

# MAE 271B Project

## Missile State Estimation

Aruul Mozhi Varman Senthilkumar

aruul@ucla.edu

### Abstract

*The main goal of the project is to estimate the missile's lateral position, velocity and acceleration using the line-of-sight angle measurements from the radar. A Kalman-Bucy filter is constructed for a Gauss-Markov process, and subsequently the same filter is employed to track the missile with input modeled using a more realistic random telegraph signal characterised by a similar correlation function.*

## 1. Introduction

A pursuer with radar launches a missile to intercept the target and the goal is to estimate the state of the missile as it approaches the target. The problem is formulated in a 2-dimensional world, accordingly the pursuer has a 2-dimensional radar and the objective is to track the lateral position, velocity and acceleration of the missile.

The measurement  $z$  from the radar is corrupted by fading and scintillation noise  $n$  and is characterized as

$$z = \theta + n$$

Here,  $\theta$  is the line-of-sight angle to the target. For  $|\theta| \ll 1$ , it is approximated as

$$\theta \approx \frac{y}{V_c(t_f - t)}$$

where  $V_c = 300 \text{ ft s}^{-1}$  and  $t_f = 10 \text{ s}$  is the terminal time. The statistics of the measurement process is as follows

$$E[n(t)] = 0, E[n(t)n(\tau)] = V\delta(t - \tau) = [R_1 + \frac{R_2}{(t_f - t)^2}]\delta(t - \tau)$$

Here,  $R_1 = 15 \times 10^{-6}$  and  $R_2 = 1.67 \times 10^{-3} \text{ rad}^2 \text{ s}^3$ .  $\delta$  is the Dirac delta function and  $\tau = 2$  is the correlation time variable.

The dynamics of the missile are

$$\dot{y} = v, \dot{v} = a_p - a_T;$$

Here,  $y$  and  $v$  are lateral position and velocity of the missile.  $a_p$  is the missile acceleration and it is zero in this case.  $a_T$  is the input to the system and it is modeled on the target acceleration as a random forcing function with exponential correlation. The auto-correlation function of  $a_T$  is given by eq. (1) and the statistics are as follows:

$$E[a_T] = 0, [a_T^2] = (100ft\ s^{-2})^2$$

$$E[a_T(t)a_T(s)] = E[a_T^2]e^{\frac{-|t-s|}{\tau}} \quad (1)$$

The initial lateral position  $y$  is zero and the initial lateral velocity  $v$  is assumed to be normally distributed with the following statistics to capture the launching error:

$$E[y(t_0)] = 0, E[v(t_0)] = 0, E[y(t_0)^2] = 0, E[y(t_0)v(t_0)] = 0, E[v(t_0)^2] = (200ft\ s^{-1})^2$$

## 2. Theory

The estimator is based on the Kalman-Bucy filter, a continuous-time linear minimum variance estimator. The filter is an optimal estimator and a conditional mean estimator for Gauss-Markov process. If the additive noise is uncorrelated the filter is equivalent to Kalman Filter in structure.

In continuous-time filter there is no separation between propagation and the measurement update steps and they occur simultaneously as in eq. (2).

$$d\hat{x}(t) = F(t)\hat{x}(t)dt + K(t)(dz(t) - H(t)\hat{x}(t)dt) \quad (2)$$

Here,  $F(t)$  is the system matrix and  $H(t)$  is the observation model.  $dz(t)$  is the measurement at time  $t$  and the Kalman gain  $K(t)$  given by eq. (3) is used to incorporate the measurement in the estimate.

$$K(t) = P(t)H(t)^TV(t)^{-1} \quad (3)$$

Here,  $P(t)$  is the error variance estimate and  $V(t)$  is the power spectral density matrix of the additive white noise  $n$  in the measurement process.

The dynamics of the process error variance estimate  $P(t)$  is determined to be the famous continuous-time algebraic Riccati equation as in the eq. (4).

$$\dot{P}(t) = F(t)P(t) + P(t)F(t)^T - P(t)H(t)^TV(t)^{-1}H(t)P(t) + G(t)W(t)G(t)^T \quad (4)$$

Here,  $G(t)$  is the measurement noise model and  $W$  is the power spectral density matrix corresponding to the additive process noise  $w_{a_T}$ .

## 3. Algorithm

A state-space model of the dynamics is developed for use in the estimator and the state-space equation is as follows:

$$\dot{x} = Fx + Gw_{a_T}, x = \begin{bmatrix} \dot{y} \\ \dot{v} \\ \dot{a}_T \end{bmatrix}, F = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & -1 \\ 0 & 0 & -\frac{1}{\tau} \end{bmatrix}, G = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

An additional state  $a_T$  based on the input is added to the state-space equations of the missile to track the dynamics of the exponentially correlated input process  $a_T$ . This is based on the fact continuous-time Gauss-Markov process evolve exponentially with time and the input in this case target acceleration  $a_T$  is modelled as an exponentially correlated process. The state-space model of the measurement process is

$$z = Hx + n, H = \begin{bmatrix} \frac{1}{V_c(t_f - t)} & 0 & 0 \end{bmatrix}$$

In the state-space equations the random variables are denoted by small letters and accordingly the deterministic components are labeled with capital letters.

The initial a priori error covariance is determined from the problem parameters as follows:

$$P(0) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & (200ft\ s^{-1})^2 & 0 \\ 0 & 0 & (100ft\ s^{-2})^2 \end{bmatrix}$$

Similarly, the power spectral density matrices for the additive process noise  $W$  and the additive non-stationary white measurement noise  $V$  are determined to be

$$V = [R_1 + \frac{R_2}{(t_f - t)^2}], W = [(100ft\ s^{-2})^2]$$

Once, the estimator performs well on the Gauss-Markov process then estimator is used on a more realistic process modeled after the random telegraph signal. In this improved process model, the value of input  $a_T$  is  $\pm 100ft\ s^{-2}$  that changes sign at random times given by a Poisson probability. The initial state of the input  $a_T(0)$  is assumed to be  $\pm a_T$  with a probability of 0.5 and  $a_T(t)$  changes polarity at Poisson times. The probability of  $k$  sign changes in a time interval of length  $T$ ,  $P(k(T))$  is given by

$$P(k(T)) = \frac{(\lambda T)^k e^{-\lambda T}}{k!}$$

where  $\lambda$  is the rate and is equal to  $0.25\ s^{-1}$  for our problem. The mean of the input acceleration  $a_T$  based on the random telegraph signal is found to be  $E[a_T] = 0$  and the auto-correlation function of  $a_T$  for this model is given by eq. (5)

$$R_{a_T a_T}(t, s) = a_T^2 e^{-2\lambda|t-s|} \quad (5)$$

In this case  $\frac{1}{\tau} = 2\lambda = 0.5\ s^{-1}$ , then as we can see the auto-correlation function of the Gauss-Markov process given by equation (1) is same as the random telegraph signal given by eq. (5) where  $t$  and  $s$  are correlation time variables. Also, the mean of both the process are zero.

The theory for developing estimators for Gauss-Markov process is well understood and once we have an estimator for the Gauss-Markov process the main objective is use to the estimator with a more realistic model of the increment process  $a_T$  given by random telegraph signal.

A discretized implementation of the filter with discretization time  $dt = 0.01\ s$  is run till the terminal time  $t_f = 10\ s$ . The power spectral density matrices  $W$  and  $V$  are divided by the discretization time  $dt$  as in eq. (6) to account for the discretization.

$$W_{discrete} = \frac{W}{dt}, V_{discrete} = \frac{V}{dt} \quad (6)$$

In the discretized implementation, the base of  $\delta$  is made to be  $dt$  and decreasing the amplitude of the  $\delta$  by a factor of  $dt$  as in eq. (6) ensures that the area or magnitude of the impulse remains unchanged. Essentially the power of the discretized system is modified so that the energy of the discretized system is same as that of the original continuous-time system.

The switching times for the random telegraph signal is given by Poisson probability and the next switching time can be computed from the previous switching time as in eq. (7).

$$t_{n+1} = t_n - \frac{1}{\lambda} \ln(U) \quad (7)$$

where  $t_n$  is the time of  $n^{\text{th}}$  sign change and  $t_{n+1}$  is the time for the following sign change.  $U$  is a random variable with uniform density function defined on the range  $[0, 1]$ . In the discrete time implementation of the estimator the switching times  $t_n$  are rounded to the closest valid discrete time intervals.

