MECH 6325 - Final Project

Jonas Wagner

2020, December 4

Abstract

In the assignment, multiple dynamic systems are considered and different Kalman filters are developed to estimate the states of the system. In problem 1, a standard Discrete-Time Kalman Filter process is used to monitor the population and food supply of Wombats. In problem 2, a basic RLC circuit with process and measurement noise is observed with a Recursive Kalman Filter. In Problem 3, the steady-state Kalman Filter is explored by solving the Continuous-Time Algebraic Riccati Equation for different system definitions. In Problem 4, a simple second-order Sampled Data System is observed with a Discrete-Time Kalman Filter. In Problem 5, a nonlinear model of an orbiting satellite is observed using both a Linear Hybrid Kalman Filter and a Hybrid Extended Kalman Filter.

Contents

1 Problem 1 - Wombat Population

A linear system describing the population and food supply for wombats is described as follows:

$$\begin{aligned} p_{k+1} &= \frac{1}{2}p_k + 2f_k \\ f_k &= f_0 + w_k \\ y_k &= p_k + v_k \end{aligned} \tag{1}$$

where p_k and f_k are the Wombat Population and food supply respectively; f_0 is the constant mean food supply; y is the measurement of wombat population; additionally, w and v are the process and measurement noise respectively.
The process noise w and measurement noise v are defined as:

$$\begin{aligned} w &\sim (0, Q),\ Q = 10 \\ v &\sim (0, R),\ R = 10 \end{aligned} \tag{2}$$

The initial states of the system are defined as:

$$\begin{aligned} p_0 &= 650 \\ f_0 &= 250 \end{aligned} \tag{3}$$

A standard Discrete-Time Kalman Filter is constructed for the system with the following initial conditions and covariance:

$$\begin{aligned} \hat{p}_0 &= 600, \quad E[(\hat{p}_0 - p_0)^2] = 500 \\ \hat{f}_0 &= 200, \quad E[(\hat{f}_0 - f_0)^2] = 200 \end{aligned} \tag{4}$$

1.1 Modeling Method 1

The system can be written in standard form as:

$$\begin{aligned} x_{k+1} &= Fx_k + Gu + Lw \\ y_k &= Hx_k + v \end{aligned} \tag{5}$$

where x_k is defined as:

$$x_k = \begin{bmatrix} p_k \\ f_k \end{bmatrix} \tag{6}$$

For the first modeling method, it is taken that the food supply is described as stated in the problem, $f_{k+1} = f_0 + w$, so the system matrices are defined as:

$$\begin{gathered} F = \begin{bmatrix} \frac{1}{2} & 2 \\ 0 & 0 \end{bmatrix} \quad G = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad L = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \\ H = \begin{bmatrix} 1 & 0 \end{bmatrix} \quad u_k \equiv f_0, \ \forall \ k \end{gathered} \tag{7}$$

A standard Discrete-Time Kalman Filter is then implemented as described in Subsection 1.3. The code to implement this can be found in Script 2 within Appendix A. The results are seen in Fig. 1.

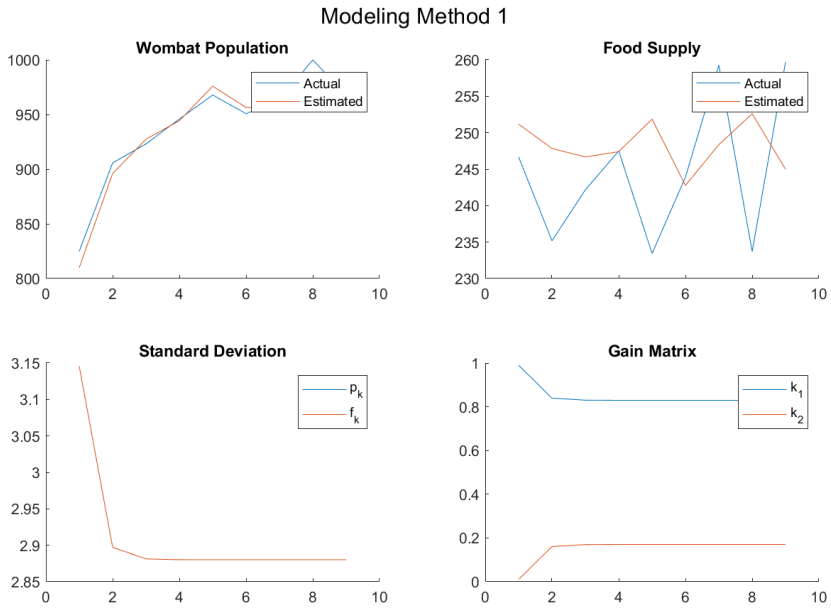


Fig. 1: Simulation results for problem 1 using modeling method 1

1.2 Modeling Method 2

The second method of modeling is also based on the same structure as Subsection 1.1, but instead of a constant input, $f_{k+1} = f_0 + w$, the previous state is used, $f_{k+1} = f_k + w$. This therefore defines these new state matrices:

$$F = \begin{bmatrix} \frac{1}{2} & 2 \\ 0 & 1 \end{bmatrix} \quad G = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad L = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$
$$H = \begin{bmatrix} 1 & 0 \end{bmatrix} \tag{8}$$

A standard Discrete-Time Kalman Filter is then implemented as described in Subsection 1.3. The code to implement this can be found in Script 2 within Appendix A. The results are seen in Fig. 2.

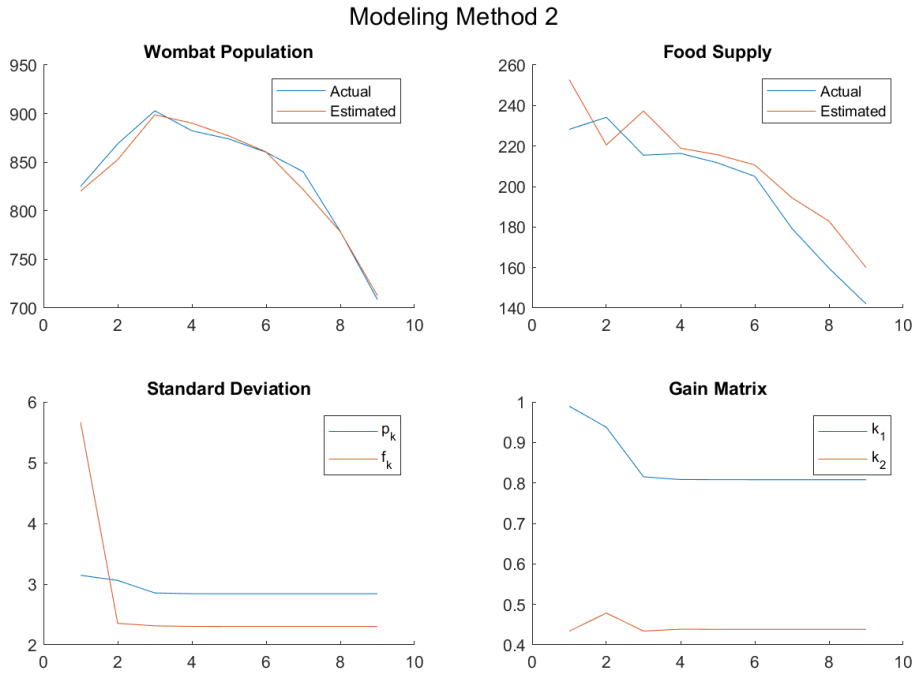


Fig. 2: Simulation results for problem 1 using modeling method 2

1.3 Discrete-Time Kalman Filter

The simulation of the Discrete-Time Kalman Filter is done using a separate function that takes the parameters of the Discrete-Time Linear Time-Invarient system, noise properties, initial conditions, and simulation parameters. It then follows the following procedure to produce and return the actual, measured, and estimated states for each step alongside Kalman Filter values at each step.

1. Initialize the following:

   (a) Arrays that store the data for each step: X, Y, K, X_hat_pre, X_hat_post, P_pre, P_post

   (b) Local variables for each iteration: x, y, p_pre, p_post, u, x_hat_pre, k, x_hat_post

```
x = x_0;
y = H * x + R * randn;
p_pre = P_0;
p_post = p_pre;

u = U(1);
x_hat_pre = x_hat_0;
k = p_pre * H' * inv(R);
x_hat_post = x_hat_pre;
```

2. Begin a for loop to iterate through each time step, `for i = 1:(N-1)`.

   (a) Simulate the system for one step:

```
u = U(i);
x = F * x + G * u + L * Q * randn;
y = H * x + R * randn;
```

   (b) Perform the standard Discrete-Time Kalman Filter update equations:

```
p_pre = F * p_post * F' + L' * Q * L;
k = p_pre * H' * inv(H * p_pre * H' + R);
x_hat_pre = F * x_hat_post + G * u;
x_hat_post = x_hat_pre + k * (y - H * x_hat_pre);
p_post = (eye(n) - k * H) *p_pre * (eye(n) - k * H)' + k * R * k';
```

   (c) Save the local variables into the appropriate arrays.

3. Set the output variables and return the simulated system and kalman filter states and values.

This is implemented in MATLAB within Script 3.

1.4 Theoretical Error Comparison

The theoretical steady state a priori covariance matrix for a Discrete-Time Kalman Filter, P_∞^-, can be solved for using the Discrete-Time Algebraic Riccati Equation (DARE):

$$\begin{aligned} P_\infty^- =& FP_\infty^-F^T - \\ & F(P_\infty^-H^T + M)(HP_\infty^- + HM + M^TH^T + R)^{-1}\times \\ & (HP_\infty^- + M^T)F^T + Q \end{aligned} \tag{9}$$

This can then be used to solve for the steady-state Kalman gain, K_∞:

$$K_\infty = (P_\infty H^T + M)(HP_\infty H^T + HM + M^TH^T + R)^{-1} \tag{10}$$

The theoretical steady state a postari covariance matrix for a Discrete-Time Kalman Filter, P_∞^+, can be calculated as:

$$P_\infty^+ = (I - K_\infty H)P_\infty^- \tag{11}$$

1.4.1 Modeling Method 1:

This can then be applied to find the theoretical standard deviation for each state. Within Script 2 the following is found for the first method:

$$\sqrt{P_\infty^-} = \begin{bmatrix} 2.897 & 0 \\ 0 & 3.163 \end{bmatrix} \tag{12}$$

Comparing this to the steady-state experimental estimation error standard deviation for both states, see Fig. 3, there is a noticeable discrepancy for the second state. This is believed to be because the second state is not based on the previous value, so the standard deviation follows that of the directly measured state.

1.4.2 Modeling Method 2:

This can then be applied to find the theoretical standard deviation for each state. Within Script 2 the following is found for the second method:

$$\sqrt{P_\infty^-} = \begin{bmatrix} 2.9212 & 1.9569 \\ 1.9569 & 3.4784 \end{bmatrix} \tag{13}$$

Comparing this to the steady-state experimental estimation error standard deviation for both states, see Fig. 4, there is a noticeable discrepancy for the second state. In this case, the second state has a better standard deviation then the theoretical one. This could be due to the fact that the expected value for the food-supply never deviates and is more predictable then if an input were included.

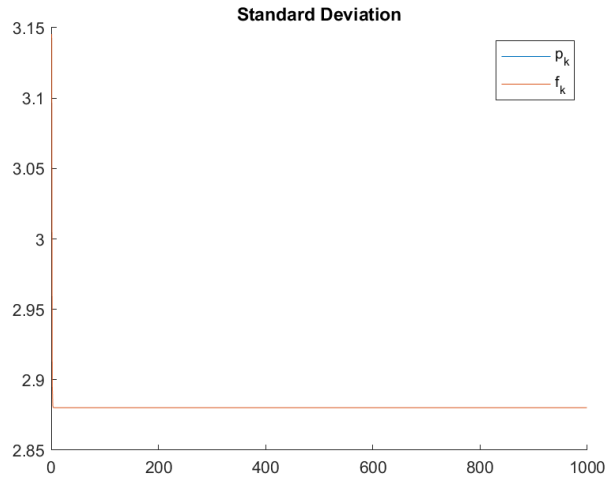


Fig. 3: Experimental estimation error standard deviation for problem 1 using modeling method 1

2 Problem 2 - Electrical Network

Given the electrical network seen in Fig. 5, a Continuous-Time Linear Time-Invarient system can be developed that models the network. This can then be discretized and a sequential Kalman Filtercan be developed.
In this RLC circuit, the following parameters are defined:

$$\begin{aligned} R &= 100 \\ L &= 1 \\ C &= 1 \\ u &\sim (0, (3)^2) \end{aligned} \tag{14}$$

It is also known that the system is relaxed at $t = 0$. The system states are to be directly measured at increments of $T = 0.1$ s with unity variance noise and a sequential Kalman Filter will be used to estimate the Capacitor Voltage and Inductor Current.

