Assignment 5

TTK4215

Elias Olsen Almenningen

Version: 1.0	Date: October 2, 2023

Contents

Problem 12

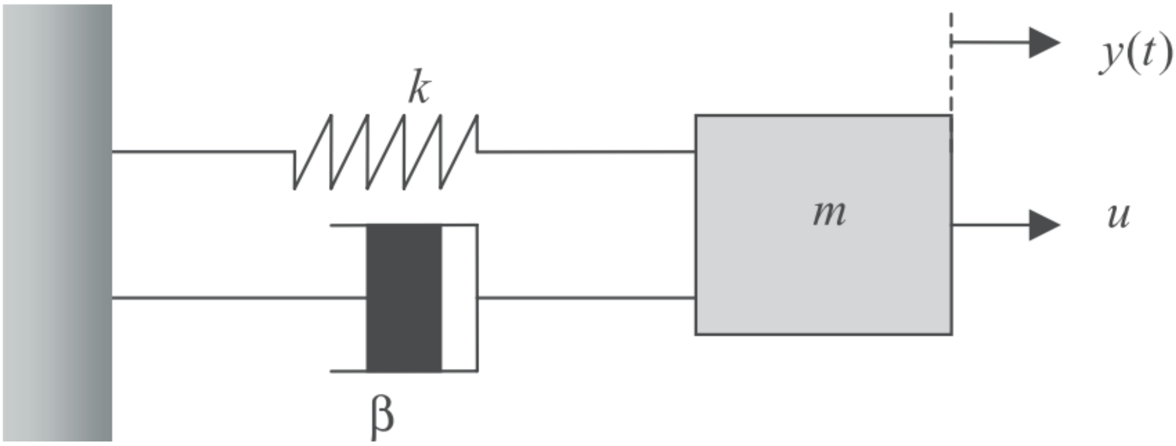


Figure 1: Mass-spring-damper system

Consider the mass–spring–damper system shown in Figure 1, where β is the damping coefficient, k is the spring constant, u is the external force, and $y(t)$ is the displacement of the mass m resulting from the force u.

c) ***Design an LS algorithm to estimate the constants m, β, k when y, u can be measured at each time t:***

Transforming the equation of motion into an SPM:

$$z = {\theta^*}^\top \phi \tag{1}$$

by taking the Laplace transform, defining the filter $\Lambda(s) = s^2 + \lambda_1 s + \lambda_0$ and rearranging the equation gives:

$$\frac{ms^2y + \beta sy + k}{\Lambda(s)} = \frac{u}{\Lambda(s)} \tag{2}$$

making the SPM:

$$\underbrace{\frac{s^2}{\Lambda(s)}[y]}_{z} = \underbrace{\begin{bmatrix} \frac{\beta}{m} & \frac{k}{m} & \frac{1}{m} \end{bmatrix}}_{\theta^{*T}} \underbrace{\begin{bmatrix} -\frac{s}{\Lambda(s)}[y] \\ -\frac{1}{\Lambda(s)}[y] \\ \frac{1}{\Lambda(s)}[u] \end{bmatrix}}_{\phi} \tag{3}$$

Defining the estimate of z as

$$\hat{z} = \theta^\top \phi \tag{4}$$

and the normalized estimation error as

$$\epsilon = \frac{z - \hat{z}}{m^2} \tag{5}$$

where $m = 1 + n_s^2$, where n_s is chosen s.t. $\frac{\phi}{m} \in \mathcal{L}_\infty$. The normal value for this is often $n_s^2 = \alpha \phi^\top \phi$, $\alpha > 0$.

The following seciton is almost copied directly from [1] and the lecture notes, and is mostly for my own learning.

The cost function is defined as

$$J(\theta) = \frac{1}{2}\int_0^t e^{-\beta(t-\tau)} \frac{\left[z(\tau) - \theta^T(t)\phi(\tau)\right]^2}{m_s^2(\tau)} d\tau + \frac{1}{2} e^{-\beta t} \left(\theta(t) - \theta_0\right)^T Q_0 \left(\theta(t) - \theta_0\right) \tag{6}$$

where $Q_0 = Q_0^T > 0, \beta \geq 0$ are the design constants and $\theta_0 = \theta(0)$ is the initial parameter estimate. This cost function is a generalization of $J(\theta) = \frac{1}{2}\int_0^t |z(\tau) - \theta(t)\phi(\tau)|^2 d\tau$ to include the possible discounting of past data and a penalty on the initial error between the estimate θ_0 and $\theta(0)$.

