```
function MotorProb(dt)

% Optimal State Estimation Solution Manual, by Dan Simon
% Problem 1.17

% Two-state DC motor simulation - rectangular integration.
% INPUT: dt = integration step size
% Use dt = 0.05 for a good simulation, dt = 0.2 for a marginal simulation,
% and dt = 0.5 for an unstable simulation.

J = 10; % moment of inertia
F = 100; % coefficient of viscous friction
A = [ 0 1 ; 0 -F/J ];
B = [ 0 ; 1/J ];
x = [ 0 ; 0 ]; % initial state

if ~exist('dt', 'var')
    dt = 0.05;
end

tf = 5; % simulation length

xArr = [x];

for t = dt : dt : tf+dt/10
    u = 10;
    xdot = A * x + B * u;
    x = x + xdot * dt;
    xArr = [xArr x];
end

t = 0 : dt : tf;

close all;
figure;
plot(t, xArr(1,:), 'b-', t, xArr(2,:), 'r:');
set(gca,'FontSize',12); set(gcf,'Color','White');
xlabel('Seconds');
legend('position', 'velocity');
title(['dt = ', num2str(dt)]);
```

```
function RLC(IntFlag, dt)

% Optimal State Estimation Solution Manual, by Dan Simon
% Problem 1.18

% Simulate a series RLC circuit.
% INPUT: dt = simulation step size
%        IntFlag = 0 for rectangular, 1 for trapezoidal, 2 for Runge Kutta

% The output is the voltage across the capacitor.

tf = 5; % simulation length

% Set the R, L, and C values.
R = 3;
L = 1;
C = 0.5;

% Set the system matrices; xdot = Ax + Bu, and y = Cx + Du.
A = [0 1/C; -1/L -R/L];
B = [0; 1/L];
C = [1 0];
D = [0];

x = [0 ; 0]; % initial condition

yarray = x(1);
for t = 0 : dt : tf - dt + dt/10
    u = exp(-2*t);
    if (IntFlag == 0)
        % Rectangular integration
        xdot = A * x + B * u;
        x = x + xdot * dt;
    elseif (IntFlag == 1)
        % Trapezoidal integration
        dx1 = (A * x + B * u) * dt;
        dx2 = (A * (x + dx1) + B * u) * dt;
        x = x + (dx1 + dx2) / 2;
    else
        u1 = exp(-2*(t+dt/2));
        dx1 = (A * x + B * u) * dt;
        dx2 = (A * (x + dx1 / 2) + B * u1) * dt;
        dx3 = (A * (x + dx2 / 2) + B * u1) * dt;
        dx4 = (A * (x + dx3) + B * u1) * dt;
        x = x + (dx1 + 2 * dx2 + 2 * dx3 + dx4) / 6;
    end
    y = C * x + D * u;
    yarray = [yarray y];
end
```

```
% Close all figures.
close all;
figure;
t = 0 : dt : tf;
plot(t, yarray, 'b-');
yAnalytical = 2 * (exp(-t) - exp(-2*t) - t .* exp(-2*t));
hold;
plot(t, yAnalytical, 'r:');
legend('Numerical solution', 'Analytical solution');

Err = sqrt((norm(yarray - yAnalytical))^2 / length(yarray))
```

```
function Rocket(ControlMag)

% Optimal State Estimation Solution Manual, by Dan Simon
% Problem 1.19

% Hovering rocket simulation to show the effects of linearization.

% Close all figures.
close all;
figure;

% Define the step size and the simulation length.
dt = 0.01;
tf = 5;

G = 6.673e-11; % gravitational constant (m^3/kg/s^2)
M = 5.98e24; % earth mass (kg)
m0 = 1000; % initial rocket mass (kg)
R = 6.37e6; % earth radius (m)
K = 1000; % thrust proportionality constant
g = 50; % drag constant

ControlArr = [10 100 300];
for i = 1 : 3
    ControlMag = ControlArr(i);
    % Nonlinear simulation.
    x1 = 0;
    x2 = 0;
    x3 = m0;
    x1array = [];
    x2array = [];
    x3array = [];

    for t = 0 : dt : tf + dt/10
        u = G * M * x3 / K / R / R + ControlMag * abs(cos(t));
        x1dot = x2;
        x2dot = (K * u - g * x2) / x3 - G * M / (R + x1)^2;
        x3dot = -u;
        x1 = x1 + dt * x1dot;
        x2 = x2 + dt * x2dot;
        x3 = x3 + dt * x3dot;
        x1array = [x1array x1];
        x2array = [x2array x2];
        x3array = [x3array x3];
    end

    % Linearized simulation.
    x1bar = 0;
    x2bar = 0;
    x3bar = 0;
```

```
    x1arrayLin = [];
    x2arrayLin = [];
    x3arrayLin = [];

    for t = 0 : dt : tf + dt/10
        m = m0 * exp(-G * M * t / K / R / R);
        ubar = ControlMag * abs(cos(2*pi*t));
        x1bardot = x2bar;
        x2bardot = (2 * G * M / R / R / R) * x1bar - (g / m) * x2bar - (G
* M / R / R / m) * x3bar + (K / m) * ubar;
        x3bardot = -ubar;
        x1bar = x1bar + dt * x1bardot;
        x2bar = x2bar + dt * x2bardot;
        x3bar = x3bar + dt * x3bardot;
        x1arrayLin = [x1arrayLin 0 + x1bar];
        x2arrayLin = [x2arrayLin 0 + x2bar];
        x3arrayLin = [x3arrayLin m + x3bar];
    end

    % Plot the results.
    t = [0 : dt : tf];
    subplot(3,1,i);
    plot(t, x1array, '-', t, x1arrayLin, '--');
    set(gca,'FontSize',12); set(gcf,'Color','White');
    if (i == 3), xlabel('Time (seconds)'); end;
    if (i == 2), ylabel('Altitude (meters)'); end;
    if (i == 1), legend('Nonlinear', 'Linearized'); end;
    title(['\Delta u = ', num2str(ControlMag)]);
end
```

```
function Uniform(N)

% Optimal State Estimation Solution Manual, by Dan Simon
% Problem 2.15

% Plot a histogram of uniformly distributed random numbers.
% INPUT: N = number of random numbers to generate

if ~exist('N', 'var')
    N = 50;
end

close all;
figure;
set(gcf,'Color','White');
rand('state', sum(100*clock)); % initialize the random number generator
for k = 1 : 3
    for i = 1 : N
        x(i) = rand;
    end
    xmean = mean(x);
    xstd = std(x);
    disp(['mean = ', num2str(xmean), ', std dev = ', num2str(xstd)]);
    subplot(3,1,k);
    hist(x,10); set(gca,'FontSize',12);
    legend(['N = ', num2str(N)]);
    N = 10 * N;
end
```

```
function Central

% Optimal State Estimation Solution Manual, by Dan Simon
% Problem 2.16

% Illustration of the Central Limit Theorem

close all;
figure; set(gcf,'Color','White');
rand('state', sum(100*clock)); % initialize the random number generator
for N = 2 : 2 : 4
    for i = 1 : 10000
        x(i) = 0;
        for j = 1 : N
            x(i) = x(i) + (rand-0.5) / N;
        end
    end
    subplot(2,1,N/2); set(gca,'FontSize',12);
    hist(x,50);
    legend(['N = ', num2str(N)]);
end
```

```
function LeastSteel

% Optimal State Estimation Solution Manual, by Dan Simon
% Problem 3.13

% Least squares curve fit for steel production data.

y = [66.6, 84.9, 88.6, 78.0, 96.8, 105.2, 93.2, 111.6, 88.3, 117.0,↵
115.2];
N = length(y);

close all; % close all figures
figure; % open a new figure
set(gcf,'Color','White');

% Linear curve fit.
xhat = [0; 0];
P = 100000 * eye(2);
R = 1;
for t = 1 : N
   H = [1 t];
   K = P * H' * inv(H * P * H' + R);
   xhat = xhat + K * (y(t) - H * xhat);
   P = (eye(2) - K * H) * P * (eye(2) - K * H)' + K * R * K';
end
% Compute the RMS error.
err = 0;
for t = 1 : N
   H = [1 t];
   err = err + (y(t) - H * xhat)^2;
end
err = sqrt(err / N);
disp(['Linear RMS Error = ',num2str(err)]);
H = [1 N+1];
disp(['Value predicted at next time = ', num2str(H*xhat)]);
% Plot the results.
t = 0 : N-1;
subplot(2,2,1);
plot(t, y, 'o', t, xhat(1)+xhat(2)*t);
legend('linear');
set(gca,'FontSize',12);

% Quadratic curve fit.
xhat = [0; 0; 0];
P = 100000 * eye(3);
R = 1;
for t = 1 : N
   H = [1 t t*t];
   K = P * H' * inv(H * P * H' + R);
   xhat = xhat + K * (y(t) - H * xhat);
```

```
   P = (eye(3) - K * H) * P * (eye(3) - K * H)' + K * R * K';
end
% Compute the mean square error.
err = 0;
for t = 1 : N
   H = [1 t t*t];
   err = err + (y(t) - H * xhat)^2;
end
err = sqrt(err / N);
disp(['Quadratic RMS Error = ',num2str(err)]);
H = [1 N+1 (N+1)^2];
disp(['Value predicted at next time = ', num2str(H*xhat)]);
% Plot the results.
t = 0 : N-1;
subplot(2,2,2);
plot(t, y, 'o', t, xhat(1)+xhat(2)*t+xhat(3)*t.*t);
legend('quadratic');
set(gca,'FontSize',12);

% Cubic curve fit.
xhat = [0; 0; 0; 0];
P = 100000 * eye(4);
R = 1;
for t = 1 : N
   H = [1 t t*t t*t*t];
   K = P * H' * inv(H * P * H' + R);
   xhat = xhat + K * (y(t) - H * xhat);
   P = (eye(4) - K * H) * P * (eye(4) - K * H)' + K * R * K';
end
% Compute the mean square error.
err = 0;
for t = 1 : N
   H = [1 t t*t t*t*t];
   err = err + (y(t) - H * xhat)^2;
end
err = sqrt(err / N);
disp(['Cubic RMS Error = ',num2str(err)]);
H = [1 N+1 (N+1)^2 (N+1)^3];
disp(['Value predicted at next time = ', num2str(H*xhat)]);
% Plot the results.
t = 0 : N-1;
subplot(2,2,3);
plot(t, y, 'o', t, xhat(1)+xhat(2)*t+xhat(3)*t.*t+xhat(4)*t.*t.*t);
legend('cubic');
set(gca,'FontSize',12);

% Quartic curve fit.
xhat = [0; 0; 0; 0; 0];
P = 100000 * eye(5);
R = 1;
```

```
for t = 1 : N
   H = [1 t t*t t*t*t t*t*t*t];
   K = P * H' * inv(H * P * H' + R);
   xhat = xhat + K * (y(t) - H * xhat);
   P = (eye(5) - K * H) * P * (eye(5) - K * H)' + K * R * K';
end
% Compute the mean square error.
err = 0;
for t = 1 : N
   H = [1 t t*t t*t*t t*t*t*t];
   err = err + (y(t) - H * xhat)^2;
end
err = sqrt(err / N);
disp(['Quartic RMS Error = ',num2str(err)]);
H = [1 N+1 (N+1)^2 (N+1)^3 (N+1)^4];
disp(['Value predicted at next time = ', num2str(H*xhat)]);
% Plot the results.
t = 0 : N-1;
subplot(2,2,4);
plot(t, y, 'o', t, xhat(1)+xhat(2)*t+xhat(3)*t.*t+xhat(4)*t.*t.*t+xhat(5)↙
*t.*t.*t.*t);
legend('quartic');
set(gca,'FontSize',12);
```

```
function [ErrW, ErrWC, ErrWN, ErrK] = Wiener1(dt)

% Optimal State Estimation Solution Manual, by Dan Simon
% Problem 3.14

% Wiener filter problem.
% Inputs:
%    dt = simulation step size (typically 0.1)
% Ouputs:
%    ErrW = E(e^2) for the parameter Wiener filter
%    ErrWC = E(e^2) for the causal Wiener filter
%    ErrWN = E(e^2) for the noncausal Wiener filter
%    ErrK = E(e^2) for the time varying Kalman filter

if ~exist('dt', 'var')
    dt = 0.1;
end
tf = 100; % simulation length
Qc = 2; % continuous time process noise
Rc = 1; % continuous time measurement noise
T = 1 / (sqrt(2) - 1); % parametric Wiener filter
x = 0; % initial state
xhatW = 0; % initial parametric Wiener estimate
xhatWC = 0; % initial causal Wiener estimate
xhatK = 0; % initial Kalman estimate
K = sqrt(3) - 1; % steady state Kalman gain
P = 0; % initial Kalman filter error covariance

xArr = [x];
yArr = [x];
xhatWArr = [xhatW];
xhatWCArr = [xhatWC];
xhatWNArr = [];
xhatKArr = [xhatK];

for t = dt : dt : tf+dt/10
    % System simulation
    x = exp(-dt) * x + sqrt(Qc*dt) * randn;
    y = x + sqrt(Rc/dt) * randn;
    % Parametric Wiener filter simulation
    xhatWdot = -xhatW / T + y / T;
    xhatW = xhatW + xhatWdot * dt;
    % Causal Wiener filter simulation
    xhatWCdot = -sqrt(3) * xhatWC + (sqrt(3) - 1) * y;
    xhatWC = xhatWC + xhatWCdot * dt;
    % Noncausal Wiener filter simulation
    xhatWCdot = -sqrt(3) * xhatWC + (sqrt(3) - 1) * y;
    xhatWC = xhatWC + xhatWCdot * dt;
    % Kalman filter simulation
    K = P * inv(Rc);
```

```
    xhatKdot = -xhatK + K * (y - xhatK);
    xhatK = xhatK + xhatKdot * dt;
    Pdot = -P^2 - 2 * P + 2;
    P = P + Pdot * dt;
    % Save data in arrays
    xArr = [xArr x];
    yArr = [yArr y];
    xhatWArr = [xhatWArr xhatW];
    xhatWCArr = [xhatWCArr xhatWC];
    xhatKArr = [xhatKArr xhatK];
end

% Noncausal Wiener filter
N = length(yArr);
xhatcArr = zeros(1,N);
xhataArr = zeros(1,N);
i = 0;
xhatc = 0; % causal part of Wiener filter output
xhata = 0; % anticausal part of Wiener filter output
for t = dt : dt : tf+dt/10
    xhatadot = sqrt(3) * xhata - yArr(N-i) / sqrt(3);
    xhata = xhata - xhatadot * dt;
    xhataArr(N-i) = xhata;
    i = i + 1;
    xhatcdot = -sqrt(3) * xhatc + yArr(i) / sqrt(3);
    xhatc = xhatc + xhatcdot * dt;
    xhatcArr = [xhatcArr xhatc];
end
for i = 1 : N
    xhatWNArr = [xhatWNArr xhatcArr(i)+xhataArr(i)];
end

close all;
figure; hold on;
plot(xArr,'r'); plot(xhatWArr,'b'); plot(xhatKArr,'k'); plot
(xhatWNArr,'g');
legend('true', 'Causal Wiener', 'Kalman', 'Noncausal Wiener');
ErrW = sum((xArr-xhatWArr).^2) / length(xArr);
ErrWC = sum((xArr-xhatWCArr).^2) / length(xArr);
ErrWN = sum((xArr-xhatWNArr).^2) / length(xArr);
ErrK = sum((xArr-xhatKArr).^2) / length(xArr);
disp(['Parametric Wiener Err^2 = ', num2str(ErrW)]);
disp(['Causal Wiener Err^2 = ', num2str(ErrWC)]);
disp(['Noncausal Wiener Err^2 = ', num2str(ErrWN)]);
disp(['Kalman Err^2 = ', num2str(ErrK)]);
```

```
function Prop1

% Optimal State Estimation Solution Manual, by Dan Simon
% Problem 4.11

% Mean and variance propagation for a linear system.

tf = 5;
dt = 0.1;

f = -0.5;
qc = 1;
m = 1;

P1 = 0;
P2 = 2;

mArray = m;
P1Array = P1;
P2Array = P2;

for t = dt : dt : tf+dt/10
   m = exp(f * dt) * m;
   P1 = (1 + 2 * f * dt) * P1 + qc * dt;
   P2 = (1 + 2 * f * dt) * P2 + qc * dt;
   mArray = [mArray m];
   P1Array = [P1Array P1];
   P2Array = [P2Array P2];
end

close all;
t = 0 : dt : tf+dt/10;
figure;
set(gcf,'Color','White');
set(gca,'FontSize',12);

subplot(2,1,1);
plot(t, mArray);
legend('mean');
set(gca,'FontSize',12);

subplot(2,1,2);
plot(t, P2Array, 'r-', t, P1Array, 'b:');
legend('variance (P_0 = 2)', 'variance (P_0 = 0)');
set(gca,'FontSize',12);

xlabel('time');
```

```
function Prop2

% Optimal State Estimation Solution Manual, by Dan Simon
% Problem 4.12

% Mean and variance propagation for an RLC circuit.