The results and performance of the estimator on Gauss-Markov process and the increment process produced by random telegraph signal is discussed in the following sections 4 and 5.

## 4. Results

The estimator is run for 1000 different trial runs for terminal time  $t_f = 10\text{ s}$  and the results are presented as figures. The Kalman filter gains  $K(t)$ , the actual RMS estimation error and the Kalman filter estimated RMS error are presented in Fig. 1 and 2 respectively. The error variance corresponding to the states are presented in 3. The true state values and the filter estimates are presented in Fig. 5, 6 and 7 corresponding to lateral position, velocity and acceleration for Gauss-Markov process. The measurements  $z$  corresponding to presented trial is show in Fig. 4. The error in state estimates  $e = x - \hat{x}$  corresponding to the run in the previous results are present in Fig. 8, 9 and 10 along side the estimated error standard deviation  $\pm 1\sigma$ .

In the same way, the performance of the filter for random telegraph signals are illustrated for one of the experimental runs. The Fig. 11 represents the measurements  $z$  for the run. The Fig. 12, 13 and 14 compare the true values of the states with the estimated values for random telegraph signal. In addition the error  $e$  in the estimates are presented in Fig. 15, 16 and 17 for all the states along with their estimated  $\pm 1\sigma$  error standard deviations.

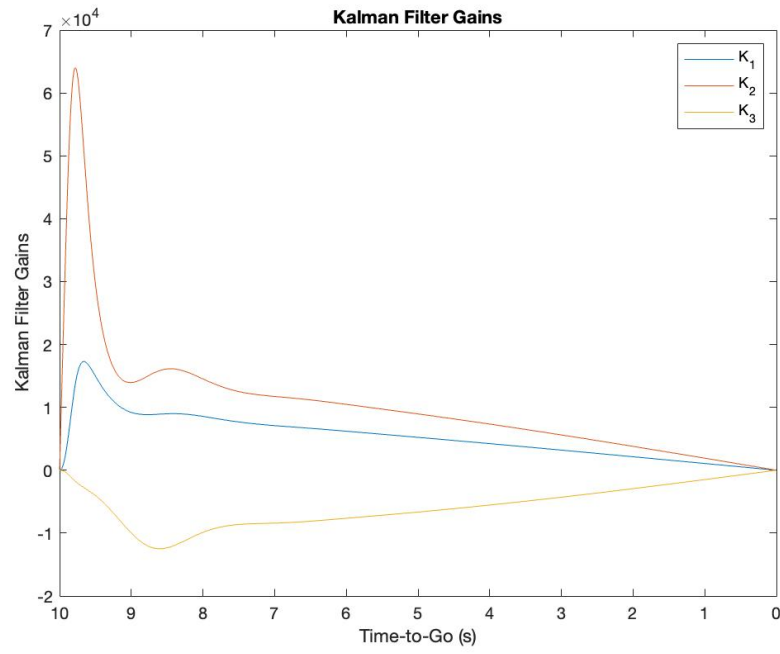


Figure 1: Kalman Filter Gain History

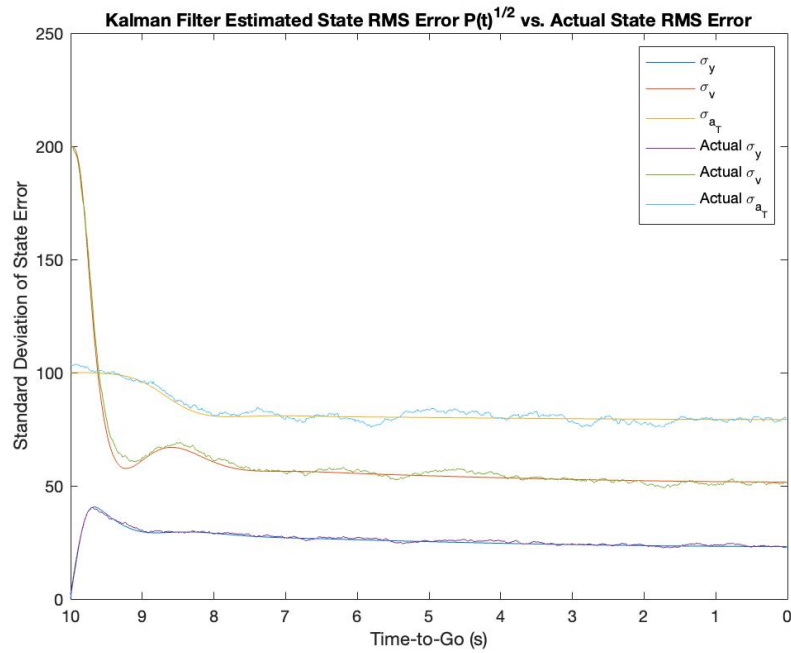


Figure 2: Comparison of Kalman filter estimated state RMS error and actual state RMS error in estimation.

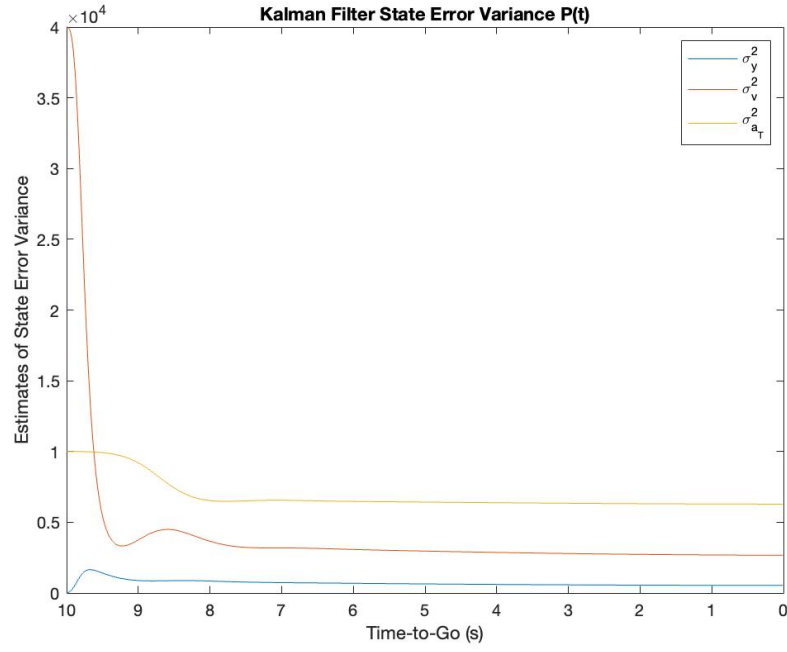


Figure 3: Kalman filter estimated state error variance  $P(t)$

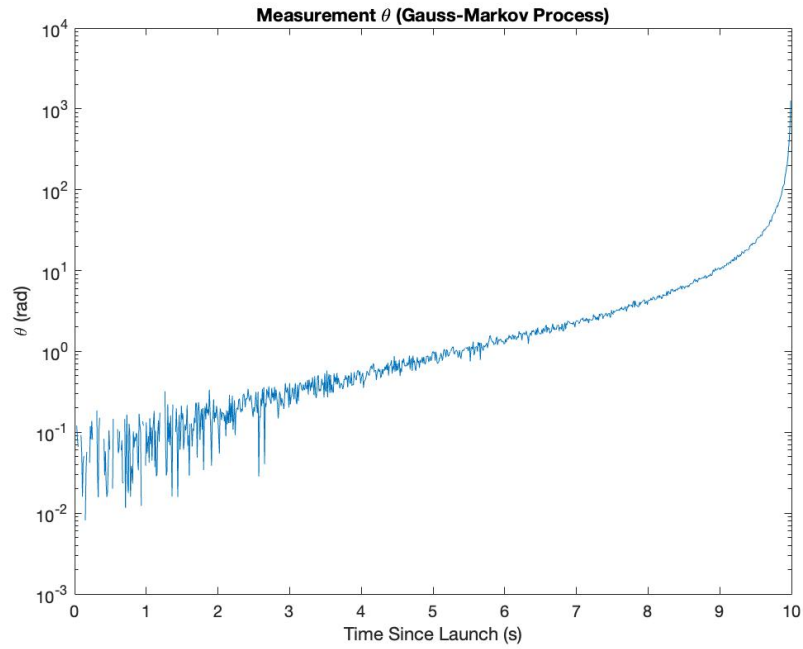


Figure 4: Measurements  $z$  for Gauss-Markov process.

