Stochastic Adaptive Control Assignment 3

Kristoffer Erbo Kjær - s203829

May 6, 2025

Question 1.1

To design the stationary ordinary Kalman filter the assosiated Discrete algebraic Riccati equation needs to be solved. For convenience in the Matlab implementation, the Riccati equation of the stationary covariance of the predictive Kalman filter is solved Eq. (1).

$$P_\infty^{pp} = AP_\infty^{pp}A^T + R_1 - AP_\infty^{pp}C^T(CP_\infty^{pp}C^T + R_2)^{-1}CP_\infty^{pp}A^T \tag{1}$$

This is done since the equations' structure is very similar to the matlap solver: **[X,K,L] = idare(A,B,Q,R,S,E)** Eq. (2).

$$A^TXA - E^TXE - (A^TXB + S)(B^TXB + R)^{-1}(A^TXB + S)^T + Q = 0 \tag{2}$$

Calling the **idare** function as follows: **"[$\mathbf{P_\infty^{pp}}, \sim, \sim$] = idare(A', C', R1, R2)"** solves the Discrete algebraic Riccati equation of the stationary covariance of the predictive Kalman filter.

The stationary covariance of the ordinary Kalman filter can then be found using Eq. (3).

$$(P_\infty^o)^{-1} = (P_\infty^p)^{-1} + C^TR_2^{-1}C \tag{3}$$

The Kalman filter gain can the be calculated:

$$K_f = P_\infty^o \cdot C'(C \cdot P_\infty^o \cdot C' + R_2)^{-1} = \begin{bmatrix} 0.4738 & 0.0008 \\ 0.0008 & 0.4738 \\ 0.1811 & 0.0034 \\ 0.0034 & 0.1811 \end{bmatrix} \tag{4}$$

Implimenting a simulation with the decrubed noice and no control input, the following simulation can then be obtained:

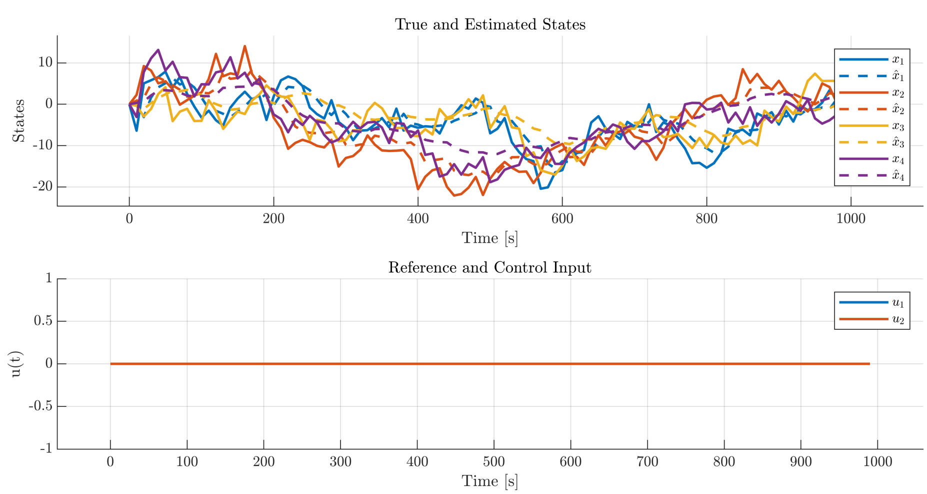

Figure 1: The uncontroled system with a Kalman filter as a state estimator.

From the simulation it is clear that the state estimates are following the actual states to some degree, but that especially the unmeasured states are deviating from the actual value of the state. This is, however, to be expected since the system is in steady state with zero input. The expected value is therefore dependent on the noise term which has a mean value of 0. Any deviations from this value are caused by the noise and it is therefore reasonable that the Kalman filter state estimates are not more precise.

Question 1.2

In this section, the aim is to implement a general predictive controller following the control law described in Eq. (5).

$$U_N = -\left(\Gamma_{yu}^T Q_y \Gamma_{yu} + Q_u\right)^{-1} \Gamma_{yu}^T Q_y \left(\Phi_{yx}\mathbb{E}[x_0] - W_N\right) \tag{5}$$

Where Q_y and Q_u are weighting matrices from the cost function describing the cost of reference deviation and utilization of control input respectivly.

For the prediction the following matrices in Eq. (6) are used.

$$\Gamma_{yu}^N = \begin{bmatrix} D & & & \\ CB & D & & \\ \vdots & \ddots & \ddots & \\ CA^{N-1}B & \cdots & CB & D \end{bmatrix}, \ \Phi_{yx}^N = \begin{bmatrix} C \\ CA \\ \vdots \\ CA^N \end{bmatrix} \tag{6}$$

The input prediction is updated for each iteration and the only control input utilized is therefore described in Eq. (7).

$$u_t = \begin{bmatrix} I & 0 & \cdots & 0 \end{bmatrix} U_N \tag{7}$$

Implementing this together with the Kalman filter (Fig. 2), it is shown that the state estimation is satisfactory for the control problem. It can be seen that state/output 1 and 2 follow their respective references. The only significant deviations are when the setpoint is changed. The deviation is both before and after the reference change due to the predictive element of the controller with even weights of all steps. The prediction horizon is limited to 10 steps / 100 s but similar performance can be shown with an increased horizon.

The deviation from the reference could be minimized by lowering the values of Q_u. The values chosen (**diag([100,100])**) are to limit the control effort while keeping reasonable response time of the thermal system. The settling time is around 50 s which is around one deg per second. In addition to limiting the control utilization the temperature of state 3 and 4 is also kept in a reasonable range when using a lower control effort. This increases the chance of the linearization being an acceptable estimation.