tf = 5;
dt = 0.1;

A = [-2 1 ; -1 0];
B = [1 ; 1];
Qc = B * 1 * B';

m = [1 ; 2];
P = [1 0 ; 0 2];

mArray = m;
PArray(:,:,1) = P;

k = 2;
for t = dt : dt : tf+dt/10
   u = 0;
   mdot = A * m + B * u;
   m = m + mdot * dt;
   Pdot = A * P + P * A' + Qc;
   P = P + Pdot * dt;
   mArray = [mArray m];
   PArray(:,:,k) = P;
   k = k + 1;
end

close all;
t = 0 : dt : tf+dt/10;
figure;
set(gcf,'Color','White');

subplot(2,1,1);
plot(t, mArray(1,:), 'r-', t, mArray(2,:), 'b:');
legend('m(1)', 'm(2)');
set(gca,'FontSize',12);

subplot(2,1,2);
plot(t, squeeze(PArray(1,1,:)), 'r-', t, squeeze(PArray(1,2,:)), 'm--', t,
squeeze(PArray(2,2,:)), 'b:');
legend('P(1,1)', 'P(1,2)', 'P(2,2)');
set(gca,'FontSize',12);

xlabel('time');
```

```
function Prop3

% Optimal State Estimation Solution Manual, by Dan Simon
% Problem 4.13

% Show the difference between the continuous steady state covariance
% and the discretized steady state covariance for an RLC circuit.

A = [0 2 ; -1 -3];
Qc = [0 0 ; 0 1];

k = 1;
for T = 0.01 : 0.01 : 1
    F = expm(A*T);
    Q = T * Qc;
    err(k) = norm(lyap(A, Qc) - dlyap(F, Q), 'fro');
    k = k + 1;
end

T = 0.01 : 0.01 : 1;
close all;
plot(T, err);
set(gcf,'Color','White');
set(gca,'FontSize',12);
xlabel('discretization step size');
ylabel('steady state covariance error');
```

```
function Population

% Optimal State Estimation Solution Manual, by Dan Simon
% Problem 5.11

% Kalman filter for wombat population estimation.

tf = 10; % simulation length

F = [1/2 2 ; 0 1];
Q = [0 0 ; 0 10];
H = [1 0];
R = 10;
Pplus = [500 0 ; 0 200]; % initial estimation error covariance

x = [650 ; 250]; % initial state
xhat = [600 ; 200]; % initial estimate

% Initialize arrays
xArr = x;
xhatArr = xhat;
PArr = [Pplus(1,1) ; Pplus(2,2)];
KArr = [];

for k = 1 : tf
    % System simulation
    x = F * x + sqrt(Q) * randn(2,1);
    y = H * x + sqrt(R) * randn;
    % Kalman filter simulation
    Pminus = F * Pplus * F' + Q;
    K = Pminus * H' * inv(H * Pminus * H' + R);
    xhat = F * xhat;
    xhat = xhat + K * (y - H * xhat);
    Pplus = (eye(2) - K * H) * Pminus;
    % Save data for plotting
    xArr = [xArr x];
    xhatArr = [xhatArr xhat];
    PArr = [PArr [Pplus(1,1) ; Pplus(2,2)]];
    KArr = [KArr K];
end

% Plot the results
close all;
k = 0 : tf;

figure; set(gca,'FontSize',12); set(gcf,'Color','White');
plot(k, xArr(1,:), 'r:', k, xhatArr(1,:), 'b-'); xlabel('time');
legend('true population', 'estimated population');

figure; set(gca,'FontSize',12); set(gcf,'Color','White');
```

```
plot(k, xArr(2,:), 'r:', k, xhatArr(2,:), 'b-'); xlabel('time');
legend('true food supply', 'estimated food supply');

figure; set(gca,'FontSize',12); set(gcf,'Color','White');
plot(k, sqrt(PArr(1,:)), 'r:', k, sqrt(PArr(2,:)), 'b-'); xlabel('time');
legend('populaticn est std dev', 'food supply est std dev');

k1 = 1 : tf;
figure; set(gca,'FontSize',12); set(gcf,'Color','White');
plot(k1, KArr(1,:), 'r:', k1, KArr(2,:), 'b-'); xlabel('time');
legend('K(1)', 'K(2)');

figure; set(gcf,'Color','White');
subplot(2,2,1);
plot(k, xArr(1,:), 'r:', k, xhatArr(1,:), 'b-'); set(gca,'FontSize',12);
ylabel('population');
legend('true', 'estimated');
subplot(2,2,2);
plot(k, xArr(2,:), 'r:', k, xhatArr(2,:), 'b-'); set(gca,'FontSize',12);
ylabel('food supply');
legend('true', 'estimated');
subplot(2,2,3);
plot(k, sqrt(PArr(1,:)), 'r:', k, sqrt(PArr(2,:)), 'b-'); set↙
(gca,'FontSize',12);
ylabel('std dev of est');
legend('population', 'food supply');
xlabel('time');
subplot(2,2,4);
plot(k1, KArr(1,:), 'r:', k1, KArr(2,:), 'b-'); xlabel('time'); set↙
(gca,'FontSize',12);
legend('K(1)', 'K(2)');
xlabel('time');

EstStd = std(xArr' - xhatArr');
disp(['Experimental std dev of estimation error = ', num2str(EstStd(1)),↙
', ', num2str(EstStd(2))]);
EstStd = sqrt(Pplus);
disp(['Theoretical std dev of estimation error = ', num2str(EstStd(1,1)),↙
', ', num2str(EstStd(2,2))]);
```

```
function RLC

% Optimal State Estimation Solution Manual, by Dan Simon
% Problem 5.12

% Kalman filter for RLC circuit state estimation.

T = 0.1;
F = exp(-T) * [2 2;-1 -1] + exp(-2*T) * [-1 -2;1 2];
H = [1 0];
q = 1;
Q = [0 0; 0 q];
R = 1;

x = [0 ; 0];
xhat = x;
Pplus = [0 0; 0 0];

N = 21;

Varminus = [];
Varplus = [Pplus(2,2)];
KArray = [];
xArray = [];
xhatArray = [];
yArray = [];

for k = 1 : N
    % Simulate the system and measurement
    x = F * x + [0; 1] * sqrt(q) * randn;
    y = H * x + sqrt(R) * randn;
    % Estimate the state
    Pminus = F * Pplus * F' + Q;
    K = Pminus * H' * inv(H * Pminus * H' + R);
    xhat = F * xhat;
    xhat = xhat + K * (y - H * xhat);
    Pplus = Pminus - K * H * Pminus;
    % Save data for plotting
    xArray = [xArray x];
    xhatArray = [xhatArray xhat];
    yArray = [yArray y];
    Varminus = [Varminus Pminus(2,2)];
    Varplus = [Varplus Pplus(2,2)];
    KArray = [KArray K];
end

disp(['inductor current variance = ', num2str(Pminus(2,2))]);

% Plot the results
close all;
```

```
k = 1 : N;
plot(k, yArray-xArray(1,:), 'r:');
hold;
plot(k, xhatArray(1,:)-xArray(1,:), 'b-');
set(gca,'FontSize',12); set(gcf,'Color','White');
xlabel('time step'); ylabel('position');
legend('measurement error', 'estimation error');

figure; hold;
for k = 1 : N-1
    plot([k-1 k], [Varplus(k) Varminus(k+1)]);
    plot([k k], [Varminus(k+1) Varplus(k+1)]);
end
set(gca,'FontSize',12); set(gcf,'Color','White'); set(gca,'Box','on');
xlabel('time step');
ylabel('estimation error variance');

Psi = [(F + Q * inv(F') * H' * inv(R) * H) (Q * inv(F')) ; (inv(F') * H' *
inv(R) * H) (inv(F'))];
Psip = Psi^2;
AB = Psip * [0 0;0 0; eye(2)];
A = AB(1:2,1:2);
B = AB(3:4,1:2);
A*inv(B)

Psip = Psip^2;
AB = Psip * [0 0;0 0; eye(2)];
A = AB(1:2,1:2);
B = AB(3:4,1:2);
A*inv(B)

Psip = Psip^2;
AB = Psip * [0 0;0 0; eye(2)];
A = AB(1:2,1:2);
B = AB(3:4,1:2);
A*inv(B)

Psip = Psip^2;
AB = Psip * [0 0;0 0; eye(2)];
A = AB(1:2,1:2);
B = AB(3:4,1:2);
A*inv(B)
```

```
function AltRLC

% Optimal State Estimation Solution Manual, by Dan Simon
% Problem 6.14

% Sequential Kalman filter for RLC circuit.

R = 100;
L = 1;
C = 1;
A = [-2/R/C 1/C ; -1/L 0];
B = [1/R/C ; 1/L];
H = eye(2); % measurement matrix
qc = 9; % variance of continuous time voltage input
Qc = B * qc * B'; % variance of continuous time process noise
R = eye(2); % covariance of discrete time measurement noise
tf = 2; % end time of simulation

T = 0.1; % discretization step size
F = expm(A*T); % state transition matrix
Q = Qc * T; % variance of discrete time process noise

x = [0 ; 0]; % initial state
[D, Lambda] = eig(Q); % eigendata of correlated process noise covariance
sqrtLambda = sqrt(Lambda);

xhatplus = x; % initial state estimate
Pplus = 0.1 * eye(2); % initial a posteriori estimation covariance
var0 = Pplus(1,1); % initial a posteriori variance of first state

% Initialize arrays
xArr = x;
xhatArr = xhatplus;
yArr = [];
varArr = [];

for t = T : T : tf
    % System simulation
    v = sqrtLambda * randn(2,1);
    w = D * v; % correlated process noise sample
    x = F * x + w; % discretized system dynamics
    y = H * x + sqrt(R) * randn(2,1); % measurement
    % Sequential Kalman filter simulation
    Pminus = F * Pplus * F' + Q;
    xhatminus = F * xhatplus;
    xhatplus = xhatminus;
    Pplus = Pminus;
    var(1,1) = Pplus(1,1);
    for i = 1 : 2
        K = Pplus * H(i,:)' / (H(i,:) * Pplus * H(i,:)' + R(i,i));
```

```
        xhatplus = xhatplus + K * (y(i) - H(i,:) * xhatplus);
        Pplus = Pplus - K * H(i,:) * Pplus;
        var(i+1,1) = Pplus(1,1);
    end
    % Save data in arrays
    xArr = [xArr x];
    xhatArr = [xhatArr xhatplus];
    yArr = [yArr y];
    varArr = [varArr var];
end

close all;

% Plot the true, estimated, and measured capacitor voltage
t = 0 : T : tf;
figure; hold on; set(gcf,'Color','White');
plot(t, xArr(1,:), 'r-', t, xhatArr(1,:), 'k:');
t = T : T : tf;
plot(t, yArr(1,:), 'b--');
set(gca,'FontSize',12);
xlabel('time'); ylabel('capacitor voltage');
legend('true', 'estimated', 'measured');
box('on');

% Plot the variance of the estimated capacitor voltage
t = T : T : tf;
figure; hold on; set(gcf,'Color','White');
plot(0, var0, 'o');
for i = 1 : 3
    plot(t, varArr(i,:), 'o');
end
tAll = [0 reshape([t ; t ; t], 1, 3*length(t))];
varAll = [var0 reshape(varArr, 1, 3*size(varArr,2))];
plot(tAll, varAll, 'k:');
set(gca,'FontSize',12);
xlabel('time'); ylabel('variance of capacitor voltage estimate');
box('on');
```

```
function Pitch

% Optimal State Estimation Solution Manual, by Dan Simon
% Problem 6.15

% Square root filter for aircraft state estimation

Mq = -0.568;
Ma = 17.98;
LaV = 1.237;
MdE = 0.175;
MdF = MdE;
LdEV = 0.001;
LdFV = LdEV;
qc = 0.001;

A = [Mq Ma; 1 -LaV];
B = [MdE MdF; -LdEV -LdFV];
Qc = [Ma; -LaV] * qc * [Ma -LaV];
H = eye(2);
R = diag([0.3 0.3]);
sqrtR = sqrt(R);

T = 0.01;
kf = 100;
F = expm(A*T);
G = (F * inv(A) - inv(A)) * B;
P = 0.01 * eye(2); % covariance of initial estimation error
S = sqrt(P);

Q = T * Qc;
[D, Lambda] = eig(Q);
sqrtLambda = sqrt(Lambda);
if norm(D*D' - eye(size(Q)), 'fro') > 100*eps
    disp('cannot find orthonormal eigenvector matrix for Q');
    return;
end

x = [0 ; 0]; % initial state
xhat = x; % initial state estimate

% Initialize arrays
xArr = x;
xhatArr = xhat;
yArr = x;
PArr(:,:,1) = P;

for k = 1 : kf
    % System simulation
    u = [0; 0];
```

```
    x = F * x + G * u;
    w = D * sqrtLambda * randn(2,1);
    x = x + w;
    y = H * x + sqrtR * randn(2,1);
    % Square root filter
    [W, T] = House1([S' * F' ; mychol(Q)]);
    S = W';
    xhat = F * xhat + G * u;
    for i = 1 : 2
        phi = S' * H(i,:)';
        a = 1 / (phi' * phi + R(i,i));
        gamma = 1 / (1 + sqrt(a * R(i,i)));
        S = S - S * a * gamma * phi * phi';
        K = a * S * phi;
        xhat = xhat + K * (y(i) - H(i,:) * xhat);
    end
    % Save data in arrays
    xArr = [xArr x];
    yArr = [yArr y];
    xhatArr = [xhatArr xhat];
    PArr(:,:,k+1) = S * S';
end

close all;
figure; set(gcf,'Color','White');
k = 0 : kf;
plot(k, xArr(1,:), 'k-', k, xhatArr(1,:), 'r--', k, yArr(1,:), 'b:');
set(gca,'FontSize',12);
xlabel('time'); ylabel('pitch rate');
legend('true', 'estimated', 'measured');

figure; set(gcf,'Color','White');
plot(k, xArr(2,:), 'k-', k, xhatArr(2,:), 'r--', k, yArr(2,:), 'b:');
set(gca,'FontSize',12);
xlabel('time'); ylabel('angle of attach');
legend('true', 'estimated', 'measured');

figure; set(gcf,'Color','White');
plot(k, squeeze(PArr(1,1,:)), 'r:', k, squeeze(PArr(2,2,:)), 'b-');
set(gca,'FontSize',12);
xlabel('time'); ylabel('estimation error covariances');
legend('pitch rate', 'angle of attack');
```

```
function [S] = mychol(P)

% Optimal State Estimation Solution Manual, by Dan Simon
% Problem 6.15

% Cholesky Decomposition
% Compute S such that S*S' = P for a (2 x 2) matrix P

S(2,2) = sqrt(P(2,2));
S(1,2) = P(1,2) / S(2,2);
S(2,1) = 0;
S(1,1) = sqrt(P(1,1) - S(1,2)^2);
if ~isreal(S(1,1))
    S(1,1) = 0;
end
```

```
function ColorScalar

% Optimal State Estimation Solution Manual, by Dan Simon
% Problem 7.10

% Kalman filter with colored noise

Q = 1; % variance of process noise
Qv = 1; % variance of white noise that affects measurement

v = 0; % measurement noise
x0 = 0; % initial state
x = x0;
xhatw0 = 0;
xhatw = xhatw0; % state estimate using filter that assumes white noise
xhatc = [xhatw0 ; 0]; % state estimate using filter that assumes colored
noise
% white Kalman filter steady state solution
Pw = (-7 + sqrt(65)) / 2;
Kw = Pw / Qv;
% colored Kalman filter steady stae solution
F = [1/2 0; 0 1/2];
H = [1 1];
Pcminus = dare(F', H', [Q 0; 0 Qv], 0);
Kc = Pcminus * H' * inv(H * Pcminus * H');

% initialize arrays
ErrwArray = []; % estimation error of filter that assumes white noise
ErrcArray = []; % estimation error of filter that assumes colored noise
xhatwArray = []; % estimate of filter that assumes white noise
xhatcArray = []; % estimate of filter that assumes colored noise
xArray = []; % true state

kf = 2000;
for k = 1 : kf
    % System
    w = randn;
    x = x / 2 + sqrt(Q) * w;
    v = v / 2 + sqrt(Qv) * randn;
    y = x + v;
    % Filters
    xhatw = xhatw / 2 + Kw * (y - xhatw / 2);
    xhatc = F * xhatc;
    xhatc = xhatc + Kc * (y - H * xhatc);
    % Save data in arrays
    ErrwArray = [ErrwArray x-xhatw];
    ErrcArray = [ErrcArray x-xhatc(1)];
    xhatwArray = [xhatwArray xhatw];
    xhatcArray = [xhatcArray xhatc];
    xArray = [xArray x];
```

```
end

Err2wAve = norm(ErrwArray,2)^2 / length(ErrwArray);
disp(['white E(e^2) = ', num2str(Err2wAve)]);
Err2cAve = norm(ErrcArray,2)^2 / length(ErrcArray);
disp(['colored E(e^2) = ', num2str(Err2cAve)]);

% Analytical estimation error variance for white Kalman filter
Ee2w = (1 - Kw)^2 * Q + Kw^2 * 4 * Qv / 3;
Ee2w = Ee2w + Kw * (1 - Kw) * 2 * Kw * Qv / 3;
Ee2w = Ee2w / (1 - (1 - Kw)^2 / 4);
disp(['Analytical error variance for white Kalman filter = ', num2str↵
(Ee2w)]);
% Analytical estimation error variance for colored Kalman filter
Pcplus = Pcminus - Kc * H * Pcminus;
disp(['Analytical error variance for colored Kalman filter = ', num2str↵
(Pcplus(1,1))]);

close all;
k = 1 : kf;
plot(k, xArray, 'k-', k, xhatwArray, 'b:', k, xhatcArray, 'r--');
legend('true', 'white filter', 'colored filter');

figure;
plot(k, ErrwArray, 'k-', k, ErrcArray, 'b:')
legend('white filter', 'colored filter');
```

```
function AlphaBeta1

% Optimal State Estimation Solution Manual, by Dan Simon
% Problem 7.11

% Plot alpha and beta parameters as a function of lambda (tracking index)

lambda = logspace(-3,3,100);
alpha = -1 / 8 * (lambda.^2 + 8 * lambda - (lambda + 4) .* sqrt(lambda.^2↙
+ 8 * lambda));
beta = 1 / 4 * (lambda.^2 + 4 * lambda - lambda .* sqrt(lambda.^2 + 8 *↙
lambda));
close all;
semilogx(lambda, alpha, 'k-', lambda, beta, 'r:');
set(gca,'FontSize',12); set(gcf,'Color','White');
legend('\alpha', '\beta');
xlabel('\lambda');
axisDef = axis;
axis([1e-3 1e+3 axisDef(3) axisDef(4)]);
```

```
function AlphaBetaGamma1

% Optimal State Estimation Solution Manual, by Dan Simon
% Problem 7.12

% Plot alpha, beta, and gamma parameters as a function of lambda
% (tracking index) for the alpha-beta-gamma filter.