Since $\frac{z}{m_s}, \frac{\phi}{m_s} \in \mathcal{L}_\infty$, $J(\theta)$ is a convex function of θ over $\mathcal{R}^t$ at each time t. Therefore, any local minimum is also global and satisfies

$$\nabla J(\theta(t)) = 0 \forall t \geq 0. \tag{7}$$

The LS algorithm for generating $\theta(t)$, the estimate of θ^*, in 4 is therefore obtained by solving

$$\nabla J(\theta) = 0 \forall t \geq 0. \tag{8}$$

for $\theta(t)$:

$$\begin{aligned}
\nabla_\theta J(\theta) &= \int_0^t e^{-\beta(t-\tau)} \frac{\left[z(\tau) - \theta^T(t)\phi(\tau)\right]}{m_s^2(\tau)} \left(-\phi(t)\right) d\tau + e^{-\beta t} Q_0(\theta - \theta_0) \\
&= -\int_0^t e^{-\beta(t-\tau)} \frac{z(\tau)\phi(\tau)}{m^2(\tau)} d\tau + \left[\int_0^t e^{-\beta(t-\tau)} \frac{\phi(\tau)\phi^T(\tau)}{m^2(\tau)} d\tau\right] \theta(t) + e^{-\beta t)} Q_0 \theta(t) - e^{-\beta t} Q_0 \theta_0 \\
&= \underbrace{\left[e^{-\beta t)} Q_0 + \int_0^t e^{-\beta(t-\tau)} \frac{\phi(\tau)\phi^T(\tau)}{m^2(\tau)} d\tau\right]}_{P^{-1}} \theta(t) - \left[e^{-\beta t} Q_0 \theta_0 + \int_0^t e^{-\beta(t-\tau)} \frac{z(\tau)\phi(\tau)}{m^2(\tau)} d\tau\right] \\
&= 0
\end{aligned}$$

where the matrix P^{-1} is defined the way it is sucht that it can be inverted later on. We can then write that

$$\begin{gathered}
P^{-1}\theta(t) = e^{-\beta t} Q_0 \theta_0 + \int_0^t e^{-\beta(t-\tau)} \frac{z(\tau)\phi(\tau)}{m^2(\tau)} d\tau \\
\downarrow \\
\theta(t) = P(t) \left[e^{-\beta t} Q_0 \theta_0 + \int_0^t e^{-\beta(t-\tau)} \frac{z(\tau)\phi(\tau)}{m^2(\tau)} d\tau\right]
\end{gathered}$$

The LS-algorithm can then be found by taking the derivative of this expression:

$$\begin{aligned}
\dot{\theta} &= P(t)\left[-\beta e^{-\beta t}Q_0\theta_0 + \frac{z(t)\phi(t)}{m^2} - \underbrace{\beta\int_0^t e^{-\beta(t-\tau)}\frac{z(\tau)\phi(\tau)}{m^2(\tau)}d\tau}_{\text{Comes from Leibniz' rule}}\right] + \dot{P}P^{-1}\theta(t) \\
&= P(t)\left(-\beta P^{-1}(t)\theta(t) + \frac{z(t)\phi(t)}{m^2(t)}\right) + \dot{P}(t)P^{-1}\theta(t) \\
&= -\beta\theta(t) + P(t)\frac{z(t)\phi(t)}{m^2(t)} + \dot{P}(t)P^{-1}\theta(t)
\end{aligned}$$

We then need to find an expression for $\dot{P}$. Using the following definitions:

$$\frac{d}{dt}P^{-1} = \beta P^{-1} + \frac{\phi(t)\phi^T(t)}{m^2}, P^{-1}(0) = Q_0 \tag{9}$$

$$0 = \frac{d}{dt}I = \frac{d}{dt}\left(PP^{-1}\right) = \dot{P}P^{-1} + P\frac{d}{dt}P^{-1} \tag{10}$$

$\dot{P}$ becomes:

$$\dot{P} = -P\frac{d}{dt}\left(P^{-1}\right)P = -P\left(\beta P^{-1} + \frac{\phi(t)\phi^T(t)}{m^2}\right)P \tag{11}$$

$$= \beta P - P\frac{\phi(t)\phi^T(t)}{m^2}P, \tag{12}$$

$$P(0) = Q_0^{-1} \tag{13}$$

We can then finally state our algorithm by inserting this into our equation for $\dot{\theta}$:

$$\begin{aligned}
\dot{\theta} &= -\beta\theta(t) + P(t)\frac{z(t)\phi(t)}{m^2(t)} + \dot{P}(t)P^{-1}\theta(t) \\
&= -\beta\theta(t) + P(t)\frac{z(t)\phi(t)}{m^2(t)} + \left(\beta P - P\frac{\phi(t)\phi^T(t)}{m^2}P\right)P^{-1}\theta(t) \\
&= -\beta\theta(t) + P(t)\frac{z(t)\phi(t)}{m^2(t)} + \beta\theta(t) - P(t)\frac{\phi(t)\phi^T(t)}{m^2}\theta(t) \\
&= P(t)\left(\frac{z(t)\phi(t)}{m^2(t)} - \underbrace{\frac{\phi(t)\phi^T(t)}{m^2(t)}\theta(t)}_{=\frac{\theta^T\phi\phi}{m^2}}\right) \\
&= P(t)\underbrace{\left(\frac{z(t)-\theta^T\phi(t)}{m^2(t)}\right)}_{\varepsilon}\phi(t) \\
&= P(t)\varepsilon(t)\phi(t)
\end{aligned}$$

The LS algorithm is thus:

$$\dot{\theta} = P(t)\varepsilon(t)\phi(t),\ \theta(0) = \theta_0$$
$$\dot{P} = \beta P - P\frac{\phi(t)\phi^T(t)}{m^2}P,\ P(0) = Q_0^{-1} > 0$$

d/e) *Simulate your algorithms assuming* $m = 20kg$*,* $\beta = 0.1kg/s$*,* $k = 5kg/s^2$*, and inputs* u *of your choice*