2.1 System Modeling

Let x be defined as:

$$x = \begin{bmatrix} v_c \\ i_L \end{bmatrix} \tag{15}$$

The system can then be modeled in the standard Continuous-Time Linear Time-Invarient form:

$$\dot{x} = Ax + Bu \tag{16}$$

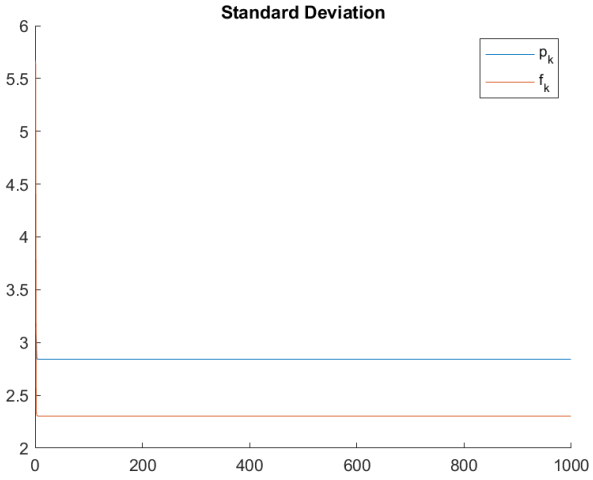


Fig. 4: Experimental estimation error standard deviation for problem 1 using modeling method 1

where A and B are defined as:

$$A = \begin{bmatrix} -2/RC & 1/C \\ -1/L & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 1/RC \\ 1/L \end{bmatrix} \tag{17}$$

The input is set to the process noise term as:

$$u \equiv w \sim (0, Q_c) \tag{18}$$

where $Q_c = 9$.

2.2 Discrete-Time Modeling

Equivalent Discrete-Time Linear Time-Invarient state-space matrices can be calculated as:

$$\begin{aligned} F &= e^{AT} \\ G &= F \int_0^T I - e^{-A\tau} \, \mathrm{d}\tau \, B \\ Q &= Q_c T \end{aligned} \tag{19}$$

With $L = G$ and $H = I_2$, the Discrete-Time Linear Time-Invarient model is then defined as:

$$\begin{aligned} x_{k+1} &= Fx_k + Lw \\ y_k &= Hx_k + v \end{aligned} \tag{20}$$

where the measurement noise is defined as $v \sim (0, I_2)$.

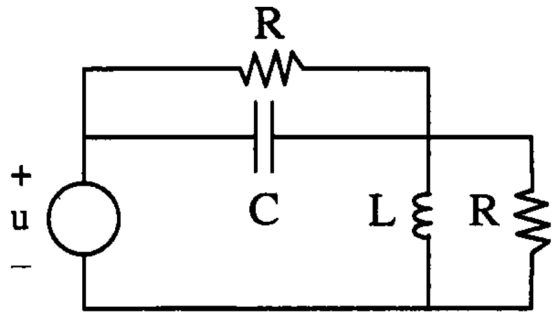


Fig. 5: Electrical network modeling in problem 2.

The system itself is calculated within Script 4 and 5. Numerically the state matrices are defined as follows:

$$F = \begin{bmatrix} 0.9930 & 0.0997 \\ -0.0997 & 0.9950 \end{bmatrix} \quad G = \begin{bmatrix} 0.0060 \\ 0.0998 \end{bmatrix}$$
$$H = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{21}$$

2.3 Sequential Kalman Filter

The simulation of the Sequential Kalman Filter is done using a separate function that takes the parameters of the Discrete-Time Linear Time-Invarient system, noise properties, initial conditions, and simulation parameters. It then follows the following procedure to produce and return the actual, measured, and estimated states for each step alongside Kalman Filter values at each step.

1. Initialize the following:

   (a) Arrays that store the data for each step: X, Y, K, X_hat_pre, X_hat_post, P_pre, P_post, P_all

   (b) Local variables for each iteration: x, y, p_pre, p_post, u, x_hat_pre, k, x_hat_post

```
x = x_0;
y = H * x + R * randn;
p_pre = P_0;
p_post = p_pre;

u = U(1);
x_hat_pre = x_hat_0;
k = p_pre * H' * inv(R);
x_hat_post = x_hat_pre;
```

2. Begin a for loop to iterate through each time step, `for i = 1:N`.

(a) Simulate the system for one step:

```
u = U(i);
x = F * x + G * u + L * Q * randn(size(Q,1),1);
y = H * x + R * randn(size(R,1),1);
```

(b) Perform the standard Discrete-Time Kalman Filter time update equations:

```
p_pre = F * p_post * F' + L' * Q * L;
x_hat_pre = F * x_hat_post + G * u;
```

(c) Initialize the measurement update loop by solving for the first a postari estimate:

```
k(:,1) = p_pre * H(1,:)' * inv(H(1,:) * p_pre * H(1,:)' + R(1,1));
x_hat(:,1) = x_hat_pre + k(:,1)*(y(1) - H(1,:) * x_hat_pre);
p(:,:,1) = (eye(n) - k(:,1) * H(1,:))* p_pre;
```

(d) For the rest of the sensors perform measurement updates as well, `for j in 2:r`.

```
k(:,j) = p(:,:,j-1) * H(j,:)' * inv(H(j,:) * p(:,:,j-1) * H(j,:)' + R(j,
    j));
x_hat(:,j) = x_hat(:,j-1) + k(:,j-1) * (y(j) - H(j,:) * x_hat(:,j-1));
p(:,:,j) = (eye(n) - k(:,j-1) * H(j,:))* p(:,:,j-1);
```

(e) Save the local variables into the appropriate arrays.

3. Set the output variables and return the simulated system and Kalman Filterstates and values.

This is implemented in MATLAB within Script 6.

2.4 Simulation Results

The simulated results of the Sequential Kalman Filter for the Capacitor Voltage Estimation Variance can be seen in Fig. 6.

2.5 Capacitor Voltage Comparison

A comparison of actual, measured, and estimated Capacitor Voltages can be seen in Fig. 7.
An analysis of the measurement and estimation errors yields that the standard deviations of error for each are given as follows:

$$\begin{aligned} \text{std}(Y_{V_C} - X_{V_C}) &= 1.186 \\ \text{std}(\hat{X}_{V_C} - X_{V_C}) &= 0.376 \end{aligned} \tag{22}$$

which clearly demonstrates the effectiveness of the Kalman Filter.

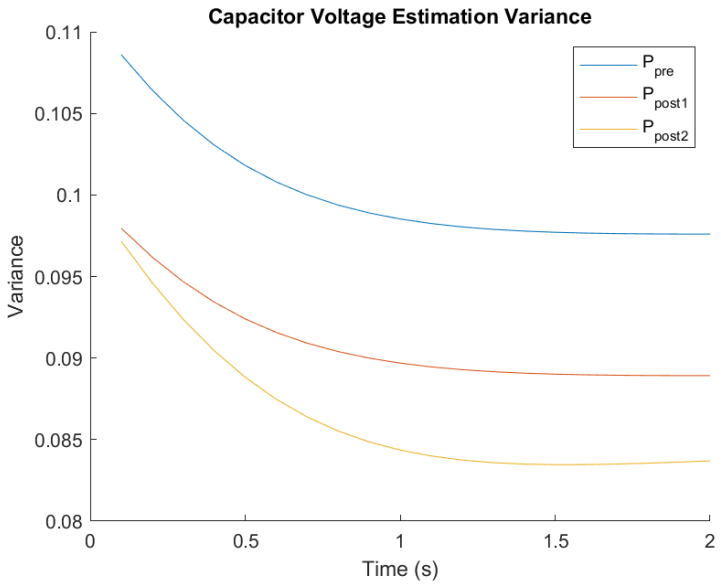


Fig. 6: Capacitor Voltage Estimation Variance for Problem 2

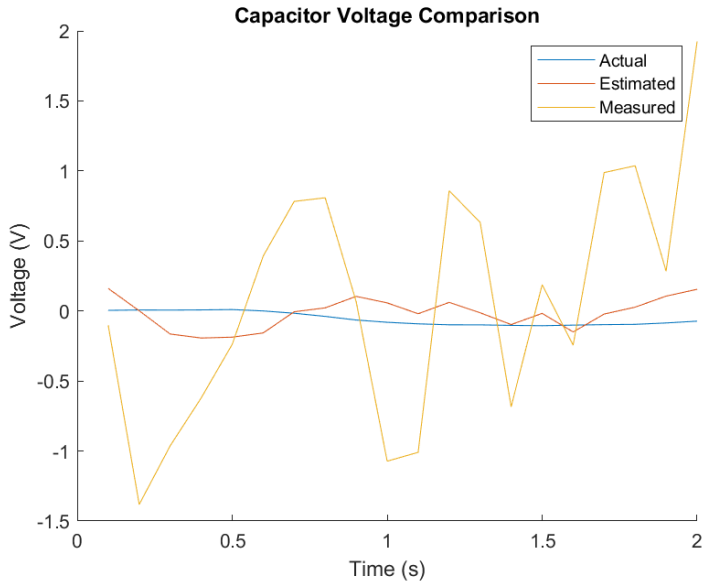


Fig. 7: Capacitor Voltage Comparison for Problem 2

3 Problem 3 - Steady State Riccati Equations

Considering system given as:

$$\begin{aligned} \dot{x} &= Ax + w, & w &\sim (0, Q) \\ y &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} x + v & v &\sim (0, R),\ R = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \end{aligned} \tag{23}$$

the steady-state Kalman Filter can be found by solving the Continuous-Time Algebraic Riccati Equation (CARE). By integrating the CARE until steady-state with various values for A, Q, and P_0, the existence of a stable steady-state Kalman Filter can be determined.

3.1 Integrating the Continuous-Time Algebraic Riccati Equation

Integrating the Continuous-Time Algebraic Riccati Equation (CARE) is done using a separate function that takes the parameters of the CARE and integration specific parameters to numerically solve for a steady-state solution to the CARE. The following procedure is followed when integrating.

1. Initialize the loop by defining:

```
p_dot = - P_0 * C' * inv(R) * C * P_0 + A * P_0 + P_0 * A' + Q;
p = P_0 + p_dot * T;
P(:,:,1) = p;
```

2. Begin a for loop to iterate through each time step, `for i = 2:N`.
   (a) Calculate for each time step:

```
p_dot = - p * C' * inv(R) * C * p + A * p + p * A' + Q;
p = p + p_dot * T;
```

   (b) Save the current value into the output array: `p(:,:,i)=p`

This is implemented in MATLAB within Script 7.

3.2 Results and icare() function comparison

3.2.1 Part a

Let,

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}, \quad Q = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \quad P(0) = I_2 \tag{24}$$

Solving the CARE through numerical integration resulted in the following:

$$P_\infty = \begin{bmatrix} 2.3868 & 0.2774 \\ 0.2774 & 4.2188 \end{bmatrix} \tag{25}$$

Solving the CARE with icare() resulted in the following:

$$P_\infty = \begin{bmatrix} 2.3868 & 0.2774 \\ 0.2774 & 4.2188 \end{bmatrix} \tag{26}$$

As is evident, there is no discrepancy.

3.2.2 Part b

Let,

$$A = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}, \quad Q = \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix}, \quad P(0) = I_2 \tag{27}$$

Solving the CARE through numerical integration resulted in the following:

$$P_\infty = \begin{bmatrix} 0.2899 & 0.5798 \\ 0.5798 & 1.1596 \end{bmatrix} \tag{28}$$

Solving the CARE with icare() resulted in the following:

$$P_\infty = \begin{bmatrix} 0.2899 & 0.5798 \\ 0.5798 & 1.1596 \end{bmatrix} \tag{29}$$

As is evident, there is no discrepancy.

3.2.3 Part c

Let,

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad Q = \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix}, \quad P(0) = I_2 \tag{30}$$

Solving the CARE through numerical integration resulted in the following:

$$P_\infty = \begin{bmatrix} 2.2899 & 0.5798 \\ 0.5798 & 3.1596 \end{bmatrix} \tag{31}$$

Solving the CARE with icare() resulted in the following:

$$P_\infty = \begin{bmatrix} 2.2899 & 0.5798 \\ 0.5798 & 3.1596 \end{bmatrix} \tag{32}$$

As is evident, there is no discrepancy.

3.2.4 Part d

Let,

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad Q = \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix}, \quad P(0) = 0 \tag{33}$$

Solving the CARE through numerical integration resulted in the following:

$$P_\infty = \begin{bmatrix} 0.6899 & 1.3798 \\ 1.3798 & 2.7596 \end{bmatrix} \tag{34}$$

Solving the CARE with icare() resulted in the following:

$$P_\infty = \begin{bmatrix} 2.2899 & 0.5798 \\ 0.5798 & 3.1596 \end{bmatrix} \tag{35}$$

As is evident, there is a discrepancy

3.3 Testing Viability of Steady-State Kalman Filter

Given the P_∞ defined in (34), a steady state Kalman gain can be calculated as:

$$K_\infty = P_\infty C^T R^{-1} = \begin{bmatrix} 0.6899 & 1.3798 \\ 1.3798 & 2.7596 \end{bmatrix} \tag{36}$$

The stability of the steady-state Kalman Filter can then be tested by testing if $(A - K_\infty C)$ is stable.

$$(A - K_\infty C) = \begin{bmatrix} 0.3101 & -1.3798 \\ -1.3798 & -1.7596 \end{bmatrix} \tag{37}$$

This can be found to be unstable as $\lambda(A - K_\infty C) = \{-2.5, 1\}$, and the root at 1 is unstable.

