

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	3
1.1 Modeling Method 1	4
1.2 Modeling Method 2	5
1.3 Discrete-Time Kalman Filter	6
1.4 Theoretical Error Comparison	7
1.4.1 Modeling Method 1:	7
1.4.2 Modeling Method 2:	7
2 Problem 2 - Electrical Network	8
2.1 System Modeling	8
2.2 Discrete-Time Modeling	9
2.3 Sequential Kalman Filter	10
2.4 Simulation Results	11
2.5 Capacitor Voltage Comparison	11
3 Problem 3 - Steady State Riccati Equations	13
3.1 Integrating the Continuous-Time Algebraic Riccati Equation	13
3.2 Results and <code>icare()</code> function comparison	14
3.2.1 Part a	14
3.2.2 Part b	14
3.2.3 Part c	14
3.2.4 Part d	15
3.3 Testing Viability of Steady-State Kalman Filter	15
4 Problem 4 - Second Order Sampled Data System	16
4.1 Discrete-Time System	16
4.2 Sampled Data System Discrete-Time Kalman Filter	16
4.3 Simulation Results	17
5 Problem 5 - Satellite Orbit	19
5.1 Nonlinear State-Space Model	19
5.1.1 <code>pblm5_nonlin</code> Function	19
5.2 Steady-State Radial Velocity	20
5.3 Continuous-Time Linear Time-Invariant Model	20
5.4 Linear Hybrid Kalman Filter	22
5.5 Hybrid Extended Kalman Filter	23
5.6 Simulation Results	24
5.6.1 Linear Hybrid Kalman Filter	24
5.6.2 Hybrid Extended Kalman Filter	26
A MATLAB Code	28
A.1 Problem 1	29
A.2 Problem 2	35
A.3 Problem 3	40
A.4 Problem 4	42
A.5 Problem 5	46
B MATLAB Output	55
C Hand Calculations	64

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:

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

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:

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

where x_k is defined as:

$$x_k = \begin{bmatrix} p_k \\ f_k \end{bmatrix} \quad (6)$$

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:

$$F = \begin{bmatrix} \frac{1}{2} & 2 \\ 0 & 0 \end{bmatrix} \quad G = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad L = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad (7)$$

$$H = [1 \ 0] \quad u_k \equiv f_0, \forall k$$

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.

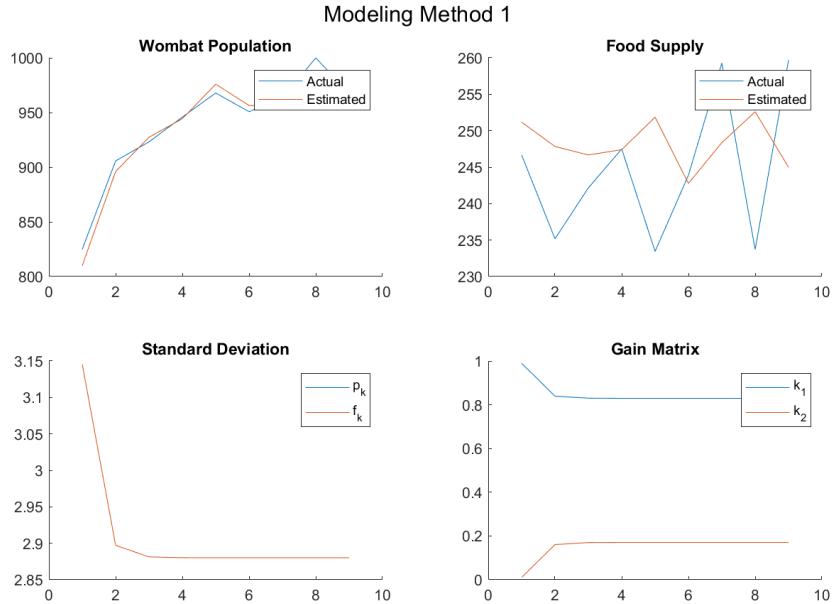


Fig. 1: Simulation results for problem 1 using modeling method 1

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:

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

$$H = [1 \quad 0]$$

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.

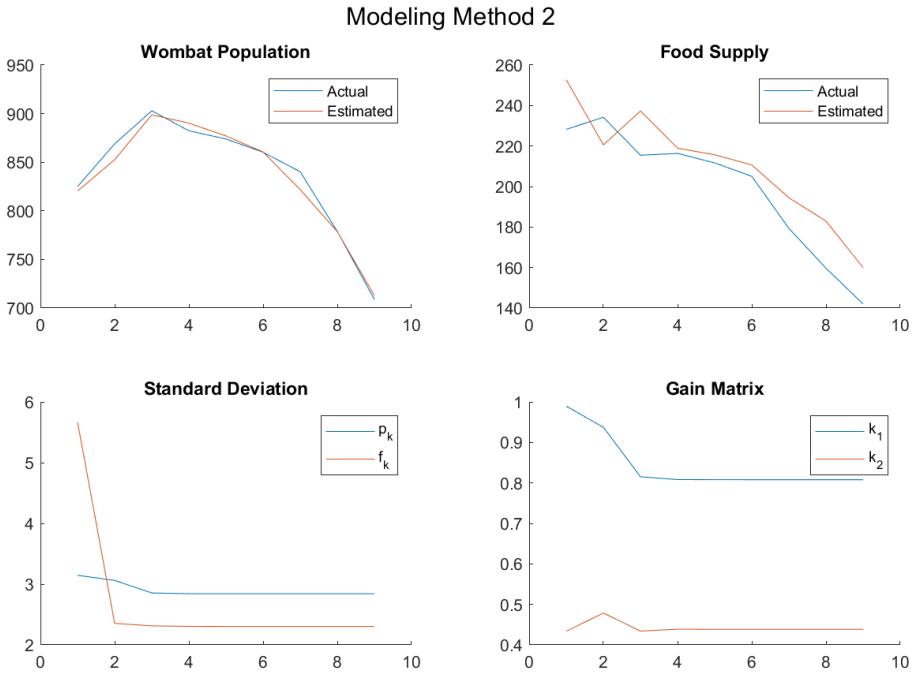


Fig. 2: Simulation results for problem 1 using modeling method 2

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-Invariant 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:

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

- Local variables for each iteration: x, y, p_pre, p_post, u, x_hat_pre, k, x_hat_post

```
1 x = x_0;
2 y = H * x + R * randn;
3 p_pre = P_0;
4 p_post = p_pre;
5
6 u = U(1);
7 x_hat_pre = x_hat_0;
8 k = p_pre * H' * inv(R);
9 x_hat_post = x_hat_pre;
```

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

- Simulate the system for one step:

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

- Perform the standard Discrete-Time Kalman Filter update equations:

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

- 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):

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

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

$$K_\infty = (P_\infty H^T + M)(H P_\infty H^T + H M + M^T H^T + R)^{-1} \quad (10)$$

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

$$P_\infty^+ = (I - K_\infty H) P_\infty^- \quad (11)$$

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:

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

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:

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

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 than the theoretical one. This could be due to the fact that the expected value for the food-supply never deviates and is more predictable than if an input were included.

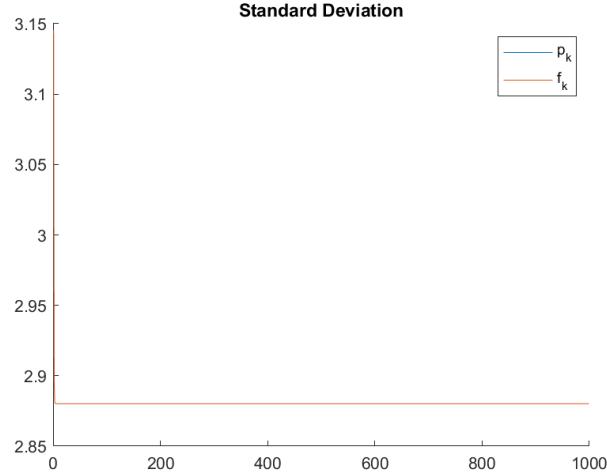


Fig. 3: Experimental estimation error standard deviation for problem 1 using modeling method 1

2 Problem 2 - Electrical Network

Given the electrical network seen in Fig. 5, a Continuous-Time Linear Time-Invariant system can be developed that models the network. This can then be discretized and a sequential Kalman Filter can 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-Invariant form:

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

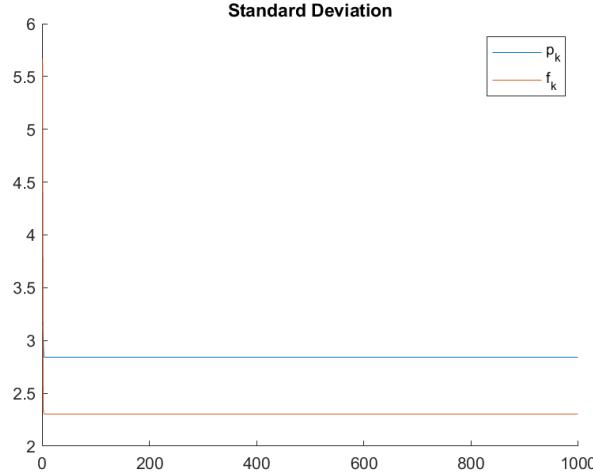


Fig. 4: Experimental estimation error standard deviation for problem 1 using modeling method 1

where A and B are defined as:

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

The input is set to the process noise term as:

$$u \equiv w \sim (0, Q_c) \quad (18)$$

where $Q_c = 9$.

2.2 Discrete-Time Modeling

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

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

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

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

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

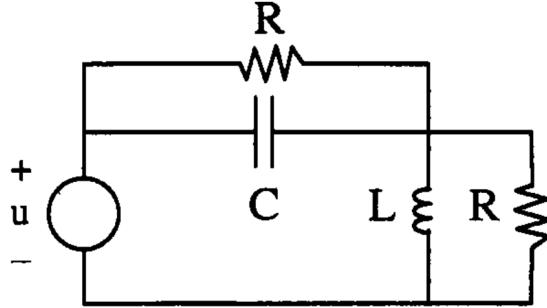


Fig. 5: Electrical network modeling in problem 2.