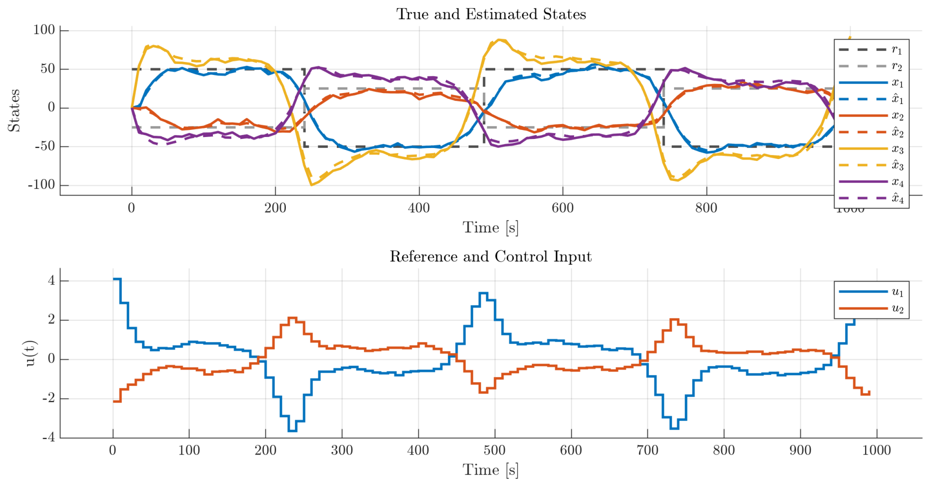

Figure 2: The system with a General predictive controller implimented and Kalman filter state estimation.

Question 1.3

Since the system has two independent measurements, it is possible to estimate two constant independent disturbances. Since a disturbance for each state cannot be estimated, a choice of where to model the disturbances must be made. Due to the actuator placement and the strong link between state 1 and 3, as well as state 2 and 4, the disturbances are modeled as entering state 3 and 4. This ensures that any offset in the actuators will effectively be counteracted by an integral action, ensuring that the system reaches steady state when control is implemented.

Ideally, a constant disturbance would be modeled as $\kappa = 1$, but as I understand it, classmates had problems with no solution to the Riccati equation existing and a value close to is therefore chosen. I can reproduce this issue when adding more than two disturbances, but since the system then becomes not observable, I would argue that adding more disturbances becomes meaningless. Especially since the system is a two input two output system and two disturbances should be enough for a future implimentation of a controller.

$\sigma_d^2 = 4$ is iteratively tuned until a satisfactory disturbance estimation is reached which will be descussed below.

$$A_{\text{aug}} = \begin{bmatrix} 0.7901 & 0.0238 & 0.1667 & 0.0081 & 0 & 0 \\ 0.0238 & 0.7901 & 0.0081 & 0.1667 & 0 & 0 \\ 0.1667 & 0.0081 & 0.7236 & 0.0014 & 1.0000 & 0 \\ 0.0081 & 0.1667 & 0.0014 & 0.7236 & 0 & 1.0000 \\ 0 & 0 & 0 & 0 & 0.9990 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.9990 \end{bmatrix}, \tag{8}$$

$$B_{\text{aug}} = \begin{bmatrix} 1.4459 & 0.0622 \\ 0.0622 & 1.4459 \\ 13.3615 & 0.0077 \\ 0.0077 & 13.3615 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}, C_{\text{aug}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}, D_{\text{aug}} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{9}$$

$$R_{1_{\text{aug}}} = \begin{bmatrix} 8.0209 & 0.2260 & 1.5334 & 0.0812 & 0 & 0 \\ 0.2260 & 8.0209 & 0.0812 & 1.5334 & 0 & 0 \\ 1.5334 & 0.0812 & 7.4082 & 0.0172 & 0 & 0 \\ 0.0812 & 1.5334 & 0.0172 & 7.4082 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4.0000 & 0 \\ 0 & 0 & 0 & 0 & 0 & 4.0000 \end{bmatrix} \tag{10}$$

To test the the Kalman filter disturbance estimation, the unforced system is simulated in Fig. 3. It is clear here that the disturbance estimation is working since the disturbance is tracked and the state estimation is still close to the actual values of the states. It is, however, clear that when the disturbance is changing rapidly, the state estimation of especially state 3 and 4 becomes more unreliable until the disturbance estimate is converged again. This could be done faster but since a constant disturbance is assumed a slower response is prioritized to minimize noise in the steady state of the disturbance.

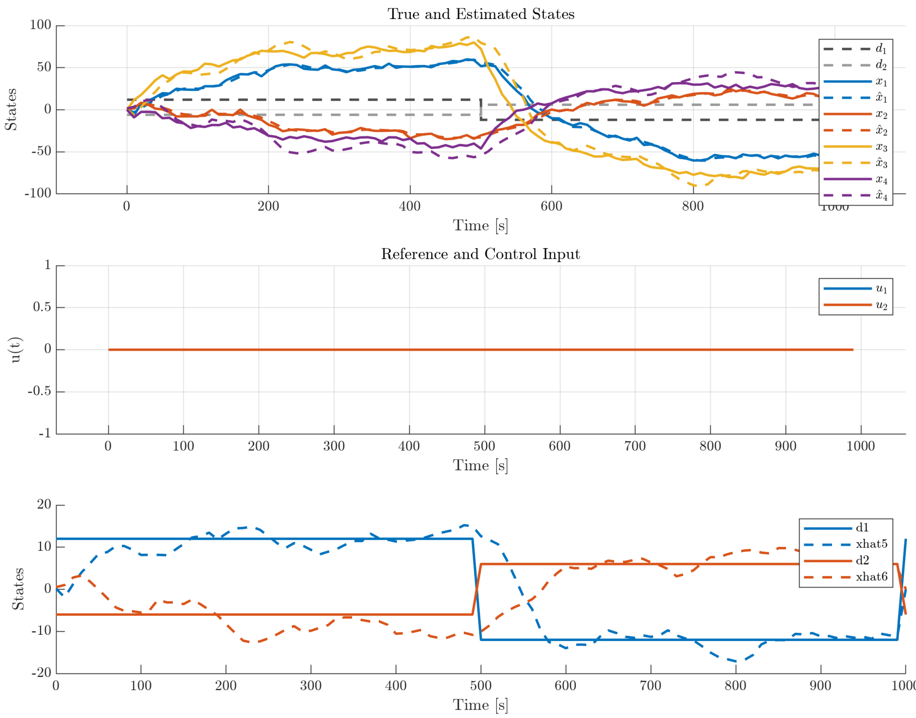

Figure 3: Desturbance estimation with no control input.