lambda = logspace(-3,3,100);
b = lambda / 2 - 3;
c = lambda / 2 + 3;
p = c - b.^2 / 3;
q = 2 * b.^3 / 27 - b .* c / 3 - 1;
z = ((-q + sqrt(q.^2 + 4 * p.^3 / 27)) / 2).^(1/3);
s = z - p / 3 ./ z - b / 3;
alpha = 1 - s.^2;
beta = 2 * (1 - s).^2;
gamma = 2 * lambda .* s;
close all;
semilogx(lambda, alpha, 'k-', lambda, beta, 'r:', lambda, gamma, 'b--');
set(gca,'FontSize',12); set(gcf,'Color','White');
legend('\alpha', '\beta', '\gamma');
xlabel('\lambda');
axisDef = axis;
axis([1e-3 1e+3 axisDef(3) axisDef(4)]);
```

```
function Drug

% Optimal State Estimation Solution Manual, by Dan Simon
% Problem 7.13

% Constrained Kalman filter for drug quantity estimation.

k1 = 1;
k2 = 1;
A = [-k1 0; k1 -k2];
B = [1; 0];
H = [0 1];

tf = 3; % simulation time (days)
Qc = [0 0; 0 0]; % variance of continuous time process noise
R = 1; % variance of measurement noise
x = [0.8; 0]; %initial state
xmax = 1; % maximum possible value of x(1)
xmin = 0.8; % minimum possible value of x(1)

% initialize Kalman filter
xhat = x;
P = [1 0; 0 1];
% initialize constrained Kalman filter (W = I)
xhatI = x;
D = [1 0];
% initialize truncated Kalman filter
xhatT = x;

% discretization
dt = 1/24; % step size (days)
Q = Qc * dt;
F = expm(A*dt);
G = (F * inv(A) - inv(A)) * B;

% initialize arrays
xArr = x;
xhatArr = xhat;
xhatIArr = xhatI;
xhatTArr = xhatT;

for t = dt : dt : tf+dt/10
    % system simulation
    u = 1;
    x = F * x + G * u + sqrt(Q) * randn(size(x));
    y = H * x + sqrt(R) * randn;
    % Kalman filter
    P = F * P * F' + Q;
    K = P * H' * (H * P * H' + R)^(-1);
    xhat = F * xhat + G * u;
```

```
    xhat = xhat + K * (y - H * xhat);
    P = P - K * H * P;
    % Constrained Kalman filter (W = I)
    if D * xhat > xmax
        xhatI = xhat - D' * inv(D * D') * (D * xhat - xmax);
    elseif D * xhat < xmin
        xhatI = xhat - D' * inv(D * D') * (D * xhat - xmin);
    else
        xhatI = xhat;
    end
    % Truncated Kalman filter
    xhatT = xhat;
    PT = P;
    [T, W] = eig(PT);
    rho = rhoCalc(sqrt(W)*T'*D', sqrt(D*PT*D'));
    cki = (xmin - D * xhatT) / sqrt(D*PT*D');
    dki = (xmax - D * xhatT) / sqrt(D*PT*D');
    alpha = sqrt(2/pi) / (erf(dki/sqrt(2)) - erf(cki/sqrt(2)));
    mu = alpha * (exp(-cki^2/2) - exp(-dki^2/2));
    sigma2 = alpha * (exp(-cki^2/2) * (cki - 2 * mu) - exp(-dki^2/2) *
(dki - 2 * mu)) + mu^2 + 1;
    ztilde = [mu; 0];
    ztildeCov = diag([sigma2 1]);
    xhatT = T * sqrt(W) * rho' * ztilde + xhatT;
    PT = T * sqrt(W) * rho' * ztildeCov * rho * sqrt(W) * T';
    % Save data
    xArr = [xArr x];
    xhatArr = [xhatArr xhat];
    xhatIArr = [xhatIArr xhatI];
    xhatTArr = [xhatTArr xhatT];
end

% plot data
close all;
t = 0 : dt : tf;

figure;
plot(t, xArr(1,:), 'b-', t, xhatArr(1,:), 'r:', t, xhatIArr(1,:), 'k--',
t, xhatTArr(1,:), 'm-.');
set(gca,'FontSize',12); set(gcf,'Color','White');
xlabel('days'); ylabel('drug mass');
legend('true', 'unconstrained', 'projected', 'truncated');

figure;
plot(t, abs(xArr(1,:)-xhatArr(1,:)), 'b-', t, abs(xArr(1,:)-xhatIArr
(1,:)), 'r:', t, abs(xArr(1,:)-xhatTArr(1,:)), 'm--');
set(gca,'FontSize',12); set(gcf,'Color','White');
xlabel('days'); ylabel('drug mass estimation error');
legend('unconstrained', 'projected', 'truncated');
```

```
function [rho] = rhoCalc(x, y1)

% Optimal State Estimation Solution Manual, by Dan Simon
% Problem 7.13

% Compute an n x n matrix rho such that
% rho * x = [y1 ... 0]' and rho * rho' = I.
% This assumes that x is of the form W^(1/2)*T^T*phi, and
% y1 is of the form (phi^T*Q*phi)^(1/2).
% This function can be used as part of the pdf truncation algorithm
% for the inequality constrained Kalman filter.

n = length(x);
rho = zeros(n, n);
rho(1,:) = x' / y1;
I = eye(n);
for k = 2 : n
    rho(k,:) = I(k,:);
    for i = 1 : k-1
        rho(k,:) = rho(k,:) - I(k,:) * rho(i,:)' * rho(i,:);
    end
    if norm(rho(i,:)) < 100 * eps
        rho(k,:) = I(1,:);
        for i = 1 : k-1
            rho(k,:) = rho(k,:) - I(1,:) * rho(i,:)' * rho(i,:);
        end
    end
    rho(k,:) = rho(k,:) / norm(rho(k,:));
end
```

```
function ProcessNoise(dt)

% Optimal State Estimation Solution Manual, by Dan Simon
% Problem 8.11

% This program shows that a simulation gives the same state variance
% regardless of the simulation step size dt, as long as dt is small
% enough to give a stable simulation, and as long as the variance of
% the process noise is adjusted correctly with changes in dt.
% INPUT: dt = simulation step size

if ~exist('dt', 'var')
    dt = 0.5;
end

Q0 = 2;
Q = Q0 * dt;
tf = 10; % final simulation time

N = 5000; % number of Monte Carlo runs to estimate E(x^2(tf))
Expx2 = 0;
for i = 1 : N
    x = 0; % initial state
    for t = dt : dt : tf + dt/10
        x = x + sqrt(Q) * randn;
    end
    Expx2 = Expx2 + x^2;
end
Expx2 = Expx2 / N;
disp(['dt = ', num2str(dt), ' : Exp(x^2) = ', num2str(Expx2)]);
```

```
function Discretize(dt)

% Optimal State Estimation Solution Manual, by Dan Simon
% Problem 8.12

% Continuous time Kalman filter for a scalar system.
% INPUT: dt = integration step size

if ~exist('dt', 'var')
    dt = 0.4;
end

tf = 1000; % simulation length
Qc = 2; % continuous time process noise
Rc = 1; % continuous time measurement noise
x = 0; % initial state
xhat = 0; % initial Kalman estimate
P = 0; % initial Kalman filter error covariance

xArr = [x];
xhatArr = [xhat];
EstErr = 0;
for t = dt : dt : tf+dt/10
    % System simulation
    x = exp(-dt) * x + sqrt(Qc*dt) * randn;
    y = x + sqrt(Rc/dt) * randn;
    % Kalman filter simulation
    K = P * inv(Rc);
    xhatdot = -xhat + K * (y - xhat);
    xhat = xhat + xhatdot * dt;
    Pdot = -P^2 - 2 * P + 2;
    P = P + Pdot * dt;
    % Compute the estimation error
    EstErr = EstErr + (x - xhat)^2;
end
EstErr = EstErr / (tf / dt);
disp(['dt = ', num2str(dt), ' : Kalman Est Var = ', num2str(EstErr)]);
```

```
function ContKFGyro(SSFlag, M, MFilter)

% Optimal State Estimation Solution Manual, by Dan Simon
% Problems 8.13 and 8.14

% Continuous time Kalman filter
% INPUTS:
%     SSFlag = steady state flag
%     M = process noise / measurement noise correlation
%     MFilter = value of M that is used by the Kalman filter

if ~exist('SSFlag', 'var')
    SSFlag = 1;
end
if ~exist('M', 'var')
    M = 0;
end
if ~exist('MFilter', 'var')
    MFilter = 0;
end

A = [0 1 ; 0 0];
C = [1 0];
q = 2;
Q = [0 0 ; 0 q];
R = 3;

dt = 0.01;
tf = 10;

x = [0 ; 0];
xhat = x;
P = [1 0 ; 0 1];

xArr = x;
xhatArr = xhat;
k = 1;
PArr(:,:,k) = P;

Pss = [sqrt(2*R^(3/2)*q^(1/2)) sqrt(R*q) ; sqrt(R*q) sqrt(2*R^(1/2)*q^(3/2))];
Kss = Pss * C' * inv(R);

[v, d] = eig([q*dt M ; M R/dt]);

for t = dt : dt : tf + dt/10
    noise = v * sqrt(d) * randn(2,1);
    % System simulation
    xdot = A * x + sqrt(Q*dt) * randn(size(x));
    xdot = A * x + noise(1);
```

```
    x = x + xdot * dt;
    y = C * x + sqrt(R/dt) * randn;
    y = C * x + noise(2);
    % Kalman filter
    if SSFlag
        K = Kss;
        Pdot = A * P + P * A' - P * C' * inv(R) * C * P + Q;
    else
        K = (P * C' + MFilter) * inv(R);
        Pdot = A * P + P * A' - K * R * K' + Q;
    end
    xhatdot = A * xhat + K * (y - C * xhat);
    xhat = xhat + xhatdot * dt;
    P = P + Pdot * dt;
    % Save data for plotting
    xArr = [xArr x];
    xhatArr = [xhatArr xhat];
    k = k + 1;
    PArr(:,:,k) = P;
end

close all;
figure;
t = 0 : dt : tf + dt/10;
plot(t, xArr(1,:), 'r-', t, xhatArr(1,:), 'b:');
set(gca,'FontSize',12); set(gcf,'Color','White');
xlabel('time');
legend('true x(1)', 'estimated x(1)');

figure;
plot(t, squeeze(PArr(1,1,:)), 'r-', t, squeeze(PArr(2,2,:)), 'b:', t,
squeeze(PArr(1,2,:)), 'k--');
set(gca,'FontSize',12); set(gcf,'Color','White');
xlabel('time');
legend('P(1,1)', 'P(2,2)', 'P(1,2)');

disp(['Theoretical (M=0) steady state covariance = ', num2str(sqrt(2*R^
(3/2)*q^(1/2))), ', ', ...
    num2str(sqrt(R*q)), ', ', num2str(sqrt(2*R^(1/2)*q^(3/2)))]);
disp(['Numerical steady state covariance = ', num2str(P(1,1)), ', ',
num2str(P(1,2)), ...
    ', ', num2str(P(2,2))]);

EstErr = xArr - xhatArr;
PExp = EstErr * EstErr' / length(t);
disp(['Simulation error covariance = ', num2str(PExp(1,1)), ', ', num2str
(PExp(1,2)), ...
    ', ', num2str(PExp(2,2))]);
```

```
function ContEx

% Optimal State Estimation Solution Manual, by Dan Simon
% Problem 8.15

% Continuous Kalman filter for a two-state problem.

close all;
figure;
subplot(2,2,1);
CAREProblem(1,2,1,1,1,1,1,1,0,1);
subplot(2,2,2);
CAREProblem(-1,-1,1,2,4,1,1,1,0,1);
subplot(2,2,3);
CAREProblem(1,1,1,2,4,1,1,1,0,1);
subplot(2,2,4);
CAREProblem(1,1,1,2,4,1,1,0,0,0);
return;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function CAREProblem(a1, a2, q11, q12, q22, r1, r2, p11, p12, p22)

disp('CARE Solution:')
disp(care([a1 0;0 a2]', [1 0;0 1]', [q11 q12;q12 q22], [r1 0;0 r2]));

PArr = [];
dt = 0.01;
tf = 4;
% Plot the time varying solution.
for t = 0 : dt : tf
    p11dot = 2 * a1 * p11 - p11^2 / r1 - p12^2 / r2 + q11;
    p12dot = (a1 + a2) * p12 - p11 * p12 / r1 - p12 * p22 / r2 + q12;
    p22dot = 2 * a2 * p22 - p12^2 / r1 - p22^2 / r2 + q22;
    p11 = p11 + p11dot * dt;
    p12 = p12 + p12dot * dt;
    p22 = p22 + p22dot * dt;
    PArr = [PArr ; p11 p12 p22];
end
t = 0 : dt : tf;
plot(t, PArr(:, 1), 'r:', t, PArr(:, 2), 'k--', t, PArr(:, 3), 'b-');
set(gca,'FontSize',12); set(gcf,'Color','White');
xlabel('time');
legend('P(1,1)', 'P(1,2)', 'P(2,2)');
% Compute the steady state solution.
Cond1 = (a1 ~= a2) && (q12 ~= 0);
Cond2 = (a1 == a2) && (a1 < 0) && (q12 ~= 0) && (q11*q22-q12*q12 == 0);
Cond3 = (a1 == a2) && (a1 > 0) && (q12 ~= 0) && (q11*q22-q12*q12 == 0);
if Cond1 || Cond3
    gamma1 = q11 / r1 + a1^2;
    gamma2 = q22 / r2 + a2^2;
```

```
    p12 = q12 / ( gamma1 + gamma2 + 2 * ( gamma1 * gamma2 - q12^2 / r1 /↵
r2 )^(1/2) )^(1/2);
    p11 = r1 * ( a1 + ( gamma1 - p12^2 / r1 / r2 )^(1/2) );
    p22 = r2 * ( a2 + ( gamma2 - p12^2 / r1 / r2 )^(1/2) );
    disp('Analytical Solution:');
    disp([p11 p12;p12 p22]);
    lambda = eig([p11 p12; p12 p22]);
    disp(['Eigenvalues of P = ', num2str(lambda(1)), ', ', num2str(lambda↵
(2))]);
end
if Cond2 || Cond3
    gamma3 = -a1 + ( a1^2 + q11 / r1 + q22 / r2 )^(1/2);
    p11 = q11 / gamma3;
    p22 = q22 / gamma3;
    p12 = q12 / gamma3;
    disp('Analytical Solution:');
    disp([p11 p12;p12 p22]);
    lambda = eig([p11 p12; p12 p22]);
    disp(['Eigenvalues of P = ', num2str(lambda(1)), ', ', num2str(lambda↵
(2))]);
end
if ~Cond1 && ~Cond2 && ~Cond3
    disp('Numerical Solution:');
    disp([p11 p12;p12 p22]);
    lambda = eig([p11 p12; p12 p22]);
    disp(['Eigenvalues of P = ', num2str(lambda(1)), ', ', num2str(lambda↵
(2))]);
end
```

```
function FixPt1

% Optimal State Estimation Solution Manual, by Dan Simon
% Problems 9.4, 9.5

% Fixed point smoother for a scalar problem.