4 Problem 4 - Second Order Sampled Data System

A second-order Continuous-Time Linear Time-Invarient system is given in standard form:

$$\begin{aligned} \dot{x} &= Ax + Lw \\ A &= \begin{bmatrix} 0 & 1 \\ -\omega^2 & -2\zeta\omega \end{bmatrix}, \quad L = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad w \sim (0, Q_c) \end{aligned} \tag{38}$$

numerically, the system is defined with $\omega = 6$ rad/s, $\zeta = 0.16$, and $Q_c = 0.01$.

The system is sampled with $T = 0.5$ s with a measurement noise $v \sim (0, R)$, $R = 10^{-4}$.
Additionally, the initial system and Kalman Filter states are known to be:

$$x(0) = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad \hat{x}(0) = x(0), \quad P(0) = \begin{bmatrix} 10^{-5} & 0 \\ 0 & 10^{-2} \end{bmatrix} \tag{39}$$

4.1 Discrete-Time System

The system is descritized as described in Subsection 2.2. This results in the following numerical results:

$$\begin{aligned} F &= \begin{bmatrix} -0.5908 & 0.01873 \\ -0.6743 & -0.6267 \end{bmatrix} \quad L = \begin{bmatrix} 0.04419 \\ 0.0187 \end{bmatrix} \\ H &= \begin{bmatrix} 1 & 0 \end{bmatrix}, \quad Q = 0.005 \end{aligned} \tag{40}$$

4.2 Sampled Data System Discrete-Time Kalman Filter

The simulation of the Sampled Data System Kalman Filter is done using a separate function that takes the parameters of the ss model, the Discrete-Time Linear Time-Invarient system, noise properties, initial conditions, and simulation parameters. It then follows the following procedure to produce and return the actual, measured, and estimated states for each step alongside Kalman Filter values and the associate time.

1. Generate the time values for the Continuous-Time calcualtions: `Xt = (T/dtdiff)*(0:(N*dtdiff));`
2. Generate the random input signal for all time steps: `W = randn(N*dtdiff+1,1);`
3. Calculate the Continuous-Time Linear Time-Invarient system response and store to X: `X = lsim(sys,U + Q*W,Xt,x_0)';`
4. Initialize the following:
   (a) Arrays that store the data for each step: Y, K, X_hat_pre, X_hat_post, P_pre, P_post
   (b) Local variables for each iteration: x, y, p_pre, p_post, u, x_hat_pre, k, x_hat_post

```
x = x_0;
y = H * x + R * randn;
p_pre = P_0;
p_post = p_pre;

x_hat_pre = x_hat_0;
k = p_pre * H' * inv(R);
x_hat_post = x_hat_pre;
```

5. Begin a for loop to iterate through each time step, `for i = 1:(N-1)`.

   (a) Simulate the system for one step:

```
t = i*T;
x = X(:,dtdiff*i+1);
y = H * x + R * randn;
```

   (b) Perform the standard Discrete-Time Kalman Filter equations:

```
p_pre = F * p_post * F' + L' * Q * L;
k = p_pre * H' * inv(H * p_pre * H' + R);
x_hat_pre = F * x_hat_post + G * u;
x_hat_post = x_hat_pre + k * (y - H * x_hat_pre);
p_post = (eye(n) - k * H) *p_pre * (eye(n) - k * H)' + k * R * k';
```

   (c) Save the local variables into the appropriate arrays.

6. Set the output variables and return the simulated system and Kalman Filterstates and values.

This is implemented in MATLAB within Script 9.

4.3 Simulation Results

The simulation results for the Sampled Data System Kalman Filter can be seen in Fig. 8. Given the simplistic nature of the system and minimal noise, the Kalman Filter does a good job at estimating the state; however, the sampling rate is far too slow to effectively capture the higher frequencies of the actual system response.

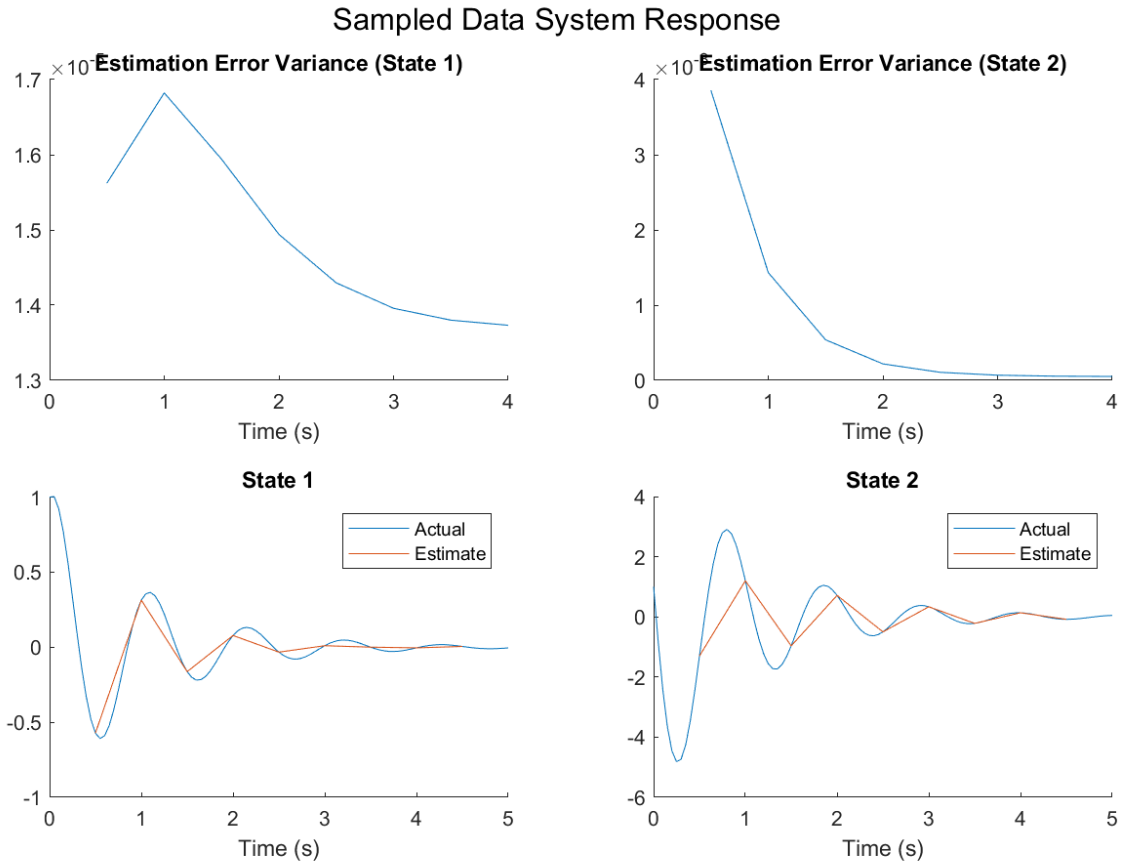


Fig. 8: Simulation Results for problem 4

5 Problem 5 - Satellite Orbit

All calculations and simulations were done in MATLAB, and can be seen in Script 10.

A planar model for an orbiting satalite is given as:

$$\begin{aligned} \ddot{r} &= r\dot{\theta}^2 - \frac{GM}{r^2} + w \\ \ddot{\theta} &= \frac{-2\dot{\theta}\dot{r}}{r} \end{aligned} \tag{41}$$

where the gravitational constant $G = 6.674e{-}11\left[\frac{m^2}{kg/s^2}\right]$ and the mass of the earth $M = 5.98e24[kg]$, and $w \sim (0, Q),\ Q = 10^{-6}$.

5.1 Nonlinear State-Space Model

Let,

$$x = \begin{bmatrix} r \\ \dot{r} \\ \theta \\ \dot{\theta} \end{bmatrix} \tag{42}$$

The state space model can then be defined as:

$$\dot{x} = f(x) = \begin{bmatrix} \dot{r} \\ r\dot{\theta}^2 - \frac{GM}{r^2} + w \\ \dot{\theta} \\ \frac{-2\dot{\theta}\dot{r}}{r} \end{bmatrix} \tag{43}$$

5.1.1 pblm5_nonlin Function

When working with the nonlinear model, a function was defined to evaluate the system as defined in the original nonlinear state-space model. This is viewable in Script 11, but the main functionality is given by:

```
x_dot(1) = x(2);
x_dot(2) = x(1) * x(4)^2 - (G*M)/(x(1)^2) + w;
x_dot(3) = x(4);
x_dot(4) = -2*x(4)*x(2)/x(1);
```

5.2 Steady-State Radial Velocity

When $w = 0$, $\dot{\theta}$ can be solved for such that r remains constant:

$$0 = \ddot{r} = r\dot{\theta}^2 - \frac{GM}{r^2} \tag{44}$$

$$\frac{GM}{r^2} = r\dot{\theta}^2 \tag{45}$$

$$\dot{\theta}_0 = \pm\sqrt{\frac{GM}{r^3}} \tag{46}$$

5.3 Continuous-Time Linear Time-Invarient Model

A Continuous-Time Linear Time-Invarient model can be developed around an equilibrium point as follows:

Let,

$$x_0 = \begin{bmatrix} r_0 \\ 0 \\ \omega_0 T \\ \omega_0 \end{bmatrix} \tag{47}$$

The state matrix A can then be found by evaluating the Jacobian of $f(x)$ at x_0:

$$A = \left.\frac{\mathrm{d}f}{\mathrm{d}x}\right|_{x_0} = \left[\begin{array}{cccc} 0 & 1 & 0 & 0 \\ \dot{\theta}^2 + \dfrac{GM}{r^3} & 0 & 0 & 2r\dot{\theta} \\ 0 & 0 & 0 & 1 \\ 2\dot{\theta}\dot{r} & \dfrac{-2\dot{\theta}}{r} & 0 & \dfrac{-2\dot{r}}{r} \end{array}\right]\Bigg|_{x_0}$$

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \omega_0^2 + \dfrac{GM}{r_0^3} & 0 & 0 & 2r_0\omega_0 \\ 0 & 0 & 0 & 1 \\ 0 & \dfrac{-2\omega_0}{r_0} & 0 & 0 \end{bmatrix} \tag{48}$$

Numerically, with $r_0 = 6.57 \times 10^6[m]$ the A matrix is calculated to be:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 4.22 \times 10^{-6} & 0 & 0 & 1.56 \times 10^4 \\ 0 & 0 & 0 & 1 \\ 0 & -3.61 \times 10^{-10} & 0 & 0 \end{bmatrix} \tag{49}$$

The eigen-values of the A matrix can be calculated to be:

$$\lambda(A) = \{0, 0, \pm j1.2 \times 10^{-3}\} \tag{50}$$

The time constant of the system τ can then be calculated as follows:

$$\begin{aligned} 2\pi f_0 = \frac{2\pi}{\tau} &= 1.2 \times 10^{-3} \\ \tau &\approx 5.2 \times 10^3[s] \end{aligned} \tag{51}$$

The largest integration step to capture the important frequencies can be estimated as:

$$0.1\tau = 520[s] \tag{52}$$

5.4 Linear Hybrid Kalman Filter

The simulation of the Linear Hybrid Kalman Filter is done using a separate function that takes the nonlinear system function, the parameters of the Continuous-Time Linear Time-Invarient system, noise properties, initial conditions, and simulation parameters. It then follows the following procedure to produce and return the actual, measured, and estimated states for each step alongside Kalman Filter values and time-step values.

1. Initialize the following:
   (a) Arrays that store the data for each step: X, Y, K, X_hat_pre, X_hat_post, P_pre, P_post, Xt, sensor_t
   (b) Local variables for each iteration: x, p, x_hat, k, (y)

```
x = x_0;
p = P_0;
x_hat = x_hat_0;
k = p * H' * inv(R);
```

2. Begin a for loop to iterate through each time step, `for i = 1:N`.
   (a) Simulate the system for one step:

```
t = i * sensorT;
x_dot = nonlinsys(x);
x = x + x_dot * sensorT;
y = H * x + R * randn(size(R,2),1);
```

   (b) Store values into their respective arrays: X, Y, sensor_t
   (c) Perform the standard Continuous-Time Kalman Filter time update equations:

```
x_hat_dot = A * x_hat;
x_hat = x_hat + x_hat_dot * sensorT;
p_dot = A * p + p * A' + L*Q_c*L';
p = p + p_dot * sensorT;
```

   (d) Store values into their respective arrays: X_hat_pre, P_pre
   (e) Perform the standard Discrete-Time Kalman Filter measurment update equations:

```
k = p*H'*inv(H*p*H' + R);
x_hat = x_hat + k * (y - H * x_hat);
p = (eye(n)-k*H)*p*(eye(n)-k*H)' + k*R*k';
```

   (f) Store values into their respective arrays: K, X_hat_post, P_post
3. Set the output variables and return the simulated system and Kalman Filterstates and values.

This is implemented in MATLAB within Script 12.

5.5 Hybrid Extended Kalman Filter

The Hybrid Extended Kalman Filter is nearly identical to the Linear Hybrid Kalman Filter version, except for the non-linear time update step, **Step 2c**.

The simulation of the Extended Hybrid Kalman Filter is done using a separate function that takes the nonlinear system function, the parameters of the Continuous-Time Linear Time-Invarient system, noise properties, initial conditions, and simulation parameters. It then follows the following procedure to produce and return the actual, measured, and estimated states for each step alongside Kalman Filter values and time-step values.