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

$$F = \begin{bmatrix} 0.9930 & 0.0997 \\ -0.0997 & 0.9950 \end{bmatrix} \quad G = \begin{bmatrix} 0.0060 \\ 0.0998 \end{bmatrix}$$

$$H = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad (21)$$

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-Invariant 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

```

1 x = x_0;
2 y = H * x + R * randn;
3 p_pre = P_0;
4 p_post = p_pre;
5
6 u = U(1);
7 x_hat_pre = x_hat_0;
8 k = p_pre * H' * inv(R);
9 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:

```

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

```

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

```

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

```

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

```

1 k(:,1) = p_pre * H(1,:) * inv(H(1,:)*p_pre * H(1,:)' + R(1,1));
2 x_hat(:,1) = x_hat_pre + k(:,1)*(y(1) - H(1,:)*x_hat_pre);
3 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`.

```

1 k(:,j) = p(:,:,j-1) * H(j,:)' * inv(H(j,:)*p(:,:,j-1) * H(j,:)' + R(j,
j));
2 x_hat(:,j) = x_hat(:,j-1) + k(:,j-1) * (y(j) - H(j,:)*x_hat(:,j-1));
3 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.

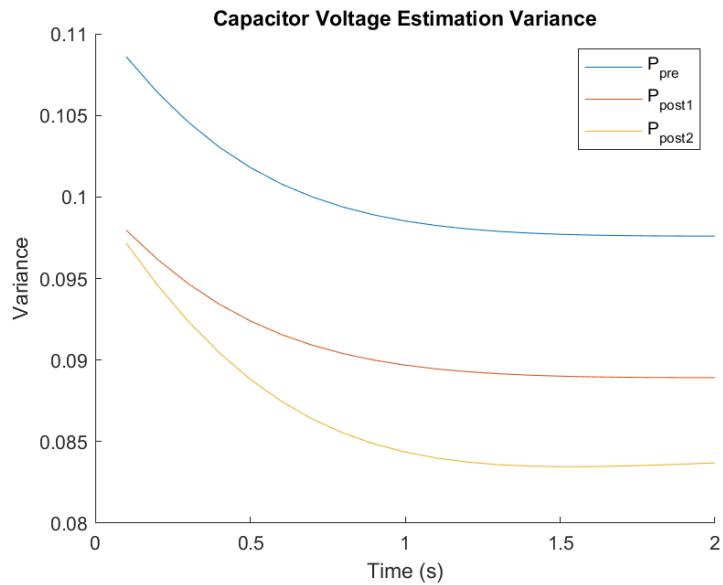


Fig. 6: Capacitor Voltage Estimation Variance for Problem 2

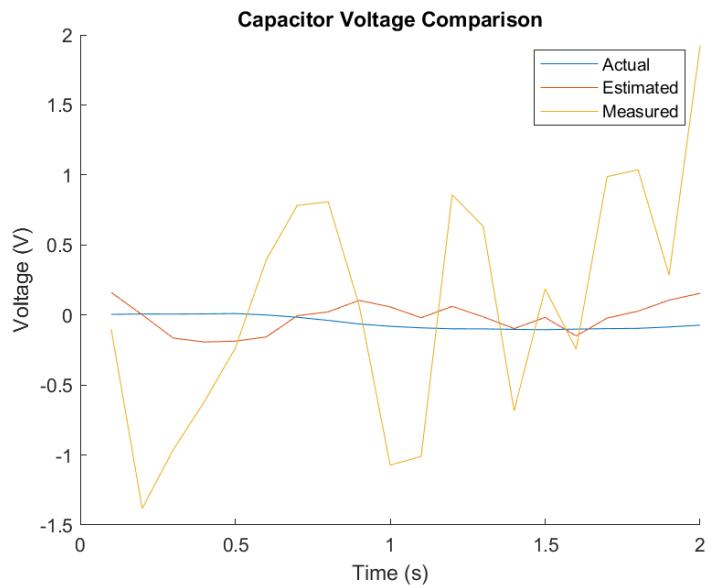


Fig. 7: Capacitor Voltage Comparison for Problem 2

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:

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

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

- (a) Calculate for each time step:

```
1 p_dot = - p * C' * inv(R) * C * p + A * p + p * A' + Q;
2 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,

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

Solving the CARE through numerical integration resulted in the following:

$$P_\infty = \begin{bmatrix} 2.3868 & 0.2774 \\ 0.2774 & 4.2188 \end{bmatrix} \quad (25)$$

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

$$P_\infty = \begin{bmatrix} 2.3868 & 0.2774 \\ 0.2774 & 4.2188 \end{bmatrix} \quad (26)$$

As is evident, there is no discrepancy.

3.2.2 Part b

Let,

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

Solving the CARE through numerical integration resulted in the following:

$$P_\infty = \begin{bmatrix} 0.2899 & 0.5798 \\ 0.5798 & 1.1596 \end{bmatrix} \quad (28)$$

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

$$P_\infty = \begin{bmatrix} 0.2899 & 0.5798 \\ 0.5798 & 1.1596 \end{bmatrix} \quad (29)$$

As is evident, there is no discrepancy.

3.2.3 Part c

Let,

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

Solving the CARE through numerical integration resulted in the following:

$$P_\infty = \begin{bmatrix} 2.2899 & 0.5798 \\ 0.5798 & 3.1596 \end{bmatrix} \quad (31)$$

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

$$P_\infty = \begin{bmatrix} 2.2899 & 0.5798 \\ 0.5798 & 3.1596 \end{bmatrix} \quad (32)$$

As is evident, there is no discrepancy.

3.2.4 Part d

Let,

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

Solving the CARE through numerical integration resulted in the following:

$$P_\infty = \begin{bmatrix} 0.6899 & 1.3798 \\ 1.3798 & 2.7596 \end{bmatrix} \quad (34)$$

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

$$P_\infty = \begin{bmatrix} 2.2899 & 0.5798 \\ 0.5798 & 3.1596 \end{bmatrix} \quad (35)$$

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:

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

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

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

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-Invariant system is given in standard form:

$$\dot{x} = Ax + Lw \quad (38)$$

$$A = \begin{bmatrix} 0 & 1 \\ -\omega^2 & -2\zeta\omega \end{bmatrix}, \quad L = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad w \sim (0, Q_c)$$

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:

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

4.1 Discrete-Time System

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

$$F = \begin{bmatrix} -0.5908 & 0.01873 \\ -0.6743 & -0.6267 \end{bmatrix} \quad L = \begin{bmatrix} 0.04419 \\ 0.0187 \end{bmatrix} \quad (40)$$

$$H = [1 \ 0], \quad Q = 0.005$$

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-Invariant 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-Invariant 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

```

1 x = x_0;
2 y = H * x + R * randn;
3 p_pre = P_0;
4 p_post = p_pre;
5
6 x_hat_pre = x_hat_0;
7 k = p_pre * H' * inv(R);
8 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:

```

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

```

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

```

1 p_pre = F * p_post * F' + L' * Q * L;
2 k = p_pre * H' * inv(H * p_pre * H' + R);
3 x_hat_pre = F * x_hat_post + G * u;
4 x_hat_post = x_hat_pre + k * (y - H * x_hat_pre);
5 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 Filter states 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.

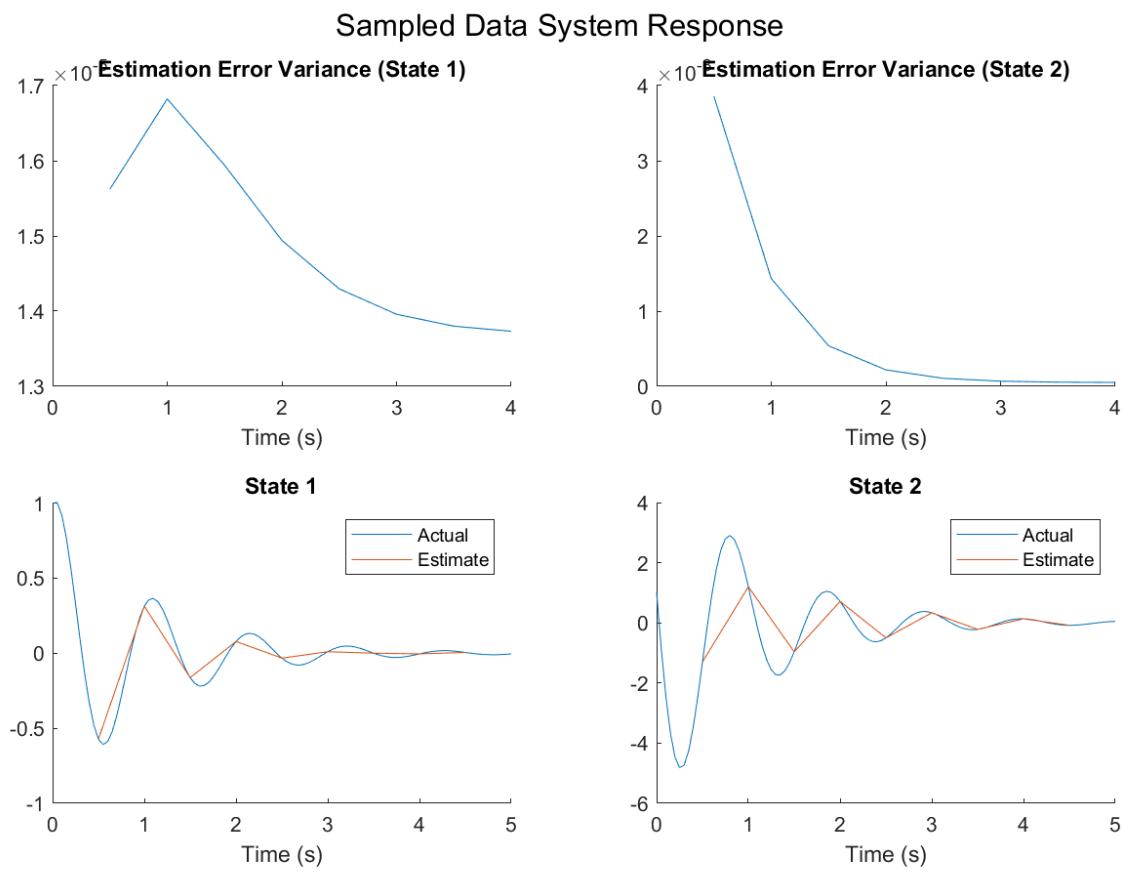


Fig. 8: Simulation Results for problem 4

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 satellite is given as:

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

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,

$$x = \begin{bmatrix} r \\ \dot{r} \\ \theta \\ \dot{\theta} \end{bmatrix} \quad (42)$$

The state space model can then be defined as:

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

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:

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

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

$$\frac{GM}{r^2} = r\dot{\theta}^2 \quad (45)$$

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

5.3 Continuous-Time Linear Time-Invariant Model

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

Let,

$$x_0 = \begin{bmatrix} r_0 \\ 0 \\ \omega_0 T \\ \omega_0 \end{bmatrix} \quad (47)$$

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

$$A = \left. \frac{df}{dx} \right|_{x_0} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \dot{\theta}^2 + \frac{GM}{r^3} & 0 & 0 & 2r\dot{\theta} \\ 0 & 0 & 0 & 1 \\ 2\dot{\theta}\ddot{r} & \frac{-2\dot{\theta}}{r} & 0 & \frac{-2\ddot{r}}{r} \end{bmatrix}_{x_0}$$

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \omega_0^2 + \frac{GM}{r_0^3} & 0 & 0 & 2r_0\omega_0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{-2\omega_0}{r_0} & 0 & 0 \end{bmatrix} \quad (48)$$

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

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

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

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

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

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

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

$$0.1\tau = 520[s] \quad (52)$$

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-Invariant 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)

```

1 x = x_0;
2 p = P_0;
3 x_hat = x_hat_0;
4 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:

```

1 t = i * sensorT;
2 x_dot = nonlinsys(x);
3 x = x + x_dot * sensorT;
4 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:

```

1 x_hat_dot = A * x_hat;
2 x_hat = x_hat + x_hat_dot * sensorT;
3 p_dot = A * p + p * A' + L*Q_c*L';
4 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 measurement update equations:

```

1 k = p*H'*inv(H*p*H' + R);
2 x_hat = x_hat + k * (y - H * x_hat);
3 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 Filter states 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-Invariant 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)


```

1 x = x_0;
2 p = P_0;
3 x_hat = x_hat_0;
4 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:


```

1 t = i * sensorT;
2 x_dot = nonlinsys(x);
3 x = x + x_dot * sensorT;
4 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:


```

1 x_hat_dot = nonlinsys(x_hat,0);
2 x_hat = x_hat + x_hat_dot * sensorT;
3 p_dot = A * p + p * A' + L*Q_c*L';
4 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 measurement update equations:


```

1 k = p*H'*inv(H*p*H' + R);
2 x_hat = x_hat + k * (y - H * x_hat);
3 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 Filter states 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.

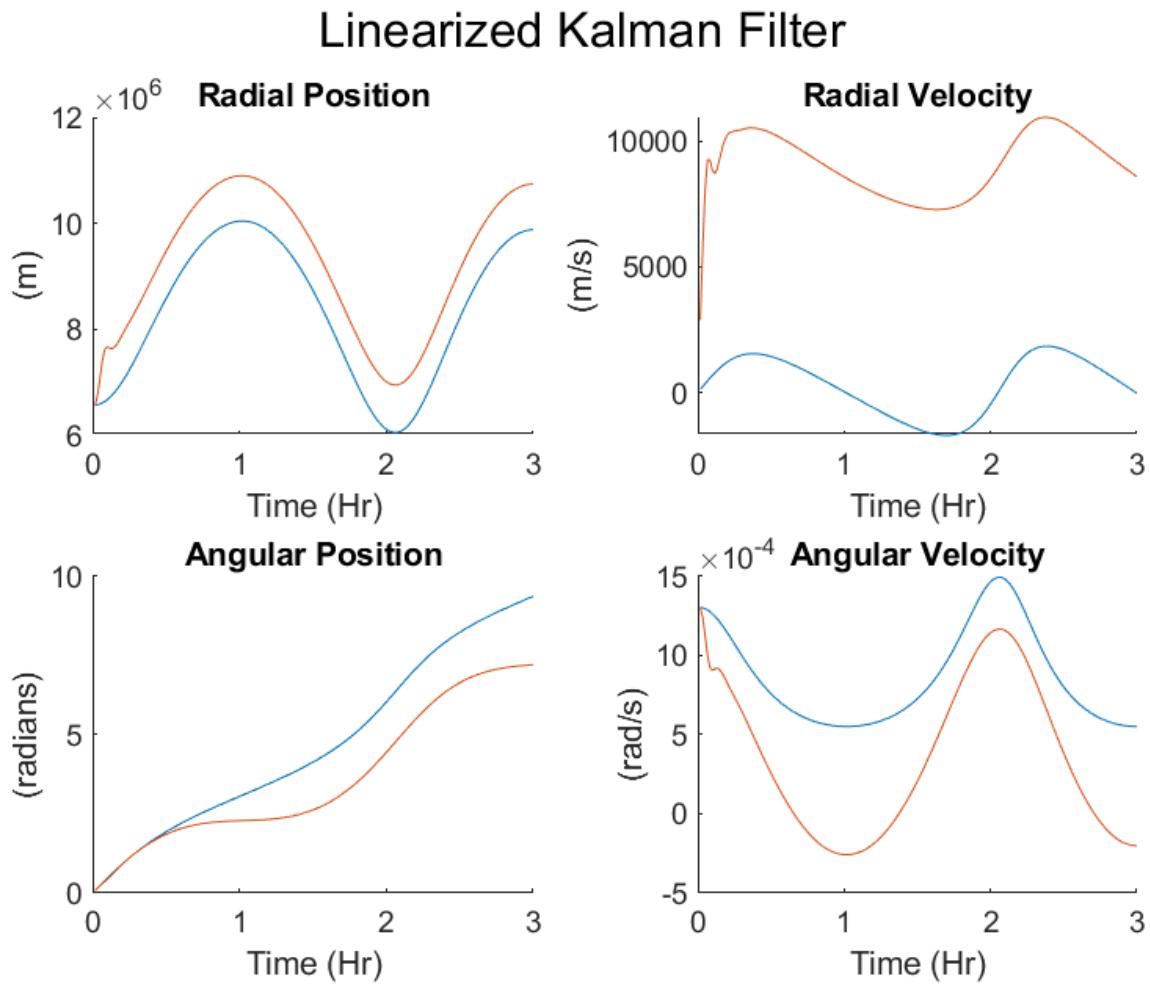


Fig. 9: Simulation Results for the Linear Hybrid Kalman Filter for problem 5

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

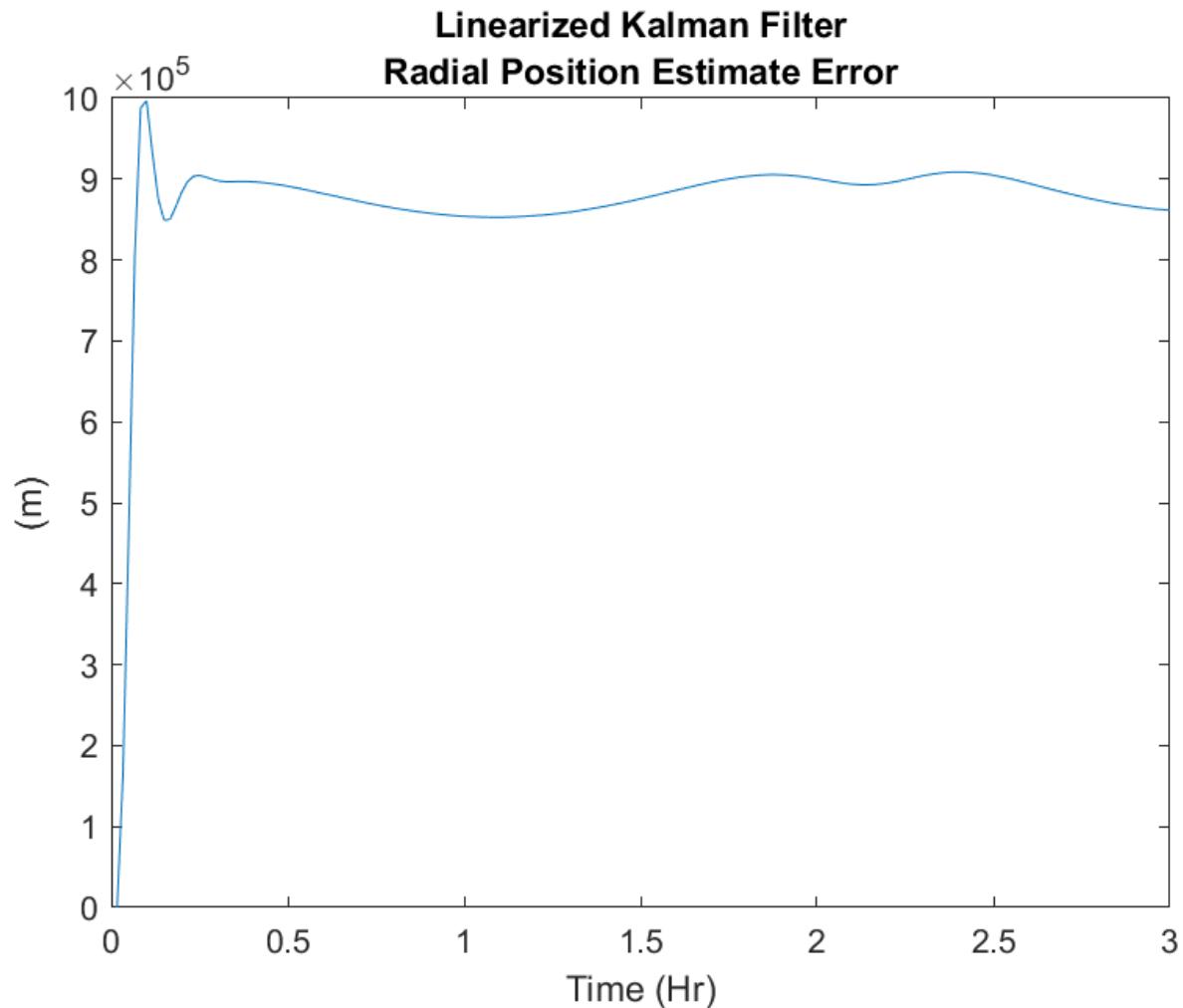


Fig. 10: Estimation Error for the Linear Hybrid Kalman Filter for problem 5

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 measurements 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.