% Verify the results of the written solution.

Q = 1;
R = 2 * Q;
R = 12 * Q;

P = (Q + sqrt(Q*Q+4*Q*R)) / 2;
Sigma = P;
Pi = P;
F = 1;
H = 1;
for k = 1 : 100
    L = F * P * H' * inv(H * P * H' + R);
    lambda = Sigma * H' * inv(H * P * H' + R);
    P = F * P * (F - L * H)' + Q;
    Pi = Pi - Sigma * H' * lambda';
    Sigma = Sigma * (F - L * H)';
end
```

```
function RTS2(Qc)

% Optimal State Estimation Solution Manual, by Dan Simon
% Problems 9.15 and 9.16

% RTS smoother applied to a second order system.
% INPUT: Qc = continuous time noise covariance

if ~exist('Qc', 'var')
    Qc = 0.01;
end

w = 6; % natural frequency of the system
z = 0.16; % damping ratio of the system
A = [0 1 ; -w^2 -2*z*w]; % system matrix
H = [1 0]; % measurement matrix
Qc = [0 0 ; 0 Qc]; % process noise covariance
R = 1e-4; % measurement noise covariance

dt = 0.5; % discretization step size
Q = Qc * dt; % discrete time process noise covariance
F = expm(A * dt); % state transition matrix

x = [1 ; 1]; % initial state
xhatf = x; % initial forward state estimate
Pfplus = diag([1e-5 1e-2]); % initial forward estimation covariance
PfplusArr(:,:,1) = Pfplus;
xhatfArr(:,1) = xhatf;
xhatfminusArr = [];

kf = 10;
for k = 1 : kf
    % system simulation
    x = F * x + sqrt(Q) * randn(2,1);
    y = H * x + sqrt(R) * randn;
    % forward Kalman filter
    Pfminus = F * Pfplus * F' + Q;
    Kf = Pfminus * H' * inv(H * Pfminus * H' + R);
    xhatfminus = F * xhatf;
    xhatf = xhatfminus + Kf * (y - H * xhatfminus);
    Pfplus = Pfminus - Kf * H * Pfminus;
    % save covariances for later
    PfminusArr(:,:,k) = Pfminus;
    PfplusArr(:,:,k+1) = Pfplus;
    xhatfminusArr = [xhatfminusArr xhatfminus];
    xhatfArr = [xhatfArr xhatf];
end
% RTS equations
xhat = xhatf;
P = Pfplus;
```

```
PArr(:,:,kf+1) = P;
for k = kf-1 : -1 : 0
    Ifminus = inv(PfminusArr(:,:,k+1));
    K = PfplusArr(:,:,k+1) * F' * Ifminus;
    P = PfplusArr(:,:,k+1) - K * (PfminusArr(:,:,k+1) - P) * K';
    xhat = xhatfArr(:,k+1) + K * (xhat - xhatfminusArr(:,k+1));
    % save covariance for later
    PArr(:,:,k+1) = P;
end

% Plot stuff
close all;
k = 0 : kf;
figure;
subplot(2,1,1);
plot(k, squeeze(PfplusArr(1,1,:)), 'k-', k, squeeze(PArr(1,1,:)), 'r:');
set(gca,'FontSize',12); set(gcf,'Color','White');
ylabel('first state');
legend('forward covariance', 'smoothed covariance');
subplot(2,1,2);
plot(k, squeeze(PfplusArr(2,2,:)), 'k-', k, squeeze(PArr(2,2,:)), 'r:');
legend('forward covariance', 'smoothed covariance');
set(gca,'FontSize',12);
xlabel('time');
ylabel('second state');
```

```
function [Err0] = PopSmooth(PlotFlag)

% Optimal State Estimation Solution Manual, by Dan Simon
% Problem 9.17

% RTS fixed interval smoother for population estimation.
% INPUT: PlotFlag = flag indicating whether or not to plot stuff
% OUTPUT: Err0 = estimation error of smoothed estimate at initial time

if ~exist('PlotFlag', 'var')
    PlotFlag = true;
end

kf = 10; % simulation length

F = [1/2 2 ; 0 1];
Q = [0 0 ; 0 10];
H = [1 0];
R = 10;
Pfplus = [500 0 ; 0 200]; % initial estimation error covariance

x0 = [650 ; 250]; % initial state
x = x0; % state
xhatf = [600 ; 200]; % initial estimate

% Initialize arrays
xArr = x;
xhatfminusArr = [];
xhatfArr = xhatf;
PfplusArr(:,:,1) = Pfplus;

for k = 1 : kf
    % System simulation
    x = F * x + sqrt(Q) * randn(2,1);
    y = H * x + sqrt(R) * randn;
    % Kalman filter simulation
    Pfminus = F * Pfplus * F' + Q;
    Kf = Pfminus * H' * inv(H * Pfminus * H' + R);
    xhatfminus = F * xhatf;
    xhatf = xhatfminus + Kf * (y - H * xhatfminus);
    Pfplus = Pfminus - Kf * H * Pfminus;
    % Save data for plotting
    xArr = [xArr x];
    xhatfminusArr = [xhatfminusArr xhatfminus];
    xhatfArr = [xhatfArr xhatf];
    PfminusArr(:,:,k) = Pfminus;
    PfplusArr(:,:,k+1) = Pfplus;
end

% RTS equations
```

```
xhat = xhatf;
xhatArr(:,:,kf+1) = xhat;
P = Pfplus;
PArr(:,:,kf+1) = P;
for k = kf-1 : -1 : 0
    Ifminus = inv(PfminusArr(:,:,k+1));
    K = PfplusArr(:,:,k+1) * F' * Ifminus;
    P = PfplusArr(:,:,k+1) - K * (PfminusArr(:,:,k+1) - P) * K';
    xhat = xhatfArr(:,k+1) + K * (xhat - xhatfminusArr(:,k+1));
    % save for later
    PArr(:,:,k+1) = P;
    xhatArr(:,:,k+1) = xhat;
end
Err0 = x0 - xhat;

if ~PlotFlag
    return;
end

close all;
figure; k = 0 : kf;
subplot(2,1,1);
plot(k, xArr(1,:), 'r-', k, xhatfArr(1,:), 'b:', k, xhatArr(1,:), 'k--');
set(gca,'FontSize',12); set(gcf,'Color','White');
legend('true', 'forward estimate', 'smoothed estimate');
ylabel('first state');
subplot(2,1,2);
plot(k, xArr(2,:), 'r-', k, xhatfArr(2,:), 'b:', k, xhatArr(2,:), 'k--');
set(gca,'FontSize',12); set(gcf,'Color','White');
ylabel('second state');
xlabel('time');

% Plot stuff
figure;
subplot(2,1,1);
plot(k, squeeze(PfplusArr(1,1,:)), 'k-', k, squeeze(PArr(1,1,:)), 'r:');
set(gca,'FontSize',12); set(gcf,'Color','White');
ylabel('first state');
legend('forward a posteriori covariance', 'smoothed covariance');
subplot(2,1,2);
plot(k, squeeze(PfplusArr(2,2,:)), 'k-', k, squeeze(PArr(2,2,:)), 'r:');
legend('forward a posteriori covariance', 'smoothed covariance');
set(gca,'FontSize',12);
xlabel('time');
ylabel('second state');

Cov1Improve = 100 * (PfplusArr(1,1,1) - PArr(1,1,1)) / PfplusArr(1,1,1);
Cov2Improve = 100 * (PfplusArr(2,2,1) - PArr(2,2,1)) / PfplusArr(2,2,1);
disp(['Improvement in 1st state covariance = ', num2str(Cov1Improve), '↵
%']);
```

```
disp(['Improvement in 2nd state covariance = ', num2str(Cov2Improve), '↙
%']);

PArr(:,:,1)
```

```
function PopSmooth1(PlotFlag)

% Optimal State Estimation Solution Manual, by Dan Simon
% Problem 9.17

% Forward-backward fixed interval smoother for population estimation.
% INPUT: PlotFlag = flag indicating whether or not to plot stuff

if ~exist('PlotFlag', 'var')
    PlotFlag = true;
end

kf = 10; % simulation length

F = [1/2 2 ; 0 1];
Q = [0 0 ; 0 10];
H = [1 0];
R = 10;

x0 = [650 ; 250]; % initial state
x = x0; % state
xhatf = [600 ; 200]; % initial estimate

yArr = [];
% Simulate the system and collect the measurements.
% Use a constant commanded acceleration.
for k = 0 : kf
    x = F * x + sqrt(Q) * randn(2,1);
    y = H * x + sqrt(R) * randn;
    yArr = [yArr y];
end

% Obtain the forward estimate.
Pfplus = [500 0 ; 0 200]; % initial estimation error covariance
PfplusArr(:,:,1) = Pfplus;
xhatf = x0; % initial state estimate
for k = 1 : kf
   % Extrapolate the most recent state estimate to the present time.
   xhatf = F * xhatf;
   % Form the Innovation vector.
   Inn = yArr(k) - H * xhatf;
   % Compute the covariance of the a priori estimation error
   Pfminus = F * Pfplus * F' + Q;
   % Compute the covariance of the Innovation.
   CovInn = H * Pfminus * H' + R;
   % Form the Kalman Gain matrix.
   K = Pfminus * H' * inv(CovInn);
   % Compute the covariance of the estimation error.
   Pfplus = Pfminus - K * H * Pfminus;
   % Update the state estimate.
```

```
   xhatf = xhatf + K * Inn;
   % Save some parameters for plotting later.
   PfplusArr(:,:,k+1) = Pfplus;
end

% Obtain the backward estimate.
% The initial backward information matrix Ibminus needs to be set to a
% small nonzero matrix so that the first calculated Ibplus is invertible.
Ibminus = [1e-12 0; 0 1e-12];
PbArr(:,:,kf+1) = inv(Ibminus);
sminus = zeros(size(x)); % initial backward modified estimate
Q = Q + [0 0; 0 0];
for k = kf : -1 : 1
    Ibplus = Ibminus + H' * inv(R) * H;
    splus = sminus + H' * inv(R) * yArr(k);
    % The following line does not work unless Sw is invertible.
    %Ibminus = inv(Sw) - inv(Sw) * inv(a) * inv(Ibplus + inv(a)' * inv(Sw)↙
* inv(a)) * inv(a)' * inv(Sw);
    %Ibminus = inv(inv(F) * inv(Ibplus) * inv(F)' + inv(F) * Q * inv(F)');
    Ibminus = F' * inv(Q + inv(Ibplus)) * F;
    sminus = Ibminus * inv(F) * inv(Ibplus) * splus;
    % Save some parameters for plotting later.
    Pbminus = inv(Ibminus);
    PbArr(:,:,k) = Pbminus;
end

% Compute the smoothed estimation covariance.
for k = 0 : kf
    P(:,:,k+1) = inv(inv(PfplusArr(:,:,k+1)) + inv(PbArr(:,:,k+1)));
end

close all;
figure;
set(gca,'FontSize',12); set(gcf,'Color','White');
k = 0 : kf;

subplot(2,1,1);
plot(k, squeeze(PfplusArr(1,1,:)), 'r-', k, squeeze(PbArr(1,1,:)), 'b--',↙
k, squeeze(P(1,1,:)), 'k:');
axis([0 kf 0 max(PfplusArr(1,1,:))]);
legend('forward', 'backward', 'smoothed');
ylabel('first state');

subplot(2,1,2);
plot(k, squeeze(PfplusArr(2,2,:)), 'r-', k, squeeze(PbArr(2,2,:)), 'b--',↙
k, squeeze(P(2,2,:)), 'k:');
axis([0 kf 0 max(PfplusArr(2,2,:))]);
ylabel('second state');
```

```
function Tire(Fmodel)

% Optimal State Estimation Solution Manual, by Dan Simon
% Problem 10.14

% Tire tread estimation.
% INPUT: Fmodel = assumed value of F matrix. This may or may not match
% the true value of the F matrix.

if ~exist('Fmodel', 'var')
    Fmodel = 0.8;
end
tf = 10; % simulation length
Ftrue = 0.8;
Q = 0.01;
H = 1;
R = 0.01;
Pplus = 0; % initial estimation error covariance
ResMeanArr = [];
ResStdArr = [];
for NumTires = 1 : 1000
    x = 1; % initial state
    xhat = 1; % initial estimate
    % Initialize arrays
    ResArr = [];
    for k = 1 : tf
        % System simulation
        x = Ftrue * x + sqrt(Q) * randn;
        y = H * x + sqrt(R) * randn;
        % Kalman filter simulation
        Pminus = Fmodel * Pplus * Fmodel' + Q;
        K = Pminus * H' * inv(H * Pminus * H' + R);
        xhatminus = Fmodel * xhat;
        xhat = xhatminus + K * (y - H * xhatminus);
        Pplus = (1 - K * H) * Pminus;
        % Save data
        ResArr = [ResArr y - xhatminus];
    end
    ResMeanArr = [ResMeanArr mean(ResArr)];
    ResStdArr = std(ResArr, 1);
end
disp(['Mean of residuals = ', num2str(mean(ResMeanArr))]);
% Compare the experimental and theoretical standard deviation of the
residual
disp(['Experimental std dev of residual = ', num2str(mean(ResStdArr))]);
disp(['Theoretical std dev of estimation error = ', num2str(sqrt(Pminus +
R))]);
```

```
function TireMulti

% Optimal State Estimation Solution Manual, by Dan Simon
% Problem 10.15

% Multiple model Kalman filtering for tire tread estimation.

global F
F = [0.8 ; 0.85 ; 0.9]; % possible values of F
NumTires = 100;
for i = 1 : NumTires
    [FhatArr(i,:), prArr(i,:,:), Fhat] = TireMultiSim(false);
end
Fhat1 = mean(FhatArr,1);
pr1(1,:) = mean(squeeze(prArr(:,1,:)),1);
pr1(2,:) = mean(squeeze(prArr(:,2,:)),1);
pr1(3,:) = mean(squeeze(prArr(:,3,:)),1);

close all
t = 0 : length(Fhat1) - 1;
figure;
subplot(2,1,1);
plot(t, Fhat1);
set(gca,'FontSize',12); set(gcf,'Color','White');
ylabel('Estimate of f');
subplot(2,1,2);
plot(t, pr1(1,:), 'b-', t, pr1(2,:), 'k--', t, pr1(3,:), 'r:');
set(gca,'FontSize',12); set(gcf,'Color','White');
ylabel('Probabilities of f');
xlabel('time step');
legend(['Probability that f = ',num2str(F(1))], ['Probability that f = ',
num2str(F(2))], ['Probability that f = ',num2str(F(3))]);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [FhatArray, prArray, Fhat] = TireMultiSim(PlotFlag)

% INPUT: PlotFlag = 1 means plot results
% OUTPUT:   FhatArray = array of F estimates as a function of time
%           prArray = array of F probabilities as a function of time
%           Fhat = final estimate of F

if ~exist('PlotFlag', 'var')
    PlotFlag = true;
end

global F
kf = 10; % Length of simulation
N = size(F, 1); % number of parameter sets
pr = ones(size(F)) / N; % a priori probabilities
% Compute the initial estimate of wn
```

```
Fhat = 0;
for i = 1 : N
   Fhat = Fhat + F(i) * pr(i);
end
Q = 0.01; % discrete time process noise variance
R = 0.01; % discrete time measurement noise covariance
H = 1; % measurement matrix
x = 1; % initial state

n = length(x); % number of states
q = size(H, 1); % number of measurements

% Initialize the N state estimators
for i = 1 : N
   Pplus(i) = zeros(n);
   xhat(i) = x;
end

% Create arrays for later plotting
FhatArray = [Fhat];
prArray = [pr];
for k = 1 : kf
   % Simulate the system.
   x = F(1) * x + sqrt(Q) * randn;
   y = H * x + sqrt(R) * randn;
   % Run a separate Kalman filter for each parameter set.
   for i = 1 : N
      Pminus(i) = F(i) * Pplus(i) * F(i)' + Q;
      K = Pminus(i) * H' * inv(H * Pminus(i) * H' + R);
      xhat(i) = F(i) * xhat(i);
      r = y - H * xhat(i); % measurment residual
      S = H * Pminus(i) * H' + R; % covariance of measurement residual
      pdf(i) = exp(-r'*inv(S)*r/2) / ((2*pi)^(q/2)) / sqrt(det(S));
      xhat(i) = xhat(i) + K * (y - H * xhat(i));
      Pplus(i) = (eye(n) - K * H) * Pminus(i) * (eye(n) - K * H)' + K * R
* K';
   end
   % Compute the sum that appears in the denominator of the probability
expression.
   Prsum = 0;
   for i = 1 : N
      Prsum = Prsum + pdf(i) * pr(i);
   end
   % Update the probability of each parameter set.
   for i = 1 : N
      pr(i) = pdf(i) * pr(i) / Prsum;
   end
   % Compute the best state estimate and the best parameter estimate.
   xhatbest = 0;
   Fhat = 0;
```

```
   for i = 1 : N
      xhatbest = xhatbest + pr(i) * xhat(:,i);
      Fhat = Fhat + pr(i) * F(i);
   end
   % Save data for plotting.