1. Initialize the following:
   (a) Arrays that store the data for each step: X, Y, K, X_hat_pre, X_hat_post, P_pre, P_post, Xt, sensor_t

   (b) Local variables for each iteration: x, p, x_hat, k, (y)

```
x = x_0;
p = P_0;
x_hat = x_hat_0;
k = p * H' * inv(R);
```

2. Begin a for loop to iterate through each time step, `for i = 1:N`.
   (a) Simulate the system for one step:

```
t = i * sensorT;
x_dot = nonlinsys(x);
x = x + x_dot * sensorT;
y = H * x + R * randn(size(R,2),1);
```

   (b) Store values into their respective arrays: X, Y, sensor_t
   (c) Perform the following nonlinear time update equations:

```
x_hat_dot = nonlinsys(x_hat,0);
x_hat = x_hat + x_hat_dot * sensorT;
p_dot = A * p + p * A' + L*Q_c*L';
p = p + p_dot * sensorT;
```

   (d) Store values into their respective arrays: X_hat_pre, P_pre
   (e) Perform the standard Discrete-Time Kalman Filter measurment update equations:

```
k = p*H'*inv(H*p*H' + R);
x_hat = x_hat + k * (y - H * x_hat);
p = (eye(n)-k*H)*p*(eye(n)-k*H)' + k*R*k';
```

   (f) Store values into their respective arrays: K, X_hat_post, P_post
3. Set the output variables and return the simulated system and Kalman Filterstates and values.

This is implemented in MATLAB within Script 13.

5.6 Simulation Results

5.6.1 Linear Hybrid Kalman Filter

The Linear Hybrid Kalman Filter was simulated using Subsection 5.4. The system response can be seen in Fig. 9.

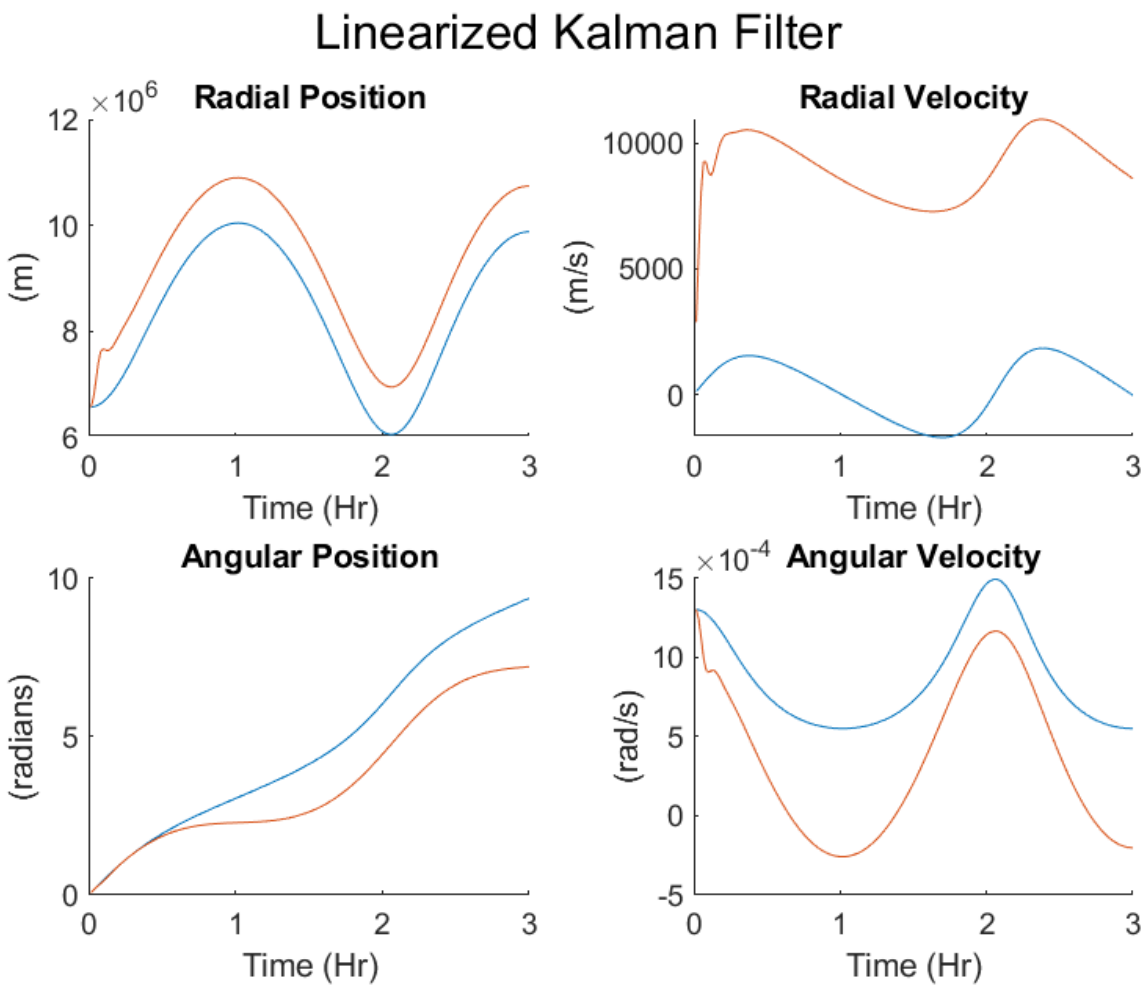


Fig. 9: Simulation Results for the Linear Hybrid Kalman Filter for problem 5

The estimation error for the Linear Hybrid Kalman Filter can be seen in Fig. 10.

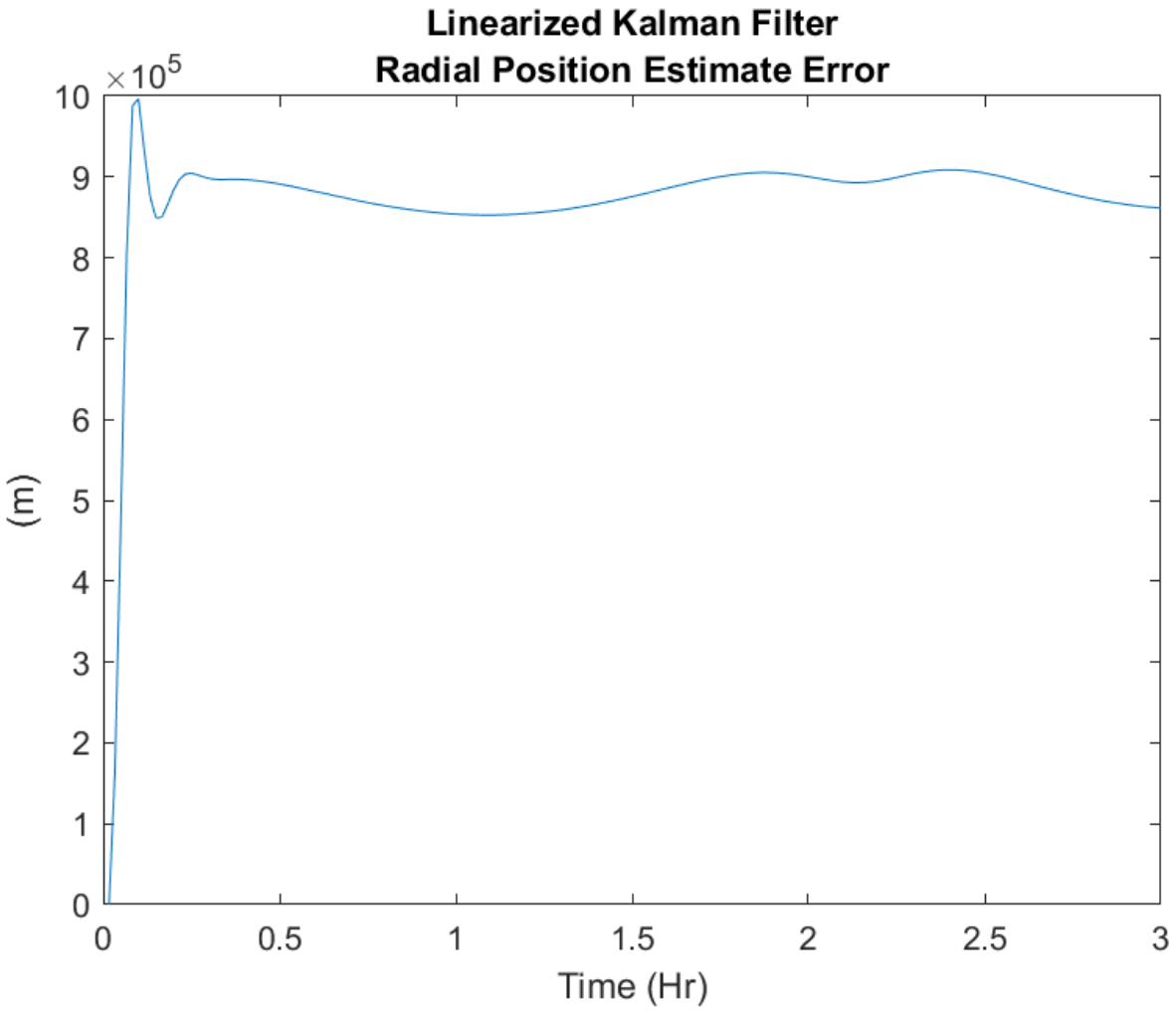


Fig. 10: Estimation Error for the Linear Hybrid Kalman Filter for problem 5

The performance of the linear Kalman Filter is reduced by the lack of linearity in the actual system dynamics. This may be improved with faster measurments and Kalman Filter updates; however this is not guaranteed. Using an Extended Kalman Filter would be the better solution.

5.6.2 Hybrid Extended Kalman Filter

The Hybrid Extended Kalman Filter was simulated using Subsection 5.5. The system response can be seen in Fig. 11.

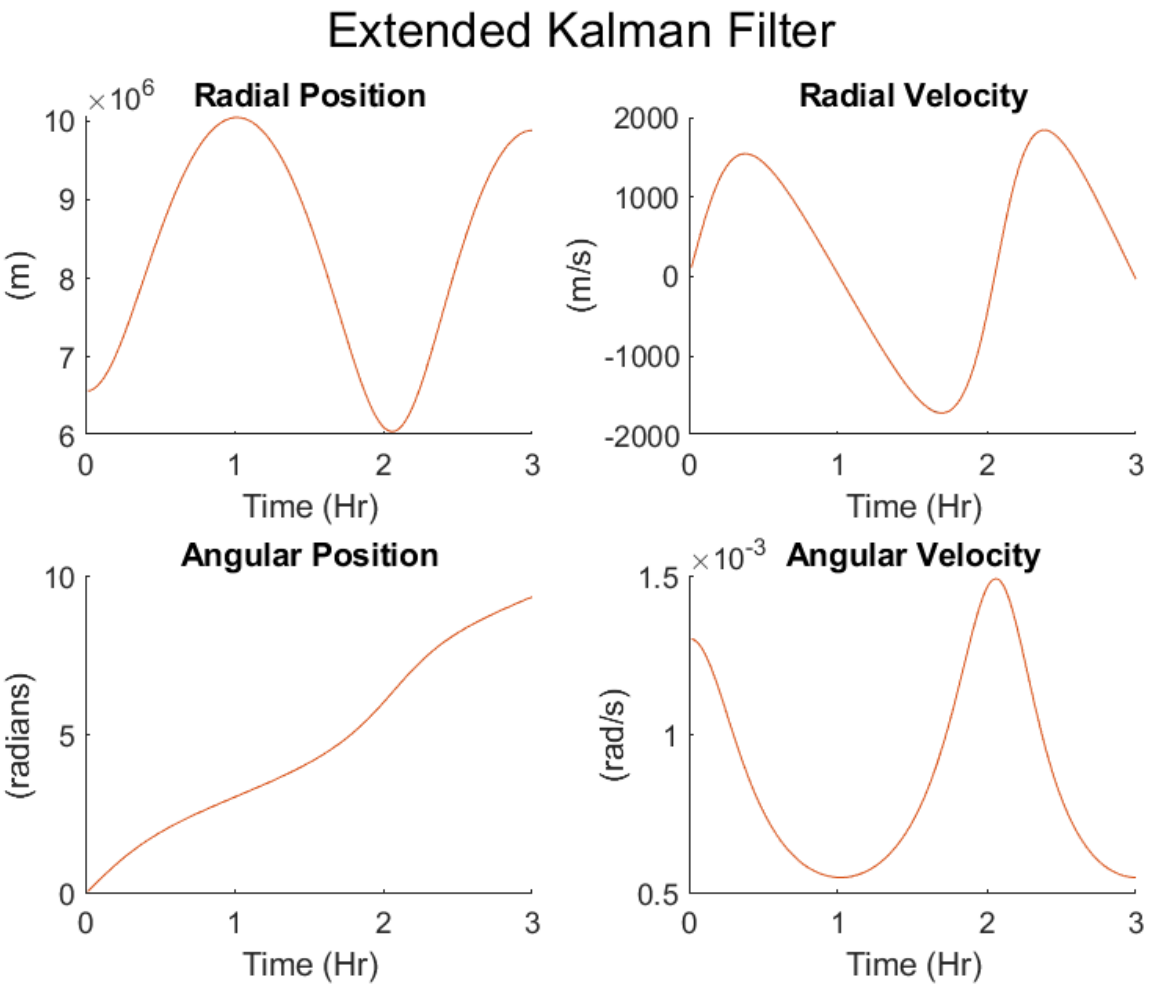


Fig. 11: Simulation Results for the Hybrid Extended Kalman Filter for problem 5

The estimation error for the Hybrid Extended Kalman Filter can be seen in Fig. 12.