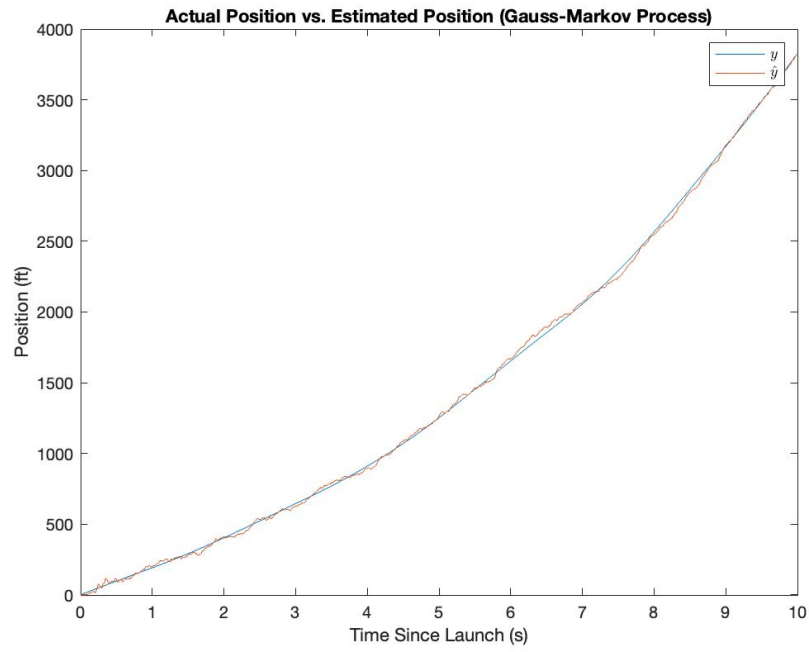


Figure 5: Comparison of actual position and estimated position for Gauss-Markov process.

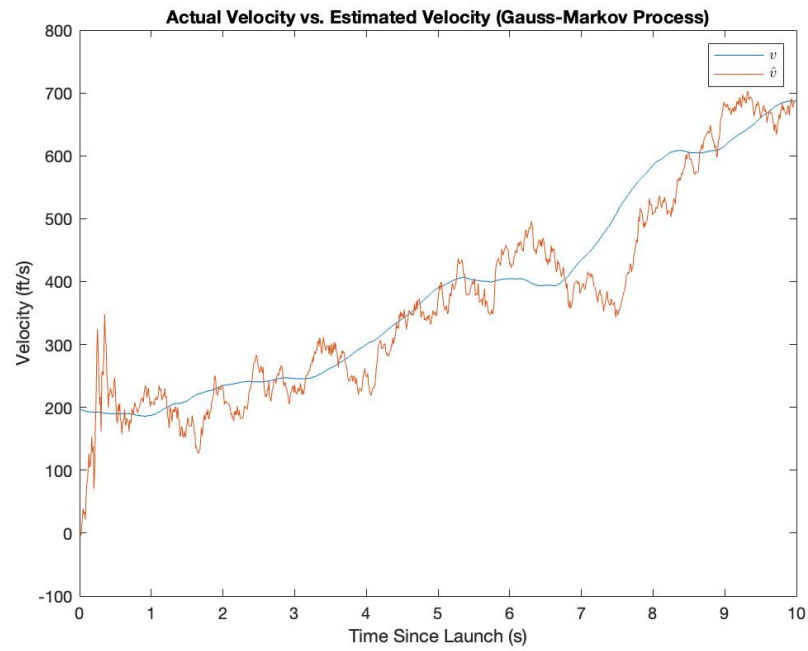


Figure 6: Comparison of actual velocity and estimated velocity for Gauss-Markov process.

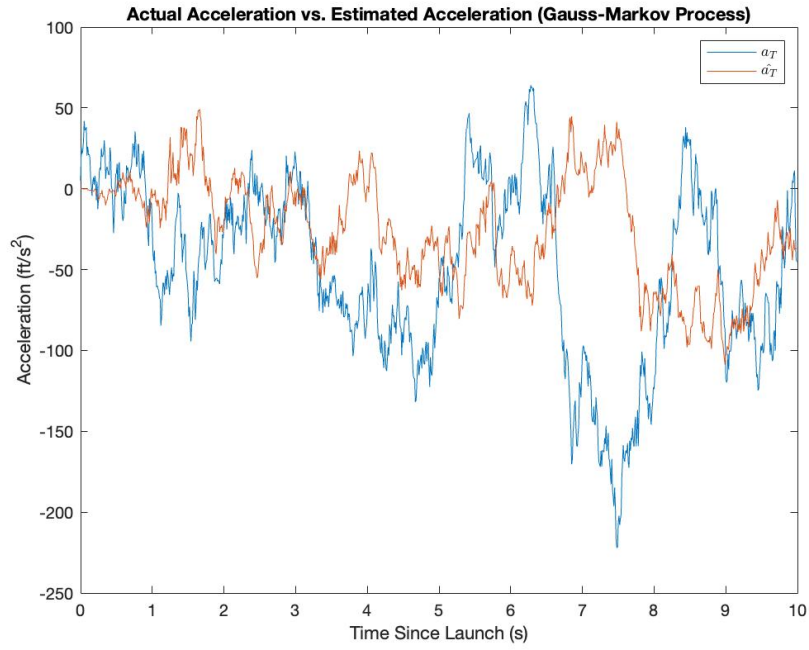


Figure 7: Comparison of actual acceleration and estimated acceleration for Gauss-Markov process.

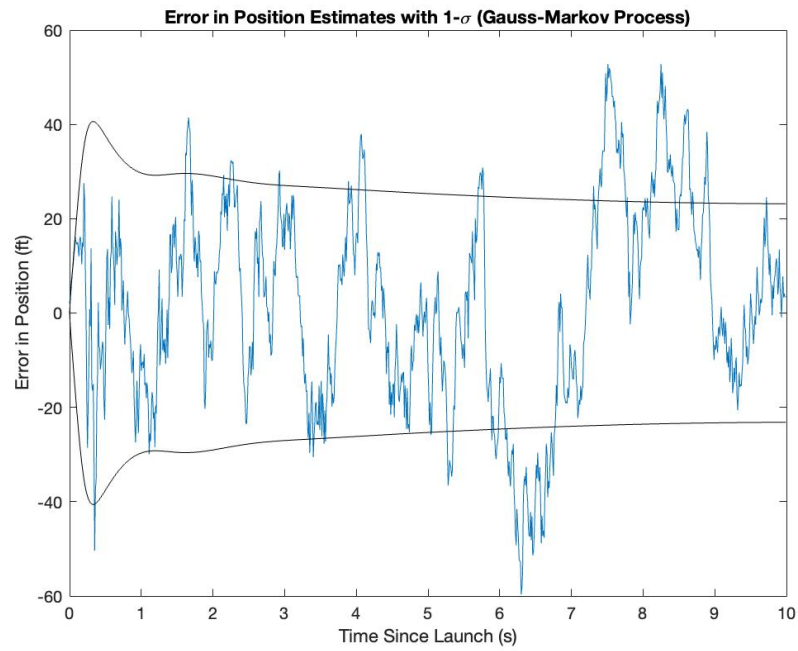


Figure 8: Error in position estimates with 1- $\sigma$  for Gauss-Markov process.