   FhatArray = [FhatArray Fhat];
   prArray = [prArray pr];
end

if PlotFlag
    close all;
    t = 0 : kf;
    figure;
    plot(t, FhatArray);
    title('Estimate of f', 'FontSize', 12);
    set(gca,'FontSize',12); set(gcf,'Color','White');
    xlabel('time step');
    figure;
    plot(t, prArray(1,:), 'b-', t, prArray(2,:), 'k--', t, prArray(3,:),↵
'r:');
    title('Probabilities of f', 'FontSize', 12);
    set(gca,'FontSize',12); set(gcf,'Color','White');
    xlabel('time step');
    legend(['Probability that f = ',num2str(F(1))], ['Probability that f =↵
',num2str(F(2))], ['Probability that f = ',num2str(F(3))]);
end

return;
```

```
function RobustScalar

% Optimal State Estimation Solution Manual, by Dan Simon
% Problem 10.16

% Robust Kalman filter for a scalar system.

Q = 5;
sigma1 = 0.5;
R = 5;
sigma2 = 1;
Kmin = 0.3;
Kmax = 0.7;
K = Kmin : (Kmax - Kmin)/100 : Kmax;
P = ((1 - K).^2 * Q + K.^2 * R) ./ (2 * K - K.^2);
VardP = (sigma1^2 * (1 - K).^2 * Q^2 + sigma2^2 * K.^4 * R^2) ./ (2 * K -
K.^2);

close all;
figure;
plot(K, P+VardP, 'k:', K, VardP, 'b-', K, P, 'r--');
set(gca,'FontSize',12); set(gcf,'Color','White');
xlabel('K');
legend('P + E(\Delta P^2)', 'E(\Delta P^2)', 'P');

[temp, i] = min(P);
disp(['min P occurs at K = ', num2str(K(i))]);
[temp, i] = min(VardP);
disp(['min E(dP^2) occurs at K = ', num2str(K(i))]);
[temp, i] = min(P+VardP);
disp(['min P+E(dP^2) occurs at K = ', num2str(K(i))]);
```

```
function HinfRadio(kf, theta)

% Optimal State Estimation Solution Manual, by Dan Simon
% Problem 11.14

% H-infinity estimators for radioactive decay problem.
% INPUTS:   kf = simulation length
%           theta = performance bound

if ~exist('kf', 'var')
    kf = 20;
end
if ~exist('theta', 'var')
    theta = 1;
end

P = 1;
Q = 1;
R = 1;
x = 0; % initial state
xhat = x; % H-infinity estimate
xArr = [x];
xhatArr = [xhat];
PArr = [P];
for k = 1 : kf
    % System simulation
    y = x + sqrt(R) * randn;
    x = x + sqrt(Q) * randn;
    % H-infinity filter simulation
    K = P * inv(1 - theta * P + P / R) / R;
    P = 1 / 4 * P * inv(1 - theta * P + P / R) + Q;
    xhat = xhat + K * (y - xhat);
    % Check for necessary conditions.
    if P <= 0
        disp('P <= 0'); return;
    elseif (1/P - theta + 1/R) <= 0
        disp('1/P - theta + 1/R <= 0'); return;
    end
    % Save data in arrays
    xArr = [xArr x];
    xhatArr = [xhatArr xhat];
    PArr = [PArr P];
end

k = 0 : kf;
close all;
figure; hold on;
subplot(2,1,1);
plot(k, xArr, 'k', 'LineWidth', 2.5); hold on;
subplot(2,1,1);
```

```
plot(k, xhatArr, 'r--');
set(gca,'FontSize',12); set(gcf,'Color','White');
legend('true state', 'H_{\infty} estimate');
xlabel('time'); ylabel('state value');
set(gca,'box','on');

subplot(2,1,2);
plot(k, PArr);
ylabel('P');

P = (1 - 4*theta + sqrt((1-4*theta)^2 + 64 * (1 - theta))) / 8 / (1 -
theta);
disp(['Theoretical steady state P = ', num2str(P)]);
```

```
function HinfVehicle(theta)

% Optimal State Estimation Solution Manual, by Dan Simon
% Problem 11.15

% function HinfVehicle
% This m-file simulates a vehicle tracking problem.
% The vehicle state is estimated with a Kalman and an H-infinity filter.
% The state consists of the north and east position, and the
% north and east velocity of the vehicle.
% The measurement consists of north and east positions.
% INPUT:    theta = H-infinity performance parameter

if ~exist('theta', 'var')
    theta = 0.0005;
end
close all;
figure;
HinfVehicleSim(theta, 1, 1);
HinfVehicleSim(theta, 2, 2);
return;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function HinfVehicleSim(theta, uTrue, FigNum)

tf = 300; % final time (seconds)
T = 3; % time step (seconds)

Q = diag([4, 4, 1, 1]); % Process noise covariance (m, m, m/sec, m/sec)
Qsqrt = sqrt(Q);

R = diag([900, 900]); % Measurement noise covariance (m, m)
Rsqrt = sqrt(R);

angle = 0.9 * pi; % heading angle (measured CCW from east)

S = eye(4);
L = eye(4);
Sbar = L' * S * L;

% Define the initial state x, initial unconstrained filter estimate xhat,
% and initial constrained Kalman filter estimate xtilde.
x = [0; 0; 0; 0];
xhatK = x; % Unconstrained Kalman filter
xhatH = x; % Kalman filter with perfect measurements

% Initial estimation error covariance
PK = diag([0, 0, 0, 0]);
PH = PK;
```

```
F = [1 0 T 0 ; 0 1 0 T ; 0 0 1 0 ; 0 0 0 1]; % system matrix
B = [0; 0; T*sin(angle); T*cos(angle)]; % input matrix
H = [1 0 0 0 ; 0 1 0 0]; % measurement matrix

% Initialize arrays for saving data for plotting.
xarray = x;
xhatKarray = xhatK;
xhatHarray = xhatH;

randn('state', sum(100*clock)); % initialize random number generator

% Get the measurement at the initial time.
y = H * x + Rsqrt * randn(size(Rsqrt,1),1);

% Begin the simulation.
for t = T : T : tf+eps
    % Set the "known" input.
    uModel = 1;
    % Simulate the Kalman filter.
    temp = inv( eye(4) + H' * inv(R) * H * PK );
    KK = PK * temp * H' * inv(R);
    xhatK = F * xhatK + B * uModel + F * KK * (y - H * xhatK);
    PK = F * PK * temp * F' + Q;
    % Simulate the H-infinity filter.
    temp = eye(4) - theta * Sbar * PH + H' * inv(R) * H * PH;
    if cond(temp) > 1e10
        disp('ill conditioned H-infinity estimation problem');
        return;
    end
    temp = inv(temp);
    KH = PH * temp * H' * inv(R);
    xhatH = F * xhatH + B * uModel + F * KH * (y - H * xhatH);
    PH = F * PH * temp * F' + Q;
    lambda = eig(inv(PH) - theta * Sbar + H' * inv(R) * H);
    if min(real(lambda)) <= 0
        disp('H-infinity performance bound out of range');
        return;
    end
    % Simulate the system.
    x = F*x + B*uTrue + Qsqrt*randn(size(x));
    % Get the measurement.
    y = H * x + Rsqrt * randn(size(Rsqrt,1),1);
    % Save data in arrays.
    xhatKarray = [xhatKarray xhatK];
    xhatHarray = [xhatHarray xhatH];
    xarray = [xarray x];
end

% Compute averages.
EstErrorK = xarray - xhatKarray;
```

```
EstErrorK = sqrt(EstErrorK(1,:).^2 + EstErrorK(2,:).^2);
EstError = mean(EstErrorK);
disp(['Kalman Average Position Estimation Error = ', num2str(EstError)]);

EstErrorH = xarray - xhatHarray;
EstErrorH = sqrt(EstErrorH(1,:).^2 + EstErrorH(2,:).^2);
EstError = mean(EstErrorH);
disp(['H-infinity Average Position Estimation Error = ', num2str↙
(EstError)]);

% Plot data.
t = 0 : T : tf;
subplot(2,1,FigNum);
plot(t, EstErrorK, 'r:', 'LineWidth', 2.5); hold on;
plot(t, EstErrorH, 'b-', 'LineWidth', 1.5);
set(gca,'FontSize',12); set(gcf,'Color','White');
title(['True Input = ', num2str(uTrue)]);
if FigNum == 2
    xlabel('seconds');
end
ylabel('pos. est. errors');
legend('Kalman', 'H_{\infty}');

EstErrorH = xarray - xhatHarray;
stdH = std(EstErrorH, 0, 2);
```

```
function HinfVehicle1(theta)

% Optimal State Estimation Solution Manual, by Dan Simon
% Problem 11.15

% Calculate closed loop Kalman and H-infinity filter eigenvalues for
% the vehicle navigation problem.

if ~exist('theta', 'var')
    theta = 0.0005;
end

T = 3; % time step (seconds)
Q = diag([4, 4, 1, 1]); % Process noise covariance (m, m, m/sec, m/sec)
R = diag([900, 900]); % Measurement noise covariance (m, m)
S = eye(4);
L = eye(4);
Sbar = L' * S * L;
F = [1 0 T 0 ; 0 1 0 T ; 0 0 1 0 ; 0 0 0 1]; % system matrix
H = [1 0 0 0 ; 0 1 0 0]; % measurement matrix
PK = dare(F', H', Q, R);

% Calculate closed loop Kalman filter eigenvalues
KK = PK * inv(eye(4) + H' * inv(R) * H * PK) * H' * inv(R);
abs(eig(F*(eye(4)-KK*H)))

PH = dare(F', eye(4), Q, (H' * R^(-1) * H - theta * Sbar)^(-1), zeros
(4,4), eye(4))
% Calculate closed loop H-infinity filter eigenvalues
temp = eye(4) - theta * Sbar * PH + H' * inv(R) * H * PH;
if cond(temp) > 1e10
    disp('ill conditioned H-infinity estimation problem');
    return;
end
KH = PH * inv(temp) * H' * inv(R);
abs(eig(F*(eye(4)-KH*H)))
```

```
function HybridNewton(theta)

% Optimal State Estimation Solution Manual, by Dan Simon
% Problem 12.8

% Kalman, H-infinity, and hybrid filter computation for a
% two-state Newtonian system.

T = 1; % step size
a = 1; % acceleration noise
R = 1; % measurement noise

F = [1 T; 0 1];
H = [1 0];
Q = a^2 * [T^4/2 T^3/2; T^3/2 T^2];

P = dare(F', H', Q, R);
KK = P * H' * inv(H * P * H' + R);

P = dare(F', eye(2), Q, inv(H'*inv(R)*H+theta*eye(2)));
lambda = min(real(eig(inv(P) - theta*eye(2) + H'*inv(R)*H)));
if lambda <= 0
    disp(['theta is too big - min(lambda) = ', num2str(lambda)]);
    return;
end
KH = P * inv(eye(2) - theta * P + H' * inv(R) * H * P) * H' * inv(R);

lambda = [];
for d = 0 : 0.01 : 1
    K = d * KK + (1 - d) * KH;
    lambda = [lambda max(abs(eig(F * (eye(2) - K * H))))];
end

close all;
figure;
set(gca,'FontSize',12); set(gcf,'Color','White');
plot(0 : 0.01 : 1, lambda, 'LineWidth', 2);
xlabel('d');
ylabel('max magnitude of estimator')
```

```
function HungScalar(F, Ml, alpha, N, eps, theta)

% Optimal State Estimation Solution Manual, by Dan Simon
% Problem 12.9

% Robust mixed Kalman/H-infinity filter for a scalar system.

Ptilde = roots([N^2, -alpha+alpha*F^2-N^2-alpha*F^2*N^2-eps*N^2,
alpha+alpha^2*Ml^2+alpha*eps]);
i = 1;
while i <= length(Ptilde)
    if ~isreal(Ptilde(i)) | (Ptilde(i) <= 0) | (alpha - N * Ptilde(i) * N
<= 0)
        Ptilde = [Ptilde(1:i-1) Ptilde(i+1:end)];
        i = i - 1;
    end
    i = i + 1;
end
if isempty(Ptilde)
    disp('Ptilde condition failed');
    return;
end
Ptilde = Ptilde(1);

R11 = 1 + alpha * M1^2;
Y = R11 + R11^2 * (alpha - Ptilde * N^2) / alpha / Ptilde / F^2 + eps;
a = theta^2 * (theta^2 - 1);
b = 1 - 2 * theta^2 + F^2 * theta^2 + theta^2 * (1 - theta^2) * Y;
c = 1 - F^2 + (2 * theta^2 - 1) * Y;
d = -Y;

P = roots([a b c d]);
i = 1;
while i <= length(P)
    if ~isreal(P(i)) | (P(i) <= 0) | (1 / theta^2 - P(i) <= 0)
        P = [P(1:i-1) P(i+1:end)];
        i = i - 1;
    end
    i = i + 1;
end
if isempty(P)
    disp('P condition failed');
    return;
end
P = P(1)

% Kalman gain calculation
KK = (sqrt(5) - 1) / 2;

% Robust filter gain calculation
```

```
T = 1 / (1 / P - theta^2);
Rtilde = T + 1;
R1 = F / (1 / Ptilde - N^2 / alpha);
F1 = F + R11 / R1;
KH = F1 * T / Rtilde;
Fhat = F1 - KH;
if (abs(Fhat) >= 1)
    disp('unstable estimator');
    return;
end

x = 0;
xhatK = 0;
xhatH = xhatK;
ErrK = [];
ErrH = ErrK;
kf = 1000;
for i = 1 : kf
    % System measurement
    y = x + randn;
    % Kalman filter
    xhatK = F * (1 - KK) * xhatK + F * KK * y;
    % Robust filter
    xhatH = Fhat * xhatH + KH * y;
    % System simulation
    x = (F + M1 * N) * x + randn;
    % Save data for plotting
    ErrK = [ErrK x-xhatK];
    ErrH = [ErrH x-xhatH];
end

disp(['std dev of est error (Kalman, robust) = ', num2str(std(ErrK)), ',↙
', num2str(std(ErrH))]);
```

```
function HinfConstr1(H1, G1)

% Optimal State Estimation Solution Manual, by Dan Simon
% Problem 12.10

% Constrained H-infinity state estimation for a two-state system.

if ~exist('H1', 'var')
    H1 = 1;
end
if ~exist('G1', 'var')
    G1 = 0.1;
end

F = [1 1; 0 1];
%F = [1 1/2; 1/2 0];

H = [H1 0];
G = [G1 0];

D = [1 1];
Q = [1 0; 0 1];

P = [2 0; 0 1];
for k = 1 : 2000
    Sigma = inv(P * H' * H - P * G' * G + eye(2)) * P;
    temp = (eye(2) - D' * D) * F;
    P = temp * Sigma * temp' + Q;
    if k == 1000
        P
    end
end
P
eig(P)
1-G*P*G'
```

```
function SquareRoot

% Optimal State Estimation Solution Manual, by Dan Simon
% Problem 13.15

% Moving average filter and EKF for nonlinear scalar system.

kf = 100;
R = 1;
x = 1;
xhat1 = 2; % initial simple filter estimate
xhat2 = xhat1; % initial Kalman filter estimate
P = 1; % initial Kalman filter variance

% initialize arrays
xhat1Arr = [];
xhat2Arr = [];

for k = 1 : kf
    % Measurement
    y = sqrt(x) * (1 + randn(R));
    % Simple filter
    xhat1 = ((k - 1) * xhat1 + y^2) / k;
    % Kalman filter
    yhat = sqrt(xhat2);
    H = 1 / 2 / yhat;
    M = yhat;
    K = P * H' * inv(H * P * H' + M * R * M');
    xhat2 = xhat2 + K * (y - yhat);
    P = (1 - K * H) * P;
    % Save estimates in arrays
    xhat1Arr = [xhat1Arr xhat1];
    xhat2Arr = [xhat2Arr xhat2];
end

close all;
figure;
k = 1 : kf;
set(gca,'FontSize',12); set(gcf,'Color','White');
plot(k, xhat1Arr, 'b-', k, xhat2Arr, 'r:');
xlabel('time step'); ylabel('estimate');
legend('Simple filter', 'Kalman filter');
```

```
function Orbit

% Optimal State Estimation Solution Manual, by Dan Simon
% Problem 13.16

% Linearized and extended Kalman filters for orbiting satellite
% state estimation.

dt = 60; % simulation step size
tf = 180 * 60; % simulation length

G = 6.6742e-11; % gravitational constant (m^3/kg/s^2)
M = 5.98e24; % earth mass (kg)
Radius = 6.37e6; % earth radius (m)

% Note if Q = 0, PL remains 0 and the KF does not respond to measurements
Qc = 1e-6; % continuous time process noise variance for radius accel.