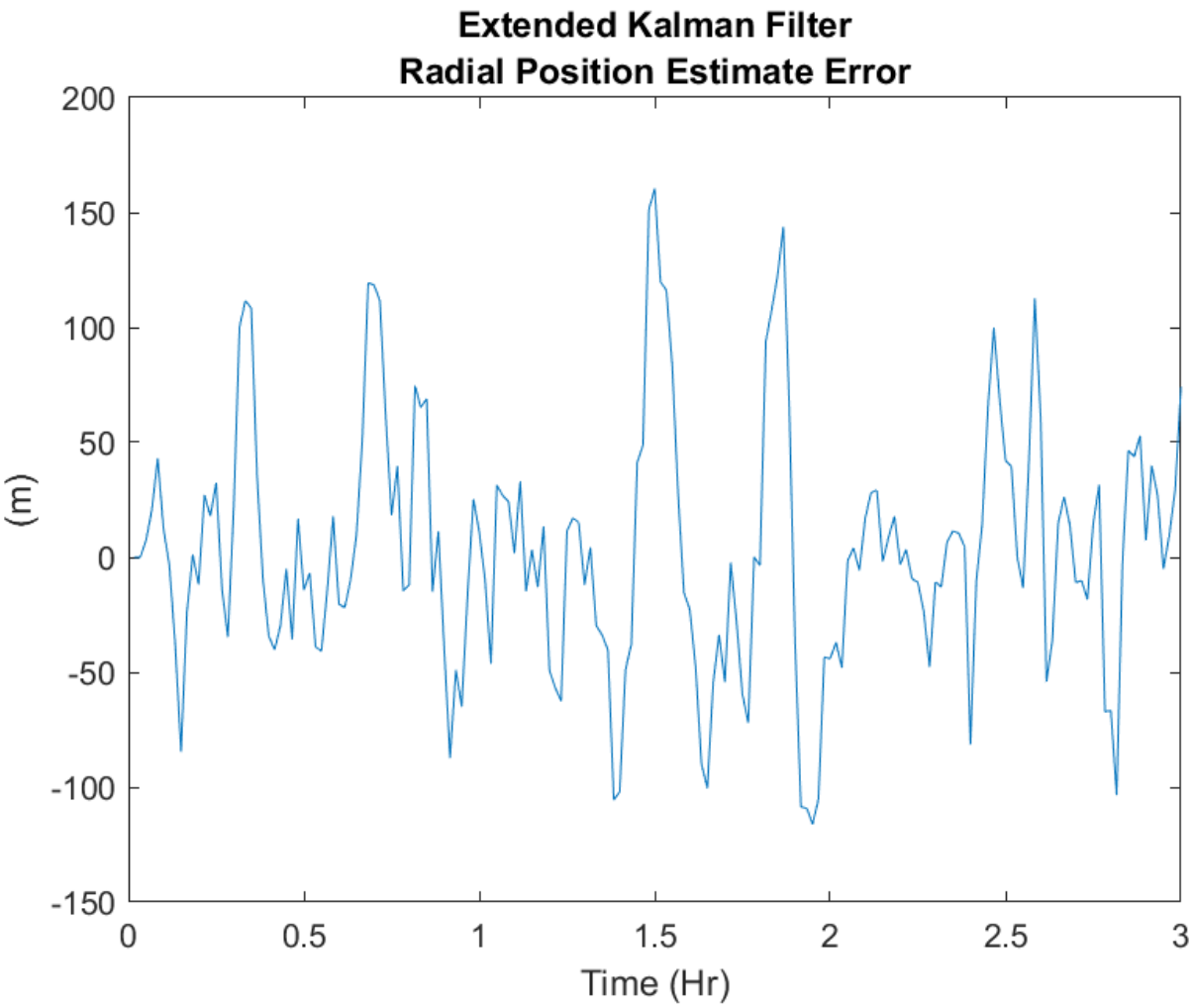


Fig. 12: Estimation Error for the Hybrid Extended Kalman Filter for problem 5

It is clear that the Extended Kalman Filter performs much better then the Linear one. Although the error is more sporadic, the magnitude is order smaller then the linear one. This is likely due to the ability for the Extended Kalman Filter to account for the non linear dynamics during the time-update stage.

A MATLAB Code

Script 1: MECH6325_FinalProject.m

```
% MECH 6325 - Final Project
%Jonas Wagner
%2020-11-28

clear
close all;

% Problem 1 ---------------------------------------------------------------
disp('Problem 1 ----------------------------------------------------------')
pblm1()

clear
% Problem 2 ---------------------------------------------------------------
disp('Problem 2 ----------------------------------------------------------')
pblm2()

clear
% Problem 3 ---------------------------------------------------------------
disp('Problem 3 ----------------------------------------------------------')
pblm3()

clear
% Probelm 4 ---------------------------------------------------------------
disp('Problem 4 ----------------------------------------------------------')
pblm4()

clear
% Problem 5 ---------------------------------------------------------------
disp('Problem 5 ----------------------------------------------------------')
pblm5()

clear
close all %Comment this out if figures are wanted...
```

A.1 Problem 1

Script 2: pblm1.m

```
function [] = pblm1()
% Problem 1 Scipt/Function
% MECH 6325 - Final Probject

% System Definition
F = [1/2, 2;
     0, 0];
G = [0; 1];
f_0 = 250;
L = [0; 1];
H = [1, 0];

x_0 = [650; 250];
Q = 10;
R = 10;

x_hat_0 = [600; 200];
P_0 = [500, 0;
       0, 200];

% Part a - KF Simulation (10 steps) ------------------------------------
n = size(F,1);
T = 1;
T_max = 10;
N = floor(T_max/T);
U = f_0 * ones(N+1,1);

[X,~,~,X_hat,P,K] = KalmanFilter_DT(F,G,L,H,x_0,Q,R,x_hat_0,P_0,N,n,U);


figure('position',[0,0,3*300,2*300])
sgtitle('Modeling Method 1')

subplot(2,2,1)
hold on
plot(X(1,:))
plot(X_hat(1,:))
hold off
title('Wombat Population')
legend('Actual','Estimated')

subplot(2,2,2)
hold on
```

```
plot(X(2,:))
plot(X_hat(2,:))
hold off
title('Food Supply')
legend('Actual','Estimated')

subplot(2,2,3)
hold on
plot(sqrt(reshape(P(1,1,:),[],1)))
plot(sqrt(reshape(P(2,2,:),[],1)))
hold off
title('Standard Deviation')
legend('p_k','f_k')

subplot(2,2,4)
hold on
plot(K(1,:))
plot(K(2,:))
hold off
title('Gain Matrix')
legend('k_1','k_2')

saveas(gcf,[pwd '/fig/pblm1_results_method1.png'])


% Part b - P_inf comparison -----------------------------------------------
disp('If using the following F, there is a descrepancy for state 2')
F = [1/2, 2;
    0, 0]
P_minus_inf = idare(F',H',Q,R)
k_inf = P_minus_inf*H'/(H*P_minus_inf*H' + R)
P_plus_inf = (eye(2)-k_inf*H)*P_minus_inf
P_plus_inf_std = sqrt(P_plus_inf)

% Part c - Simulation (1000 steps)
T_max = 1000;
N = floor(T_max/T);
U = f_0 * ones(N+1,1);
[X1,~,~,X_hat1,P1,~] = KalmanFilter_DT(F,G,L,H,x_0,Q,R,x_hat_0,P_0,N,n,U);

figure()
hold on
plot(sqrt(reshape(P1(1,1,:),[],1)))
plot(sqrt(reshape(P1(2,2,:),[],1)))
hold off
title('Standard Deviation')
```

```
legend('p_k','f_k')
saveas(gcf,[pwd '/fig/pblm1_std_method1.png'])

disp(['There is a decrepancy for the secound state.', newline,...
    'This is belived to be becouse of the constant input not = 0,',...
    newline, 'so the steady-state Kalman filter for that state is ',...
    'not as accurate.']);


disp('------------------------------------------------------------------')


disp('The alternative method of simulation:')

F = [1/2, 2;
    0, 1]
G = [0; 1]

n = size(F,1);
T = 1;
T_max = 10;
N = floor(T_max/T);
U = 0 * ones(N+1,1);

[X,~,~,X_hat,P,K] = KalmanFilter_DT(F,G,L,H,x_0,Q,R,x_hat_0,P_0,N,n,U);


figure('position',[0,0,3*300,2*300])
sgtitle('Modeling Method 2')

subplot(2,2,1)
hold on
plot(X(1,:))
plot(X_hat(1,:))
hold off
title('Wombat Population')
legend('Actual','Estimated')

subplot(2,2,2)
hold on
plot(X(2,:))
plot(X_hat(2,:))
hold off
title('Food Supply')
legend('Actual','Estimated')

```

```
subplot(2,2,3)
hold on
plot(sqrt(reshape(P(1,1,:),[],1)))
plot(sqrt(reshape(P(2,2,:),[],1)))
hold off
title('Standard Deviation')
legend('p_k','f_k')

subplot(2,2,4)
hold on
plot(K(1,:))
plot(K(2,:))
hold off
title('Gain Matrix')
legend('k_1','k_2')

saveas(gcf,[pwd '/fig/pblm1_results_method2.png'])

P_minus_inf = idare(F',H',L*Q*L',R)
k_inf = P_minus_inf*H'/(H*P_minus_inf*H' + R)
P_plus_inf = (eye(2)-k_inf*H)*P_minus_inf
P_plus_inf_std = sqrt(P_plus_inf)

% x_hat_error_inf =

T_max = 1000;
N = floor(T_max/T);
U = 0 * ones(N+1,1);
[X2,~,~,X_hat2,P2,~] = KalmanFilter_DT(F,G,L,H,x_0,Q,R,x_hat_0,P_0,N,n,U);

figure()
hold on
plot(sqrt(reshape(P2(1,1,:),[],1)))
plot(sqrt(reshape(P2(2,2,:),[],1)))
hold off
title('Standard Deviation')
legend('p_k','f_k')
saveas(gcf,[pwd '/fig/pblm1_std_method2.png'])


disp('The alternative method has a descrepancy for the food term as well')

end
```

Script 3: KalmanFilter_DT.m

```
function [X,Y,X_hat_pre,X_hat_post,P,K] = KalmanFilter_DT(F,G,L,H,x_0,Q,R,...
                                        x_hat_0,P_0,N,n,U)
    arguments
        F
        G
        L
        H
        x_0
        Q
        R
        x_hat_0
        P_0
        N
        n = size(F,1)
        U = zeros(N+1,1);
    end

    x = x_0;
    y = H * x + R * randn(size(R,1),1);
    p_pre = P_0;
    p_post = p_pre;

    u = U(1);
    x_hat_pre = x_hat_0;
    k = p_pre * H' * inv(R);
    x_hat_post = x_hat_pre;

    X = [];
    Y = [];

    K = [];
    X_hat_pre= [];
    X_hat_post = [];
    P_pre = zeros(n,n,N-1);
    P_post = zeros(n,n,N-1);

    for i = 1:(N-1)
        u = U(i);
        x = F * x + G * u + L * Q * randn(size(Q,1),1);
        y = H * x + R * randn(size(R,1),1);

        p_pre = F * p_post * F' + L' * Q * L;
        k = p_pre * H' * inv(H * p_pre * H' + R);
        x_hat_pre = F * x_hat_post + G * u;
        x_hat_post = x_hat_pre + k * (y - H * x_hat_pre);
```

```
        p_post = (eye(n) - k * H) *p_pre * (eye(n) - k * H)' + k * R * k';

        X = [X x];
        Y = [Y y];

        K = [K k];
        X_hat_pre = [X_hat_pre x_hat_pre];
        X_hat_post = [X_hat_post x_hat_post];
        P_pre(:,:,i) = p_pre;
        P_post(:,:,i) = p_post;
    end


    X = X;
    Y = Y;

    X_hat_pre = X_hat_pre;
    X_hat_post = X_hat_post;
    P = P_post;

end
```