Question 1.4

Now combining the disturbance estimation Kalman filter and the general predictive controller, the results seen in Fig. 4 are achieved. Since the same seed as in question 1.3 is used, the system is subject to the same noise. Observing Fig. 3 and Fig. 4 it appears that the disturbance estimate is the same as with no control. This is reasonable since the Kalman filter accounts for the input; the only unexplained factors are therefore the noise and the disturbances which are the same in both cases (For confirmation it was checked that changing the seed results in different results).

The estimation performance is also similar to the one seen in question 1.3. After stepping the disturbance, the estimation of states 3 and 4 is worse. In this particular case, it appears to affect the step in state 1 around 500 s since the disturbance is lowered in the same instance. This results in the settling time being lower since the controller expects state 3 to be higher than it actually is.

Overall, the disturbance estimation and controller appear to function well when combined since the reference is followed and the disturbance is successfully estimated.

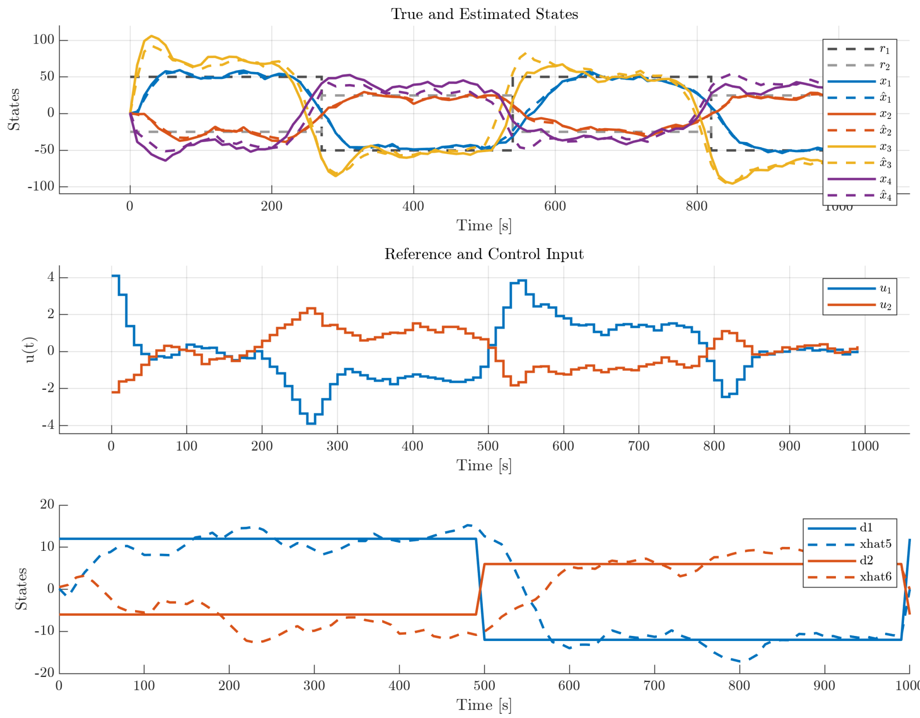

Figure 4: General predictive control with disturbance estimation with two disturbances.

To further test the performance of the closed loop system, a disturbance is now added to all states while the disturbance estimation still only assumes the disturbances enter through states 3 and 4. This can be seen in Fig. 5. The disturbance estimation is now significantly off for states 3 and 4, which are the ones modeled since the disturbances in states 1 and 2 dominate the effect on the output. The state estimates of states 3 and 4 are therefore also significantly off. However, since the disturbance estimation acts as integrators on the reference error, it can be observed that the reference tracking is still satisfactory when there are no rapid changes in the disturbances.