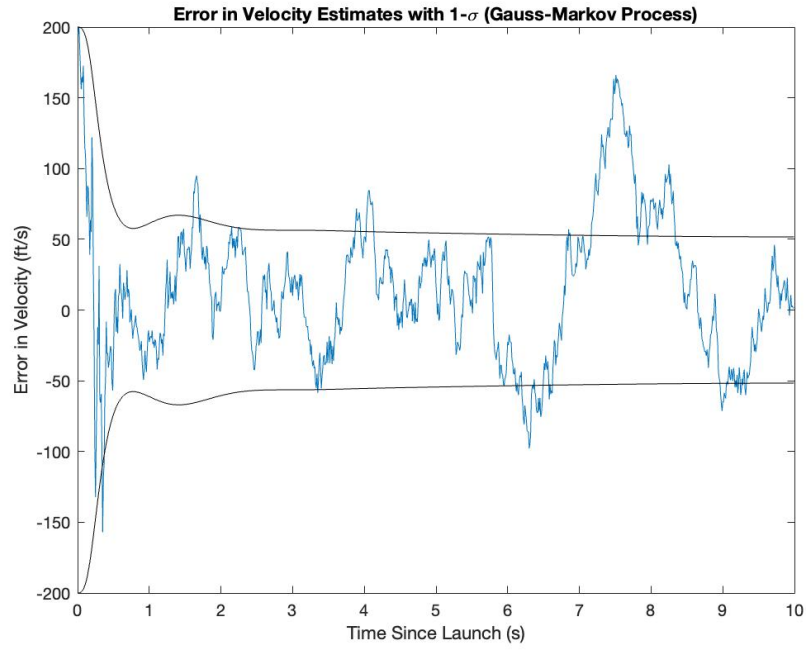


Figure 9: Error in velocity estimates with 1- $\sigma$  for Gauss-Markov process.

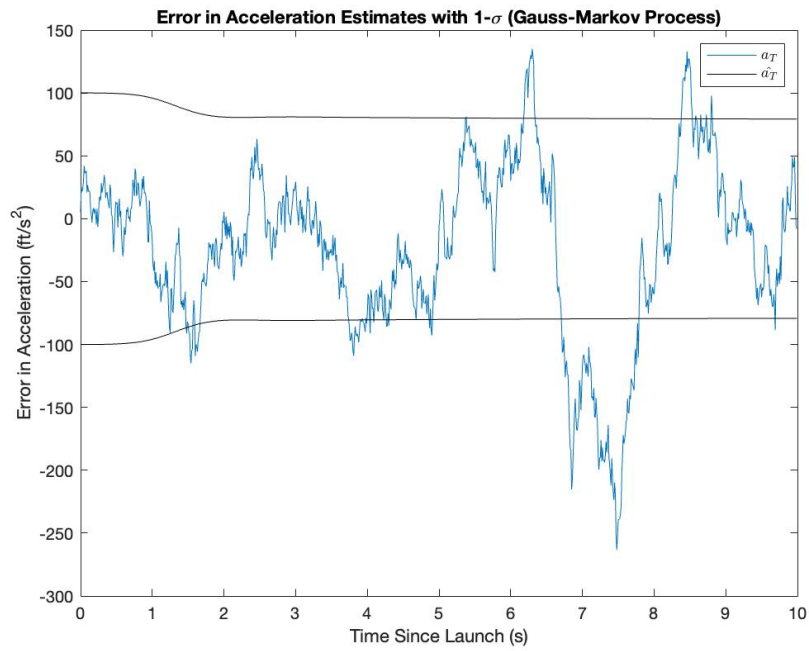


Figure 10: Error in acceleration estimates with 1- $\sigma$  for Gauss-Markov process.

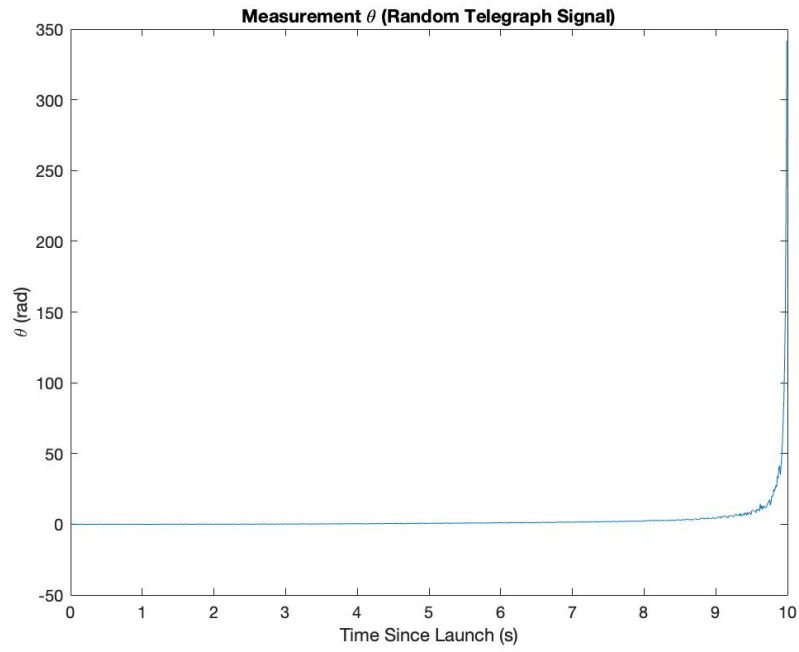


Figure 11: Measurements  $z$  for random telegraph signal.

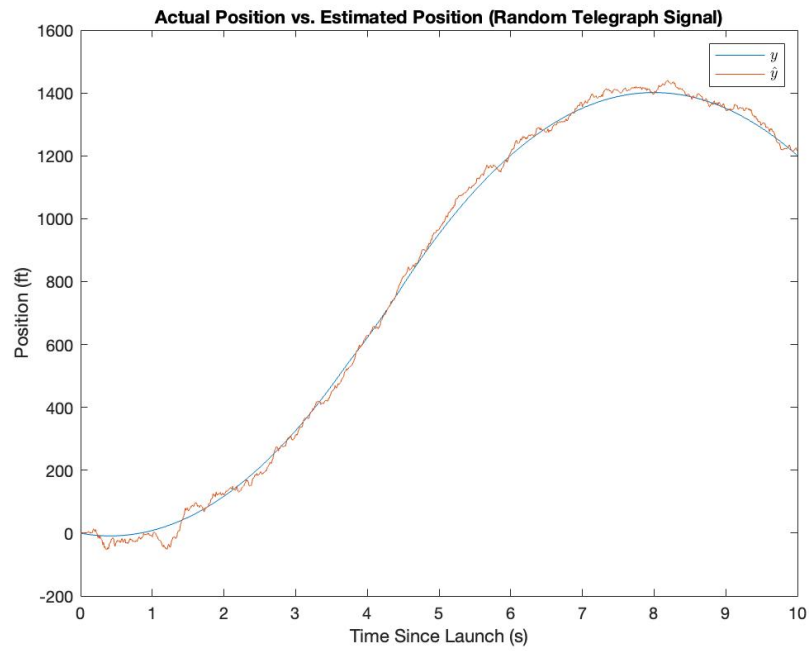


Figure 12: Comparison of actual position and estimated position for random telegraph signal.

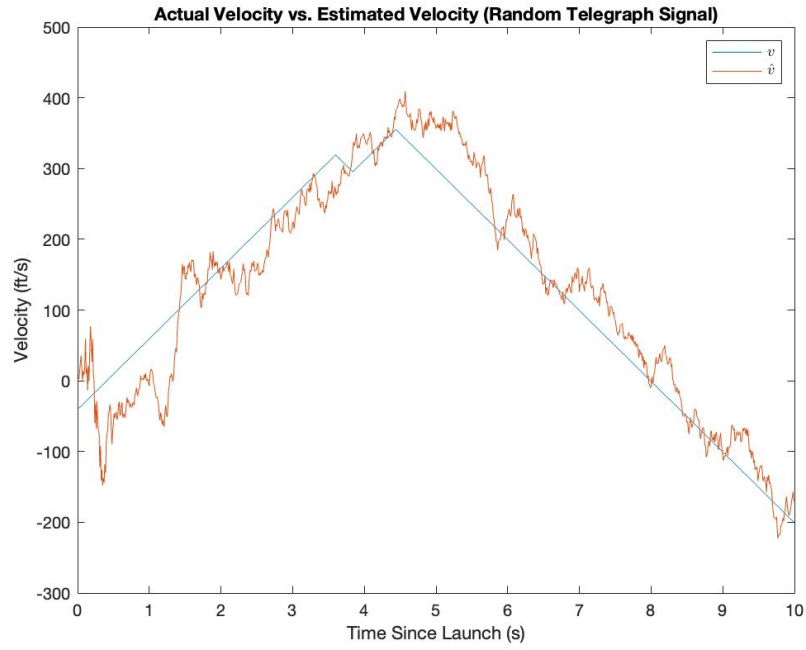


Figure 13: Comparison of actual velocity and estimated velocity for random telegraph signal.