A.2 Problem 2

Script 4: pblm2.m

```
function [] = pblm2()
% Problem 2 Scipt/Function
% MECH 6325 - Final Probject

% Parameters
R = 100;
L = 1;
C = 1;

% CT System Definitions
A = [-2/(R*C), 1/C;
     -1/L, 0];
B = [1/(R*C);
     1/L];
C = eye(2);
Q_c = 3^2;

T = 0.1;
T_max = 2;
N = floor(T_max/T);

disp('Discretized System:')
[F,G,H,Q,~] = DiscretizeSystem(A,B,C,Q_c,0,T)
% L = G

n = size(F,1);
x_0 = zeros(n,1);
R = eye(2);

x_hat_0 = x_0;
P_0 = 0.1 * eye(n);

[X,Y,~,X_hat_post,P_pre,P_post,~] = ...
    KalmanFilter_Sequential(F,G,G,H,x_0,Q,R,x_hat_0,P_0,N,n);

%Plotting
t = T:T:T_max;
% Part a
figure()
hold on
plot(t,reshape(P_pre(1,1,:),[],1))
plot(t,reshape(P_post(1,1,:,1),[],1))
plot(t,reshape(P_post(1,1,:,2),[],1))
```

```
hold off
title('Capacitor Voltage Estimation Variance')
legend('P_{pre}','P_{post}_1','P_{post}_2')
xlabel('Time (s)')
ylabel('Variance')
saveas(gcf,[pwd '/fig/pblm2_est_var.png'])

% Part b
figure()
hold on
plot(t,X(1,:))
plot(t,X_hat_post(1,:))
plot(t,Y(1,:))
hold off
title('Capacitor Voltage Comparison')
legend('Actual','Estimated','Measured')
xlabel('Time (s)')
ylabel('Voltage (V)')
saveas(gcf,[pwd '/fig/pblm2_comparison.png'])

disp('Standard diviations of errors')
Y_error_std = std(Y(1,:)-X(1,:))
X_hat_error_std = std(X_hat_post(1,:)-X(1,:))

end
```

Script 5: DiscretizeSystem.m.m

```
function [F,G,H,Q,R] = DiscretizeSystem(A,B,C,Q_c,R_c,T)
    n = size(A,1);
    if B == 0
        B = zeros(n,1);
    end
    if C == 0
        C = zeros(1,n);
    end
    sys = ss(A,B,C,0);
    dt_sys = c2d(sys,T);
    [F,G,H,~,T] = ssdata(dt_sys);
% Original Code... issues with stability using this method...
% n = size(A,1);
% F = exp(A * T);
% syms tau
% G = double(F * int(eye(n) - exp(-A * tau),tau,0,T) * B);
% H = C;
    Q = Q_c * T;
    R = R_c * T;
end
```

Script 6: KalmanFilter_Sequential.m

```
function [X,Y,X_hat_pre,X_hat_post,P_pre,P_post,K] = ...
    KalmanFilter_Sequential(F,G,L,H,x_0,Q,R,x_hat_0,P_0,N,n,r,U)
    arguments
        F
        G
        L
        H
        x_0
        Q
        R
        x_hat_0
        P_0
        N = 100;
        n = size(F,1);
        r = size(H,1)
        U = zeros(N+1,1);
    end

    x = x_0;
    y = H * x + R * randn;
    p_pre = P_0;
    p_post = p_pre;

    u = U(1);
    x_hat_pre = x_hat_0;
    x_hat_post = x_hat_pre;

    X = [];
    Y = [];

    K = zeros(n,r,N);
    X_hat_pre= [];
    X_hat_post = [];
    P_pre = zeros(n,n,N);
    P_post = zeros(n,n,N);

    for i = 1:N
        u = U(i);
        x = F * x + G * u + L * Q * randn(size(Q,1),1);
        y = H * x + R * randn(size(R,1),1);

        % Time Update
        p_pre = F * p_post * F' + L' * Q * L;
        x_hat_pre = F * x_hat_post + G * u;

```

```
        k = zeros(n,r);
        x_hat = zeros(n,r);
        p = zeros(n,n,r);

        %Measurement Update
        k(:,1) = p_pre * H(1,:)' * inv(H(1,:) * p_pre * H(1,:)' + R(1,1));
        x_hat(:,1) = x_hat_pre + k(:,1)*(y(1) - H(1,:) * x_hat_pre);
        p(:,:,1) = (eye(n) - k(:,1) * H(1,:))* p_pre;

        for j = 2:r
            k(:,j) = p(:,:,j-1) * H(j,:)' * inv(H(j,:) * p(:,:,j-1) * H(j,:)' + R(j
                ,j));
            x_hat(:,j) = x_hat(:,j-1) + k(:,j-1) * (y(j) - H(j,:) * x_hat(:,j-1));
            p(:,:,j) = (eye(n) - k(:,j-1) * H(j,:))* p(:,:,j-1);
        end

        x_hat_post = x_hat(:,r);
        p_post = p(:,:,r);

        X = [X x];
        Y = [Y y];

        K(:,:,i) = k;
        X_hat_pre = [X_hat_pre x_hat_pre];
        X_hat_post = [X_hat_post x_hat_post];
        P_all(:,:,i,:) = p(:,:,:);
        P_pre(:,:,i) = p_pre;
        P_post(:,:,i) = p_post;
    end

    X = X;
    Y = Y;

    X_hat_pre = X_hat_pre;
    X_hat_post = X_hat_post;
    P_pre = P_pre;
    P_post = P_all;
end
```

A.3 Problem 3

Script 7: pblm3.m

```
function [] = pblm3()
% Problem 3 Scipt/Function
% MECH 6325 - Final Probject

C = eye(2);
R = eye(2);
N = 1e3;
T = 1e-2;

% Part a
disp('Part a ------------------')
A = diag([1,2]);
Q = [1,1;
     1,1];
P_0 = eye(2);
P = careInt(A,C,Q,R,P_0,N,T);
P_ss_a = P(:,:,end)

P_ss_a_care = care(A',C',Q,R)

% Part b
disp('Part b ------------------')
A = diag([-1,-1]);
Q = [1,2;
     2,4];
P_0 = eye(2);
P = careInt(A,C,Q,R,P_0,N,T);
P_ss_b = P(:,:,end)

P_ss_b_care = care(A',C',Q,R)


% Part c
disp('Part c ------------------')
A = diag([1,1]);
Q = [1,2;
     2,4];
P_0 = eye(2);
P = careInt(A,C,Q,R,P_0,N,T);
P_ss_c = P(:,:,end)

P_ss_c_care = care(A',C',Q,R)

```

```
N = 1e5;

% Part d
disp('Part d -----------------')
A = diag([1,1]);
Q = [1,2;
     2,4];
P_0 = 0;
P = careInt(A,C,Q,R,P_0,N,T);
P_ss_d = P(:,:,end)

P_ss_d_care = care(A',C',Q,R)

K_int = P_ss_d * C' * inv(R)
A_KC_int = A-K_int*C
eig_int = eig(A_KC_int)
disp(['This is an unstable observer, so it does not result in a' ,...
      'steady-state Kalman Filter'])
end


function P = careInt(A,C,Q,R,P_0,N,T)
    arguments
        A
        C
        Q
        R
        P_0
        N = 100;
        T = 0.01;
    end
    p_dot = - P_0 * C' * inv(R) * C * P_0 + A * P_0 + P_0 * A' + Q;
    p = P_0 + p_dot * T;
    P(:,:,1) = p;
    for i = 2:N
        p_dot = - p * C' * inv(R) * C * p + A * p + p * A' + Q;
        p = p + p_dot * T;
        P(:,:,i) = p;
    end
end
```

A.4 Problem 4

Script 8: pblm4.m

```
function [] = pblm4()
% Problem 4 Scipt/Function
% MECH 6325 - Final Probject

% Parameters
T = 0.5;
T_max = 5;

omega = 6;
zeta = 0.16;

% System Definition
A = [0, 1;
    -omega^2, -2*zeta*omega];
B = [0;1];
C = eye(2);
Q_c = 0.01;

sys = ss(A,B,C,0)


% Part a - Discretization
[F,G,~,Q,~] = DiscretizeSystem(A,B,C,Q_c,0,T);
H = [1,0];
Q
L=G;
R = 1e-4;


dt_sys = ss(F,G,H,0,T)

% Part b - Simulation
x_0 = [1;1];
x_hat_0 = x_0;
P_0 = diag([1e-5,1e-2]);


[X,Xt,~,~,~,X_hat,P,~,hat_t] = KalmanFilter_DT_SDS(sys,F,0,L,H,...
                                        x_0,Q,R,x_hat_0,P_0,T,T_max);



figure('position',[0,0,3*300,2*300])
```

```
sgtitle('Sampled Data System Response')

subplot(2,2,1)
hold on
plot(hat_t(1:end-1),reshape(P(1,1,1:end-1),[],1))
hold off
title('Estimation Error Variance (State 1)')
xlabel('Time (s)')

subplot(2,2,2)
hold on
plot(hat_t(1:end-1),reshape(P(2,2,1:end-1),[],1))
hold off
title('Estimation Error Variance (State 2)')
xlabel('Time (s)')

subplot(2,2,3)
hold on
plot(Xt,X(1,:))
plot(hat_t,X_hat(1,:))
hold off
title('State 1')
legend('Actual','Estimate')
xlabel('Time (s)')

subplot(2,2,4)
hold on
plot(Xt,X(2,:))
plot(hat_t,X_hat(2,:))
hold off
title('State 2')
legend('Actual','Estimate')
xlabel('Time (s)')
saveas(gcf,[pwd '/fig/pblm4_results.png'])
end
```

Script 9: KalmanFilter_DT_SDS.m

```
function [X,Xt,Y,W,X_hat_pre,X_hat_post,P,K,hat_t] = KalmanFilter_DT_SDS(sys,F,G,
    L,H,...
                                        x_0,Q,R,...
                                         x_hat_0,P_0,T,T_max,n,dtdiff,U)
    arguments
        sys % CT LTI sys... should have direct output of states
        F
        G
        L %Assumed L = G... if not W or L should be modified...
        H
        x_0
        Q
        R
        x_hat_0
        P_0
        T
        T_max
        n = size(F,1);
        dtdiff = 10;
        U = zeros(dtdiff*floor(T_max/T)+1,1);
    end
    N = floor(T_max/T);

    x = x_0;
    y = H * x + R * randn;
    p_pre = P_0;
    p_post = p_pre;

    x_hat_pre = x_hat_0;
    k = p_pre * H' * inv(R);
    x_hat_post = x_hat_pre;

    Xt = (T/dtdiff)*(0:(N*dtdiff));
    W = randn(N*dtdiff+1,1);
    X = lsim(sys,U + Q*W,Xt,x_0)';
    Y = [];
    hat_t = [];

    K = [];
    X_hat_pre= [];
    X_hat_post = [];
    P_pre = zeros(n,n,N-1);
    P_post = zeros(n,n,N-1);

    for i = 1:(N-1)
```

```
        u = U(i);
        t = i*T;
        x = X(:,dtdiff*i+1);%F * x + G * u + L * Q * randn;
        y = H * x + R * randn;

        p_pre = F * p_post * F' + L' * Q * L;
        k = p_pre * H' * inv(H * p_pre * H' + R);
        x_hat_pre = F * x_hat_post + G * u;
        x_hat_post = x_hat_pre + k * (y - H * x_hat_pre);
        p_post = (eye(n) - k * H) *p_pre * (eye(n) - k * H)' + k * R * k';

        Y = [Y y];
        hat_t = [hat_t t];

        K = [K k];
        X_hat_pre = [X_hat_pre x_hat_pre];
        X_hat_post = [X_hat_post x_hat_post];
        P_pre(:,:,i) = p_pre;
        P_post(:,:,i) = p_post;
    end

    P = P_post;

end
```

A.5 Problem 5

Script 10: pblm5.m

```
function [] = pblm5()
% Problem 5 Scipt/Function
% MECH 6325 - Final Probject
G_const = 6.674e-11;
M_const = 5.98e24;

% Part a --------------------------
disp('Part a ------------------------------')
syms r r_dot theta theta_dot w G M
x = [r; r_dot; theta; theta_dot];
x_dot_sym = pblm5_nonlin(x,w,G,M)

% Part b --------------------------
disp('Part b ------------------------------')
eq = 0 == subs(x_dot_sym(2),w,0)
theta_dot_sym = solve(eq,theta_dot)

% Part c --------------------------
disp('Part c ------------------------------')
syms r_0 omega_0 T
x_0 = [r_0; 0; omega_0 * T; omega_0];
omega_static = subs(theta_dot_sym,[G, M, r],[G_const, M_const, r_0]);

A = subs(jacobian(x_dot_sym,x), x, x_0)
L = jacobian(x_dot_sym,w)


r_0_const = 6.57e6;
omega_0_const = double(subs(omega_static(1), r_0,r_0_const))
v_const = omega_0_const * r_0_const % This checks orbital speed... should be
    around 7-8 km/s
A_const = double(subs(A,[omega_0, r_0, G, M],...
                      [omega_0_const, r_0_const, G_const, M_const]))
eig_A = eig(A_const)
tau = 2*pi / (imag(eig_A(2)))
min_int_step = 0.1 * tau


% Part d ----------------------------------
disp('Part d ----------------------------------')
T = 5;
sensorT = 60;
T_max = 3 * 60 * 60;
```

```
H = [1,0,0,0;
     0,0,1,0];
R = diag([100,0.1]);

x_0 = [r_0_const; 0; 0; 1.1*omega_0_const];
Q_c = 10e-6;
x_hat_0 = x_0;
P_0 = zeros(4);



[X,Xt,Y,X_hat,P,K,sensor_t] = KalmanFilter_Hybrid_nonlinsys(@pblm5_nonlin,...
                                        A_const,0,L,H,...
                                        x_0, Q_c, R,...
                                        x_hat_0, P_0,...
                                        T,T_max,sensorT);

figure()
sgtitle('Linearized Kalman Filter')
subplot(2,2,1);
hold on
plot(Xt/3600,X(1,:))
plot(sensor_t/3600,X_hat(1,:))
hold off
title('Radial Position')
xlabel('Time (Hr)')
ylabel('(m)')

subplot(2,2,2);
hold on
plot(Xt/3600,X(2,:))
plot(sensor_t/3600,X_hat(2,:))
hold off
title('Radial Velocity')
xlabel('Time (Hr)')
ylabel('(m/s)')

subplot(2,2,3);
hold on
plot(Xt/3600,X(3,:))
plot(sensor_t/3600,X_hat(3,:))
hold off
title('Angular Position')
xlabel('Time (Hr)')
ylabel('(radians)')