Q = diag([0 Qc*dt 0 0]); % discretized process noise covariance
R = diag([10000 0.01]); % measurement noise covariance
H = [1 0 0 0 ; 0 0 1 0]; % measurement matrix

x = [Radius+2e5 ; 0 ; 0 ; 0]; % initial state
x(4) = sqrt(M * G / x(1)^3);
x(4) = 1.1 * x(4);
x0 = x; % nominal state
dxhatL = zeros(size(x)); % initial estimate for linearized Kalman filter
xhatL = x0 + dxhatL;
PL = diag([0 0 0 0]); % initial est. error covariance for linearized KF
xhatE = x; % initial estimate for extended Kalman filter
PE = PL; % initial estimation error covariance for extended KF

% initialize arrays
xArr = x;
xhatLArr = xhatL;
xhatEArr = xhatE;

for t = dt : dt : tf+dt/10
    % system simulation
    xdot(1,1) = x(2);
    xdot(2,1) = x(1) * x(4)^2 - M * G / x(1) / x(1);
    xdot(3,1) = x(4);
    xdot(4,1) = -2 * x(4) * x(2) / x(1);
    xdot = xdot + sqrt(Q) * randn(4,1);
    x = x + xdot * dt;
    x(3) = mod(x(3), 2*pi);
    y = H * x + sqrt(R) * randn(2,1);
    % linearized Kalman filter
    FL = [0 1 0 0 ; 3*x0(4)^2 0 0 2*x0(1)*x0(4) ; 0 0 0 1 ; 0 -2*x0(4)/x0(
(1) 0 0];
    dxhatLdot = FL * dxhatL;
```

```
    dxhatL = dxhatL + dxhatLdot * dt;
    PLdot = FL * PL + PL * FL' + Q;
    PL = PL + PLdot * dt;
    KL = PL * H' * inv(H * PL * H' + R);
    dxhatL = dxhatL + KL * (y - H * (x0 + dxhatL));
    x0(3) = x0(4) * t;
    xhatL = x0 + dxhatL;
    xhatL(3) = mod(xhatL(3), 2*pi);
    PL = PL - KL * H * PL;
    % extended Kalman filter
    FE = [0 1 0 0 ; xhatE(4)^2+2*G*M/xhatE(1)^3 0 0 2*xhatE(1)*xhatE(4) ;
0 0 0 1 ;
        2*xhatE(4)*xhatE(2)/xhatE(1)^2 -2*xhatE(4)/xhatE(1) 0 -2*xhatE(2)
/xhatE(1)];
    xhatEdot(1) = xhatE(2);
    xhatEdot(2) = xhatE(1) * xhatE(4)^2 - M * G / xhatE(1) / xhatE(1);
    xhatEdot(3) = xhatE(4);
    xhatEdot(4) = -2 * xhatE(4) * xhatE(2) / xhatE(1);
    xhatE = xhatE + xhatEdot' * dt;
    PEdot = FE * PE + PE * FE' + Q;
    PE = PE + PEdot * dt;
    KE = PE * H' * inv(H * PE * H' + R);
    xhatE = xhatE + KE * (y - H * xhatE);
    xhatE(3) = mod(xhatE(3), 2*pi);
    PE = PE - KE * H * PE;
    % save data in arrays
    xArr = [xArr x];
    xhatLArr = [xhatLArr xhatL];
    xhatEArr = [xhatEArr xhatE];
end

close all;
t = 0 : dt : tf;
t = t / 60;

% Plot true states
figure;
set(gcf,'Color','White');
subplot(2,2,1); plot(t, xArr(1,:), 'k-', t, xhatLArr(1,:), 'r:');
set(gca,'FontSize',12); ylabel('range (meters)');
subplot(2,2,2); plot(t, xArr(2,:), 'k-', t, xhatLArr(2,:), 'r:');
set(gca,'FontSize',12); ylabel('range rate (meters/s)');
subplot(2,2,3); plot(t, xArr(3,:), 'k-', t, xhatLArr(3,:), 'r:');
set(gca,'FontSize',12); xlabel('minutes'); ylabel('angle (rad)');
subplot(2,2,4); plot(t, xArr(4,:), 'k-', t, xhatLArr(4,:), 'r:');
set(gca,'FontSize',12); xlabel('minutes'); ylabel('angle rate (rad/s)');

% Plot EKF radius estimation error
figure;
plot(t, xArr(1,:)-xhatEArr(1,:), 'b-');
```

```
set(gcf,'Color','White'); set(gca,'FontSize',12);
xlabel('minutes'); ylabel('meters');

% Plot linearized Kalman filter radius estimation error
figure;
plot(t, xArr(1,:)-xhatLArr(1,:), 'b-');
set(gcf,'Color','White'); set(gca,'FontSize',12);
xlabel('minutes'); ylabel('meters');
```

```
function Motor(ControlNoise)

% Optimal State Estimation Solution Manual, by Dan Simon
% Problem 13.17

% Hybrid extended Kalman filter simulation for two-phase step motor.
% Estimate the stator currents, and the rotor position and velocity, on
% the basis of noisy measurements of the stator currents.

% Initialize the random number generator
randn('state', 1);

T = 0.1; % measurement period

Ra = 1.9; % Winding resistance
L = 0.003; % Winding inductance
lambda = 0.1; % Motor constant
J = 0.00018; % Moment of inertia
B = 0.001; % Coefficient of viscous friction

if ~exist('ControlNoise', 'var')
    ControlNoise = 0.01; % std dev of uncertainty in control inputs
end

AccelNoise = 0.5; % std dev of shaft acceleration noise
MeasNoise = 0.1; % standard deviation of measurement noise

R = [MeasNoise^2 0; 0 MeasNoise^2]; % Measurement noise covariance
xdotNoise = [ControlNoise/L ; ControlNoise/L ; 0.5 ; 0];
Q = xdotNoise * xdotNoise'; % Process noise covariance
P = 1*eye(4); % Initial state estimation covariance

dt = 0.0005; % Integration step size
tf = 3; % Simulation length

x = [0; 0; 0; 0]; % Initial state
xhat = x; % State estimate
w = 2 * pi; % Control input frequency

% Initialize arrays for plotting at the end of the program
xArray = [];
xhatArray = [];
trPArray = [];
tArray = [];

% Begin simulation loop
for t = 0 : T : tf
    xArray = [xArray x];
    xhatArray = [xhatArray xhat];
    trPArray = [trPArray trace(P)];
```

```
    tArray = [tArray t];
    % Nonlinear simulation
    for tau = dt : dt : T+dt/10
        ua0 = sin(w*t);
        ub0 = cos(w*t);
        s4 = sin(x(4));
        c4 = cos(x(4));
        xdot = [-Ra/L*x(1) + x(3)*lambda/L*s4 + ua0/L;
            -Ra/L*x(2) - x(3)*lambda/L*c4 + ub0/L;
            -3/2*lambda/J*x(1)*s4 + 3/2*lambda/J*x(2)*c4 - B/J*x(3);
            x(3)];
        xdot = xdot + xdotNoise .* randn(4,1);
        x = x + xdot * dt;
        x(4) = mod(x(4), 2*pi);
    end
    H = [1 0 0 0; 0 1 0 0];
    z = H * x + [MeasNoise*randn; MeasNoise*randn];
    % Kalman filter time update equations
    for tau = dt : dt : T+dt/10
        s4 = sin(xhat(4));
        c4 = cos(xhat(4));
        F = [-Ra/L 0 lambda/L*s4 xhat(3)*lambda/L*c4;
            0 -Ra/L -lambda/L*c4 xhat(3)*lambda/L*s4;
            -3/2*lambda/J*s4 3/2*lambda/J*c4 -B/J -3/2*lambda/J*(xhat(1)*
*c4+xhat(2)*s4);
            0 0 1 0];
        xhatdot = [-Ra/L*xhat(1) + xhat(3)*lambda/L*sin(xhat(4)) + ua0/L;
            -Ra/L*xhat(2) - xhat(3)*lambda/L*cos(xhat(4)) + ub0/L;
            -3/2*lambda/J*xhat(1)*sin(xhat(4)) + 3/2*lambda/J*xhat(2)*cos*
(xhat(4)) - B/J*xhat(3);
            xhat(3)];
        xhat = xhat + xhatdot * dt;
        Pdot = F * P + P * F' + Q;
        P = P + Pdot * dt;
    end
    % Kalman filter measurement update equations
    K = P * H' * inv(H * P * H' + R);
    xhat = xhat + K * (z - H * xhat);
    xhat(4) = mod(xhat(4), 2*pi);
    P = (eye(4) - K * H) * P * (eye(4) - K * H)' + K * R * K';
end

% Plot data.
close all;
figure;
plot(tArray, xArray(1,:), tArray,xhatArray(1,:),'r:')
set(gca,'FontSize',12); set(gcf,'Color','White');
xlabel('Time (Seconds)'); ylabel('Winding A Current (Amps)');
legend('Truc', 'Estimated');
```

```
figure;
plot(tArray, xArray(2,:), tArray,xhatArray(2,:),'r:')
set(gca,'FontSize',12); set(gcf,'Color','White');
xlabel('Time (Seconds)'); ylabel('Winding B Current (Amps)');
legend('True', 'Estimated');

figure;
plot(tArray, xArray(3,:), tArray,xhatArray(3,:),'r:')
set(gca,'FontSize',12); set(gcf,'Color','White');
xlabel('Time (Seconds)'); ylabel('Rotor Speed (Radians / Sec)');
legend('True', 'Estimated');

figure;
plot(tArray, xArray(4,:), tArray,xhatArray(4,:),'r:')
set(gca,'FontSize',12); set(gcf,'Color','White');
xlabel('Time (Seconds)'); ylabel('Rotor Position (Radians)');
legend('True', 'Estimated');

% Compute the std dev of the estimation errors
N = size(xArray, 2);
N2 = round(N / 2);
xArray = xArray(:,N2:N);
xhatArray = xhatArray(:,N2:N);
iaEstErr = sqrt(norm(xArray(1,:)-xhatArray(1,:))^2 / size(xArray,2));
ibEstErr = sqrt(norm(xArray(2,:)-xhatArray(2,:))^2 / size(xArray,2));
wEstErr = sqrt(norm(xArray(3,:)-xhatArray(3,:))^2 / size(xArray,2));
thetaEstErr = sqrt(norm(xArray(4,:)-xhatArray(4,:))^2 / size(xArray,2));
disp(['Std Dev of Estimation Errors = ',num2str(iaEstErr),', ',num2str↙
(ibEstErr),', ',num2str(wEstErr),', ',num2str(thetaEstErr)]);

% Display the P version of the estimation error standard deviations
disp(['Sqrt(P) = ',num2str(sqrt(P(1,1))),', ',num2str(sqrt(P(2,2))),', ',↙
num2str(sqrt(P(3,3))),', ',num2str(sqrt(P(4,4)))]);
```

```
function [Est1Err, Est2Err, EstiErr] = Hen1(PlotFlag)

% Optimal State Estimation Solution Manual, by Dan Simon
% Problem 13.18

% Discrete time 1st order and 2nd order EKF, and iterated EKF.
% 2nd order EKF algorithm is based on Henriksen's 1982 paper.

% INPUTS:
%   PlotFlag = flag saying whether or not to plot results
% OUTPUTS:
%   Est1Err = 1st order EKF estimation error (RMS)
%   Est2Err = 2nd order EKF estimation error (RMS)
%   EstiErr = iterated EKF estimation error (RMS)

if ~exist('PlotFlag', 'var')
    PlotFlag = true;
end

Q = 0.1; % process noise covariance
R = 1; % measurement noise variance

x = 1; % true state

P = 1; % initial estimation error variance
P2 = P;
Pi = P;

kf = 5; % simulation length

N = 2; % number of iterations in iterated EKF

xhat = 2; % first order EKF estimate
xhat2 = xhat; % second order EKF estimate
xhati = xhat; % iterated EKF estimate

randn('state', sum(100*clock)); % random number generator seed

xArray = x;
xhatArray = xhat;
xhat2Array = xhat2;
xhatiArray = xhat2;
Parray = P;
P2array = P2;
Piarray = Pi;

for k = 1 : kf

    % Simulate the system.
    x = x^2 + sqrt(Q) * randn;
```

```
    % Simulate the noisy measurement.
    y = x^2 + sqrt(R) * randn;

    % First order Kalman filter.
    F = 2 * xhat;
    xhat = xhat^2;
    P = F * P * F' + Q;
    H = 2 * xhat;
    K = P * H' * inv(H * P * H' + R);
    yhat = xhat^2;
    xhat = xhat + K * (y - yhat);
    P = (1 - K * H) * P;

    % Second order Kalman filter.
    % Continuous-time part of the 2nd order Kalman filter (time update).
    F2 = 2 * xhat2;
    xhat2 = xhat2^2 + (1/2) * 2 * P2;
    P2 = F2 * P2 * F2' + Q;
    H2 = 2 * xhat2;
    K2 = P2 * H2' * inv(H2 * P2 * H2' + R);
    yhat2 = xhat2^2 + (1/2) * 2 * P2;
    xhat2 = xhat2 + K2 * (y - yhat2);
    P2 = (1 - K2 * H2) * P2;

    % Iterated Kalman filter.
    % Continuous-time part of the iterated Kalman filter (time update).
    Fi = 2 * xhati;
    xhati = xhati^2;
    Pi = Fi * Pi * Fi' + Q;
    xhatminus = xhati;
    Pminus = Pi;
    for i = 1 : N
        Hi = 2 * xhati;
        Ki = Pminus * Hi' * inv(Hi * Pminus * Hi' + R);
        yhati = xhati^2;
        xhati = xhati + Ki * (y - yhati - Hi * (xhatminus - xhati));
        Pi = (1 - Ki * Hi) * Pminus;
    end

    % Save data for plotting.
    xArray = [xArray x];
    xhatArray = [xhatArray xhat];
    xhat2Array = [xhat2Array xhat2];
    xhatiArray = [xhatiArray xhati];
    Parray = [Parray diag(P)];
    P2array = [P2array diag(P2)];
    Piarray = [Piarray diag(Pi)];
end

Est1Err = sqrt((norm(xArray - xhatArray))^2/length(xArray));
```

```
Est2Err = sqrt((norm(xArray - xhat2Array))^2/length(xArray));
EstiErr = sqrt((norm(xArray - xhatiArray))^2/length(xArray));

if ~PlotFlag
    return;
end

close all;
k = 0 : kf;

figure; hold;
plot(k, xArray, 'b', k, xhatArray, 'r:', k, xhat2Array, 'k--', k,
xhatiArray, 'm-.');
set(gca,'FontSize',12); set(gcf,'Color','White');
xlabel('Seconds');
legend('True state', 'First order estimate', 'Second order estimate',
'Iterated estimate');

disp(['1st Order EKF RMS estimation error = ', num2str(Est1Err)]);
disp(['2nd Order EKF RMS estimation error = ', num2str(Est2Err)]);
disp(['Itereated EKF RMS estimation error = ', num2str(EstiErr)]);
```

```
function pdfApprox

% Optimal State Estimation Solution Manual, by Dan Simon
% Problem 13.19

% Approximate a pdf that is uniform on (-1,+1) with a sum of Gaussians.
% The Gaussian pdfs have means that are equally spaced on (-1,+1), and
% they have identical variances. Find the variance the minimizes the
% RMS error between the uniform pdf and the sum of Gaussian pdfs.

global muGauss NumPts dx

dx = 0.01;
NumPts = 2 / dx + 1;
MArray = [3 5 10]; % number of Gaussians used to estimation pdf
close all;
figure;
for k = 1 : length(MArray)
    M = MArray(k);
    muGauss(1) = -1 + 2 / (M + 1);
    for i = 2 : M
        muGauss(i) = muGauss(i-1) + 2 / (M + 1);
    end

    % Golden search code - copied from
    % http://www.cse.ucsc.edu/~hongwang/Codes/Golden_search
    a=0.01;
    b=2;
    tol=1.0e-10;
    n=0;
    g=(sqrt(5)-1)/2;
    r1=a+(b-a)*(1-g);
    f1=f(r1, M);
    r2=a+(b-a)*g;
    f2=f(r2, M);
    while (b-a) > tol,
        n=n+1;
        if f1 < f2,
            b=r2;
            r2=r1;
            f2=f1;
            r1=a+(b-a)*(1-g);
            f1=f(r1, M);
        else
            a=r1;
            r1=r2;
            f1=f2;
            r2=a+(b-a)*g;
            f2=f(r2, M);
        end
```

```
    end
    x0=(a+b)/2;
    % Plot the final solution and compute the RMS error.
    subplot(3,1,k);
    ErrRMS = f(x0, M, true)
end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function ErrRMS = f(sigma, M, PlotFlag)
global muGauss NumPts dx
ErrRMS = 0;
j = 1;
for x = -1 : dx : 1
    Approx(j) = 0;
    for i = 1 : M
        Approx(j) = Approx(j) + (1/M) * exp(-(x-muGauss(i))^2 / 2 /↵
sigma^2);
    end
    ErrRMS = ErrRMS + (Approx(j) - 1/2)^2;
    j = j + 1;
end
ErrRMS = sqrt(ErrRMS / NumPts);
if ~exist('PlotFlag', 'var')
    PlotFlag = false;
end
if PlotFlag
    set(gca,'FontSize',12); set(gcf,'Color','White');
    x = -1 : dx : 1;
    plot(x, (1/2)*ones(size(x)), 'k-', 'LineWidth', 2);
    hold on;
    plot(x, Approx, 'r:', 'LineWidth', 2);
    text(-0.5, 0.8, ['M = ', num2str(M), ' : sigma = ', num2str(sigma), ',↵
RMS Error = ', num2str(ErrRMS)]);
    v = axis;
    axis([v(1) v(2) 0 v(4)]);
end
end
```

```
function GaussFilter

% Optimal State Estimation Solution Manual, by Dan Simon
% Problem 13.20

% EKF and Gaussian sum filter for scalar system.

