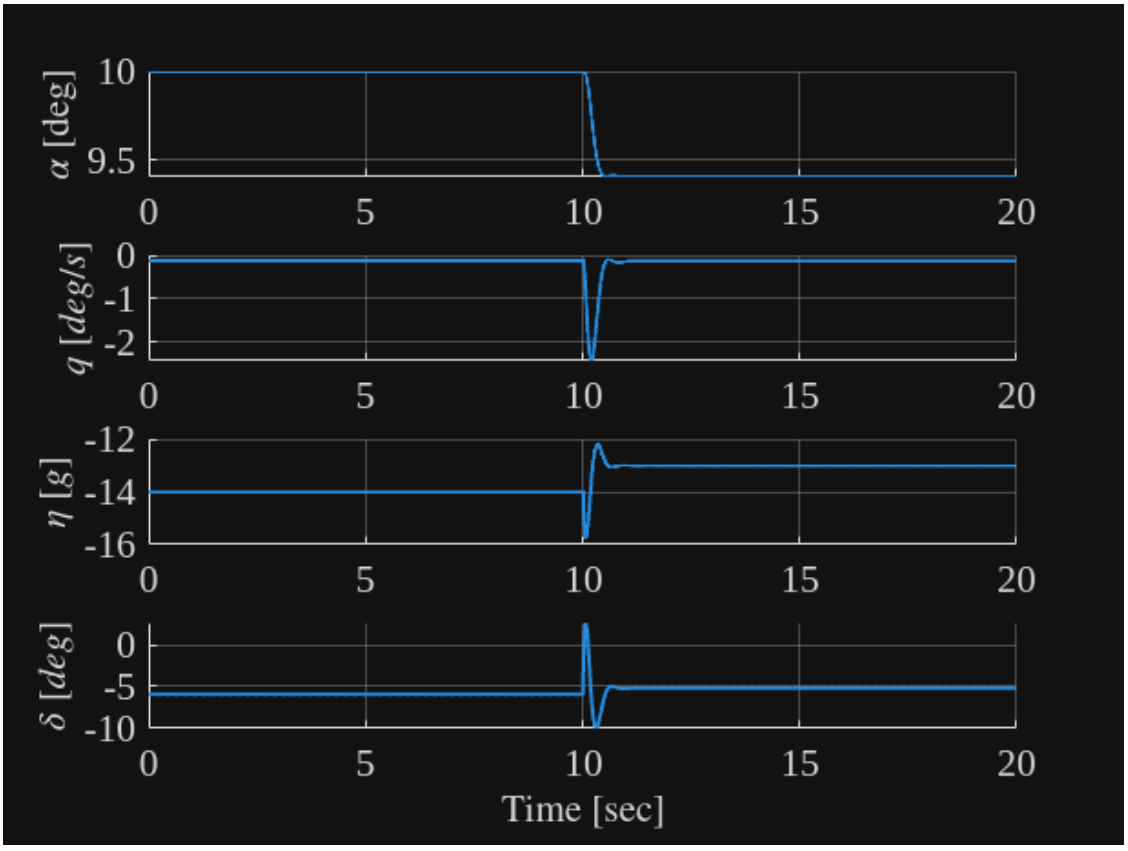

```
% Verification on the nonlinear model

% The stationary condition is verified through a simulation of the nonlinear
system
SIM_TIME = 20;
SIMULINK_FILENAME = 'pitch_axis_missile_IntegralControl_Simulink2022b';
StepAmplitude = 1;
simOut = sim(SIMULINK_FILENAME,SIM_TIME);

% Plot temporal behaviour of the state of the nonlinear model
alpha = simOut.logsout.getElement(1).Values.Data;
q = simOut.logsout.getElement(2).Values.Data;
eta = simOut.logsout.getElement(3).Values.Data/g; % normalize by the gravity
constant to obtain the normal acceleration in g's
delta = simOut.logsout.getElement(4).Values.Data;
time = simOut.logsout.getElement(3).Values.Time;
time_delta = simOut.logsout.getElement(4).Values.Time;

e_eta = (yss(1)/g+StepAmplitude) - eta(end)
```

```
e_eta = 2.3093e-14
```

```
max_delta = max(delta)
```

```
max_delta = 2.6800
```

```
figure, h1 = subplot(4,1,1); set(h1,'FontName','times','FontSize',16)
hold on, grid on
plot(time,alpha,'LineWidth',1.5)
ylabel('$\alpha$
[deg]','FontName','times','FontSize',16,'Interpreter','latex')
h2 = subplot(4,1,2); set(h2,'FontName','times','FontSize',16)
hold on, grid on
plot(time,q,'LineWidth',1.5)
ylabel('$q$ [$deg/
s$]','FontName','times','FontSize',16,'Interpreter','latex')
h3 = subplot(4,1,3); set(h3,'FontName','times','FontSize',16)
hold on, grid on
plot(time,eta,'LineWidth',1.5)
ylabel('$\eta$ [$g$]','FontName','times','FontSize',16,'Interpreter','latex')
h4 = subplot(4,1,4); set(h4,'FontName','times','FontSize',16)
hold on, grid on
plot(time_delta,delta,'LineWidth',1.5)
ylabel('$\delta$
[$deg$]','FontName','times','FontSize',16,'Interpreter','latex')
xlabel('Time [sec]','FontName','times','FontSize',16,'Interpreter','latex')
```