```

```
subplot(2,2,4);
hold on
plot(Xt/3600,X(4,:))
plot(sensor_t/3600,X_hat(4,:))
hold off
title('Angular Velocity')
xlabel('Time (Hr)')
ylabel('(rad/s)')

saveas(gcf,[pwd '/fig/pblm5_results_hybrid_lin.png'])

test_t = 0:sensorT:T_max;
test_x = interp1(Xt',X',test_t)';
test_x_hat = interp1(sensor_t',X_hat',test_t)';

x_error = test_x_hat - test_x;

figure()
plot(test_t/3600,x_error(1,:));
title(['Linearized Kalman Filter', newline,...
       'Radial Position Estimate Error'])
xlabel('Time (Hr)')
ylabel('(m)')

saveas(gcf,[pwd '/fig/pblm5_est_error_hybrid_lin.png'])

disp(['The performance of the linear Kalman Filter is reduced by the ',...
     'lack of ', newline, 'linearity in the actual system dynamics.',...
     ' This may be improved with faster', newline, 'measurments',...
     ' and Kalman Filter updates; however this is not gaurenteed.']);




% Part e ------------------------------------------------------
disp('Part e --------------------------------')
[X,Xt,Y,X_hat,P,K,sensor_t] = KalmanFilter_Extended_Hybrid(@pblm5_nonlin,...
                                        A_const,0,L,H,...
                                        x_0, Q_c, R,...
                                        x_hat_0, P_0,...
                                        T,T_max,sensorT);

figure()
sgtitle('Extended Kalman Filter')
subplot(2,2,1);
```

```
hold on
plot(Xt/3600,X(1,:))
plot(sensor_t/3600,X_hat(1,:))
hold off
title('Radial Position')
xlabel('Time (Hr)')
ylabel('(m)')

subplot(2,2,2);
hold on
plot(Xt/3600,X(2,:))
plot(sensor_t/3600,X_hat(2,:))
hold off
title('Radial Velocity')
xlabel('Time (Hr)')
ylabel('(m/s)')

subplot(2,2,3);
hold on
plot(Xt/3600,X(3,:))
plot(sensor_t/3600,X_hat(3,:))
hold off
title('Angular Position')
xlabel('Time (Hr)')
ylabel('(radians)')

subplot(2,2,4);
hold on
plot(Xt/3600,X(4,:))
plot(sensor_t/3600,X_hat(4,:))
hold off
title('Angular Velocity')
xlabel('Time (Hr)')
ylabel('(rad/s)')

saveas(gcf,[pwd '/fig/pblm5_results_hybrid_EKF.png'])

test_t = 0:sensorT:T_max;
test_x = interp1(Xt',X',test_t)';
test_x_hat = interp1(sensor_t',X_hat',test_t)';

x_error = test_x_hat - test_x;

figure()
plot(test_t/3600,x_error(1,:));
title(['Extended Kalman Filter', newline,...
```

```
       'Radial Position Estimate Error'])
xlabel('Time (Hr)')
ylabel('(m)')

saveas(gcf,[pwd '/fig/pblm5_est_error_hybrid_EKF.png'])

disp(['As is clear between the two estimate error plots, the Extended ',...
     'Kalman Filter ', newline, 'has a lot less error then the Linear one.',...
     'This is very evident from the order ', newline,...
     'of magnitude on each of the plots.'])

assignin('base','X',X)



end
```

Script 11: pblm5_nonlin.m

```
function x_dot = pblm5_nonlin(x,w,G,M)
    arguments
        x (4,1)
        w = 10e-6 * randn;
        G = 6.674e-11;
        M = 5.98e24;
    end
    x_dot = x; % to get same array type
    x_dot(1) = x(2);
    x_dot(2) = x(1) * x(4)^2 - (G*M)/(x(1)^2) + w;
    x_dot(3) = x(4);
    x_dot(4) = -2*x(4)*x(2)/x(1);
end
```

Script 12: KalmanFilter_Hybrid_nonlinsys.m

```
function [X,Xt,Y,X_hat,P,K,sensor_t] = KalmanFilter_Hybrid_nonlinsys2(...
                                        nonlinsys,A,B,L,H,...
                                        x_0,Q_c,R,...
                                        x_hat_0,P_0,...
                                        T,T_max,sensorT,n,U)
arguments
    nonlinsys
    A double
    B double
    L double
    H double
    x_0 double
    Q_c double
    R double
    x_hat_0 double
    P_0 double
    T double = 0.1;
    T_max double = 500;
    sensorT double = 0.5;
    n = size(A,1);
    U = zeros(ceil(T_max/T),1); %This code assuems U = 0... so not needed
end

% Empty Arrays
N = floor(T_max/sensorT);
X = [];
Y = [];
sensor_t = [];
K = [];
X_hat_pre = [];
X_hat_post = [];
P_pre = zeros(n,n,N);
P_post = zeros(n,n,N);


% For Loop Initiation
x = x_0;

p = P_0;
x_hat = x_hat_0;
k = p * H' * inv(R);

for i = 1:N
    % Actual System and Measurment
    t = i * sensorT;
```

```
    x_dot = nonlinsys(x);
    x = x + x_dot * sensorT;
    y = H * x + R * randn(size(R,2),1);

    X = [X x];
    Y = [Y y];
    sensor_t = [sensor_t t];

    % Time Update
    x_hat_dot = A * x_hat;
    x_hat = x_hat + x_hat_dot * sensorT;
    p_dot = A * p + p * A' + L*Q_c*L';
    p = p + p_dot * sensorT;

    X_hat_pre = [X_hat_pre x_hat];
    P_pre(:,:,i) = p;

    % Measurment Update
    k = p*H'*inv(H*p*H' + R);
    x_hat = x_hat + k * (y - H * x_hat);
    p = (eye(n)-k*H)*p*(eye(n)-k*H)' + k*R*k';

    K = [K k];
    X_hat_post = [X_hat_post x_hat];
    P_post(:,:,i) = p;
end

Xt = sensor_t;

X_hat = X_hat_post;
P = P_post;
end
```

Script 13: KalmanFilter_Extended_Hybrid.m

```
function [X,Xt,Y,X_hat,P,K,sensor_t] = KalmanFilter_Extended_Hybrid(...
                                        nonlinsys,A,B,L,H,...
                                        x_0,Q_c,R,...
                                        x_hat_0,P_0,...
                                        T,T_max,sensorT,n,U)
arguments
    nonlinsys
    A double
    B double
    L double
    H double
    x_0 double
    Q_c double
    R double
    x_hat_0 double
    P_0 double
    T double = 0.1;
    T_max double = 500;
    sensorT double = 0.5;
    n = size(A,1);
    U = zeros(ceil(T_max/T),1); %This code assuems U = 0... so not needed
end



% Empty Arrays
N = floor(T_max/sensorT);
X = [];
Y = [];
sensor_t = [];
K = [];
X_hat_pre = [];
X_hat_post = [];
P_pre = zeros(n,n,N);
P_post = zeros(n,n,N);


% For Loop Initiation
x = x_0;

p = P_0;
x_hat = x_hat_0;
k = p * H' * inv(R);

for i = 1:N
```

```
    % Actual System and Measurment
    t = i * sensorT;
% x = interp1(Xt,X,t)';

    x_dot = nonlinsys(x);
    x = x + x_dot * sensorT;

    y = H * x + R * randn(size(R,2),1);

    X = [X x];
    Y = [Y y];
    sensor_t = [sensor_t t];

    % Non-lin Time Update
    x_hat_dot = nonlinsys(x_hat,0);
    x_hat = x_hat + x_hat_dot * sensorT;
    p_dot = A * p + p * A' + L*Q_c*L';
    p = p + p_dot * sensorT;

    X_hat_pre = [X_hat_pre x_hat];
    P_pre(:,:,i) = p;

    % Measurment Update
    k = p*H'*inv(H*p*H' + R);
    x_hat = x_hat + k * (y - H * x_hat);
    p = (eye(n)-k*H)*p*(eye(n)-k*H)' + k*R*k';

    K = [K k];
    X_hat_post = [X_hat_post x_hat];
    P_post(:,:,i) = p;
end


% X = X';
% Xt = Xt';
Xt = sensor_t;

X_hat = X_hat_post;
P = P_post;
end
```

B MATLAB Output

```
% MECH 6325 - Final Project
%Jonas Wagner
%2020-11-28

clear
close all;

% Problem 1
 ---------------------------------------------------------------
disp('Problem 1
 ---------------------------------------------------------------')
pblm1()

clear
% Problem 2
 ---------------------------------------------------------------
disp('Problem 2
 ---------------------------------------------------------------')
pblm2()

clear
% Problem 3
 ---------------------------------------------------------------
disp('Problem 3
 ---------------------------------------------------------------')
pblm3()

clear
% Probelm 4
 ---------------------------------------------------------------
disp('Problem 4
 ---------------------------------------------------------------')
pblm4()

clear
% Problem 5
 ---------------------------------------------------------------
disp('Problem 5
 ---------------------------------------------------------------')
pblm5()

clear
close all %Comment this out if figures are wanted...
```

```
Problem 1 ---------------------------------------------------------
If using the following F, there is a descrepancy for state 2

F =

    0.5000    2.0000
         0         0
```

```
P_minus_inf =

   52.0974   -0.0000
   -0.0000   10.0000


k_inf =

    0.8390
   -0.0000


P_plus_inf =

    8.3896   -0.0000
   -0.0000   10.0000


P_plus_inf_std =

   2.8965 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   3.1623 + 0.0000i


X_hat_error_std1 =

    9.3206   11.1483

There is a decrepancy for the secound state.
This is belived to be becouse of the constant input not = 0,
so the steady-state Kalman filter for that state is not as accurate.
------------------------------------------------------------
The alternative method of simulation:

F =

    0.5000    2.0000
         0    1.0000


G =

     0
     1


P_minus_inf =

   58.1889   26.1130
   26.1130   22.0991


k_inf =
```

```
    0.8533
    0.3830


P_plus_inf =

    8.5335    3.8295
    3.8295   12.0991


P_plus_inf_std =

    2.9212    1.9569
    1.9569    3.4784


X_hat_error_std2 =

    8.8327   10.9833

The alternative method has a descrepancy for the food term as well
Problem 2 ------------------------------------------------------------
Discretized System:

F =

    0.9930    0.0997
   -0.0997    0.9950


G =

    0.0060
    0.0098


H =

     1     0
     0     1


Q =

    0.9000

Standard diviations of errors

Y_error_std =

    0.8822


X_hat_error_std =
```

```
    0.1330

Problem 3 ------------------------------------------------------------
Part a -----------------

P_ss_a =

    2.3868    0.2774
    0.2774    4.2188


P_ss_a_care =

    2.3868    0.2774
    0.2774    4.2188

Part b -----------------

P_ss_b =

    0.2899    0.5798
    0.5798    1.1596


P_ss_b_care =

    0.2899    0.5798
    0.5798    1.1596

Part c -----------------

P_ss_c =

    2.2899    0.5798
    0.5798    3.1596


P_ss_c_care =

    2.2899    0.5798
    0.5798    3.1596

Part d -----------------

P_ss_d =

    0.6899    1.3798
    1.3798    2.7596


P_ss_d_care =

    2.2899    0.5798
```

```
    0.5798    3.1596


K_int =

    0.6899    1.3798
    1.3798    2.7596


A_KC_int =

    0.3101   -1.3798
   -1.3798   -1.7596


eig_int =

   -2.4495
    1.0000

This is an unstable observer, so it does not result in asteady-state
 Kalman Filter
Problem 4 ----------------------------------------------------------

sys =

  A =
         x1     x2
   x1     0      1
   x2   -36  -1.92

  B =
      u1
   x1  0
   x2  1

  C =
      x1  x2
   y1  1   0
   y2  0   1

  D =
      u1
   y1  0
   y2  0

Continuous-time state-space model.


Q =

    0.0050
```

```
dt_sys =

  A =
           x1       x2
   x1  -0.5908  0.01873
   x2  -0.6743  -0.6267

  B =
           u1
   x1  0.04419
   x2  0.01873

  C =
       x1  x2
   y1   1   0

  D =
       u1
   y1   0

Sample time: 0.5 seconds
Discrete-time state-space model.

Problem 5 -------------------------------------------------------
Part a ----------------------------

x_dot_sym =

                        r_dot
w + r*theta_dot^2 - (G*M)/r^2
                    theta_dot
       -(2*r_dot*theta_dot)/r

Part b --------------------------

eq =

0 == r*theta_dot^2 - (G*M)/r^2


theta_dot_sym =

 (G^(1/2)*M^(1/2))/r^(3/2)
-(G^(1/2)*M^(1/2))/r^(3/2)

Part c --------------------------

A =

[                        0,                 1, 0,              0]
[omega_0^2 + (2*G*M)/r_0^3,                 0, 0, 2*omega_0*r_0]
[                        0,                 0, 0,              1]
[                        0, -(2*omega_0)/r_0, 0,              0]
```

```
L =

0
1
0
0


omega_0_const =

    0.0012


v_const =

   7.7940e+03


A_const =

   1.0e+04 *

         0    0.0001         0         0
    0.0000         0         0    1.5588
         0         0         0    0.0001
         0   -0.0000         0         0


eig_A =

   0.0000 + 0.0000i
   0.0000 + 0.0012i
   0.0000 - 0.0012i
   0.0000 + 0.0000i


tau =

   5.2964e+03


min_int_step =

  529.6442

Part d ---------------------------------
The performance of the linear Kalman Filter is reduced by the lack of
linearity in the actual system dynamics. This may be improved with
 faster
measurments and Kalman Filter updates; however this is not gaurenteed.
Part e ---------------------------------
As is clear between the two estimate error plots, the Extended Kalman
 Filter
```

```
has a lot less error then the Linear one.This is very evident from the
 order
of magnitude on each of the plots.
```