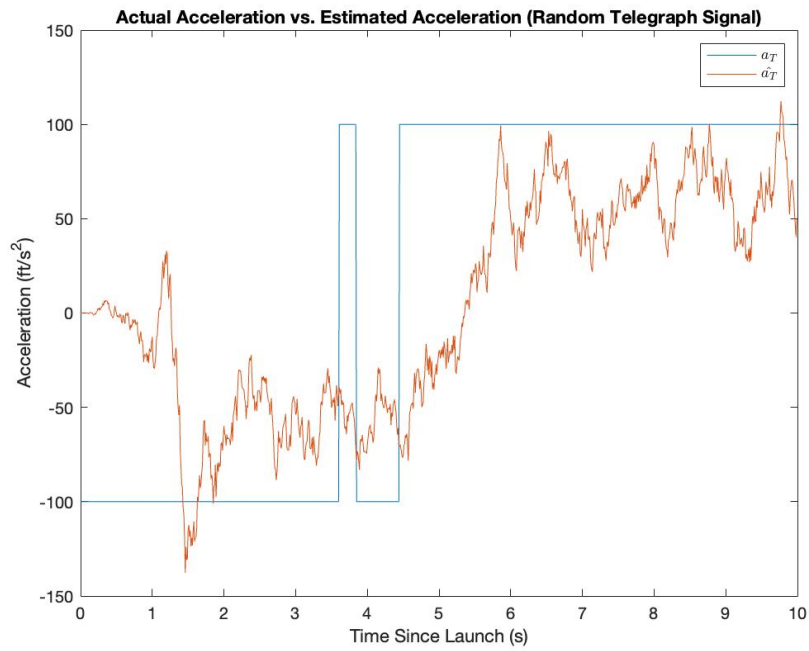


Figure 14: Comparison of actual acceleration and estimated acceleration for random telegraph signal.

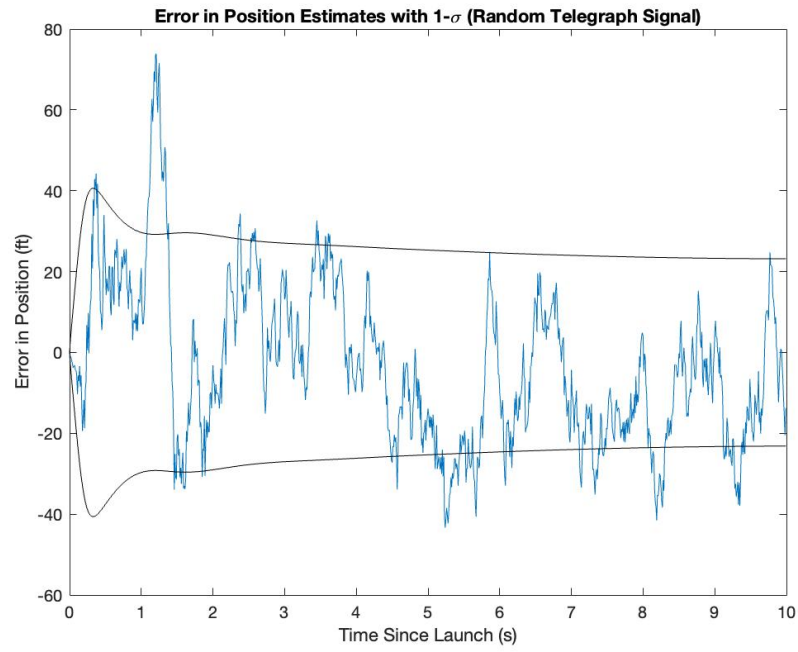


Figure 15: Error in position estimates with  $1-\sigma$  for random telegraph signal.

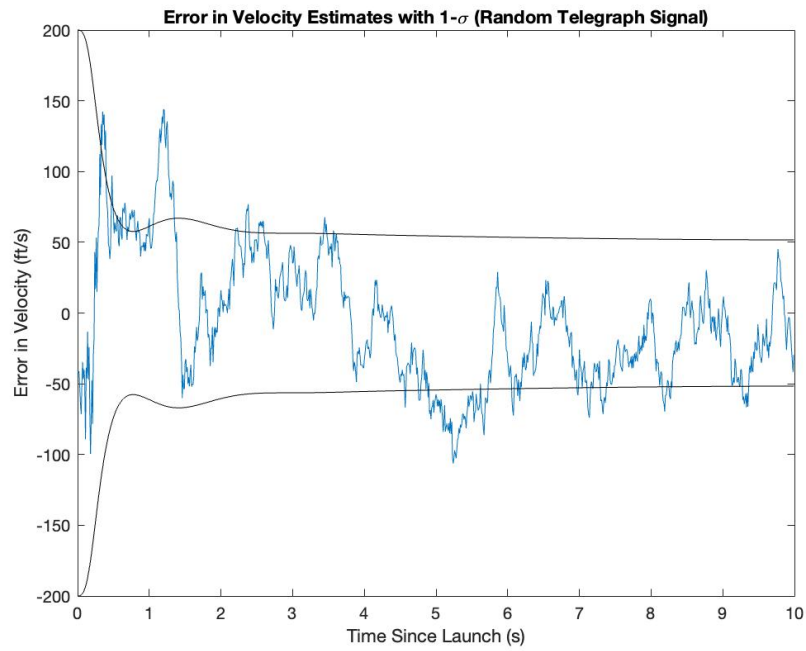


Figure 16: Error in velocity estimates with  $1-\sigma$  for random telegraph signal.

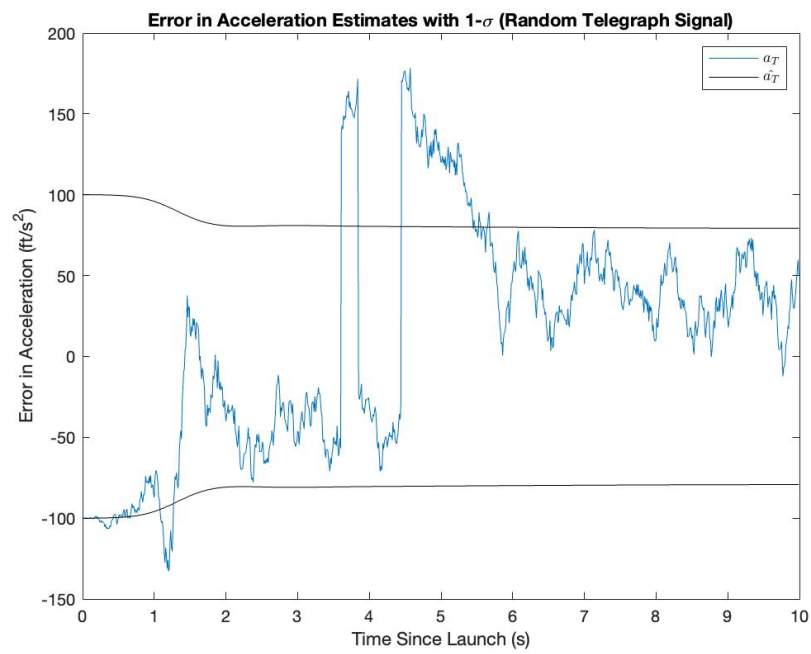


Figure 17: Error in acceleration estimates with 1- $\sigma$  for random telegraph signal.

## 5. Monte Carlo Analysis

We perform a Monte Carlo simulation with 1000 realizations of the Kalman filter to understand the actual mean error and error variance through simulation. The error covariance from the simulation is then compared with the error covariance computed by the Kalman filter to understand the performance of the filter.