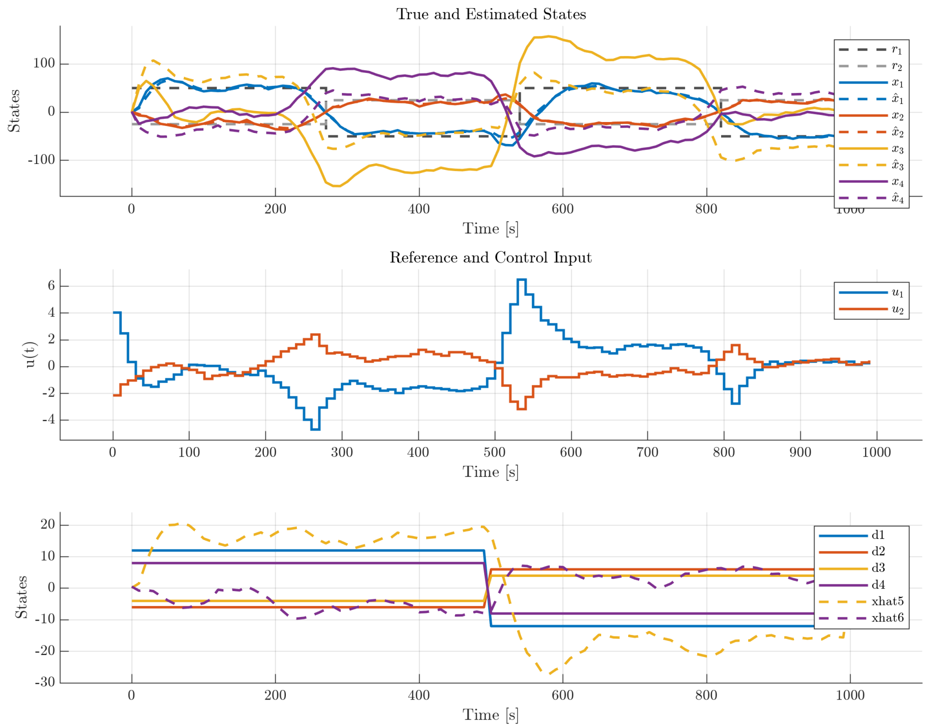

Figure 5: General predictive control with disturbance estimation with four disturbances.

Appendix

A init

```
clear
rng(203829)
addpath('functions')
set(groot, 'defaultTextInterpreter', 'latex');
set(groot, 'defaultAxesTickLabelInterpreter', 'latex');
set(groot, 'defaultLegendInterpreter', 'latex');
plotPos = [100 100 800, 400];
plotPos2 = [100 100 800, 600];
```

B Question 1.1

```
Question 1.1
%% --- system definition ---
Ts = 10;     % sample time [s]

A = [0.7901, 0.0238, 0.1667, 0.0081;
     0.0238, 0.7901, 0.0081, 0.1667;
     0.1667, 0.0081, 0.7236, 0.0014;
     0.0081, 0.1667, 0.0014, 0.7236];

B = [ 1.4459,  0.0622;
      0.0622,  1.4459;
     13.3615,  0.0077;
      0.0077, 13.3615];

C = [1, 0, 0, 0;
     0, 1, 0, 0];

% cost / noise-covariance matrices (as given)
R1= [8.0209, 0.2260, 1.5334, 0.0812;
       0.2260, 8.0209, 0.0812, 1.5334;
       1.5334, 0.0812, 7.4082, 0.0172;
       0.0812, 1.5334, 0.0172, 7.4082];

R2 = eye(2);     % = R2

%% --- LQR controller design ---
ctrlOn = 0;
[K,~,~] = dlqr(A,B,diag([1,1,0,0]),R2*250);
dcG = (C*(eye(4) - (A-B*K))^(-1) * B)^-1;
```

```

%% --- Kalman filter design (steady-state) ---
%   L is the state-estimator gain:  xhat+ = A*xhat + B*u + L*(y - C*xhat)
% Solve the discrete Riccati equation
[P,~,~] = dare(A', C', R1, R2);
[Pp,~,~] = idare(A', C', R1, R2);

Po = inv(inv(Pp) + C'*inv(R2)*C);

% Compute stationary Kalman gain
% Kf = P * C' / (C * P * C' + R2);
Kf = Po * C' / (C * Po * C' + R2);


%% --- simulation setup ---
Tfinal = 1000;            % total simulation time [s]
N = Tfinal / Ts;          % number of steps
x     = zeros(4, N+1);    % true state
xhat  = zeros(4, N+1);    % estimated state
y     = zeros(2, N);      % measurements
yhat  = zeros(2, N);      % predicted outputs
u     = zeros(2, N);      % control inputs

rng(0);  % for reproducible noise

T = N; % Periode in sampels
ref1 = 50*square(2*pi*1/T*(1:N),50);
ref2 = zeros(1,N);
ref = [ref1;ref2];


for k = 1:N
    % Simulate measurement
    v = mvnrnd(zeros(size(R2,1),1), R2)';
    y(:,k) = C*x(:,k) + v;

    % Controller (if any)
    if ctrlOn
        u(:,k) = -K*xhat(:,k) + dcG * ref(:,k);
    end
    % True system evolution
    w = mvnrnd(zeros(size(R1,1),1), R1)';
    x(:,k+1) = A*x(:,k) + B*u(:,k) + w;

    % Stationary Kalman filter update
    innov = y(:,k) - C*xhat(:,k);
```

```
    xhat(:,k + 1) = A*xhat(:,k) + B*u(:,k) + Kf*innov;
    yhat(:,k + 1) = C*xhat(:,k);
end

time = 0:Ts:Tfinal;
colors = lines(4);
%% --- plotting ---
fig = figure();
fig.Position = plotPos;
tiledlayout(2,1, 'Padding', 'compact', 'TileSpacing', 'compact');

nexttile;
names = {'$x_1$','$\hat{x}_1$','$x_2$','$\hat{x}_2$','$x_3$','$\hat{x}_3$','$x_4$','$\hat{x
if ctrlOn
    stairs(time(1:end-1), ref(1,:), '--', 'LineWidth', 1.5, 'Color', [0.3 0.3 0.3]);
    names = {'$r_1$', '$x_1$','$\hat{x}_1$','$x_2$','$\hat{x}_2$','$x_3$','$\hat{x}_3$','$x
end
hold on
for i = 1:4
    plot(time,     x(i,:),     '-',  'Color', colors(i,:), 'LineWidth', 1.5);
    plot(time, xhat(i,:), '--', 'Color', colors(i,:), 'LineWidth', 1.5);
end
xlabel('Time [s]');
ylabel('States');
legend(names, ...
       'Location','northeast');
title('True and Estimated States');
ylim padded;
xlim padded;
grid on;
hold off

% Plot control input
nexttile;
hold on
stairs(time(1:end-1), u(:,:)', '-',  'LineWidth', 1.5);
xlabel('Time [s]');
ylabel('u(t)');
legend('$u_1$','$u_2$');
title('Reference and Control Input');
ylim padded;
xlim padded;
grid on;
hold off
filePath = fullfile("output", 'question_1_1.png');
exportgraphics(fig, filePath, 'Resolution', 600);
```