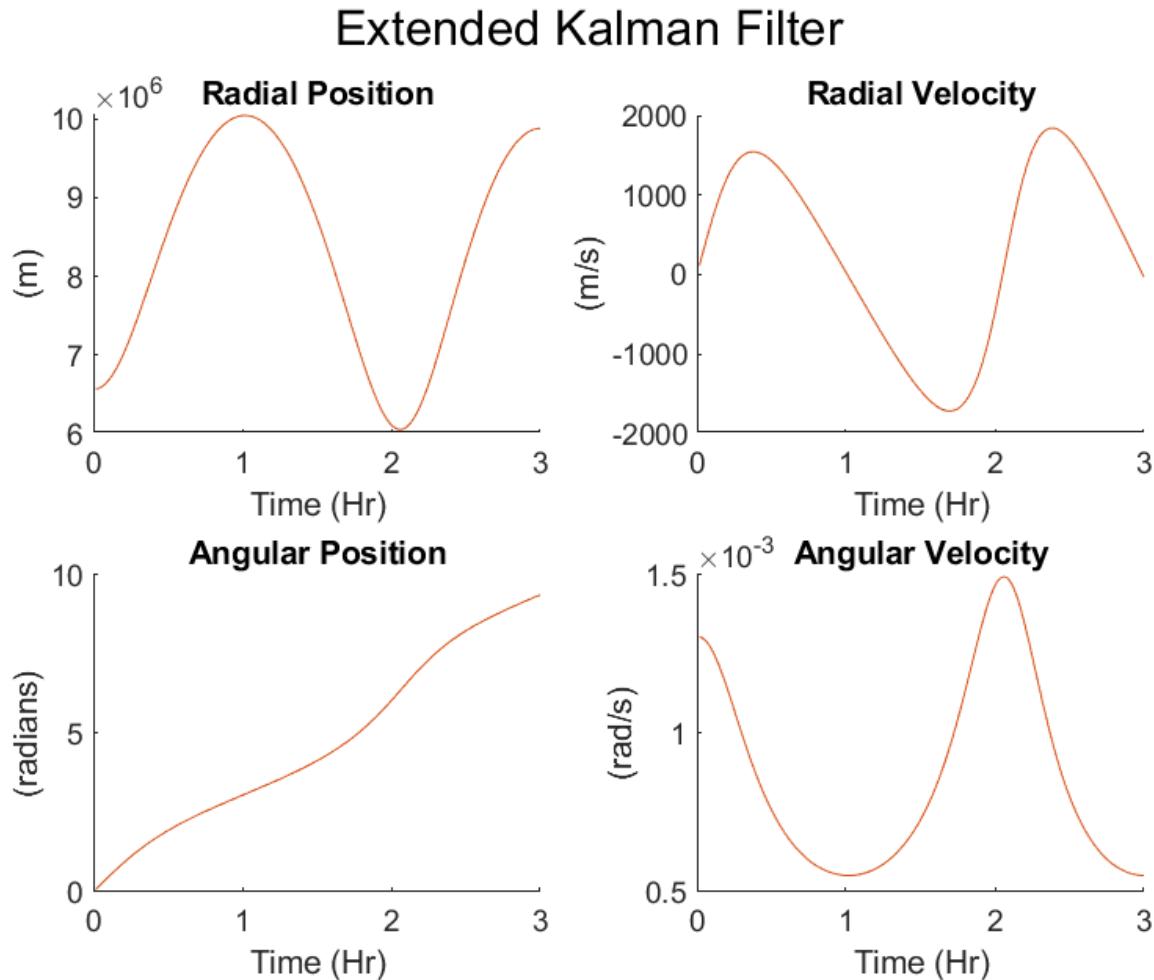


Fig. 11: Simulation Results for the Hybrid Extended Kalman Filter for problem 5

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

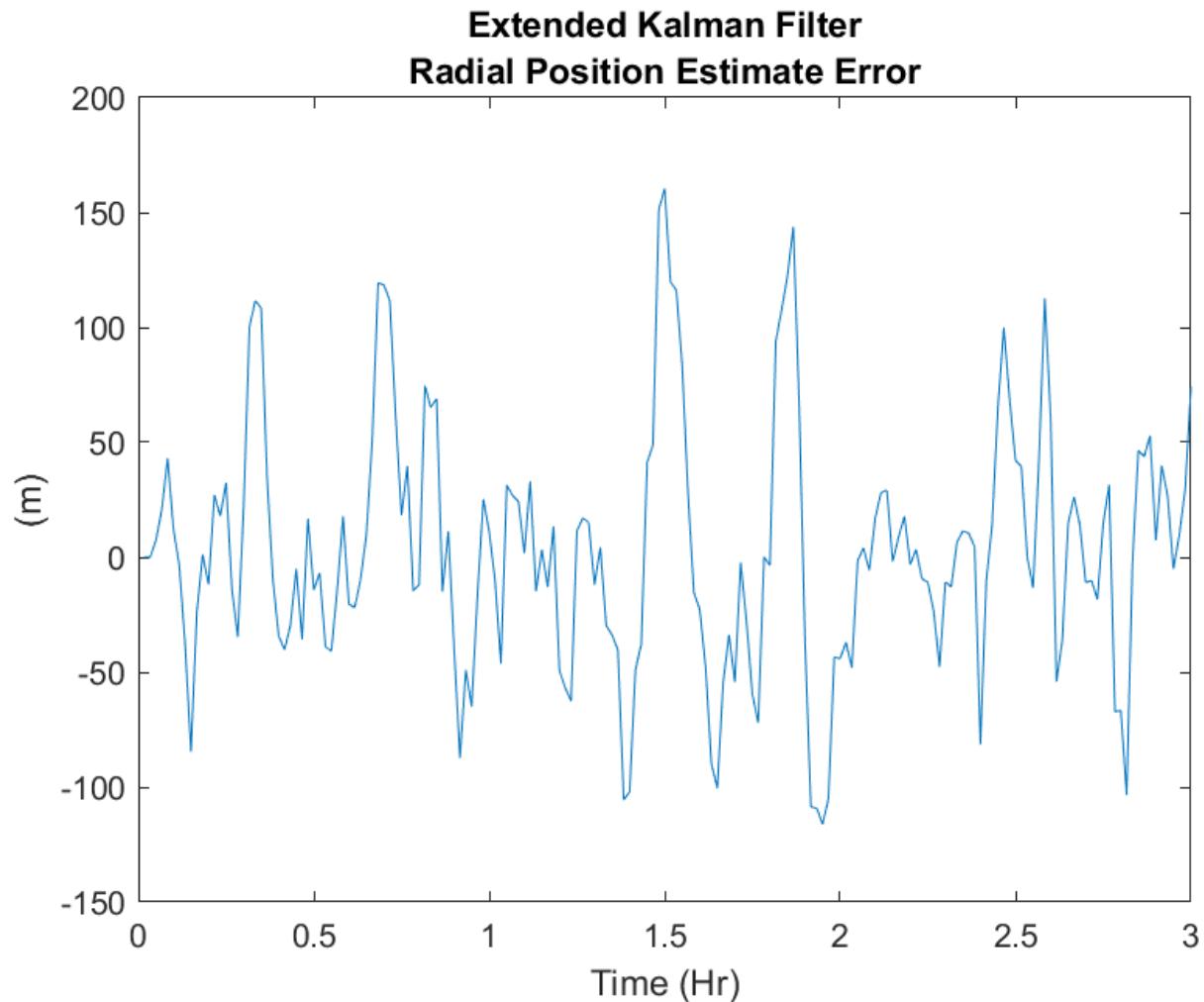


Fig. 12: Estimation Error for the Hybrid Extended Kalman Filter for problem 5

It is clear that the Extended Kalman Filter performs much better than the Linear one. Although the error is more sporadic, the magnitude is order smaller than 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

```
1 % MECH 6325 - Final Project
2 %Jonas Wagner
3 %2020-11-28
4
5 clear
6 close all;
7
8 % Problem 1 -----
9 disp('Problem 1 -----')
10 pblm1()
11
12 clear
13 % Problem 2 -----
14 disp('Problem 2 -----')
15 pblm2()
16
17 clear
18 % Problem 3 -----
19 disp('Problem 3 -----')
20 pblm3()
21
22 clear
23 % Problem 4 -----
24 disp('Problem 4 -----')
25 pblm4()
26
27 clear
28 % Problem 5 -----
29 disp('Problem 5 -----')
30 pblm5()
31
32 clear
33 close all %Comment this out if figures are wanted...
```

A.1 Problem 1

Script 2: pblm1.m

```
1 function [] = pblm1()
2 % Problem 1 Script/Function
3 % MECH 6325 - Final Project
4
5 % System Definition
6 F = [1/2, 2;
7     0, 0];
8 G = [0; 1];
9 f_0 = 250;
10 L = [0; 1];
11 H = [1, 0];
12
13 x_0 = [650; 250];
14 Q = 10;
15 R = 10;
16
17 x_hat_0 = [600; 200];
18 P_0 = [500, 0;
19     0, 200];
20
21 % Part a - KF Simulation (10 steps) -----
22 n = size(F,1);
23 T = 1;
24 T_max = 10;
25 N = floor(T_max/T);
26 U = f_0 * ones(N+1,1);
27
28 [X,~,~,X_hat,P,K] = KalmanFilter_DT(F,G,L,H,x_0,Q,R,x_hat_0,P_0,N,n,U);
29
30
31 figure('position',[0,0,3*300,2*300])
32 sgttitle('Modeling Method 1')
33
34 subplot(2,2,1)
35 hold on
36 plot(X(1,:))
37 plot(X_hat(1,:))
38 hold off
39 title('Wombat Population')
40 legend('Actual','Estimated')
41
42 subplot(2,2,2)
43 hold on
```

```

44 plot(X(2,:))
45 plot(X_hat(2,:))
46 hold off
47 title('Food Supply')
48 legend('Actual','Estimated')
49
50 subplot(2,2,3)
51 hold on
52 plot(sqrt(reshape(P(1,1,:),[],1)))
53 plot(sqrt(reshape(P(2,2,:),[],1)))
54 hold off
55 title('Standard Deviation')
56 legend('p_k','f_k')
57
58 subplot(2,2,4)
59 hold on
60 plot(K(1,:))
61 plot(K(2,:))
62 hold off
63 title('Gain Matrix')
64 legend('k_1','k_2')
65
66 saveas(gcf,[pwd '/fig/pblm1_results_method1.png'])
67
68
69 % Part b - P_inf comparison -----
70 disp('If using the following F, there is a discrepancy for state 2')
71 F = [1/2, 2;
72      0, 0]
73 P_minus_inf = idare(F',H',Q,R)
74 k_inf = P_minus_inf*H'/(H*P_minus_inf*H' + R)
75 P_plus_inf = (eye(2)-k_inf*H)*P_minus_inf
76 P_plus_inf_std = sqrt(P_plus_inf)
77
78 % Part c - Simulation (1000 steps)
79 T_max = 1000;
80 N = floor(T_max/T);
81 U = f_0 * ones(N+1,1);
82 [X1,~,~,X_hat1,P1,~] = KalmanFilter_DT(F,G,L,H,x_0,Q,R,x_hat_0,P_0,N,n,U);
83
84 figure()
85 hold on
86 plot(sqrt(reshape(P1(1,1,:),[],1)))
87 plot(sqrt(reshape(P1(2,2,:),[],1)))
88 hold off
89 title('Standard Deviation')