A Monte Carlo simulation is to be constructed to find the ensemble averages over a set of realizations. Let  $e^l(t)$  represent the actual error for realization  $l$ , then ensemble average of error  $e^l(t)$  which produces the actual mean error is given in equation (8).

$$e^{ave}(t) = \frac{1}{N_{ave}} \sum_{l=1}^{N_{ave}} e^l(t) \quad (8)$$

where  $N_{ave} = 1000$  is the number of realizations. It is expected that the  $e^{ave}(t) \approx 0$  for all  $t$  for an unbiased estimator. The ensemble average producing the actual error variance  $P^{ave}(t)$  is given by equation (9).

$$P^{ave}(t) = \frac{1}{N_{ave} - 1} \sum_{l=1}^{N_{ave}} [e^l(t) - e^{ave}(t)][e^l(t) - e^{ave}(t)]^T \quad (9)$$

The matrix  $P^{ave}(t)$  should be close to  $P(t)$  computed in the Kalman filter algorithm,  $P^{ave}(t) - P(t) \approx 0$  for all  $t$  to ensure that the Kalman filter model and algorithm are correct. The Kalman filter error variance  $P(t)$  and the actual error variance  $P^{ave}(t)$  for the Monte Carlo runs corresponding to the Gauss-Markov process is presented in Fig. 18. A similar comparison of  $P(t)$  and  $P^{ave}(t)$  for random telegraph signal is present in Fig. 19.

Finally, it is shown that the residual process is an white process and the independence of the residuals is shown as in equation (10), where the ensemble average for the correlation of the residuals at time  $t$  and  $t_i$  is zero when  $t \neq t_i$ .

$$\frac{1}{N_{ave}} \sum_{l=1}^{N_{ave}} r^l(t_i) r^l(t)^T \approx 0, \forall t \neq t_i \quad (10)$$

To illustrated that the residual process is uncorrelated in time, correlation of the residual process for 7<sup>th</sup> second with time  $t$  is shown in Fig. 20. In addition the power spectral density (PSD) of the residual process shown in Fig. 21 indicates that it is a white process.

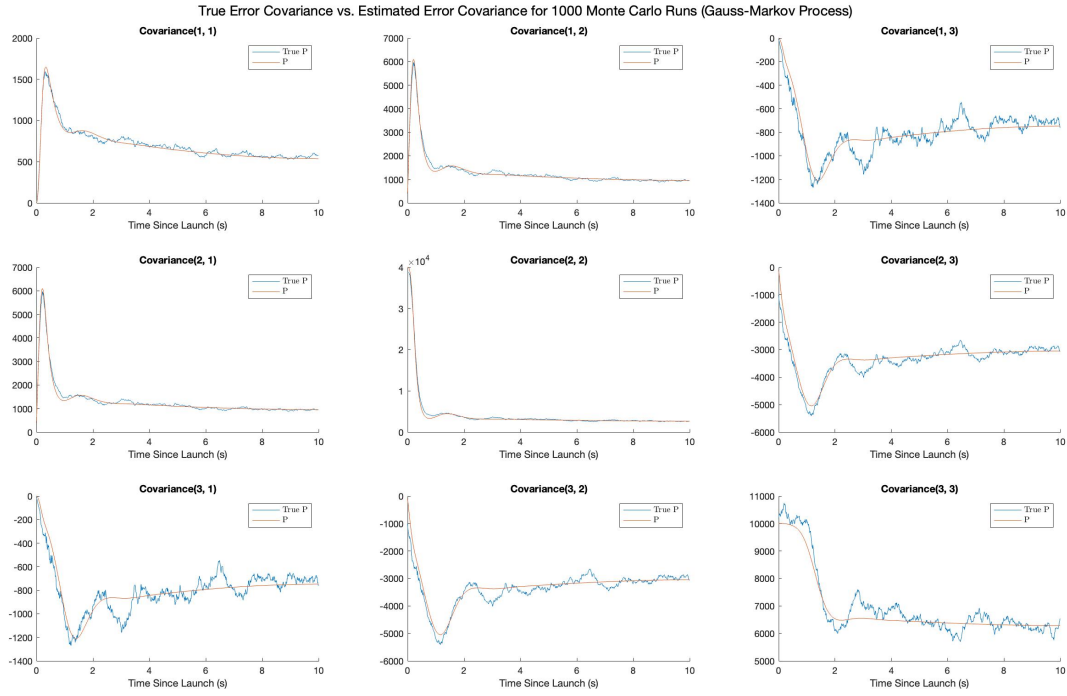


Figure 18: Comparison of actual error covariance  $P^{ave}(t)$  with the Kalman filter error covariance estimates  $P(t)$  for Gauss-Markov process.

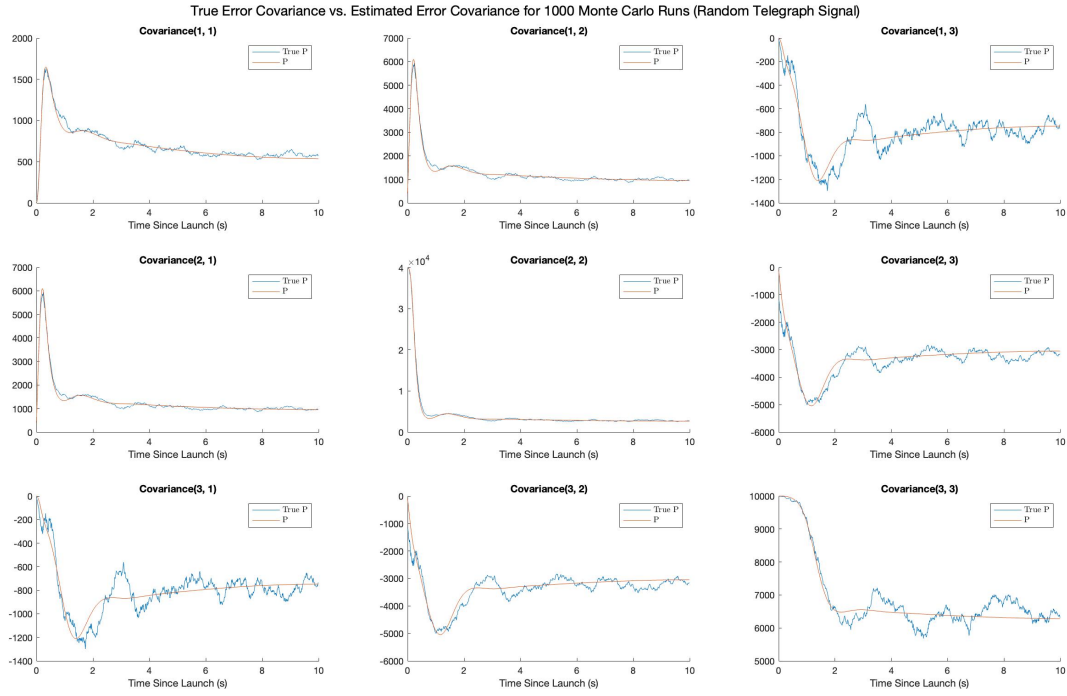


Figure 19: Comparison of actual error covariance  $P^{ave}(t)$  with the Kalman filter error covariance estimates  $P(t)$  for random telegraph signal.



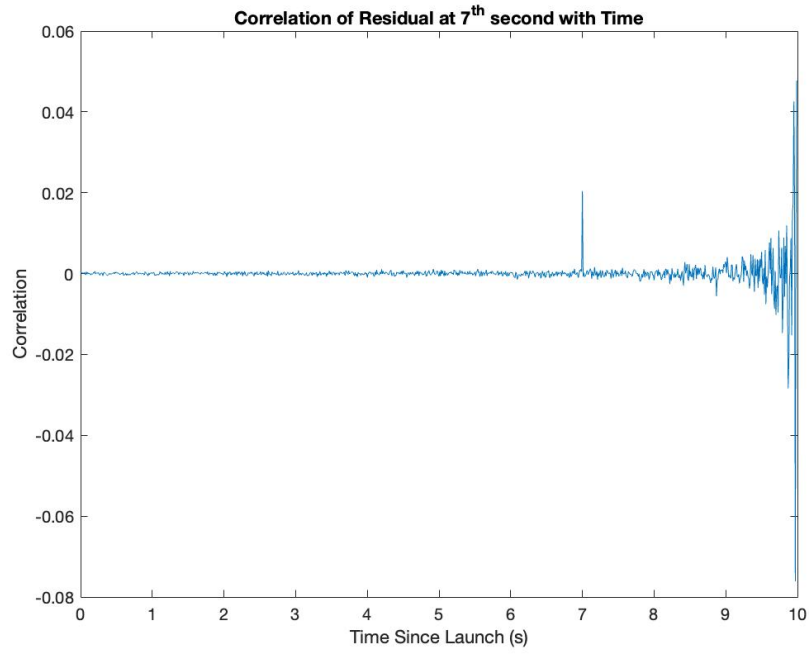


Figure 20: Correlation of the residual process at time  $t = 7$  s with time.