Kalman filtering for output feedback control

The angle of attack α is not measured, therefore the designed autopilot cannot be implemented unless the state vector is estimated based on the available measurements. Now, we assume that the accelerometer and the rate gyro are affected by white noise, i.e.

$$y_1 = \eta + \nu_\eta$$
$$y_2 = q + \nu_q$$

where ν_η is zero mean white noise with amplitude $A_{\nu_\eta} = \pm 0.003 \text{ m/s}^2$, and ν_q is zero mean white noise with amplitude $A_{\nu_q} = \pm 0.006 \text{ deg/s}$.

Problem 5 [4 points] Design a discrete time Kalman filter to estimate the state vector $\mathbf{x} = [\alpha, q]^T$ and the measured output $y_1 = \eta$.

```
% Your solution goes here
% Kalman filter architecture
A_kf = Aol;
B_kf = Bol;
C_kf = Col;
D_kf = Dol;
Bv_kf = [Aol(1,1) 0;0 Aol(2,2)];

% Observability analysis
Mo = obsv(A,C);
% Test if the observability matrix has full column rank,
% and if the system is therefore observable.
if rank(Mo) >= size(A,1)
    disp('System is observable!');
else
    disp('System is NOT observable!');
end
```

```
System is observable!
```

```
% Design of Kalman filter
v11 = 0.003;
v22 = 0.006;
V1 = [v11 0;0 v22]; % process noise intensity
V2 = zeros(2,2); % measurement noise intensity

% Discretization
V1d = Bv_kf*V1*Bv_kf'*Ts;
V2d = V2/Ts;
Gv_kf = eye(2);
[F_kf,~] = c2d(A_kf,B_kf,Ts);

[Lkf,Q_th] = lqed(F_kf,Gv_kf,C_kf,V1d,V2d,Ts);
```

```
Lkf = 2×2
   -0.0519         0
         0    1.0000
```

```
[K,S,e] = dlqr(Lkf,B,Q_th,R,N)
```

```
Error using dlqr (line 34)
The N matrix must have 2 rows and 1 columns.
```

Problem 6 [4 points] Assess the estimation performance of the Kalman filter when tested on the closed loop nonlinear system for commanded step changes of normal acceleration of $\pm 1g$.

```
% Your solution goes here

% plotting of estimates Vs true states

% comparison of numerical estimation error covariance Vs theoretical

% autocorrelation of innovation signal i = y - y_hat
```