C Question 1.2

```
Question 1.2
%% 1) System definition
Ts     = 10;              % sample time [s]
Tfinal = 1000;            % total simulation time [s]
Nsim   = Tfinal / Ts;     % number of steps

A = [0.7901, 0.0238, 0.1667, 0.0081;
     0.0238, 0.7901, 0.0081, 0.1667;
     0.1667, 0.0081, 0.7236, 0.0014;
     0.0081, 0.1667, 0.0014, 0.7236];
B = [ 1.4459,  0.0622;
      0.0622,  1.4459;
     13.3615,  0.0077;
      0.0077, 13.3615];
C = [1, 0, 0, 0;
     0, 1, 0, 0];

D = zeros(2,2);

[nx, nu] = size(B);
ny       = size(C,1);

R1 = [8.0209, 0.2260, 1.5334, 0.0812;
         0.2260, 8.0209, 0.0812, 1.5334;
         1.5334, 0.0812, 7.4082, 0.0172;
         0.0812, 1.5334, 0.0172, 7.4082];  % process-noise cov
R2 = eye(ny);                               % meas-noise cov
[ Lf, P, ~ ] = dlqe(A, eye(nx), C, R1, R2);

% [P,~,~] = dare(A', C', Q, R);
[Pp,~,~] = idare(A', C', R1, R2);

Po = inv(inv(Pp) + C'*inv(R2)*C);

% Compute stationary Kalman gain
% Kf = P * C' / (C * P * C' + R2);
Kf = Po * C' / (C * Po * C' + R2);

%% 3) GPC tuning parameters
Np  = 10; % prediction horizon
% Np = Nsim
Qy  = kron(eye(Np), 1*eye(ny));                    % output-tracking weight
Ru  = 100 * eye(nu*Np);                           % input move weight
```

```
%% 4) Build prediction matrices
Phi     = zeros(ny*Np, nx);
Gamma_y = zeros(ny*Np, nu*Np);

for i = 1:Np
    Phi((i-1)*ny+1:i*ny, :) = C * A^(i-1);
    for j = 1:i
        idx_row = (i-1)*ny+1 : i*ny;
        idx_col = (j-1)*nu+1 : j*nu;

        if i == j
            Gamma_y(idx_row, idx_col) = D;  % Diagonal: D
        else
            Gamma_y(idx_row, idx_col) = C * A^(i-j) * B;  % Lower-triangular: CA^(i-j)B
        end
    end
end
%% 5) Reference signal
ref1 = 50 * square(2*pi*(2/(Nsim))*(1:Nsim + Np), 50);
ref2 = -0.5*ref1; %zeros(1, Nsim + Np);
ref  = [ref1; ref2];

%% 6) Preallocate
x      = zeros(nx,   Nsim+1);   % true state
xhat   = zeros(nx,   Nsim+1);   % estimated state
u      = zeros(nu,   Nsim);
y      = zeros(ny,   Nsim);
xhat_pred = xhat(:,1);

rng(0);  % for reproducibility

%% 7) Main simulation loop

for k = 1:Nsim
    %--- simulate measurement y(k) with noise
    y(:,k) = C*x(:,k) + mvnrnd(zeros(ny,1), R2)';

    %--- Kalman correct
    innov      = y(:,k) - C*xhat_pred;
    xhat(:,k)  = xhat_pred + Kf * innov;

    %--- GPC: form QP linear term
    r_win = reshape(ref(:, k:k+Np-1), [], 1);% stacked reference

    %--- apply control law
```

```
    U_N = -inv(Gamma_y' * Qy * Gamma_y + Ru) * Gamma_y' * Qy * (Phi * xhat(:, k) - r_win);

    %--- apply only first move
    % u(:, k) = U_N(1:nu);
    u(:, k) = [eye(nu), zeros(nu,nu*Np - nu)] * U_N;

    %--- simulate true system with process noise
    x(:, k+1) = A * x(:, k) + B * u(:, k) + mvnrnd(zeros(nx, 1), R1)';

    %--- Kalman predict
    xhat_pred = A*xhat(:,k) + B*u(:, k);

end

%% 8) Plotting
time   = 0:Ts:Tfinal;
colors = lines(4);

fig = figure();
fig.Position = plotPos;
tiledlayout(2,1, 'Padding', 'compact', 'TileSpacing', 'compact');

nexttile;
hold on
stairs(time(1:end-1), ref(1,1:end-Np), '--', 'LineWidth', 1.5, 'Color', [0.3 0.3 0.3]);
stairs(time(1:end-1), ref(2,1:end-Np), '--', 'LineWidth', 1.5, 'Color', [0.6 0.6 0.6]);
names = {'$r_1$', '$r_2$', '$x_1$','$\hat{x}_1$','$x_2$','$\hat{x}_2$','$x_3$','$\hat{x}_3$


for i = 1:4
    plot(time,     x(i,:),     '-',  'Color', colors(i,:), 'LineWidth', 1.5);
    plot(time, xhat(i,:), '--', 'Color', colors(i,:), 'LineWidth', 1.5);
end
xlabel('Time [s]');
ylabel('States');
legend(names, ...
       'Location','northeast');
title('True and Estimated States');
ylim padded;
xlim padded;
grid on;
hold off

% Plot control input
nexttile;
hold on
```

```
stairs(time(1:end-1), u(:,:)', '-',  'LineWidth', 1.5);
xlabel('Time [s]');
ylabel('u(t)');
legend('$u_1$','$u_2$');
title('Reference and Control Input');
ylim padded;
xlim padded;
grid on;
hold off
filePath = fullfile("output", 'question_1_2.png');
exportgraphics(fig, filePath, 'Resolution', 300);