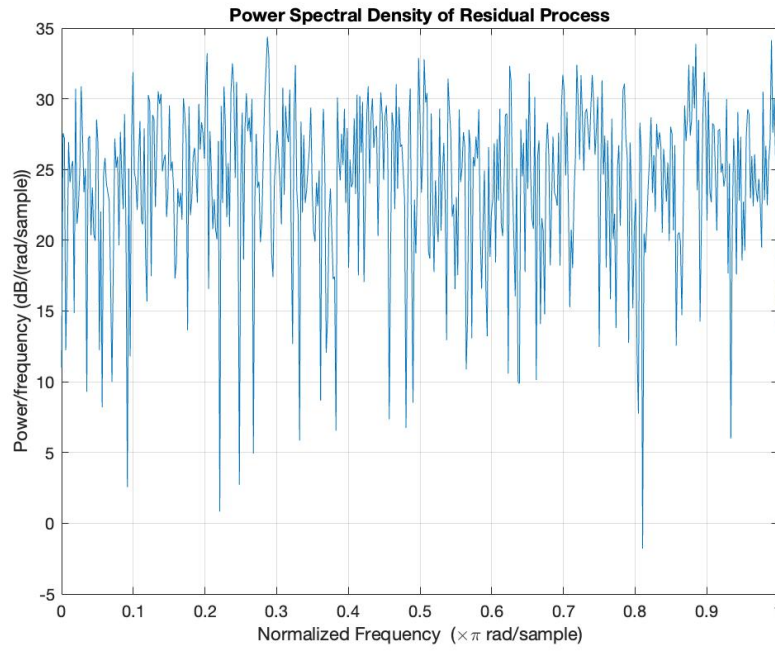


Figure 21: Power spectral density of the residual process shows that it is a white process.

## 6. Conclusion

In this project Kalman-Bucy filter was used to estimate the lateral position, velocity and acceleration of a missile launched from an aircraft. The estimator was developed on a Gauss-Markov process and tested on an random telegraph signal input with zero mean and similar correlation. This kind of random telegraph signal is more realistic, the simulations show that the estimator performs equally well on this kind of signal. Monte Carlo analysis was performed to check the correctness of the estimator and it can be seen that the estimator behaves well on the random telegraph signal. From the experiments it can be seen that using the theory for Gauss-Markov process, estimators can be designed for a more realistic model of the increment process  $a_T$  given by random telegraph signal.

## 7. MATLAB Code

```
1  clc ;
2  close all ;
3  clear ;
4
5  tau = 2;
6  Y_0_MEAN = 0;
7  V_0_MEAN = 0;
8  At_0_MEAN = 0;
9
10 V_VAR = 200^2;
11 At_VAR = 100^2;
12
13 Vc = 300;
14 Tf = 10;
15 R1 = 15e-6;
16 R2 = 1.67e-3;
17
18 AtTlg = 100;
19
20 p_0 = zeros(3, 3);
21 p_0(2, 2) = V_VAR;
22 p_0(3, 3) = At_VAR;
23
24 dt = 0.01;
25
26 ts = linspace(0, Tf, Tf / dt + 1);
27 ts = ts(1:end-1);
28
29 F = [0, 1, 0; 0, 0, -1; 0, 0, -1/tau];
30 G = [0; 0; 1];
31 W = At_VAR;
32
```

```

33 [filterKs , filterPs] = computePGM(ts , p_0 , F, G, W);
34
35 n = 1000;
36 trueXHistory= zeros(n, 3, size(ts , 2));
37 trueYHistory= zeros(n, 1, size(ts , 2));
38 xHatHistory= zeros(n, 3, size(ts , 2));
39 residualHistory = zeros(n, 1, size(ts , 2));
40
41 trueXHistoryRTS = zeros(n, 2, size(ts , 2));
42 trueYHistoryRTS = zeros(n, 1, size(ts , 2));
43 xHatHistoryRTS = zeros(n, 3, size(ts , 2));
44 residualHistoryRTS = zeros(n, 1, size(ts , 2));
45 trueAtHistoryRTS = zeros(n, 1, size(ts , 2));
46
47 for i = 1:n
48     [trueXs, trueYs] = simDynamics(ts , F, G, W);
49     trueXHistory(i, :, :) = trueXs;
50     trueYHistory(i, :, :) = trueYs;
51     [xHats, residuals] = myKfInnovate(ts , filterKs , trueYs , F);
52     xHatHistory(i, :, :) = xHats;
53     residualHistory(i, :, :) = residuals;
54
55     [trueXsRTS, trueYsRTS, atRTS] = simTelegraphDynamics(dt , ts ,
        AtTlg , Tf);
56     trueXHistoryRTS(i, :, :) = trueXsRTS;
57     trueYHistoryRTS(i, :, :) = trueYsRTS;
58     trueAtHistoryRTS(i, :, :) = atRTS;
59     [xHatsRTS, residualsRTS] = myKfInnovate(ts , filterKs , trueYsRTS ,
        F);
60     xHatHistoryRTS(i, :, :) = xHatsRTS;
61     residualHistoryRTS(i, :, :) = residualsRTS;
62 end
63
64 [trueErrorVariance] = monteCarloAnalysis(trueXHistory , xHatHistory);
65 [trueErrorVarianceRTS] = monteCarloAnalysis(cat(2, trueXHistoryRTS ,
    trueAtHistoryRTS) , xHatHistoryRTS);
66
67 plotResults1(ts , Tf, filterKs , filterPs , trueErrorVariance)
68 plotResults2(ts , trueYs , trueXs , xHats , filterPs)
69 plotResults3(ts , trueYsRTS , cat(1, trueXsRTS , atRTS), xHatsRTS ,
    filterPs)
70 plotResults4(ts , trueErrorVariance , trueErrorVarianceRTS , filterPs)
71 plotResults5(ts , dt , residualHistory)
72

```

```

73 function [ks, ps] = computePGM(ts, p_0, F, G, W)
74     dt = ts(2) - ts(1);
75     ps = zeros(1, 3, 3);
76     ps(1, :, :) = p_0;
77
78     ks = zeros(3, 1);
79     ks(:, 1) = [0; 0; 0];
80
81     R1 = 15e-6;
82     R2 = 1.67e-3;
83     Vc = 300;
84     Tf = 10;
85
86     for t = ts
87         V = R1 + R2/((Tf-t)^2);
88
89         H = [1/(Vc*(Tf - t)), 0, 0];
90         ks(:, end+1) = squeeze(ps(end, :, :)) * H' / V;
91
92         p_dot = F * squeeze(ps(end, :, :)) + squeeze(ps(end, :, :))
93             * F' - ks(:, end) * H * squeeze(ps(end, :, :)) + G * W * G
94             ' ;
95         ps(end+1, :, :) = squeeze(ps(end, :, :)) + p_dot * dt;
96     end
97
98     ks = ks(:, 2:end);
99     ps = ps(2:end, :, :);
100
101 end
102
103 function [xs, ys] = simDynamics(ts, F, G, W)
104     dt = ts(2) - ts(1);
105
106     R1 = 15e-6;
107     R2 = 1.67e-3;
108     Vc = 300;
109     Tf = 10;
110
111     Y_0_MEAN = 0;
112
113     V_0_MEAN = 0;
114     V_VAR = 200^2;
115
116     At_0_MEAN = 0;