kf = 20; % simulation length

Q = 0;
R = 0.01;
M = 2; % number of terms in Gaussian sum filter

x = -1/2; % initial state
xhat = 0; % initial estimate

for i = 1 : M
    a(i) = 1 / M;
    xhat(i) = -1/3 + (2/3) * (i-1); % initial Gaussian sum filter est.
    Pplus(i) = 0.43^2;
end

% Initialize arrays
xArr = x;
xhatArr = xhat';

for k = 1 : kf
    % System simulation
    x = x + sqrt(Q) * randn;
    y = x^2 + sqrt(R) * randn;
    % Kalman filter simulation
    for i = 1 : M
        xhat(i) = xhat(i);
        F(i) = 1;
        Pminus(i) = F(i) * Pplus(i) * F(i)' + Q;
        H(i) = 2 * xhat(i);
        S(i) = H(i) * Pminus(i) * H(i)' + R;
        K(i) = Pminus(i) * H(i)' * inv(S(i));
        Pplus(i) = (1 - K(i) * H(i)) * Pminus(i);
        r(i) = y - xhat(i)^2;
        xhat(i) = xhat(i) + K(i) * (y - xhat(i)^2);
        beta(i) = exp(-r(i)' * inv(S(i)) * r(i) / 2) / sqrt(2 * pi)^length
(x) / sqrt(det(S(i)));
    end
    a = a .* beta / sum(a.*beta);
    % Save data for plotting
    xArr = [xArr x];
    xhatArr = [xhatArr xhat'];
end
```

```
% Plot the results
close all;
k = 0 : kf;

figure; set(gca,'FontSize',12); set(gcf,'Color','White');
subplot(2,1,1);
plot(k, xArr, 'r:', k, xhatArr(1,:), 'b-', k, xhatArr(2,:), 'k.-');
xlabel('time'); ylabel('state');
legend('true state', 'estimated state 1', 'estimated state 2');

set(gca,'FontSize',12); set(gcf,'Color','White');
for i = 1 : 2
    xmin = xhat(i) - 3 * sqrt(Pplus(i));
    xmax = xhat(i) + 3 * sqrt(Pplus(i));
    dx = (xmax - xmin) / 100;
    j = 1;
    for x = xmin : dx : xmax
        pdf(j) = exp(-(x - xhat(i))^2 / 2 / Pplus(i));
        j = j + 1;
    end
    x = xmin : dx : xmax;
    subplot(2,1,2); hold on; ylabel('pdfs');
    plot(x, pdf);
end
axis([-0.7 0.7 0 1]);
```

```
function TrackEKF

% Optimal State Estimation Solution Manual, by Dan Simon
% Problem 13.21

% EKF-based ground vehicle navigation

% Define the north and east coordinates of the two tracking stations.
N1 = 20;
E1 = 0;
N2 = 0;
E2 = 20;

dt = 0.1; % simulation step size
tf = 60; % simulation length

F = [1 0 dt 0 ; 0 1 0 dt ; 0 0 1 0 ; 0 0 0 1];
Q = diag([0 0 4 4]);
R = diag([1 1]);

x = [0; 0; 50; 50]; % initial state
xhatplus = x; % initial state estimate
Pplus = diag([0 0 0 0]); % initial estimation error covariance

% Initialize arrays
xArr = x;
xhatArr = xhatplus;

for t = dt : dt : tf
    % System simulation
    x = F * x + sqrt(Q) * randn(4,1);
    y(1,1) = sqrt((x(1) - N1)^2 + (x(2) - E1)^2);
    y(2,1) = sqrt((x(1) - N2)^2 + (x(2) - E2)^2);
    % EKF time update
    Pminus = F * Pplus * F' + Q;
    xhatminus = F * xhatplus;
    % EKF measurement update
    H = zeros(2,4);
    nhat = xhatminus(1);
    ehat = xhatminus(2);
    temp = sqrt((nhat - N1)^2 + (ehat - E1)^2);
    H(1,1) = (nhat - N1) / temp;
    H(1,2) = (ehat - E1) / temp;
    temp = sqrt((nhat - N2)^2 + (ehat - E2)^2);
    H(2,1) = (nhat - N2) / temp;
    H(2,2) = (ehat - E2) / temp;
    K = Pminus * H' * inv(H * Pminus * H' + R);
    yhat(1,1) = sqrt((nhat - N1)^2 + (ehat - E1)^2);
    yhat(2,1) = sqrt((nhat - N2)^2 + (ehat - E2)^2);
    xhatplus = xhatminus + K * (y - yhat);
```

```
    Pplus = (eye(4) - K * H) * Pminus;
    % Save data for plotting
    xArr = [xArr x];
    xhatArr = [xhatArr xhatplus];
end

% Plot results
close all;
k = 0 : dt : tf;

figure; set(gca,'FontSize',12); set(gcf,'Color','White');
subplot(2,2,1);
plot(k, xArr(1,:)-xhatArr(1,:), 'b-');
title('North Position Estimation Error');

subplot(2,2,2);
plot(k, xArr(2,:)-xhatArr(2,:), 'b-');
title('East Position Estimation Error');

subplot(2,2,3);
plot(k, xArr(3,:)-xhatArr(3,:), 'b-'); xlabel('time');
title('North Velocity Estimation Error');

subplot(2,2,4);
plot(k, xArr(4,:)-xhatArr(4,:), 'b-'); xlabel('time');
title('East Velocity Estimation Error');

% Compute experimental standard deviations of estimation errors.
EstStd = std(xArr' - xhatArr');
disp(['Experimental std dev of est err = ', num2str(EstStd)]);
```

```
function SysId

% Optimal State Estimation Solution Manual, by Dan Simon
% Problem 13.22

% EKF for system identification.

phi = 0.9; % true value of phi
Q = [1 0; 0 0]; % variance of process noise

kf = 100; % number of time steps
x = 1; % initial value of x
P = eye(2); % initial Kalman filter covariance
xhat = [0 0]'; % initial Kalman filter state estimate
xhatArr = []; % array for saving state estimate

for k = 1 : kf
    % system equations
    x = phi * x + sqrt(Q(1,1)) * randn;
    y = x;
    % EKF time update
    F = [xhat(2) xhat(1) ; 0 1];
    P = F * P * F' + Q;
    xhat(1) = xhat(2) * xhat(1);
    xhat(2) = xhat(2);
    % EKF measurement update
    H = [1 0];
    K = P * H' * inv(H * P * H');
    xhat = xhat + K * (y - xhat(1));
    P = (eye(2) - K * H) * P;
    % Save data for arrays
    xhatArr = [xhatArr xhat];
end

close all;
k = 1 : kf;
figure;
set(gca,'FontSize',12); set(gcf,'Color','White');
plot(k, phi*ones(size(k)), 'k-', k, xhatArr(2,:), 'r:');
legend('true', 'estimated');
xlabel('time step'); ylabel('phi');
```

```
function Parameter

% Optimal State Estimation Solution Manual, by Dan Simon
% Problem 13.23

% Extended Kalman filter for parameter estimation.
% Estimate the natural frequency of a second order system.

tf = 100; % simulation length
dt = 0.01; % simulation step size
wn = 2; % natural frequency
zeta = 0.1; % damping ratio
b = -2 * zeta * wn;
Q2 = 1;

Q = [1000 0; 0 Q2]; % covariance of process noise
R = [10 0; 0 10]; % covariance of measurement noise
H = [1 0 0; 0 1 0]; % measurement matrix
P = [0 0 0; 0 0 0; 0 0 20]; % covariance of estimation error

close all;
figure;
set(gca,'FontSize',12); set(gcf,'Color','White');

sigma2 = [0 1 100];
for i = 1 : length(sigma2)
    Q2 = sigma2(i); % artificial noise used for parameter estimation
    x = [0; 0; -wn*wn]; % initial state
    xhat = 2 * x; % initial state estimate

    % Initialize arrays for later plotting
    xArray = x;
    xhatArray = xhat;
    P3Array = P(3,3);

    dtPlot = tf / 100; % how often to plot output data
    tPlot = 0;

    for t = dt : dt : tf+dt
        % Simulate the system.
        w = sqrt(Q(1,1)) * randn;
        xdot = [x(2); x(3)*x(1) + b*x(2) - x(3)*w; 0];
        x = x + xdot * dt;
        z = H * x + sqrt(R) * [randn; randn];
        % Simulate the Kalman filter.
        F = [0 1 0; xhat(3) b xhat(1); 0 0 0];
        L = [0 0; -xhat(3) 0; 0 1];
        Pdot = F * P + P * F' + L * Q * L' - P * H' * inv(R) * H * P;
        P = P + Pdot * dt;
        K = P * H' * inv(R);
```

```
        xhatdot = [xhat(2); xhat(3)*xhat(1) + b*xhat(2); 0];
        xhatdot = xhatdot + K * (z - H * xhat);
        xhat = xhat + xhatdot * dt;
        if (t >= tPlot + dtPlot - 100*eps)
            % Save data for plotting.
            xArray = [xArray x];
            xhatArray = [xhatArray xhat];
            P3Array = [P3Array P(3,3)];
            tPlot = t;
        end
    end
    % Plot results
    t = 0 : dtPlot : tf;
    subplot(3,1,i);
    plot(t, xArray(3,:) - xhatArray(3,:), 'LineWidth', 2);
    set(gca,'FontSize',12); set(gcf,'Color','White');
    if (i == 3)
        xlabel('Seconds');
    elseif (i == 2)
        ylabel('w_n^2 Estimation Error');
    end
    axis([0 tf -1 5]);
    legend(['\sigma_p^2 = ', num2str(sigma2(i))]);
end
```

```
function TrackUKF(tf);

% Optimal State Estimation Solution Manual, by Dan Simon
% Problem 14.14

% UKF simulation of land vehicle tracking system.
% INPUTS:
%    tf = simulation length

if ~exist('tf', 'var')
    tf = 60;
end
dt = 0.1; % simulation step size

% Define the north and east coordinates of the two tracking stations.
N1 = 20;
E1 = 0;
N2 = 0;
E2 = 20;

F = [1 0 dt 0 ; 0 1 0 dt ; 0 0 1 0 ; 0 0 0 1];
Q = diag([0 0 4 4]);
R = diag([1 1]);

x = [0; 0; 50; 50];
xhatplus = x;
Pplus = 0.01 * eye(4); % must be positive definite for UKF

% Initialize arrays
xArr = x;
xhatArr = xhatplus;
PArr(:,:,1) = Pplus;

N = length(x);
k = 1;
for t = dt : dt : tf
    % System simulation
    x = F * x + sqrt(Q) * randn(4,1);
    y(1,1) = sqrt((x(1) - N1)^2 + (x(2) - E1)^2);
    y(2,1) = sqrt((x(1) - N2)^2 + (x(2) - E2)^2);
    y = y + sqrt(R) * randn(2,1);
    % UKF time update
    [root,p] = chol(4*Pplus);
    for i = 1 : N
        sigma(:,i) = xhatplus + root(i,:)';
        sigma(:,i+N) = xhatplus - root(i,:)';
    end
    for i = 1 : 2*N
        sigma(:,i) = F * sigma(:,i);
    end
```

```
    xhatminus = zeros(N,1);
    for i = 1 : 2*N
        xhatminus = xhatminus + sigma(:,i) / 2 / N;
    end
    Pminus = zeros(N,N);
    for i = 1 : 2*N
        Pminus = Pminus + (sigma(:,i) - xhatminus) * (sigma(:,i) -
xhatminus)' / 2 / N;
    end
    Pminus = Pminus + Q;
    % UKF measurement update
    [root,p] = chol(N*Pminus);
    for i = 1 : N
        sigma(:,i) = xhatminus + root(i,:)';
        sigma(:,i+N) = xhatminus - root(i,:)';
    end
    for i = 1 : 2*N
        nhat = sigma(1,i);
        ehat = sigma(2,i);
        yukf(1,i) = sqrt((nhat - N1)^2 + (ehat - E1)^2);
        yukf(2,i) = sqrt((nhat - N2)^2 + (ehat - E2)^2);
    end
    yhat = 0;
    for i = 1 : 2*N
        yhat = yhat + yukf(:,i) / 2 / N;
    end
    Py = zeros(2,2);
    Pxy = zeros(N,2);
    for i = 1 : 2*N
        Py = Py + (yukf(:,i) - yhat) * (yukf(:,i) - yhat)' / 2 / N;
        Pxy = Pxy + (sigma(:,i) - xhatminus) * (yukf(:,i) - yhat)' / 2 /
N;
    end
    Py = Py + R;
    Kukf = Pxy * inv(Py);
    xhatplus = xhatminus + Kukf * (y - yhat);
    Pplus = Pminus - Kukf * Py * Kukf';
    % Save data for plotting
    xArr = [xArr x];
    xhatArr = [xhatArr xhatplus];
    k = k + 1;
    PArr(:,:,k) = Pplus;
end

% Plot results
close all;
k = 0 : dt : tf;

% Plot estimation errors
figure; set(gcf,'Color','White');
```

```
subplot(2,2,1); set(gca,'FontSize',12);
plot(k, xArr(1,:)-xhatArr(1,:), 'b-', 'LineWidth', 2);
title('North Position Estimation Error');

subplot(2,2,2); set(gca,'FontSize',12);
plot(k, xArr(2,:)-xhatArr(2,:), 'b-', 'LineWidth', 2);
title('East Position Estimation Error');

subplot(2,2,3); set(gca,'FontSize',12);
plot(k, xArr(3,:)-xhatArr(3,:), 'b-', 'LineWidth', 2); xlabel('time');
title('North Velocity Estimation Error');

subplot(2,2,4); set(gca,'FontSize',12);
plot(k, xArr(4,:)-xhatArr(4,:), 'b-', 'LineWidth', 2); xlabel('time');
title('East Velocity Estimation Error');

% Plot UKF variance estimates
figure; set(gcf, 'Color', 'White'); set(gca,'FontSize',12); hold on; box↙
on;
plot(k, sqrt(squeeze(PArr(1,1,:))), 'r:', 'LineWidth', 2);
plot(k, sqrt(squeeze(PArr(2,2,:))), 'b--', 'LineWidth', 2);
plot(k, sqrt(squeeze(PArr(3,3,:))), 'k-', 'LineWidth', 2);
plot(k, sqrt(squeeze(PArr(4,4,:))), 'm-.', 'LineWidth', 2);
legend('North Position', 'East Position', 'North Velocity', 'East↙
Velocity');
xlabel('time');
ylabel('UKF Std Dev Estimate');

% Compare experimental and theoretical standard deviations of estimation↙
errors.
EstStd = std(xArr' - xhatArr');
disp(['Experimental std dev of est err = ', num2str(EstStd)]);
```

```
function PendulumUKF

% Optimal State Estimation Solution Manual, by Dan Simon
% Problem 14.15

% Inverted pendulum simulation - UKF and EKF estimation

dt = 0.005; % integration step size
tf = 2; % simulation time
thetainit = 0.1; % initial pendulum angle (radians)
m = 0.2; % pendulum mass (kg)
M = 1.0; % cart mass (kg)
g = 9.81; % acceleration due to gravity (m/s^2)
B = 0.1; % coefficient of viscous friction on cart wheels
l = 1; % length of pendulum (meters)
r = 0.02; % radius of pendulum mass (meters)
J = m * r * r / 2; % moment of inertia of pendulum mass (cylinder)

theta = thetainit; % theta = angle pendulum makes with vertical
thetadot = 0;
thetadotdot = 0;
d = 0; % d = horizontal displacement of cart
ddot = 0;
ddotdot = 0;

theta_array = [theta];
thetadot_array = [thetadot];
thetadotdot_array = [thetadotdot];
d_array = [d];
ddot_array = [ddot];
ddotdot_array = [ddotdot];

% Create Jacobian matrix for EKF
temp = J * (M + m) + M * m * l^2;
F22 = -m^2 * l^2 * B / (M + m) / temp - B / (M + m);
F23 = -m^2 * l^2 * g / temp;
F42 = m * l * B / temp;
F43 = m * g * l * (M + m) / temp;
F = [0 1 0 0;
    0 F22 F23 0;
    0 0 0 1;
    0 F42 F43 0];
% Measurement matrix
H = [1 0 0 0];

% Continuous time process noise covariance
Q = diag([0 0.0004 0 0.04]);
% Discrete time process noise covariance
Q = Q * dt;
% Discrete time measurement noise covariance
```

```
R = diag([0.01]);
r = size(R,1); % number of measurements

x = [0; 0; thetainit; 0]; % initial state
xhatplus = x; % initial UKF estimate
xhatKplus = x; % initial EKF estimate
% Initial estimation error covariance - must be positive definite for UKF
Pplus = 0.00001 * eye(4);
PKplus = Pplus;

% Initialize arrays
xArr = x;
xhatArr = xhatplus;
xhatKArr = xhatKplus;

N = length(x);
for t = dt : dt : tf
    % System simulation
    theta_old = theta;
    thetadot_old = thetadot;
    thetadotdot_old = thetadotdot;
    d_old = d;
    ddot_old = ddot;
    ddotdot_old = ddotdot;
    sine = sin(theta);
    cosine = cos(theta);
    % Compute the new state values.
    u = 40 * theta_old;
    theta = theta + thetadot_old * dt;
    thetadot = thetadot + thetadotdot_old * dt + sqrt(Q(3,3)) * randn *↙
dt;
    thetadotdot = (m*g*l*sine - m*l*cosine*(u + ...
        m*l*thetadot_old*thetadot_old*sine - B*ddot_old) / (M + m)) / ...