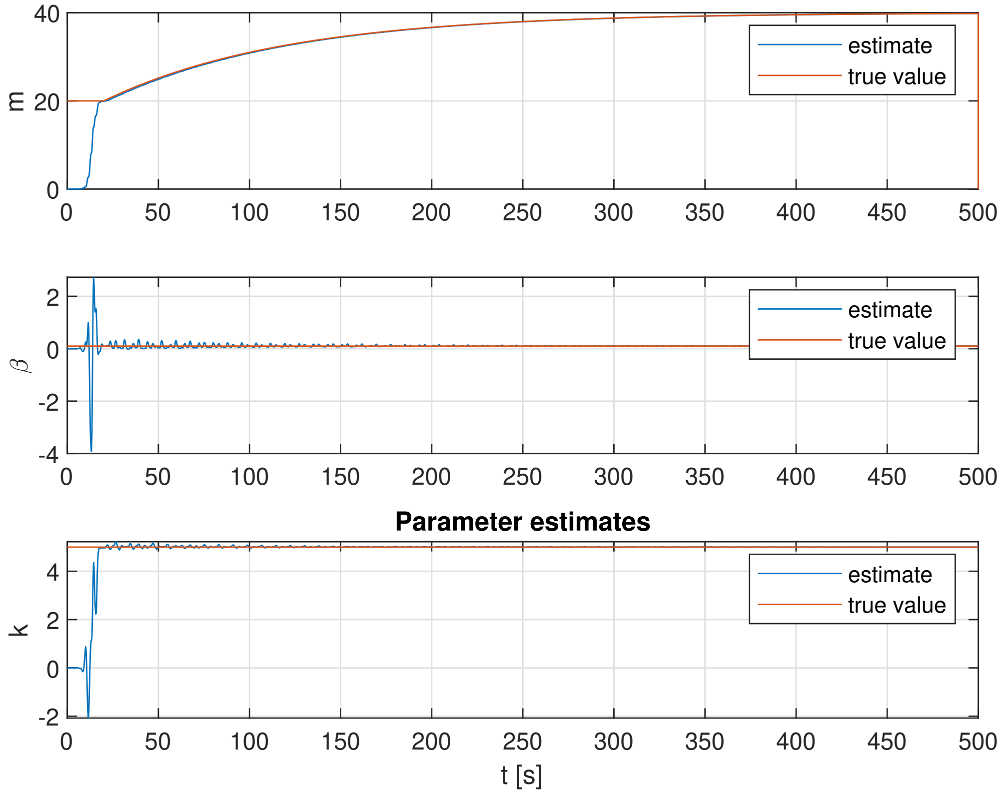


Figure 2: $\lambda_1 = \lambda_0 = 6$, $Q_0 = eye(3)$

Problem 13

Consider the mass-spring-damper system shown in Figure 3

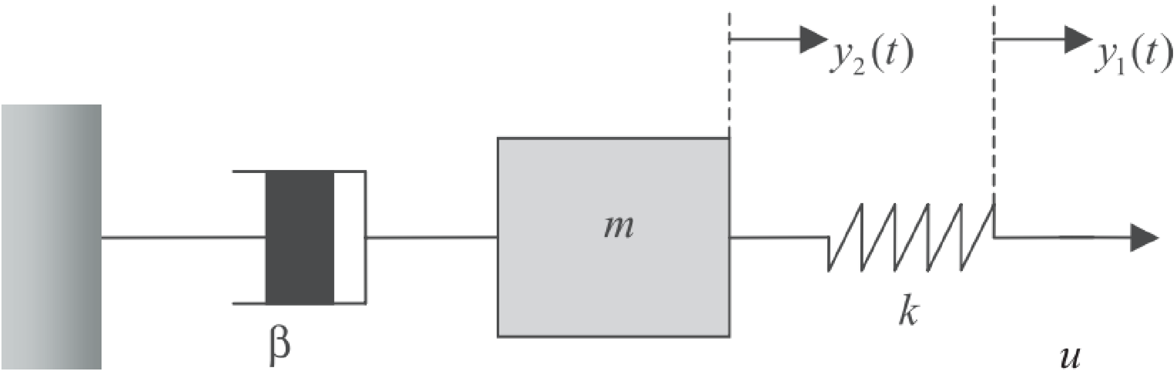


Figure 3: The mass-spring-damper system for Problem 13

a)

Verify that the equations of motion are given by

$$k(y_1 - y_2) = u, \tag{14}$$

$$k(y_1 - y_2) = m\ddot{y}_2 + \beta\dot{y}_2 \tag{15}$$

Eq. (14) is the definition of the spring force, where the applied force u is equal to the spring force ky, where the length of the spring is given by $y = y_1 - y_2$, as seen in Figure 4.

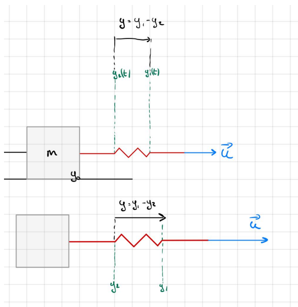


Figure 4: Lenght of spring when u increases

Eq. (15) is Newton's second law applied on the mass m:

$$\begin{aligned} F_{total} &= m\ddot{y} \\ k(y_1 - y_2) - \beta\dot{y}_2 &= m\ddot{y}_2 \\ k(y_1 - y_2) &= m\ddot{y}_2 + \beta\dot{y}_2 \end{aligned}$$

b)

If y_1, y_2, u can be measured at each time t, design an online parameter estimator to estimate the constants k, m, and β.

In order to estimate k, m and β, we have use two estimators: one to estimate k using eq. (14) and another to m and β using eq. (15), where $k(y_1 - y_2)$ is replaced with u.

Estimating k

Rewriting eq. (14), and since there are no derivatives there is no need for filtering, gives us

$$\underbrace{u}_{z_1} = \underbrace{[k]}_{\theta_1^{*T}} \underbrace{[y_1 - y_2]}_{\phi_1} \tag{16}$$

Estimating m and β, using $u = k(y_1 - y_2)$

Rewrite eq. (15) and taking the Laplace transform give us

$$u = s^2 y_2 m + s y_2 \beta \tag{17}$$

This equation has first and second derivatives so filtering is required. Defining the filter $\Lambda(s) = s^2 + \lambda_1 s + \lambda_0$ and writing the equation as an SPM gives us

$$\underbrace{\frac{1}{\Lambda(s)}[u]}_{z_2} = \underbrace{\begin{bmatrix} m & \beta \end{bmatrix}}_{\theta_2^{*T}} \underbrace{\begin{bmatrix} \frac{s^2}{\Lambda(s)}[y_2] \\ \frac{s}{\Lambda(s)}[y_2] \end{bmatrix}}_{\phi_2} \tag{18}$$

2d)