```

```

90 legend('p_k','f_k')
91 saveas(gcf,[pwd '/fig/pblm1_std_method1.png'])
92
93 disp(['There is a decrepency for the secound state.', newline, ...
94      'This is believed to be because of the constant input not = 0,', ...
95      newline, 'so the steady-state Kalman filter for that state is ', ...
96      'not as accurate.']);
97
98 disp('-----')
100
101
102 disp('The alternative method of simulation:')
103
104 F = [1/2, 2;
105      0, 1]
106 G = [0; 1]
107
108 n = size(F,1);
109 T = 1;
110 T_max = 10;
111 N = floor(T_max/T);
112 U = 0 * ones(N+1,1);
113
114 [X,~,~,X_hat,P,K] = KalmanFilter_DT(F,G,L,H,x_0,Q,R,x_hat_0,P_0,N,n,U);
115
116
117 figure('position',[0,0,3*300,2*300])
118 sgttitle('Modeling Method 2')
119
120 subplot(2,2,1)
121 hold on
122 plot(X(1,:))
123 plot(X_hat(1,:))
124 hold off
125 title('Wombat Population')
126 legend('Actual','Estimated')
127
128 subplot(2,2,2)
129 hold on
130 plot(X(2,:))
131 plot(X_hat(2,:))
132 hold off
133 title('Food Supply')
134 legend('Actual','Estimated')
135

```

```

136 subplot(2,2,3)
137 hold on
138 plot(sqrt(reshape(P(1,1,:),[],1)))
139 plot(sqrt(reshape(P(2,2,:),[],1)))
140 hold off
141 title('Standard Deviation')
142 legend('p_k','f_k')
143
144 subplot(2,2,4)
145 hold on
146 plot(K(1,:))
147 plot(K(2,:))
148 hold off
149 title('Gain Matrix')
150 legend('k_1','k_2')
151
152 saveas(gcf,[pwd '/fig/pblm1_results_method2.png'])
153
154 P_minus_inf = idare(F',H',L*Q*L',R)
155 k_inf = P_minus_inf*H'/(H*P_minus_inf*H' + R)
156 P_plus_inf = (eye(2)-k_inf*H)*P_minus_inf
157 P_plus_inf_std = sqrt(P_plus_inf)
158
159 % x_hat_error_inf =
160
161 T_max = 1000;
162 N = floor(T_max/T);
163 U = 0 * ones(N+1,1);
164 [X2,~,~,X_hat2,P2,~] = KalmanFilter_DT(F,G,L,H,x_0,Q,R,x_hat_0,P_0,N,n,U);
165
166 figure()
167 hold on
168 plot(sqrt(reshape(P2(1,1,:),[],1)))
169 plot(sqrt(reshape(P2(2,2,:),[],1)))
170 hold off
171 title('Standard Deviation')
172 legend('p_k','f_k')
173 saveas(gcf,[pwd '/fig/pblm1_std_method2.png'])
174
175
176 disp('The alternative method has a discrepancy for the food term as well')
177
178 end

```

Script 3: KalmanFilter_DT.m

```

1 function [X,Y,X_hat_pre,X_hat_post,P,K] = KalmanFilter_DT(F,G,L,H,x_0,Q,R, ...
2                                         x_hat_0,P_0,N,n,U)
3 arguments
4     F
5     G
6     L
7     H
8     x_0
9     Q
10    R
11    x_hat_0
12    P_0
13    N
14    n = size(F,1)
15    U = zeros(N+1,1);
16 end
17
18 x = x_0;
19 y = H * x + R * randn(size(R,1),1);
20 p_pre = P_0;
21 p_post = p_pre;
22
23 u = U(1);
24 x_hat_pre = x_hat_0;
25 k = p_pre * H' * inv(R);
26 x_hat_post = x_hat_pre;
27
28 X = [];
29 Y = [];
30
31 K = [];
32 X_hat_pre= [];
33 X_hat_post = [];
34 P_pre = zeros(n,n,N-1);
35 P_post = zeros(n,n,N-1);
36
37 for i = 1:(N-1)
38     u = U(i);
39     x = F * x + G * u + L * Q * randn(size(Q,1),1);
40     y = H * x + R * randn(size(R,1),1);
41
42     p_pre = F * p_post * F' + L' * Q * L;
43     k = p_pre * H' * inv(H * p_pre * H' + R);
44     x_hat_pre = F * x_hat_post + G * u;
45     x_hat_post = x_hat_pre + k * (y - H * x_hat_pre);

```

```

46 p_post = (eye(n) - k * H) * p_pre * (eye(n) - k * H)' + k * R * k';
47
48 X = [X x];
49 Y = [Y y];
50
51 K = [K k];
52 X_hat_pre = [X_hat_pre x_hat_pre];
53 X_hat_post = [X_hat_post x_hat_post];
54 P_pre(:,:,i) = p_pre;
55 P_post(:,:,i) = p_post;
56 end
57
58
59 X = X;
60 Y = Y;
61
62 X_hat_pre = X_hat_pre;
63 X_hat_post = X_hat_post;
64 P = P_post;
65
66 end

```

A.2 Problem 2

Script 4: pblm2.m

```
1 function [] = pblm2()
2 % Problem 2 Script/Function
3 % MECH 6325 - Final Project
4
5 % Parameters
6 R = 100;
7 L = 1;
8 C = 1;
9
10 % CT System Definitions
11 A = [-2/(R*C), 1/C;
12     -1/L, 0];
13 B = [1/(R*C);
14     1/L];
15 C = eye(2);
16 Q_c = 3^2;
17
18 T = 0.1;
19 T_max = 2;
20 N = floor(T_max/T);
21
22 disp('Discretized System:')
23 [F,G,H,Q,~] = DiscretizeSystem(A,B,C,Q_c,0,T)
24 % L = G
25
26 n = size(F,1);
27 x_0 = zeros(n,1);
28 R = eye(2);
29
30 x_hat_0 = x_0;
31 P_0 = 0.1 * eye(n);
32
33 [X,Y,~,X_hat_post,P_pre,P_post,~] = ...
34     KalmanFilter_Sequential(F,G,G,H,x_0,Q,R,x_hat_0,P_0,N,n);
35
36 %Plotting
37 t = T:T:T_max;
38 % Part a
39 figure()
40 hold on
41 plot(t,reshape(P_pre(1,1,:),[],1))
42 plot(t,reshape(P_post(1,1,:,:1),[],1))
43 plot(t,reshape(P_post(1,1,:,:2),[],1))
```

```

44 hold off
45 title('Capacitor Voltage Estimation Variance')
46 legend('P_{pre}', 'P_{post}_1', 'P_{post}_2')
47 xlabel('Time (s)')
48 ylabel('Variance')
49 saveas(gcf,[pwd '/fig/pblm2_est_var.png'])
50
51 % Part b
52 figure()
53 hold on
54 plot(t,X(1,:))
55 plot(t,X_hat_post(1,:))
56 plot(t,Y(1,:))
57 hold off
58 title('Capacitor Voltage Comparison')
59 legend('Actual', 'Estimated', 'Measured')
60 xlabel('Time (s)')
61 ylabel('Voltage (V)')
62 saveas(gcf,[pwd '/fig/pblm2_comparison.png'])
63
64 disp('Standard deviations of errors')
65 Y_error_std = std(Y(1,:)-X(1,:))
66 X_hat_error_std = std(X_hat_post(1,:)-X(1,:))
67
68 end

```

Script 5: DiscretizeSystem.m.m

```
1 function [F,G,H,Q,R] = DiscretizeSystem(A,B,C,Q_c,R_c,T)
2     n = size(A,1);
3     if B == 0
4         B = zeros(n,1);
5     end
6     if C == 0
7         C = zeros(1,n);
8     end
9     sys = ss(A,B,C,0);
10    dt_sys = c2d(sys,T);
11    [F,G,H,~,T] = ssdata(dt_sys);
12 % Original Code... issues with stability using this method...
13 % n = size(A,1);
14 % F = exp(A * T);
15 % syms tau
16 % G = double(F * int(eye(n) - exp(-A * tau),tau,0,T) * B);
17 % H = C;
18 % Q = Q_c * T;
19 % R = R_c * T;
20 end
```

Script 6: KalmanFilter_SequENTIAL.m

```

1 function [X,Y,X_hat_pre,X_hat_post,P_pre,P_post,K] = ...
2     KalmanFilter_Sequential(F,G,L,H,x_0,Q,R,x_hat_0,P_0,N,n,r,U)
3 arguments
4     F
5     G
6     L
7     H
8     x_0
9     Q
10    R
11    x_hat_0
12    P_0
13    N = 100;
14    n = size(F,1);
15    r = size(H,1)
16    U = zeros(N+1,1);
17 end
18
19 x = x_0;
20 y = H * x + R * randn;
21 p_pre = P_0;
22 p_post = p_pre;
23
24 u = U(1);
25 x_hat_pre = x_hat_0;
26 x_hat_post = x_hat_pre;
27
28 X = [];
29 Y = [];
30
31 K = zeros(n,r,N);
32 X_hat_pre= [];
33 X_hat_post = [];
34 P_pre = zeros(n,n,N);
35 P_post = zeros(n,n,N);
36
37 for i = 1:N
38     u = U(i);
39     x = F * x + G * u + L * Q * randn(size(Q,1),1);
40     y = H * x + R * randn(size(R,1),1);
41
42     % Time Update
43     p_pre = F * p_post * F' + L' * Q * L;
44     x_hat_pre = F * x_hat_post + G * u;
45