        (J + m*l*l - m*m*l*l*cosine*cosine / (M + m)) + sqrt(Q(4,4)) *↙
randn * dt;
    d = d + ddot_old * dt;
    ddot = ddot + ddotdot_old * dt + sqrt(Q(1,1)) * randn * dt;
    ddotdot = (u - m*l*thetadotdot_old*cosine + ...
        m*l*thetadot_old*thetadot_old*sine - B*ddot_old) / (M + m) + sqrt↙
(Q(2,2)) * randn * dt;
    % Save the state values in arrays for later plotting.
    theta_array = [theta_array theta];
    thetadot_array = [thetadot_array thetadot];
    thetadotdot_array = [thetadotdot_array thetadotdot];
    d_array = [d_array d];
    ddot_array = [ddot_array ddot];
    ddotdot_array = [ddotdot_array ddotdot];
    % Measurement simulation
    y = d + sqrt(R) * randn;
    % UKF time update
```

```
    [root,p] = chol(N*Pplus);
    for i = 1 : N
        sigma(:,i) = xhatplus + root(i,:)';
        sigma(:,i+N) = xhatplus - root(i,:)';
    end
    for i = 1 : 2*N
        % Save the old a posteriori sigma points.
        theta_old = sigma(3,i);
        thetadot_old = sigma(4,i);
        d_old = sigma(1,i);
        ddot_old = sigma(2,i);
        sine = sin(theta_old);
        cosine = cos(theta_old);
        % Compute the new sigma points (a priori).
        u = 40 * theta_old;
        sigma(3,i) = sigma(3,i) + thetadot_old * dt;
        thetadotdot_old = m * g * l * theta_old * (M + m);
        thetadotdot_old = thetadotdot_old - m * l * (u - B * ddot_old);
        thetadotdot_old = thetadotdot_old / temp;
        sigma(4,i) = sigma(4,i) + thetadotdot_old * dt;
        sigma(1,i) = sigma(1,i) + ddot_old * dt;
        ddotdot_old = (u - m * l * thetadotdot_old - B * ddot_old) / (M +
m);
        sigma(2,i) = sigma(2,i) + ddotdot_old * dt;
    end
    xhatminus = zeros(N,1);
    for i = 1 : 2*N
        xhatminus = xhatminus + sigma(:,i) / 2 / N;
    end
    Pminus = zeros(N,N);
    for i = 1 : 2*N
        Pminus = Pminus + (sigma(:,i) - xhatminus) * (sigma(:,i) -
xhatminus)' / 2 / N;
    end
    Pminus = Pminus + Q;
    % UKF measurement update
    [root,p] = chol(N*Pminus);
    for i = 1 : N
        sigma(:,i) = xhatminus + root(i,:)';
        sigma(:,i+N) = xhatminus - root(i,:)';
    end
    for i = 1 : 2*N
        nhat = sigma(1,i);
        ehat = sigma(2,i);
        yukf(1,i) = sigma(1,i);
    end
    yhat = 0;
    for i = 1 : 2*N
        yhat = yhat + yukf(:,i) / 2 / N;
    end
```

```
    Py = zeros(r,r);
    Pxy = zeros(N,r);
    for i = 1 : 2*N
        Py = Py + (yukf(:,i) - yhat) * (yukf(:,i) - yhat)' / 2 / N;
        Pxy = Pxy + (sigma(:,i) - xhatminus) * (yukf(:,i) - yhat)' / 2 /↙
' N;
    end
    Py = Py + R;
    Kukf = Pxy * inv(Py);
    xhatplus = xhatminus + Kukf * (y - yhat);
    Pplus = Pminus - Kukf * Py * Kukf';
    % Save data for plotting
    xhatArr = [xhatArr xhatplus];
    % Extended Kalman filter simulation
    % Save the old a posteriori state estimate.
    theta_old = xhatKplus(3);
    thetadot_old = xhatKplus(4);
    d_old = xhatKplus(1);
    ddot_old = xhatKplus(2);
    sine = sin(theta_old);
    cosine = cos(theta_old);
    % EKF time update.
    % The EKF works only if "knows" u perfectly.
    u = 40 * theta_old;
    %u = 40 * theta;
    xhatK(3,1) = xhatKplus(3) + thetadot_old * dt;
    thetadotdot_old = m * g * l * theta_old * (M + m);
    thetadotdot_old = thetadotdot_old - m * l * (u - B * ddot_old);
    thetadotdot_old = thetadotdot_old / temp;
    xhatK(4,1) = xhatKplus(4) + thetadotdot_old * dt;
    xhatK(1,1) = xhatKplus(1) + ddot_old * dt;
    ddotdot_old = (u - m * l * thetadotdot_old - B * ddot_old) / (M + m);
    xhatK(2,1) = xhatKplus(2) + ddotdot_old * dt;
    PK = PKplus + (F * PKplus + PKplus * F') * dt + Q;
    % EKF measurement update.
    K = PK * H' * inv(H * PK * H' + R);
    xhatKplus = xhatK + K * (y - H * xhatK);
    PKplus = (eye(N) - K * H) * PK * (eye(N) - K * H)' + K * R * K';
    % Save date for plotting.
    xhatKArr = [xhatKArr xhatKplus];
end

% Plot results
close all;
t = 0 : dt : tf;
figure; set(gcf,'Color','White');

subplot(2,2,1); set(gca,'FontSize',12);
plot(t, 180/2/pi*theta_array, 'b-', 'LineWidth', 2); hold on;
plot(t, 180/2/pi*xhatArr(3,:), 'r:', 'LineWidth', 2);
```

```
%plot(t, 180/2/pi*xhatKArr(3,:), 'm--', 'LineWidth', 2);
ylabel('angle (deg)');

subplot(2,2,2); set(gca,'FontSize',12);
plot(t, d_array, 'b-', 'LineWidth', 2); hold on;
plot(t, xhatArr(1,:), 'r:', 'LineWidth', 2);
%plot(t, xhatKArr(1,:), 'm--', 'LineWidth', 2);
ylabel('cart pos (meters)');
legend('true', 'UKF estimate');

subplot(2,2,3); set(gca,'FontSize',12);
plot(t, thetadot_array, 'b-', 'LineWidth', 2); hold on;
plot(t, xhatArr(4,:), 'r:', 'LineWidth', 2);
%plot(t, xhatKArr(4,:), 'm--', 'LineWidth', 2);
ylabel('ang vel (deg/s)');
xlabel('time (s)');

subplot(2,2,4); set(gca,'FontSize',12);
plot(t, ddot_array, 'b-', 'LineWidth', 2); hold on;
plot(t, xhatArr(2,:), 'r:', 'LineWidth', 2);
%plot(t, xhatKArr(2,:), 'm--', 'LineWidth', 2);
ylabel('cart vel (meters/s)');
xlabel('time (s)');
```

```
function Hypersphere

% Optimal State Estimation Solution Manual, by Dan Simon
% Problem 15.11

% Plot the volume of a hypersphere as a function of dimension.

v(1) = 2;
v(2) = pi;
v(3) = 4*pi/3;
for i = 4 : 20
    v(i) = 2*pi*v(i-2)/i;
end
i = 1 : 20;
close all;
figure; set(gca,'FontSize',12); set(gcf,'Color','White');
plot(i, v);
xlabel('number of dimensions');
ylabel('volume of hypersphere');
```

```
function Kernel(BWScale, KernelType)

% Optimal State Estimation Solution Manual, by Dan Simon
% Problem 15.12, 15.13, 15.14

% Epanechnikov-based pdf approximation

% INPUTS:
%     BWScale is used to scale the bandwidth from its optimal value:
%                     Bandwidth h = BWScale * h^*
%     KernelType =  1 (Epanechnikov)
%                   2 (Gaussian)
%                   3 (Uniform)
%                   4 (Triangular)
%                   5 (Biweight)

if ~exist('BWScale', 'var')
    BWScale = 1;
end
if ~exist('KernelType', 'var')
    KernelType = 1;
end

close all;
figure;

subplot(2,1,1); hold on;
EpanechSub([1 2], BWScale, KernelType);
subplot(2,1,2); hold on;
EpanechSub([1 2 3], BWScale, KernelType);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function EpanechSub(x, BWScale, KernelType)

q = ones(size(x)); % probabilities
NReg = 100; % number of probability bins for pdf approximation
n = 1; % dimension of the state vector
vol = 2; % volume of unit hypersphere in d-dimensional space
N = length(x); % number of particles in the particle filter
S = cov(x);
A = chol(S)';
A = eye(length(x));
h = (8 * vol^(-1) * (n + 4) * (2 * sqrt(pi))^n)^(1 / (n + 4)) * N^(-1 / (n
+ 4)); % bandwidth of RPF
h = BWScale * h;
% Define the domain from which we will choose a posteriori particles for
% the regularized particle filter.
xreg(1) = min(x) - std(x);
xreg(NReg) = max(x) + std(x);
dx = (xreg(NReg) - xreg(1)) / (NReg - 1);
```

```
for i = 2 : NReg - 1
    xreg(i) = xreg(i-1) + dx;
end
% Create the pdf approximation that is required for the regularized
% particle filter.
for j = 1 : NReg
    for i = 1 : N
        qreg(i,j) = 0;
        normx = norm(inv(A) * (xreg(j) - x(i)));
        if KernelType == 2
            qreg(i,j) = qreg(i,j) + q(i) * exp(-normx^2 / h^2 / 2) / h^n /↙
det(A);
        elseif normx < h
            switch KernelType
                case 1
                    qreg(i,j) = qreg(i,j) + q(i) * (n + 2) * (1 - normx^2↙
/ h^2) / 2 / vol / h^n / det(A);
                case 3
                    qreg(i,j) = qreg(i,j) + q(i) / 2 / h^n / det(A);
                case 4
                    qreg(i,j) = qreg(i,j) + q(i) * (1 - normx / h) / h^n /↙
det(A);
                case 5
                    qreg(i,j) = qreg(i,j) + q(i) * (1 - normx^2 / h^2)^2 *↙
15 / 16 / h^n / det(A);
            end
        end
    end
end
% Plot
for i = 1 : N
    plot([x(i) x(i)], [0 q(i)], 'k-');
    plot(x(i), q(i), 'ko');
    plot(xreg, qreg(i,:), 'r:');
end
plot(xreg, sum(qreg,1), 'b--');
set(gca,'FontSize',12); set(gcf,'Color','White');
xlabel('state estimate'); ylabel('pdf');
return;
```

```
function [errRMSKalman, errRMSParticle] = PartScalar(N, Q)

% Optimal State Estimation Solution Manual, by Dan Simon
% Problem 15.15

% EKF and Particle filter scalar example,
% adapted from Gordon, Salmond, and Smith paper.
% INPUTS:
%    N = number of particles
%    Q = process noise variance
% OUTPUTS:
%    errRMSKalman = Kalman filter RMS estimation error
%    errRMSParticle = Particle filter RMS estimation error

if ~exist('N', 'var')
    N = 100;
end
if ~exist('Q', 'var')
    Q = 1;
end

x = 0.1; % initial state
R = 1; % measurement noise covariance
tf = 50; % simulation length

xhat = x;
P = 2;
xhatPart = x;

% Initialize the particle filter.
for i = 1 : N
    xpart(i) = x + sqrt(P) * randn;
end

xArr = [x];
yArr = [x^2 / 20 + sqrt(R) * randn];
xhatArr = [xhat];
xhatPartArr = [xhatPart];

close all;

for k = 1 : tf
    % System simulation
    x = 0.5 * x + 25 * x / (1 + x^2) + 8 * cos(1.2*(k-1)) + sqrt(Q) *↵
randn;
    y = x^2 / 20 + sqrt(R) * randn;
    % Extended Kalman filter
    F = 0.5 + 25 * (1 - xhat^2) / (1 + xhat^2)^2;
    P = F * P * F' + Q;
    H = xhat / 10;
```

```
    K = P * H' * (H * P * H' + R)^(-1);
    xhat = 0.5 * xhat + 25 * xhat / (1 + xhat^2) + 8 * cos(1.2*(k-1));
    xhat = xhat + K * (y - xhat^2 / 20);
    P = (1 - K * H) * P;
    % Particle filter
    for i = 1 : N
        xpartminus(i) = 0.5 * xpart(i) + 25 * xpart(i) / (1 + xpart(i)^2)...
+ 8 * cos(1.2*(k-1)) + sqrt(Q) * randn;
        ypart = xpartminus(i)^2 / 20;
        vhat = y - ypart;
        q(i) = (1 / sqrt(R) / sqrt(2*pi)) * exp(-vhat^2 / 2 / R);
    end
    % Normalize the likelihood of each a priori estimate.
    qsum = sum(q);
    if qsum < eps
        q = ones(size(q)) / N;
    else
        for i = 1 : N
            q(i) = q(i) / qsum;
        end
    end
    % Resample.
    for i = 1 : N
        u = rand; % uniform random number between 0 and 1
        qtempsum = 0;
        for j = 1 : N
            qtempsum = qtempsum + q(j);
            if qtempsum >= u
                xpart(i) = xpartminus(j);
                break;
            end
        end
    end
    % The particle filter estimate is the mean of the particles.
    xhatPart = mean(xpart);
    % Save data in arrays for later plotting
    xArr = [xArr x];
    yArr = [yArr y];
    xhatArr = [xhatArr xhat];
    xhatPartArr = [xhatPartArr xhatPart];
end
errRMSKalman = sqrt((norm(xArr - xhatArr))^2 / tf);
errRMSParticle = sqrt((norm(xArr - xhatPartArr))^2 / tf);
```

```
function PartScalarMonte

% Optimal State Estimation Solution Manual, by Dan Simon
% Problem 15.15

% This is a routine to generate Monte Carlo simulation results for the
% EKF and Particle filter scalar example that is
% adapted from the Gordon, Salmond, and Smith paper.

NMonte = 100; % number of Monte Carlo simulations

N = 100; % number of particles
Q = 0.1; % process noise variance
for i = 1 : NMonte
    [errKalman(i), errParticle(i)] = PartScalar(N, Q);
    fprintf('.');
end
fprintf('\n');
disp(['N = ', num2str(N), ', Q = ', num2str(Q), ', Ave RMS Est Err = ',↵
num2str(mean(errKalman)), ' (Kalman), ', num2str(mean(errParticle)), '↵
(Particle)']);

N = 100;
Q = 1;
for i = 1 : NMonte
    [errKalman(i), errParticle(i)] = PartScalar(N, Q);
    fprintf('.');
end
fprintf('\n');
disp(['N = ', num2str(N), ', Q = ', num2str(Q), ', Ave RMS Est Err = ',↵
num2str(mean(errKalman)), ' (Kalman), ', num2str(mean(errParticle)), '↵
(Particle)']);

N = 100;
Q = 10;
for i = 1 : NMonte
    [errKalman(i), errParticle(i)] = PartScalar(N, Q);
    fprintf('.');
end
fprintf('\n');
disp(['N = ', num2str(N), ', Q = ', num2str(Q), ', Ave RMS Est Err = ',↵
num2str(mean(errKalman)), ' (Kalman), ', num2str(mean(errParticle)), '↵
(Particle)']);

N = 100;
Q = 100;
for i = 1 : NMonte
    [errKalman(i), errParticle(i)] = PartScalar(N, Q);
    fprintf('.');
end
```

```
fprintf('\n');
disp(['N = ', num2str(N), ', Q = ', num2str(Q), ', Ave RMS Est Err = ',...
num2str(mean(errKalman)), ' (Kalman), ', num2str(mean(errParticle)), '...
(Particle)']);

N = 10;
Q = 1;
for i = 1 : NMonte
    [errKalman(i), errParticle(i)] = PartScalar(N, Q);
    fprintf('.');
end
fprintf('\n');
disp(['N = ', num2str(N), ', Q = ', num2str(Q), ', Ave RMS Est Err = ',...
num2str(mean(errKalman)), ' (Kalman), ', num2str(mean(errParticle)), '...
(Particle)']);

N = 1000;
Q = 1;
for i = 1 : NMonte
    [errKalman(i), errParticle(i)] = PartScalar(N, Q);
    fprintf('.');
end
fprintf('\n');
disp(['N = ', num2str(N), ', Q = ', num2str(Q), ', Ave RMS Est Err = ',...
num2str(mean(errKalman)), ' (Kalman), ', num2str(mean(errParticle)), '...
(Particle)']);
```