Published with MATLAB® R2020b

C Hand Calculations

1) P_k — Wombat Population
f_k — Wombat Food supply

$$P_{k+1} = \frac{1}{2}P_k + 2f_k$$
$$f_{k+1} = f_0 + \tilde{w}$$
$$y = P_k + v$$

$P_0 = 650$
$f_0 = 250$
$\tilde{w} \sim (0, 10)$
$v \sim (0, 10)$
$\hat{P}_0 = 600$ $\quad E[(\hat{P}_0 - P_0)^2] = 500$
$\hat{f}_0 = 200$ $\quad E[(\hat{f}_0 - f_0)^2] = 200$

Let, $x_k = \begin{bmatrix} P_k \\ f_k \end{bmatrix}$ $\quad u_k = 0$

$$x_{k+1} = \begin{bmatrix} \frac{1}{2} & 2 \\ 0 & 0 \end{bmatrix} x_k + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u_k + \begin{bmatrix} 0 \\ 1 \end{bmatrix} w$$

$$y_{k+1} = \begin{bmatrix} 1 & 0 \end{bmatrix} x_k + v$$

$$F = \begin{bmatrix} \frac{1}{2} & 2 \\ 0 & 0 \end{bmatrix} \quad G = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad L = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$H = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

$$\hat{x}_0 = \begin{bmatrix} 600 \\ 200 \end{bmatrix} \quad P_0 = \begin{bmatrix} 500 & 0 \\ 0 & 200 \end{bmatrix}$$

a) standard KF Process

MECH 6326 Final Project Jonas Wagner 2020-11-20

1)

b) $P_\infty = F P_\infty F^T - F P H^T (H P H^T + R)^{-1} H P F^T + Q$

$P_\infty = DARE(F^T, H^T, Q, R, 0)$

c) $M = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$

wrong. $M \neq 0$

b) $P_\infty = DARE(F^T, H^T, Q, HM + M^T H^T + R, FM)$

$M = 0$

c) $P_\infty^- = idare(F^T, H^T, Q, R)$

b)

$K_\infty = P_\infty^- H^T (H P_\infty^- H^T + R)^{-1}$

$P_\infty^+ = (I - K_\infty H) P_\infty^-$

$\hat{x}_k^+ = (I - K_\infty H) F \hat{x}_{k-1}^+ + K_\infty (H x_k + v)$

$\hat{x}_k^- = F \hat{x}_{k-1}^+$

$\hat{x}_k^+ = F \hat{x}_{k-1}^+ - K_\infty H F \hat{x}_{k-1}^+ + K_\infty H x_k + K_\infty v$

ss - $\hat{x}_k^+ = \hat{x}_k^-$

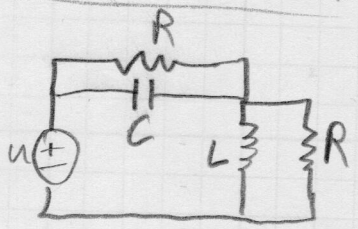
$K_\infty H (\hat{x}_k - x_k) = K_\infty v$

$\hat{x}_k - x_k = H^{-1} v$

$(0, H^{-1} R)$

EX. 1.8 in book...

a)



$$\dot{v}_c = \frac{-2}{RC} v_c + \frac{1}{C} i_L + \frac{1}{RC} u$$

$$\dot{i}_L = \frac{-1}{L} v_c + 0 + \frac{1}{L} u$$

$$\dot{x} = \begin{bmatrix} v_c \\ i_L \end{bmatrix} \qquad \dot{x} = Ax + Lw \quad (u = w)$$

$$A = \begin{bmatrix} \frac{-2}{RC} & \frac{1}{C} \\ \frac{-1}{L} & 0 \end{bmatrix} \qquad L = \begin{bmatrix} \frac{1}{RC} \\ \frac{1}{L} \end{bmatrix} \qquad w \sim (0, Q_c)$$

$Q_c = 3^2$

$$y(t_k) = H x(t_k) + v$$

$$H = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad v \sim (0, R), \quad R = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

2) cont.

Let, $F = e^{A\Delta t}$ $\quad \Delta t = 0.1$

$Q = Q_c \Delta t$

$x_0 = 0$ $\quad \tilde{w} = L\tilde{w}$

1) $x_{k+1} = F x_k + \tilde{w}$ — $\tilde{w} \sim (0, \tilde{Q})$

$y_k = H x_k + v$ — $v \sim (0, R)$

$R = I_2 I_2$

2) $\hat{x}_0^+ = 0$

$P_0^+ = 0.1\, I_4$

3) $P_k^- = F P_{k-1}^+ F^T + L Q L^T$

$\hat{x}_k^- = F \hat{x}_{k-1}^+ + \cancel{G u}^0$

4) For each sensor, $(i = 1, 2)$

$K_{ik} = \dfrac{P_{i-1,k}^+ H_{ik}^T}{H_{ik} P_{i-1,k}^+ H_{ik}^T + R_{ik}} = \dfrac{P_{ik}^+ H_{ik}^T}{R_{ik}}$

$\hat{x}_{ik}^+ = \hat{x}_{i-1}^+ + K_{ik}(y_{ik} - H_{ik} \hat{x}_{i-1,k}^+)$

$P_{ik}^+ = (I - K_{ik} H_{ik}) P_{i-1,k}^+ (I - K_{ik} H_{ik})^T + K_{ik} R_{ik} K_{ik}^T$

$\hat{x}_k^+ = \hat{x}_{rk}^+$

$\hat{P}_k^+ = P_{rk}^+$

b) Trace: just Plot each std: error between for each

3) Riccati Equation Value comparisons.

$\dot{x} = Ax + w$] $\sim A = \begin{bmatrix} a_1 & 0 \\ 0 & a_2 \end{bmatrix}$ $w \sim (0, \begin{bmatrix} q_{11} & q_{12} \\ q_{12} & q_{22} \end{bmatrix})$

$y = Hx + V$] $H = I_2$, $V \sim (0, I_2)$

a) $A = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$ $Q = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$ $P(0) = I$

b) $A = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}$ $Q = \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix}$ $P(0) = I$

c) $A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ $Q = \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix}$ $P(0) = I$

d) $A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ $Q = \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix}$ $P(0) = 0$

Extra Part ☆

Does it result in a ss-KF? ← Pg. 254

No, with the P_∞ from integration this isn't the case.

Check observer Poles for stability

$$P_\infty = \begin{bmatrix} 0.69 & 1.38 \\ 1.38 & 2.76 \end{bmatrix}$$

$$K_\infty = P_\infty C^T R^{-1} = \begin{bmatrix} 0.69 & 1.37 \\ 1.38 & 2.76 \end{bmatrix}$$

Not stable

$$(A - KC) = \begin{bmatrix} 0.31 & -1.38 \\ -1.38 & -1.76 \end{bmatrix} \rightarrow \lambda(A-KC) = (-2.5, 1)$$ ✓

4) $\dot{x} = \begin{bmatrix} 0 & 1 \\ -\omega^2 & -2\xi\omega \end{bmatrix} x + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tilde{w}(t)$ — A

$\omega = 6$ rad/s, $\xi = 0.016$

$y(t_k) = [1 \;\; 0]\, x + v(t_k)$ — $\tilde{w}(t) \sim (0, Q_c)$, $Q_c = 0.01$

$v(t_k) \sim (0, 10^{-4})$

$x(0) = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

$\hat{x}(0) = x(0)$ $\quad P(0) = \begin{bmatrix} 10^{-5} & 0 \\ 0 & 10^{-2} \end{bmatrix}$

a) $F = e^{A\Delta t}$ $\quad \Delta t = 0.5$ s

$x_{k+1} = F x_k + w$ — $w \sim (0, \underbrace{\begin{bmatrix} 0 & 0 \\ 0 & Q_c \Delta t \end{bmatrix}}_{Q})$

$y_k = H_k + V$ — $V \sim (0, 10^{-4})$ $\quad Q$

b) Standard OT KF

- Plot $E[(x-\hat{x})(x-\hat{x})^T]$ over time
- Compare states + Estimates

5)

$$\ddot{r} = r\dot{\theta}^2 - \frac{GM}{r^2} + \ddot{w}$$

$$\ddot{\theta} = -\frac{2\dot{\theta}\dot{r}}{r}$$

$G = 6.674 \times 10^{-11} \frac{m^3}{kg\,s^2}$

$M = 5.98 \times 10^{24}\ kg$

$\ddot{w} \sim (0, 10^{-6})$

a)

$$x = \begin{bmatrix} r \\ \dot{r} \\ \theta \\ \dot{\theta} \end{bmatrix}$$

$$\dot{x} = \begin{bmatrix} \dot{r} \\ r\dot{\theta}^2 - \frac{GM}{r^2} + \ddot{w} \\ \dot{\theta} \\ -\frac{2\dot{\theta}\dot{r}}{r} \end{bmatrix}$$

$$= \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} x + \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \ddot{w}$$

b) $\ddot{w} = 0$

$$0 = r\dot{\theta}^2 - \frac{GM}{r^2}$$

$$\frac{GM}{r^2} = r\dot{\theta}^2$$

$$\boxed{\dot{\theta} = \pm\sqrt{\frac{GM}{r^3}}}$$

5) cont.

c) $X_0 = \begin{bmatrix} r_0 \\ 0 \\ \omega_0 T \\ \omega_0 \end{bmatrix}$

$$A = \left.\frac{df}{dx}\right|_{X_0} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \dot{\theta}_0^2 + \frac{2GM}{r_0^3} & 0 & 0 & 2 r_0 \dot{\theta}_0 \\ 0 & 0 & 0 & 1 \\ 2\dot{\theta}_0 \dot{r}_0^{\nearrow 0} & -\frac{2\dot{\theta}_0}{r_0} & 0 & -\frac{2\dot{r}_0^{\nearrow 0}}{r_0} \end{bmatrix}$$

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \omega_0^2 + \frac{2GM}{r_0^3} & 0 & 0 & 2 r_0 \omega_0 \\ 0 & 0 & 0 & 1 \\ 0 & -\frac{2\omega_0}{r_0} & 0 & 0 \end{bmatrix} \qquad B = 0$$

$Lw \rightarrow \hat{w} \sim (0, \hat{Q})$

$\hat{Q} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 10^{-6} & 0 \\ 0 & 0 & 0 \end{bmatrix}$

~~λ~~ for $r_0 = 6.57 \times 10^6$ m:

$$\begin{bmatrix} 0 \\ 0 + j1.2e^{-3} \\ 0 - j1.2e^{-3} \\ 0 \end{bmatrix} \qquad 2\pi f_0 = 1.2 \times 10^{-3} = \frac{2\pi}{\tau}$$

Largest Integration step:

$$\tau = \frac{2\pi}{1.2 \times 10^{-3}} \approx 5.2 \times 10^3$$

$$\boxed{0.1\tau = 520\,s}$$

5) cont.

d) $\Delta t = 1$ min

$\sigma_r = 100$ m, $\sigma_\theta = 0.1$ rad

$y = [I_{2\times 2}]x + v$, $R = \begin{bmatrix} 100^2 & 0 \\ 0 & 0.1^2 \end{bmatrix}$

$\begin{bmatrix} (100)^2 & 0 \\ 0 & (0.1)^2 \end{bmatrix}$

$$\hat{x}(0) = x(0) = \begin{bmatrix} r_0 \\ \dot{r}_0 \\ \theta_0 \\ 1.1\,\omega_0 \end{bmatrix}$$

$P(0) = \text{diag}(0,0,0,0)$

Linearized CT-KF. should be hybrid

$\hat{r}$-r plot ... Modifications to improve?

e) EKF:

2) A ~ same

$$L = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

$$\hat{x}_0^+ = \begin{bmatrix} r_0 \\ 0 \\ \theta_0 \\ 1.1\,\omega_0 \end{bmatrix}$$

$P_0^+ = \text{diag}(0,0,0,0)$

3) $Q = \text{diag}(0, 10^{-6}, 0, 0)$

3a) Integrate steps:

$\dot{\hat{x}} = f(\hat{x}, u, 0, t)$

$\dot{P} = AP + PA^T + LQL^T$

b) Standard KF measurment update

- Compare with LT-KF