```

D Question 1.3

```
Question 1.3

%% 1) System definition
Ts     = 10;                 % sample time [s]
Tfinal = 1000;               % total simulation time [s]
Nsim   = Tfinal / Ts;        % number of steps

A = [0.7901, 0.0238, 0.1667, 0.0081;
     0.0238, 0.7901, 0.0081, 0.1667;
     0.1667, 0.0081, 0.7236, 0.0014;
     0.0081, 0.1667, 0.0014, 0.7236];
B = [ 1.4459,  0.0622;
      0.0622,  1.4459;
     13.3615,  0.0077;
      0.0077, 13.3615];
C = [1, 0, 0, 0;
     0, 1, 0, 0;];



[nx, nu] = size(B);
ny        = size(C,1);

%% 2) Kalman filter design (steady-state gain)
R1 = [8.0209, 0.2260, 1.5334, 0.0812;
         0.2260, 8.0209, 0.0812, 1.5334;
         1.5334, 0.0812, 7.4082, 0.0172;
         0.0812, 1.5334, 0.0172, 7.4082];  % process-noise cov
R2 = eye(ny);                               % meas-noise cov


kappa = [0.999,0.999];
% kappa = [0.99999,0.99999,0.99999,0.99999];
sigma_d = 2*[1,1];
A_aug = [A,[zeros(2,2);eye(2,2)];zeros(2,4),diag(kappa)];
% A_aug = [A,eye(4,2);zeros(2,4),diag(kappa)]
B_aug = [B;zeros(2,2)];
C_aug =  [C, zeros(2,2)];
D_aug = D;
R1_aug = [R1,zeros(4,2);zeros(2,4),diag(sigma_d.^2)];
rank(obsv(A_aug,C_aug))
```

```
[nx, nu] = size(B_aug);
ny        = size(C_aug,1);


[ Kf, ~, ~ ] = dlqe(A_aug, eye(nx), C_aug, R1_aug, R2);
[Pp,~,~] = idare(A_aug', C_aug', R1_aug, R2);

Po = inv(inv(Pp) + C_aug'*inv(R2)*C_aug);

% Compute stationary Kalman gain
% Kf = P * C' / (C * P * C' + R2);
Kf = Po * C_aug' / (C_aug * Po * C_aug' + R2);


%% 3) GPC tuning parameters
Np  = 10;                                       % prediction horizon
Qy  = kron(eye(Np), eye(ny));                   % output-tracking weight
Ru  = 100 * eye(nu*Np);                        % input move weight

%% 4) Build prediction matrices
Phi     = zeros(ny*Np, nx);
Gamma_y = zeros(ny*Np, nu*Np);

for i = 1:Np
    Phi((i-1)*ny+1:i*ny, :) = C_aug * A_aug^(i-1);
    for j = 1:i
        idx_row = (i-1)*ny+1 : i*ny;
        idx_col = (j-1)*nu+1 : j*nu;

        if i == j
            Gamma_y(idx_row, idx_col) = D_aug;  % Diagonal: D
        else
            Gamma_y(idx_row, idx_col) = C_aug * A_aug^(i-j) * B_aug;  % Lower-triangular: C
        end
    end
end

% Precompute Hessian for QP
H = Gamma_y' * Qy * Gamma_y + Ru;
opts = optimoptions('quadprog','Display','off');

%% 5) Reference signal (±10 square wave on output 1)
ref1 = 50 * square(2*pi*(2/(Nsim+ Np))*(1:Nsim + Np), 50);
ref2 = -0.5*ref1; %zeros(1, Nsim + Np);
ref  = [ref1; ref2];

```

```
%% 6) Preallocate
x     = zeros(nx-2,   Nsim+1);   % true state
xhat  = zeros(nx,   Nsim+1);   % estimated state
d  = [12* square(2*pi*(2/(2*Nsim+ 1))*(1:Nsim + 1), 50);
    -6* square(2*pi*(2/(2*Nsim+ 1))*(1:Nsim + 1), 50);
    ];   % estimated state
% d = 100*[ones(1,Nsim+1);
%      -ones(1,Nsim+1)];

u     = zeros(nu,   Nsim);
y     = zeros(ny,   Nsim);
xhat_pred = xhat(:,1);
Xpred   = zeros(nx, Np+1, Nsim);  % store Np-step predictions
rng(0);  % for reproducibility

%% 7) Main simulation loop
for k = 1:Nsim
    %--- simulate measurement y(k) with noise
    y(:,k) = C*x(:,k) + mvnrnd(zeros(ny,1), R2)';

    %--- Kalman correct
    innov      = y(:,k) - C_aug*xhat_pred;
    xhat(:,k)  = xhat_pred + Kf * innov;

    %--- GPC: form QP linear term
    r_win = reshape(ref(:, k:k+Np-1), [], 1);% stacked reference
    f = Gamma_y' * Qy * (Phi * xhat(:, k) - r_win);

    %--- apply control law
    U_N = -inv(Gamma_y' * Qy * Gamma_y + Ru) * Gamma_y' * Qy * (Phi * xhat(:, k) - r_win);

    %--- apply only first move
    % u(:, k) = U_N(1:nu);
    u(:, k) = [eye(nu), zeros(nu,nu*Np - nu)] * U_N;
    u(:, k) =  [0;0];


    % Compute Np-step ahead state predictions
    Xpred(:,1,k) = xhat(:,k);
    for j = 1:Np
    Uj = U_N((j-1)*nu+1:j*nu);
    Xpred(:,j+1,k) = A_aug*Xpred(:,j,k) + B_aug*Uj;
    end