```

```

46 k = zeros(n,r);
47 x_hat = zeros(n,r);
48 p = zeros(n,n,r);
49
50 %Measurement Update
51 k(:,1) = p_pre * H(1,:)' * inv(H(1,:)) * p_pre * H(1,:)'+ R(1,1));
52 x_hat(:,1) = x_hat_pre + k(:,1)*(y(1) - H(1,:) * x_hat_pre);
53 p(:,:,1) = (eye(n) - k(:,1) * H(1,:))* p_pre;
54
55 for j = 2:r
56     k(:,j) = p(:,:,j-1) * H(j,:)'* inv(H(j,:)) * p(:,:,j-1) * H(j,:)'+ R(j
57         ,j));
58     x_hat(:,j) = x_hat(:,j-1) + k(:,j-1) * (y(j) - H(j,:) * x_hat(:,j-1));
59     p(:,:,j) = (eye(n) - k(:,j-1) * H(j,:))* p(:,:,j-1);
60 end
61
62 x_hat_post = x_hat(:,r);
63 p_post = p(:,:,r);
64
65 X = [X x];
66 Y = [Y y];
67
68 K(:,:,i) = k;
69 X_hat_pre = [X_hat_pre x_hat_pre];
70 X_hat_post = [X_hat_post x_hat_post];
71 P_all(:,:,i,:) = p(:,:,i);
72 P_pre(:,:,i) = p_pre;
73 P_post(:,:,i) = p_post;
74 end
75
76 X = X;
77 Y = Y;
78
79 X_hat_pre = X_hat_pre;
80 X_hat_post = X_hat_post;
81 P_pre = P_pre;
82 P_post = P_all;
end

```

A.3 Problem 3

Script 7: pblm3.m

```
1 function [] = pblm3()
2 % Problem 3 Script/Function
3 % MECH 6325 - Final Project
4
5 C = eye(2);
6 R = eye(2);
7 N = 1e3;
8 T = 1e-2;
9
10 % Part a
11 disp('Part a -----')
12 A = diag([1,2]);
13 Q = [1,1;
14     1,1];
15 P_0 = eye(2);
16 P = careInt(A,C,Q,R,P_0,N,T);
17 P_ss_a = P(:,:,end)
18
19 P_ss_a_care = care(A',C',Q,R)
20
21 % Part b
22 disp('Part b -----')
23 A = diag([-1,-1]);
24 Q = [1,2;
25     2,4];
26 P_0 = eye(2);
27 P = careInt(A,C,Q,R,P_0,N,T);
28 P_ss_b = P(:,:,end)
29
30 P_ss_b_care = care(A',C',Q,R)
31
32
33 % Part c
34 disp('Part c -----')
35 A = diag([1,1]);
36 Q = [1,2;
37     2,4];
38 P_0 = eye(2);
39 P = careInt(A,C,Q,R,P_0,N,T);
40 P_ss_c = P(:,:,end)
41
42 P_ss_c_care = care(A',C',Q,R)
43
```

```

44 N = 1e5;
45
46 % Part d
47 disp('Part d -----')
48 A = diag([1,1]);
49 Q = [1,2;
50     2,4];
51 P_0 = 0;
52 P = careInt(A,C,Q,R,P_0,N,T);
53 P_ss_d = P(:,:,end)
54
55 P_ss_d_care = care(A',C',Q,R)
56
57 K_int = P_ss_d * C' * inv(R)
58 A_KC_int = A-K_int*C
59 eig_int = eig(A_KC_int)
60 disp(['This is an unstable observer, so it does not result in a' ,...
61       'steady-state Kalman Filter'])
62 end
63
64
65 function P = careInt(A,C,Q,R,P_0,N,T)
66     arguments
67         A
68         C
69         Q
70         R
71         P_0
72         N = 100;
73         T = 0.01;
74     end
75     p_dot = - P_0 * C' * inv(R) * C * P_0 + A * P_0 + P_0 * A' + Q;
76     p = P_0 + p_dot * T;
77     P(:,:,1) = p;
78     for i = 2:N
79         p_dot = - p * C' * inv(R) * C * p + A * p + p * A' + Q;
80         p = p + p_dot * T;
81         P(:,:,i) = p;
82     end
83 end

```

A.4 Problem 4

Script 8: pblm4.m

```
1 function [] = pblm4()
2 % Problem 4 Script/Function
3 % MECH 6325 - Final Project
4
5 % Parameters
6 T = 0.5;
7 T_max = 5;
8
9 omega = 6;
10 zeta = 0.16;
11
12 % System Definition
13 A = [0, 1;
14     -omega^2, -2*zeta*omega];
15 B = [0;1];
16 C = eye(2);
17 Q_c = 0.01;
18
19 sys = ss(A,B,C,0)
20
21
22 % Part a - Discretization
23 [F,G,~,~] = DiscretizeSystem(A,B,C,Q_c,0,T);
24 H = [1,0];
25 Q
26 L=G;
27 R = 1e-4;
28
29
30 dt_sys = ss(F,G,H,0,T)
31
32 % Part b - Simulation
33 x_0 = [1;1];
34 x_hat_0 = x_0;
35 P_0 = diag([1e-5,1e-2]);
36
37
38 [X,Xt,~,~,~,X_hat,P,~,hat_t] = KalmanFilter_DT_SDS(sys,F,0,L,H,....
39                                         x_0,Q,R,x_hat_0,P_0,T,T_max);
40
41
42 figure('position',[0,0,3*300,2*300])
```

```

44 sgtitle('Sampled Data System Response')
45
46 subplot(2,2,1)
47 hold on
48 plot(hat_t(1:end-1),reshape(P(1,1,1:end-1),[],1))
49 hold off
50 title('Estimation Error Variance (State 1)')
51 xlabel('Time (s)')
52
53 subplot(2,2,2)
54 hold on
55 plot(hat_t(1:end-1),reshape(P(2,2,1:end-1),[],1))
56 hold off
57 title('Estimation Error Variance (State 2)')
58 xlabel('Time (s)')
59
60 subplot(2,2,3)
61 hold on
62 plot(Xt,X(1,:))
63 plot(hat_t,X_hat(1,:))
64 hold off
65 title('State 1')
66 legend('Actual','Estimate')
67 xlabel('Time (s)')
68
69 subplot(2,2,4)
70 hold on
71 plot(Xt,X(2,:))
72 plot(hat_t,X_hat(2,:))
73 hold off
74 title('State 2')
75 legend('Actual','Estimate')
76 xlabel('Time (s)')
77 saveas(gcf,[pwd '/fig/pblm4_results.png'])
78 end

```

Script 9: KalmanFilter_DT_SDS.m

```

1 function [X,Xt,Y,W,X_hat_pre,X_hat_post,P,K,hat_t] = KalmanFilter_DT_SDS(sys,F,G,
2 L,H, ...
3 x_0,Q,R, ...
4 arguments
5 sys % CT LTI sys... should have direct output of states
6 F
7 G
8 L %Assumed L = G... if not W or L should be modified...
9 H
10 x_0
11 Q
12 R
13 x_hat_0
14 P_0
15 T
16 T_max
17 n = size(F,1);
18 dtdiff = 10;
19 U = zeros(dtdiff*floor(T_max/T)+1,1);
20 end
21 N = floor(T_max/T);
22
23 x = x_0;
24 y = H * x + R * randn;
25 p_pre = P_0;
26 p_post = p_pre;
27
28 x_hat_pre = x_hat_0;
29 k = p_pre * H' * inv(R);
30 x_hat_post = x_hat_pre;
31
32 Xt = (T/dtdiff)*(0:(N*dtdiff));
33 W = randn(N*dtdiff+1,1);
34 X = lsim(sys,U + Q*W,Xt,x_0)';
35 Y = [];
36 hat_t = [];
37
38 K = [];
39 X_hat_pre= [];
40 X_hat_post = [];
41 P_pre = zeros(n,n,N-1);
42 P_post = zeros(n,n,N-1);
43
44 for i = 1:(N-1)

```

```

45 u = U(i);
46 t = i*T;
47 x = X(:,dtdiff*i+1);%F * x + G * u + L * Q * randn;
48 y = H * x + R * randn;
49
50 p_pre = F * p_post * F' + L' * Q * L;
51 k = p_pre * H' * inv(H * p_pre * H' + R);
52 x_hat_pre = F * x_hat_post + G * u;
53 x_hat_post = x_hat_pre + k * (y - H * x_hat_pre);
54 p_post = (eye(n) - k * H) *p_pre * (eye(n) - k * H)' + k * R * k';
55
56 Y = [Y y];
57 hat_t = [hat_t t];
58
59 K = [K k];
60 X_hat_pre = [X_hat_pre x_hat_pre];
61 X_hat_post = [X_hat_post x_hat_post];
62 P_pre(:,:,i) = p_pre;
63 P_post(:,:,i) = p_post;
64 end
65
66 P = P_post;
67
68 end

```

A.5 Problem 5

Script 10: pblm5.m

```
1 function [] = pblm5()
2 % Problem 5 Script/Function
3 % MECH 6325 - Final Project
4 G_const = 6.674e-11;
5 M_const = 5.98e24;
6
7 % Part a -----
8 disp('Part a -----')
9 syms r r_dot theta theta_dot w G M
10 x = [r; r_dot; theta; theta_dot];
11 x_dot_sym = pblm5_nonlin(x,w,G,M)
12
13 % Part b -----
14 disp('Part b -----')
15 eq = 0 == subs(x_dot_sym(2),w,0)
16 theta_dot_sym = solve(eq,theta_dot)
17
18 % Part c -----
19 disp('Part c -----')
20 syms r_0 omega_0 T
21 x_0 = [r_0; 0; omega_0 * T; omega_0];
22 omega_static = subs(theta_dot_sym,[G, M, r],[G_const, M_const, r_0]);
23
24 A = subs(jacobian(x_dot_sym,x), x, x_0)
25 L = jacobian(x_dot_sym,w)
26
27
28 r_0_const = 6.57e6;
29 omega_0_const = double(subs(omega_static(1), r_0,r_0_const))
30 v_const = omega_0_const * r_0_const % This checks orbital speed... should be
   around 7-8 km/s
31 A_const = double(subs(A,[omega_0, r_0, G, M],...
   [omega_0_const, r_0_const, G_const, M_const]))
32 eig_A = eig(A_const)
33 tau = 2*pi / (imag(eig_A(2)))
34 min_int_step = 0.1 * tau
35
36
37
38 % Part d -----
39 disp('Part d -----')
40 T = 5;
41 sensorT = 60;
42 T_max = 3 * 60 * 60;
```

```

43 H = [1,0,0,0;
44     0,0,1,0];
45 R = diag([100,0.1]);
46
47 x_0 = [r_0_const; 0; 0; 1.1*omega_0_const];
48 Q_c = 10e-6;
49 x_hat_0 = x_0;
50 P_0 = zeros(4);
51
52
53
54 [X,Xt,Y,X_hat,P,K,sensor_t] = KalmanFilter_Hybrid_nonlinsys(@pbm5_nonlin, ...
55                         A_const,0,L,H, ...
56                         x_0, Q_c, R, ...
57                         x_hat_0, P_0, ...
58                         T,T_max,sensorT);
59
60 figure()
61 sgttitle('Linearized Kalman Filter')
62 subplot(2,2,1);
63 hold on
64 plot(Xt/3600,X(1,:))
65 plot(sensor_t/3600,X_hat(1,:))
66 hold off
67 title('Radial Position')
68 xlabel('Time (Hr)')
69 ylabel('(m)')
70
71 subplot(2,2,2);
72 hold on
73 plot(Xt/3600,X(2,:))
74 plot(sensor_t/3600,X_hat(2,:))
75 hold off
76 title('Radial Velocity')
77 xlabel('Time (Hr)')
78 ylabel('(m/s)')
79
80 subplot(2,2,3);
81 hold on
82 plot(Xt/3600,X(3,:))
83 plot(sensor_t/3600,X_hat(3,:))
84 hold off
85 title('Angular Position')
86 xlabel('Time (Hr)')
87 ylabel('(radians)')
88