Using least squares, some of the MatLab code becomes (... indicates unincluded code):

```
%% Insert true system
beta                    = 0.2;
m                       = 15;
k                       = 2;
% 1/Lambda [u] = s\^2 / Lambda [y\_2] * m + s/Lambda [y\_2] * beta
A                       = [0 0 0; 0 0 1;0 0 -beta/m];
B                       = [0; 0; 1/m];
...
% Define input as a function of t
u            = 5 * sin(2 * t) + 10.5;
beta_1_ls    = .01;    % Forgetting factor
beta_2_ls    = 10;    % Forgetting factor
...
% Initial estimates
```

```
theta_1(:,1) = 1;            % Initial estimate of k
theta_2(:,1) = [1; 1];    % Initial estimate of m and beta

P_1 = 1;
P_2 = inv(eye(2));

...
 % Simulate true system
    x_dot               = A*x(:, n) + B*u(n);
    x(:, n+1)           = x(:, n) + h*x_dot;

    y_2                 = x(2, n);
    y_1                 = u(n)/k + y_2;

    % Generate z and phi by filtering known signals
    x_z_2_n               = x_z_2 + (A_f*x_z_2 + B_f*u(n))*h;
    z_2                   = C_f_1*x_z_2;                   % z_2 = 1/Lambda [u]


    x_phi_2_n             = x_phi_2 + (A_f*x_phi_2 + B_f*y_2)*h;

    phi_2                 = [ (C_f_3*x_phi_2 + D_f_3*y_2);  % s^2/Lambda * y_2
                              (C_f_2*x_phi_2 + D_f_2*y_2)   % s/Lamnba * y2
                            ];

    % Generate first system
    z_1 = u(n);
    phi_1 = y_1-y_2;

    % Calculate estimation error
    % n_s_1 = phi_1'*phi_1;
    % n_s_2 = phi_2'*phi_2;

    epsilon_1             = z_1 - (theta_1(:,n)'*phi_1);%/(1+n_s_1);
    epsilon_2             = z_2 - (theta_2(:,n)'*phi_2);%/(1+n_s_2);

    % Update law
    theta_1_dot           = P_1 * epsilon_1 * phi_1;
    P_1_dot               = beta_1_ls * P_1 - P_1 * (phi_1 * phi_1') * P_1;

    theta_1(:, n+1)       = theta_1(:, n) + theta_1_dot*h;
    P_1                    = P_1 + P_1_dot * h;

    theta_2_dot           = P_2 * epsilon_2 * phi_2 ;
    P_2_dot               = beta_2_ls * P_2 - P_2 * (phi_2 * phi_2') * P_2;

    theta_2(:, n+1)       = theta_2(:, n) + theta_2_dot*h;
    P_2                   = P_2 + P_2_dot * h;

    % Set values for next iteration
    x_phi_2               = x_phi_2_n;
    x_z_2                 = x_z_2_n;
...
```

Using Inst. Cost the MatLab code becomes:

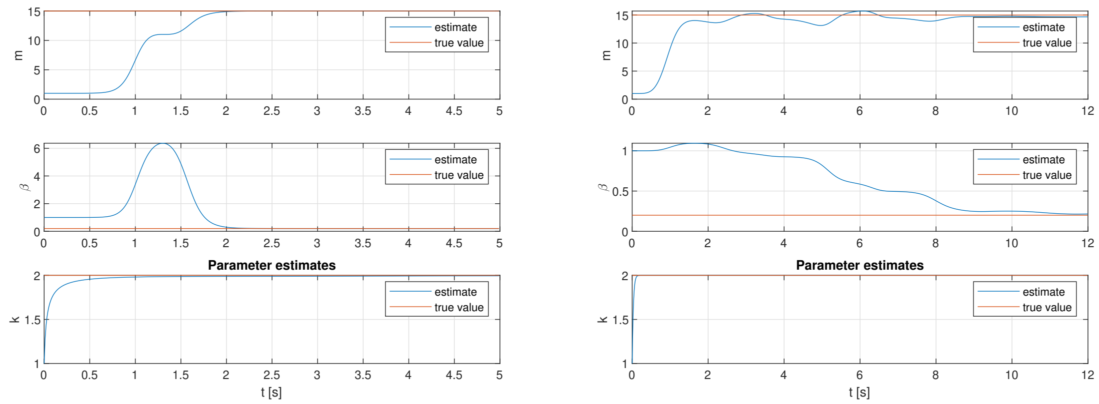
```
...
% Define input as a function of t
u             = 5 * sin(2 * t) + 10.5;
gamma_1       = 1;
gamma_2       = diag([50 1]);
...
 % Calculate estimation error
    n_s_1 = phi_1'*phi_1;
    n_s_2 = phi_2'*phi_2;

    epsilon_1            = z_1 - theta_1(:,n)' * phi_1;
    epsilon_2            = z_2 - theta_2(:,n)' * phi_2;

    % Update law
    theta_2_dot          = gamma_2 * epsilon_2 * phi_2;
    theta_2(:, n+1)      = theta_2(:, n) + theta_2_dot * h;

    theta_1_dot = gamma_1*epsilon_1*phi_1;
    theta_1(:, n+1) = theta_1(:, n) + theta_1_dot*h;
```

The plot becomes:



(a) Estimation using LS **(b)** Estimation using Inst. Cost

Figure 5: Estimation using LS and Inst. Cost

References

[1] P. Ioannou and B. Fidan. *Adaptive control tutorial*. en. Advances in Design and Control. Society for Industrial and Applied Mathematics, 2006. ISBN: 99780898716153. DOI: https://doi.org/10.1137/1.9780898718652.