```

```
    %--- simulate true system with process noise
    % x(:, k+1) = A * x(:, k) + B * u(:, k) + mvnrnd(zeros(nx-2, 1), Qk_f)' + [eye(2,2);zer
    x(:, k+1) = A * x(:, k) + B * u(:, k) + mvnrnd(zeros(nx-2, 1), R1)' + [zeros(2,2);eye(2

    %--- Kalman predict
    xhat_pred = A_aug*xhat(:,k) + B_aug*u(:, k);
end

%% 8) Plotting
time   = 0:Ts:Tfinal;
colors = lines(4);

fig = figure();
fig.Position = plotPos2;
tiledlayout(3,1, 'Padding', 'compact', 'TileSpacing', 'compact');

nexttile;
hold on
stairs(time(1:end), d(1,1:end), '--', 'LineWidth', 1.5, 'Color', [0.3 0.3 0.3]);
stairs(time(1:end), d(2,1:end), '--', 'LineWidth', 1.5, 'Color', [0.6 0.6 0.6]);
names = {'$d_1$', '$d_2$', '$x_1$','$\hat{x}_1$','$x_2$','$\hat{x}_2$','$x_3$','$\hat{x}_3$


for i = 1:4
    plot(time,     x(i,:),     '-',  'Color', colors(i,:), 'LineWidth', 1.5);
    plot(time, xhat(i,:), '--', 'Color', colors(i,:), 'LineWidth', 1.5);
end
xlabel('Time [s]');
ylabel('States');
legend(names, ...
       'Location','northeast');
title('True and Estimated States');
ylim padded;
xlim padded;
grid on;
hold off

% Plot control input
nexttile;
hold on
stairs(time(1:end-1), u(:,:)', '-',  'LineWidth', 1.5);
xlabel('Time [s]');
ylabel('u(t)');
legend('$u_1$','$u_2$');
title('Reference and Control Input');
ylim padded;
```

```
xlim padded;
grid on;
hold off
% filePath = fullfile("output", 'question_1_4.png');
% exportgraphics(fig, filePath, 'Resolution', 300);

% fig = figure();
% fig.Position = plotPos;
% tiledlayout(1,1, 'Padding', 'compact', 'TileSpacing', 'compact');
nexttile;
hold on
for i = 5:6
    plot(time,  d(i-4,:),     '-',  'Color', colors(i-4,:), 'LineWidth', 1.5);
    plot(time, xhat(i,:), '--', 'Color', colors(i-4,:), 'LineWidth', 1.5);
end
hold off
xlabel('Time [s]');
ylabel('States');
legend([strcat("d",string(1:2));strcat("xhat",string(5:6))], ...
       'Interpreter','latex','Location','northeast');
filePath = fullfile("output", 'question_1_3.png');
exportgraphics(fig, filePath, 'Resolution', 300);
```