```

```

89 subplot(2,2,4);
90 hold on
91 plot(Xt/3600,X(4,:))
92 plot(sensor_t/3600,X_hat(4,:))
93 hold off
94 title('Angular Velocity')
95 xlabel('Time (Hr)')
96 ylabel('(rad/s)')
97
98 saveas(gcf,[pwd '/fig/pblm5_results_hybrid_lin.png'])
99
100 test_t = 0:sensorT:T_max;
101 test_x = interp1(Xt',X',test_t)';
102 test_x_hat = interp1(sensor_t',X_hat',test_t)';
103
104 x_error = test_x_hat - test_x;
105
106 figure()
107 plot(test_t/3600,x_error(1,:));
108 title(['Linearized Kalman Filter', newline, ...
109         'Radial Position Estimate Error'])
110 xlabel('Time (Hr)')
111 ylabel('(m)')
112
113 saveas(gcf,[pwd '/fig/pblm5_est_error_hybrid_lin.png'])
114
115 disp(['The performance of the linear Kalman Filter is reduced by the ',...
116       'lack of ', newline, 'linearity in the actual system dynamics.',...
117       ' This may be improved with faster', newline, 'measurements',...
118       ' and Kalman Filter updates; however this is not guaranteed.']);
119
120
121
122
123
124 % Part e -----
125 disp('Part e -----')
126 [X,Xt,Y,X_hat,P,K,sensor_t] = KalmanFilter_Extended_Hybrid(@pblm5_nonlin, ...
127                           A_const,0,L,H, ...
128                           x_0, Q_c, R, ...
129                           x_hat_0, P_0, ...
130                           T,T_max,sensorT);
131
132 figure()
133 sgttitle('Extended Kalman Filter')
134 subplot(2,2,1);

```

```

135 hold on
136 plot(Xt/3600,X(1,:))
137 plot(sensor_t/3600,X_hat(1,:))
138 hold off
139 title('Radial Position')
140 xlabel('Time (Hr)')
141 ylabel('(m)')

142 subplot(2,2,2);
143 hold on
144 plot(Xt/3600,X(2,:))
145 plot(sensor_t/3600,X_hat(2,:))
146 hold off
147 title('Radial Velocity')
148 xlabel('Time (Hr)')
149 ylabel('(m/s)')

150 subplot(2,2,3);
151 hold on
152 plot(Xt/3600,X(3,:))
153 plot(sensor_t/3600,X_hat(3,:))
154 hold off
155 title('Angular Position')
156 xlabel('Time (Hr)')
157 ylabel('(radians)')

158 subplot(2,2,4);
159 hold on
160 plot(Xt/3600,X(4,:))
161 plot(sensor_t/3600,X_hat(4,:))
162 hold off
163 title('Angular Velocity')
164 xlabel('Time (Hr)')
165 ylabel('(rad/s)')

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

167 test_t = 0:sensorT:T_max;
168 test_x = interp1(Xt',X',test_t)';
169 test_x_hat = interp1(sensor_t',X_hat',test_t)';
170 x_error = test_x_hat - test_x;

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

```

```
181     'Radial Position Estimate Error'])
182 xlabel('Time (Hr)')
183 ylabel('(m)')
184
185 saveas(gcf,[pwd '/fig/pblm5_est_error_hybrid_EKF.png'])
186
187 disp(['As is clear between the two estimate error plots, the Extended',...
188       'Kalman Filter ', newline, 'has a lot less error then the Linear one.',...
189       'This is very evident from the order ', newline,...
190       'of magnitude on each of the plots.'])
191
192 assignin('base','X',X)
193
194
195
196 end
```

Script 11: pblm5_nonlin.m

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

Script 12: KalmanFilter_Hybrid_nonlinsys.m

```

1 function [X,Xt,Y,X_hat,P,K,sensor_t] = KalmanFilter_Hybrid_nonlinsys2(...
2                                     nonlinsys,A,B,L,H, ...
3                                     x_0,Q_c,R, ...
4                                     x_hat_0,P_0, ...
5                                     T,T_max,sensorT,n,U)
6 arguments
7     nonlinsys
8     A double
9     B double
10    L double
11    H double
12    x_0 double
13    Q_c double
14    R double
15    x_hat_0 double
16    P_0 double
17    T double = 0.1;
18    T_max double = 500;
19    sensorT double = 0.5;
20    n = size(A,1);
21    U = zeros(ceil(T_max/T),1); %This code assumes U = 0... so not needed
22 end
23
24 % Empty Arrays
25 N = floor(T_max/sensorT);
26 X = [];
27 Y = [];
28 sensor_t = [];
29 K = [];
30 X_hat_pre = [];
31 X_hat_post = [];
32 P_pre = zeros(n,n,N);
33 P_post = zeros(n,n,N);
34
35
36 % For Loop Initiation
37 x = x_0;
38
39 p = P_0;
40 x_hat = x_hat_0;
41 k = p * H' * inv(R);
42
43 for i = 1:N
44     % Actual System and Measurement
45     t = i * sensorT;

```

```

46 x_dot = nonlinsys(x);
47 x = x + x_dot * sensorT;
48 y = H * x + R * randn(size(R,2),1);
49
50 X = [X x];
51 Y = [Y y];
52 sensor_t = [sensor_t t];
53
54 % Time Update
55 x_hat_dot = A * x_hat;
56 x_hat = x_hat + x_hat_dot * sensorT;
57 p_dot = A * p + p * A' + L*Q_c*L';
58 p = p + p_dot * sensorT;
59
60 X_hat_pre = [X_hat_pre x_hat];
61 P_pre(:,:,i) = p;
62
63 % Measurment Update
64 k = p*H'*inv(H*p*H' + R);
65 x_hat = x_hat + k * (y - H * x_hat);
66 p = (eye(n)-k*H)*p*(eye(n)-k*H)' + k*R*k';
67
68 K = [K k];
69 X_hat_post = [X_hat_post x_hat];
70 P_post(:,:,i) = p;
71 end
72
73 Xt = sensor_t;
74
75 X_hat = X_hat_post;
76 P = P_post;
77 end

```

Script 13: KalmanFilter_Extended_Hybrid.m

```
1 function [X,Xt,Y,X_hat,P,K,sensor_t] = KalmanFilter_Extended_Hybrid(...
2     nonlinsys,A,B,L,H, ...
3     x_0,Q_c,R, ...
4     x_hat_0,P_0, ...
5     T,T_max,sensorT,n,U)
6 arguments
7     nonlinsys
8     A double
9     B double
10    L double
11    H double
12    x_0 double
13    Q_c double
14    R double
15    x_hat_0 double
16    P_0 double
17    T double = 0.1;
18    T_max double = 500;
19    sensorT double = 0.5;
20    n = size(A,1);
21    U = zeros(ceil(T_max/T),1); %This code assumes U = 0... so not needed
22 end
23
24
25
26 % Empty Arrays
27 N = floor(T_max/sensorT);
28 X = [];
29 Y = [];
30 sensor_t = [];
31 K = [];
32 X_hat_pre = [];
33 X_hat_post = [];
34 P_pre = zeros(n,n,N);
35 P_post = zeros(n,n,N);
36
37
38 % For Loop Initiation
39 x = x_0;
40
41 p = P_0;
42 x_hat = x_hat_0;
43 k = p * H' * inv(R);
44 for i = 1:N
```

```

46 % Actual System and Measurment
47 t = i * sensorT;
48 % x = interp1(Xt,X,t)';
49
50 x_dot = nonlinsys(x);
51 x = x + x_dot * sensorT;
52
53 y = H * x + R * randn(size(R,2),1);
54
55 X = [X x];
56 Y = [Y y];
57 sensor_t = [sensor_t t];
58
59 % Non-lin Time Update
60 x_hat_dot = nonlinsys(x_hat,0);
61 x_hat = x_hat + x_hat_dot * sensorT;
62 p_dot = A * p + p * A' + L*Q_c*L';
63 p = p + p_dot * sensorT;
64
65 X_hat_pre = [X_hat_pre x_hat];
66 P_pre(:,:,i) = p;
67
68 % Measurment Update
69 k = p*H'*inv(H*p*H' + R);
70 x_hat = x_hat + k * (y - H * x_hat);
71 p = (eye(n)-k*H)*p*(eye(n)-k*H)' + k*R*k';
72
73 K = [K k];
74 X_hat_post = [X_hat_post x_hat];
75 P_post(:,:,i) = p;
76 end
77
78
79 % X = X';
80 % Xt = Xt';
81 Xt = sensor_t;
82
83 X_hat = X_hat_post;
84 P = P_post;
85 end

```

B MATLAB Output

```

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

clear
close all;

% Problem 1
-----
disp('Problem 1
-----')
pbml1()

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

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

clear
% Problem 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 discrepancy for state 2

F =

$$\begin{matrix} 0.5000 & 2.0000 \\ 0 & 0 \end{matrix}$$


```

```
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 decrepency for the secound state.
This is believed to be because 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
0.0998

H =
1      0
0      1

Q =
0.9000

Standard deviations of errors

Y_error_std =
0.8822

X_hat_error_std =
```