E Question 1.4

```
Question 1.4
%% 1) System definition
Ts     = 10;                 % sample time [s]
Tfinal = 1000;               % total simulation time [s]
Nsim   = Tfinal / Ts;        % number of steps

A = [0.7901, 0.0238, 0.1667, 0.0081;
     0.0238, 0.7901, 0.0081, 0.1667;
     0.1667, 0.0081, 0.7236, 0.0014;
     0.0081, 0.1667, 0.0014, 0.7236];
B = [ 1.4459,  0.0622;
      0.0622,  1.4459;
     13.3615,  0.0077;
      0.0077, 13.3615];
C = [1, 0, 0, 0;
     0, 1, 0, 0;];
D= zeros(2,2);

[nx, nu] = size(B);
ny        = size(C,1);

%% 2) Kalman filter design (steady-state gain)
R1 = [8.0209, 0.2260, 1.5334, 0.0812;
         0.2260, 8.0209, 0.0812, 1.5334;
         1.5334, 0.0812, 7.4082, 0.0172;
         0.0812, 1.5334, 0.0172, 7.4082];  % process-noise cov
R2 = eye(ny);                             % meas-noise cov


kappa = [0.999,0.999];
% kappa = [0.99999,0.99999,0.99999,0.99999];
sigma_d = 2*[1,1];
A_aug = [A,[zeros(2,2);eye(2,2)];zeros(2,4),diag(kappa)];
% A_aug = [A,eye(4,2);zeros(2,4),diag(kappa)]
B_aug = [B;zeros(2,2)];
C_aug =  [C, zeros(2,2)];
D_aug = D;
R1_aug = [R1,zeros(4,2);zeros(2,4),diag(sigma_d.^2)];
rank(obsv(A_aug,C_aug))


[nx, nu] = size(B_aug);
ny        = size(C_aug,1);
```

```
[ Kf, ~, ~ ] = dlqe(A_aug, eye(nx), C_aug, R1_aug, R2);
[Pp,~,~] = idare(A_aug', C_aug', R1_aug, R2);

Po = inv(inv(Pp) + C_aug'*inv(R2)*C_aug);

% Compute stationary Kalman gain
% Kf = P * C' / (C * P * C' + R2);
Kf = Po * C_aug' / (C_aug * Po * C_aug' + R2);


%% 3) GPC tuning parameters
Np  = 10;                                    % prediction horizon
Qy  = kron(eye(Np), eye(ny));                % output-tracking weight
Ru  = 100 * eye(nu*Np);                     % input move weight

%% 4) Build prediction matrices
Phi     = zeros(ny*Np, nx);
Gamma_y = zeros(ny*Np, nu*Np);

for i = 1:Np
    Phi((i-1)*ny+1:i*ny, :) = C_aug * A_aug^(i-1);
    for j = 1:i
        idx_row = (i-1)*ny+1 : i*ny;
        idx_col = (j-1)*nu+1 : j*nu;

        if i == j
            Gamma_y(idx_row, idx_col) = D_aug;  % Diagonal: D
        else
            Gamma_y(idx_row, idx_col) = C_aug * A_aug^(i-j) * B_aug;  % Lower-triangular: C
        end
    end
end

% Precompute Hessian for QP
H = Gamma_y' * Qy * Gamma_y + Ru;
opts = optimoptions('quadprog','Display','off');

%% 5) Reference signal (±10 square wave on output 1)
ref1 = 50 * square(2*pi*(2/(Nsim+ Np))*(1:Nsim + Np), 50);
ref2 = -0.5*ref1; %zeros(1, Nsim + Np);
ref  = [ref1; ref2];


%% 6) Preallocate
```

```
x      = zeros(nx-2,   Nsim+1);   % true state
xhat   = zeros(nx,   Nsim+1);   % estimated state
d1   = [12* square(2*pi*(2/(2*Nsim+ 1))*(1:Nsim + 1), 50);
    -6* square(2*pi*(2/(2*Nsim+ 1))*(1:Nsim + 1), 50);
    ];   % estimated state

d2 =[12* square(2*pi*(2/(2*Nsim+ 1))*(1:Nsim + 1), 50);
    -6* square(2*pi*(2/(2*Nsim+ 1))*(1:Nsim + 1), 50);
    -4* square(2*pi*(2/(2*Nsim+ 1))*(1:Nsim + 1), 50);
    8* square(2*pi*(2/(2*Nsim+ 1))*(1:Nsim + 1), 50)];
% d = 100*[ones(1,Nsim+1);
%      -ones(1,Nsim+1)];

u      = zeros(nu,   Nsim);
y      = zeros(ny,   Nsim);
xhat_pred = xhat(:,1);
Xpred   = zeros(nx, Np+1, Nsim);  % store Np-step predictions
rng(0);  % for reproducibility

%% 7) Main simulation loop
for k = 1:Nsim
    %--- simulate measurement y(k) with noise
    y(:,k) = C*x(:,k) + mvnrnd(zeros(ny,1), R2)';

    %--- Kalman correct
    innov      = y(:,k) - C_aug*xhat_pred;
    xhat(:,k)  = xhat_pred + Kf * innov;

    %--- GPC: form QP linear term
    r_win = reshape(ref(:, k:k+Np-1), [], 1);% stacked reference
    f = Gamma_y' * Qy * (Phi * xhat(:, k) - r_win);

    %--- apply control law
    U_N = -inv(Gamma_y' * Qy * Gamma_y + Ru) * Gamma_y' * Qy * (Phi * xhat(:, k) - r_win);

    %--- apply only first move
    % u(:, k) = U_N(1:nu);
    u(:, k) = [eye(nu), zeros(nu,nu*Np - nu)] * U_N;
    % u(:, k) =  [0;0];


    % Compute Np-step ahead state predictions
    Xpred(:,1,k) = xhat(:,k);
    for j = 1:Np
    Uj = U_N((j-1)*nu+1:j*nu);
    Xpred(:,j+1,k) = A_aug*Xpred(:,j,k) + B_aug*Uj;
```

```
    end


    %--- simulate true system with process noise
    x(:, k+1) = A * x(:, k) + B * u(:, k) + mvnrnd(zeros(nx-2, 1), R1)' + [zeros(2,2);eye(2
    % x(:, k+1) = A * x(:, k) + B * u(:, k) + mvnrnd(zeros(nx-2, 1), R1)' + eye(4)*d2(:,k);

    %--- Kalman predict
    xhat_pred = A_aug*xhat(:,k) + B_aug*u(:, k);
end

%% 8) Plotting
time   = 0:Ts:Tfinal;
colors = lines(4);

fig = figure();
fig.Position = plotPos2;
tiledlayout(3,1, 'Padding', 'compact', 'TileSpacing', 'compact');

nexttile;
hold on
stairs(time(1:end-1), ref(1,1:end-Np), '--', 'LineWidth', 1.5, 'Color', [0.3 0.3 0.3]);
stairs(time(1:end-1), ref(2,1:end-Np), '--', 'LineWidth', 1.5, 'Color', [0.6 0.6 0.6]);
names = {'$r_1$', '$r_2$', '$x_1$','$\hat{x}_1$','$x_2$','$\hat{x}_2$','$x_3$','$\hat{x}_3$


for i = 1:4
    plot(time,     x(i,:),     '-',  'Color', colors(i,:), 'LineWidth', 1.5);
    plot(time, xhat(i,:), '--', 'Color', colors(i,:), 'LineWidth', 1.5);
end
xlabel('Time [s]');
ylabel('States');
legend(names, ...
       'Location','northeast');
title('True and Estimated States');
ylim padded;
xlim padded;
grid on;
hold off

% Plot control input
nexttile;
hold on
stairs(time(1:end-1), u(:,:)', '-',  'LineWidth', 1.5);
xlabel('Time [s]');
ylabel('u(t)');
```

```matlab
legend('$u_1$','$u_2$');
title('Reference and Control Input');
ylim padded;
xlim padded;
grid on;
hold off
% filePath = fullfile("output", 'question_1_4.png');
% exportgraphics(fig, filePath, 'Resolution', 300);
% fig = figure();
% fig.Position = plotPos;
% tiledlayout(1,1, 'Padding', 'compact', 'TileSpacing', 'compact');
nexttile;
hold on
for i = 5:6
    plot(time,  d1(i-4,:),     '-',  'Color', colors(i-4,:), 'LineWidth', 1.5);
    plot(time, xhat(i,:), '--', 'Color', colors(i-4,:), 'LineWidth', 1.5);
end
hold off
xlabel('Time [s]');
ylabel('States');
ylim padded;
xlim padded;
grid on
legend([strcat("d",string(1:2));strcat("xhat",string(5:6))], ...
       'Interpreter','latex','Location','northeast');
filePath = fullfile("output", 'question_1_4.png');
exportgraphics(fig, filePath, 'Resolution', 300);

```