0.1330

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

P_ss_a =

$$\begin{array}{cc} 2.3868 & 0.2774 \\ 0.2774 & 4.2188 \end{array}$$

P_ss_a_care =

$$\begin{array}{cc} 2.3868 & 0.2774 \\ 0.2774 & 4.2188 \end{array}$$

Part b -----

P_ss_b =

$$\begin{array}{cc} 0.2899 & 0.5798 \\ 0.5798 & 1.1596 \end{array}$$

P_ss_b_care =

$$\begin{array}{cc} 0.2899 & 0.5798 \\ 0.5798 & 1.1596 \end{array}$$

Part c -----

P_ss_c =

$$\begin{array}{cc} 2.2899 & 0.5798 \\ 0.5798 & 3.1596 \end{array}$$

P_ss_c_care =

$$\begin{array}{cc} 2.2899 & 0.5798 \\ 0.5798 & 3.1596 \end{array}$$

Part d -----

P_ss_d =

$$\begin{array}{cc} 0.6899 & 1.3798 \\ 1.3798 & 2.7596 \end{array}$$

P_ss_d_care =

$$\begin{array}{cc} 2.2899 & 0.5798 \end{array}$$

```
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 a steady-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
measurements and Kalman Filter updates; however this is not guaranteed.

Part e -----
As is clear between the two estimate error plots, the Extended Kalman
Filter

```

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

Published with MATLAB® R2020b

C Hand Calculations

④ P_K — Wombat Population
 f_K — Wombat Food Supply

$$P_{K+1} = \frac{1}{2} P_K + 2 f_K$$

$$P_0 = 650$$

$$f_K = f_0 + w$$

$$f_0 = 250$$

$$v = P_K + v$$

$$v \sim (0, 10)$$

Let, $x_K = \begin{bmatrix} P_K \\ f_K \end{bmatrix}$

$$x_{K+1} = \begin{bmatrix} x_K + 2u_K + 2w \\ u_K \end{bmatrix}$$

$$x_{K+1} = \begin{bmatrix} \frac{1}{2}x_K + 2u_K + 2w \\ u_K \end{bmatrix}$$

$$x_{K+1} = \begin{bmatrix} \frac{1}{2} & 2 \\ 0 & 0 \end{bmatrix} x_K + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u_K + \begin{bmatrix} 0 \\ 1 \end{bmatrix} w$$

$$\begin{aligned} P_0 &= 600 & E[(\hat{P}_0 - P_0)^2] &= 500 \\ f_0 &= 200 & E[(\hat{f}_0 - f_0)^2] &= 200 \end{aligned}$$

$$x_{K+1} = \begin{bmatrix} 1 & 0 \end{bmatrix} x_K + v$$

$$x_0 = F_0 = \begin{bmatrix} \frac{1}{2} & 2 \\ 0 & 0 \end{bmatrix} \quad b = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad L = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

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

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

a) standard KF Process

d)

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

$$\text{P}_{\infty} = \text{DARE}(F^T, H^T, Q, R, 0)$$

d)

$$M = \begin{bmatrix} I \\ 0 \end{bmatrix}$$

wrong
 $M \neq 0$

d)

$$\text{P}_{\infty} = \text{DARE}(F^T, H^T, Q, H M + M^T H^T + R, FM)$$

d)

$$\text{P}_{\infty}^- = \text{idare}(F^T, H^T, Q, R)$$

$$M = 0$$

b)

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

$$\text{P}_{\infty}^+ = (I - K_{\infty} H) \text{P}_{\infty}^-$$

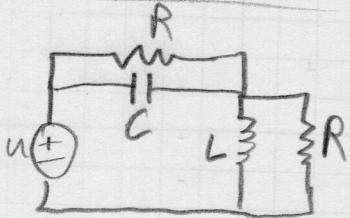
~~$\hat{x}_k^+ = (I - K_{\infty} H) F \hat{x}_{k-1}^+ + K_{\infty} (H \hat{x}_{k-1}^+ + v)$~~

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

~~ss - $\hat{x}_k^+ = \hat{x}_k^- + K_{\infty} H (\hat{x}_k^+ - \hat{x}_k^-) = K_{\infty} v$~~

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

~~(0, H^T R)~~

Ex. 1.8 in book...
a)

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

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

$$\dot{x} = \begin{bmatrix} \dot{v}_c \\ \dot{i}_L \end{bmatrix}$$

$$\dot{x} = Ax + Lw \quad \leftarrow u = w$$

$$A = \begin{bmatrix} -\frac{1}{RC} & \frac{1}{C} \\ -\frac{1}{L} & 0 \end{bmatrix}$$

$$L = \begin{bmatrix} \frac{1}{RC} \\ \frac{1}{L} \end{bmatrix}$$

$$w \sim (0, Q_C) \\ Q_C = 3^2$$

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

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

2) cont.

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

$$A\Delta t = \sigma_0 I$$

$$Q = Q_c \Delta t$$

$$x_0 = 0$$

$$\tilde{w} = L \tilde{w}$$

$$\begin{aligned} 1) \quad x_{K+1}^- &= F x_K + \tilde{w} \quad - \tilde{w} \sim \mathcal{N}(0, Q) \\ y_K &= H x_K + v \quad - v \sim \mathcal{N}(0, R) \end{aligned}$$

$$\begin{aligned} 2) \quad \hat{x}_0^+ &= 0 \\ P_0^+ &= 0.1 I_4 \end{aligned}$$

$$R = I_2 I_2^T$$

$$3) \quad P_K^- = F P_{K-1}^+ F^T + L Q L^T$$

$$\hat{x}_K^- = F \hat{x}_{K-1}^+ + \tilde{u}^0$$

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

$$K_{iK} = \frac{P_{i-1K}^+ H_{iK} H_{iK}^T}{H_{iK} P_{i-1K}^+ H_{iK}^T + R_{iK}} = \frac{P_{i-1K}^+ H_{iK}^T}{R_{iK}}$$

$$\hat{x}_{iK}^+ = \hat{x}_{i-1}^+ + K_{iK} (y_{iK} - H_{iK} \hat{x}_{i-1K}^+)$$

$$P_{iK}^+ = (I - K_{iK} H_{iK}) P_{i-1K}^+ (I - K_{iK} H_{iK})^T + K_{iK} R_{iK} K_{iK}^T$$

$$\hat{x}_K^+ = \hat{x}_{iK}^+$$

$$\hat{P}_K^+ = P_{iK}^+$$

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

③ Riccati Equation Value comparisons

$$\dot{x} = Ax + w \quad A = \begin{bmatrix} a & 0 \\ 0 & a_2 \end{bmatrix} \quad w \sim \mathcal{N}(0, \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix})$$

$$y = Hx + v \quad H = I_2, \quad v \sim \mathcal{N}(0, I_2)$$

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

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

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

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

Extra Part

Does it result in a ss-KF? Pg. 254

No, with the P_∞ from integration

this isn't the case:

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

Check
observer
poles
for stability

$$K = P_\infty C^T R^{-1} = \begin{bmatrix} 0.69 & 1.37 \\ 1.38 & 2.76 \end{bmatrix} \quad \text{Not stable}$$

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

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

$w = 6 \text{ rad/s}$
 $\xi = 0.016$

$X_k = \begin{bmatrix} 1 & 0 \end{bmatrix} X + V(t_k) \quad \tilde{w}(t) \sim (0, 0.01) \quad Q_c$

$X(0) = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad V(t_{15}) \sim (0, 10^{-4})$

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

a) $F = e^{At} \quad At = 0.5 \text{ s}$

$X_{k+1} = F X_k + w - w u(0) \begin{bmatrix} 0 & 0 \\ 0 & Q_c At \end{bmatrix})$

$Y_k = H_k + V - V u(0, 10^{-4}) \quad Q$

b) Standard OT KF

- Plot $E[(\tilde{X} - X)(\tilde{X} - X)^T]$ over time

- Compare states + Estimates

b)

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

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

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

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

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

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

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

$$\begin{bmatrix} \dot{r} \\ r \ddot{\theta}^2 - \frac{GM}{r^2} + \ddot{w} \\ \ddot{\theta} \\ -\frac{2\dot{\theta}\dot{r}}{r} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

b) $\ddot{w} = 0$

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

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

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

5) cont.

$$c) \quad \dot{x}_0 = \begin{bmatrix} r_0 \\ 0 \\ w_0^T \\ w_0 \end{bmatrix}$$

$$A = \frac{df}{dx} \Big|_{x_0} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \dot{\theta}_0^2 + \frac{2GM}{r_0^3} & 0 & 0 & 2r_0\dot{\theta}_0 \\ 0 & 0 & 0 & 1 \\ 2\dot{\theta}_0\ddot{r}_0 & -\frac{2\dot{\theta}_0}{r_0} & 0 & -\frac{2\dot{\theta}_0}{r_0} \end{bmatrix}$$

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \dot{w}_0^2 + \frac{2GM}{r_0^3} & 0 & 0 & 2r_0w_0 \\ 0 & 0 & 0 & 1 \\ 0 & -\frac{2w_0}{r_0} & 0 & 0 \end{bmatrix}$$

$$B = 0$$

$$Lw \rightarrow \tilde{w}(t, Q)$$

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

$$\lambda \text{ for } r_0 = 6.57 \times 10^6 \text{ m:}$$

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

Largest Integration step: ???

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

$$0.1\Delta t = 520 \text{ s}$$

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

5) cont.

a) $\Delta t = 1 \text{ min}$

$$x(0) = \begin{bmatrix} r_0 \\ \dot{r}_0 \\ 0 \\ 1.1w_0 \end{bmatrix}$$

$$\sigma_r = 100 \text{ m} \quad y = E_{\text{air}}^T \bar{x} + w \quad \begin{bmatrix} 100 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\sigma_\theta = 0.1 \text{ rad}$$

$$\hat{x}(0) = x(0)$$

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

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

Kinematized CT-KF:

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

c) EKF:

d) An same

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

$$\hat{x}_0^+ = \begin{bmatrix} r_0 \\ 0 \\ 0 \\ 1.1w_0 \end{bmatrix} \quad u(0^+)$$

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

3) a) Integrate steps:

$$\dot{\hat{x}} = f(\hat{x}, u, 0, +) = \text{diag}(0, 0, 0, 0.13)$$

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

b) standard KF measurement update

- Compare with CT-KF