```

```

115     At_VAR = 100^2;
116
117     xs = [[Y_0_MEAN; normrnd(V_0_MEAN, sqrt(V_VAR)); normrnd(
118         At_0_MEAN, sqrt(At_VAR))]];
119     ys = [0];
120
121     w_at = normrnd(0, sqrt(W/dt), size(ts));
122     i = 1;
123     V = R1 + R2./((Tf-ts).^2);
124     v = normrnd(0, sqrt(V/dt));
125     for t = ts
126         H = [1/(Vc*(Tf - t)), 0, 0];
127         x_dot = F * xs(:, end) + G * w_at(i);
128         xs(:, end+1) = xs(:, end) + x_dot*dt;
129         ys(end+1) = H * xs(:, end) + v(i);
130         i = i + 1;
131     end
132
133     xs = xs(:, 2:end);
134     ys = ys(2:end);
135 end
136
137 function [xHats, residuals] = myKfInnovate(ts, ks, zs, F)
138     dt = ts(2) - ts(1);
139     Vc = 300;
140     Tf = 10;
141
142     xHats = [[0; 0; 0]];
143     residuals = [0];
144     i = 1;
145     for t = ts
146         H = [1/(Vc*(Tf - t)), 0, 0];
147         residual = zs(i) - H * xHats(:, end);
148         x_dot = F*xHats(:, end) + ks(:, i) * residual;
149         xHats(:, end+1) = xHats(:, end) + x_dot * dt;
150         residuals(end+1) = residual;
151         i = i + 1;
152     end
153
154     xHats = xHats(:, 2:end);
155     residuals = residuals(2:end);
156 end

```

```

157 function trueErrorVariance = monteCarloAnalysis(trueXHistory ,
    xHatHistory)
158     error = trueXHistory - xHatHistory;
159     meanError = squeeze(mean(error , 1));
160     trueErrorVariance = zeros(size(xHatHistory , 3), 3, 3);
161     for i = 1:size(xHatHistory , 3)
162         for j = 1:size(xHatHistory , 1)
163             trueErrorVariance(i , :, :) = squeeze(trueErrorVariance(i ,
                :, :)) + ((error(j , :, i)' - meanError(:, i)) * ((
                    error(j , :, i)' - meanError(:, i))'));
164         end
165     end
166     trueErrorVariance = trueErrorVariance / (size(xHatHistory , 1)-1);
167
168 end
169
170 function aRTS = generateRTS(dt , Tf , AtTlg)
171     t = 0;
172     ts = [];
173     lambda = 0.25;
174     while t < Tf
175         t = t - log(unifrnd(0 , 1))/lambda;
176         ts(end+1) = t;
177     end
178
179     ts = unique(roundn(ts , log10(dt)));
180
181     sampleT = linspace(0 , Tf , Tf / dt + 1);
182
183     [locA , locB] = ismember(ts , sampleT);
184     locB = locB(locB ~= 0);
185
186     switchT = zeros(size(sampleT));
187     switchT(locB) = 1;
188
189     aRTS = zeros(size(sampleT)-1);
190     flip = binornd(1 , 0.5);
191     if(flip)
192         sign = 1;
193     else
194         sign = -1;
195     end
196
197     for i = 1:size(aRTS , 2)

```

```

198         if(switchT(i) == 1)
199             sign = sign * -1;
200         end
201         aRTS(i) = sign * AtTlg;
202     end
203 end
204
205 function [xs, ys, atRTS] = simTelegraphDynamics(dt, ts, AtTlg, Tf)
206     atRTS = generateRTS(dt, Tf, AtTlg);
207     F = [0, 1; 0, 0];
208     G = [0; -1];
209
210     dt = ts(2) - ts(1);
211
212     R1 = 15e-6;
213     R2 = 1.67e-3;
214     Vc = 300;
215     Tf = 10;
216
217     Y_0_MEAN = 0;
218
219     V_0_MEAN = 0;
220     V_VAR = 200^2;
221
222     xs = [[Y_0_MEAN; normrnd(V_0_MEAN, sqrt(V_VAR))]];
223     ys = [0];
224
225     i = 1;
226     V = R1 + R2./((Tf-ts).^2);
227     v = normrnd(0, sqrt(V/dt));
228     for t = ts
229         H = [1/(Vc*(Tf - t)), 0];
230         x_dot = F * xs(:, end) + G * atRTS(i);
231         xs(:, end+1) = xs(:, end) + x_dot*dt;
232         ys(end+1) = H * xs(:, end) + v(i);
233         i = i + 1;
234     end
235
236     xs = xs(:, 2:end);
237     ys = ys(2:end);
238 end
239
240 function residualCorrelations = residualAnalysis(timeToAnalysis,
    residualHistory, dt)

```

```

241 residualT = residualHistory(:, :, (timeToAnalysis/dt)+1);
242 residualCorrelations = zeros(size(residualHistory, 2), size(
    residualHistory, 3));
243 for i = 1:size(residualHistory, 1)
244     for j = 1:size(residualHistory, 3)
245         residualCorrelations(j) = residualCorrelations(j) + (
            residualHistory(i, 1, j) * residualT(i, 1)');
246     end
247 end
248 residualCorrelations = residualCorrelations / size(
    residualHistory, 1);
249 end
250
251 function plotResults1(ts, Tf, filterKs, filterPs, trueErrorVariance)
252     figure(1)
253     ax1 = axes;
254     plot(Tf - ts, filterKs(1, :))
255     hold on
256     plot(Tf - ts, filterKs(2, :))
257     plot(Tf - ts, filterKs(3, :))
258     hold off
259     set(ax1, 'Xdir', 'reverse')
260     xlabel('Time-to-Go (s)')
261     ylabel('Kalman Filter Gains')
262     legend('K_{1}', 'K_{2}', 'K_{3}')
263     set(get(gca, 'Title'), 'String', 'Kalman Filter Gains');
264
265     figure(2)
266     ax1 = axes;
267     plot(Tf - ts, sqrt(filterPs(:, 1, 1)))
268     hold on
269     plot(Tf - ts, sqrt(filterPs(:, 2, 2)))
270     plot(Tf - ts, sqrt(filterPs(:, 3, 3)))
271     plot(Tf - ts, sqrt(trueErrorVariance(:, 1, 1)))
272     plot(Tf - ts, sqrt(trueErrorVariance(:, 2, 2)))
273     plot(Tf - ts, sqrt(trueErrorVariance(:, 3, 3)))
274     hold off
275     set(ax1, 'Xdir', 'reverse')
276     xlabel('Time-to-Go (s)')
277     ylabel('Standard Deviation of State Error')
278     legend('\sigma_{y}', '\sigma_{v}', '\sigma_{a-T}', 'Actual \
        sigma_{y}', 'Actual \sigma_{v}', 'Actual \sigma_{a-T}')
279     set(get(gca, 'Title'), 'String', 'Kalman Filter Estimated State
        RMS Error P(t)^{1/2} vs. Actual State RMS Error');

```



```

280
281     figure(21)
282     ax1 = axes;
283     plot(Tf - ts, filterPs(:, 1, 1))
284     hold on
285     plot(Tf - ts, filterPs(:, 2, 2))
286     plot(Tf - ts, filterPs(:, 3, 3))
287     hold off
288     set(ax1, 'Xdir', 'reverse')
289     xlabel('Time-to-Go (s)')
290     ylabel('Estimates of State Error Variance')
291     legend('\sigma_{y}^2', '\sigma_{v}^2', '\sigma_{a_{T}}^2')
292     set(get(gca, 'Title'), 'String', 'Kalman Filter State Error
        Variance P(t)');
293
294 end
295
296 function plotResults2(ts, trueYs, trueXs, xHats, filterPs)
297     figure(3)
298     plot(ts, trueYs)
299     xlabel('Time Since Launch (s)')
300     ylabel('\theta (rad)')
301     set(gca, 'YScale', 'log')
302     set(get(gca, 'Title'), 'String', 'Measurement \theta (Gauss-
        Markov Process)');
303
304     figure(4)
305     plot(ts, trueXs(1, :))
306     hold on
307     plot(ts, xHats(1, :))
308     hold off
309     xlabel('Time Since Launch (s)')
310     ylabel('Position (ft)')
311     set(legend('$y$', '$\hat{y}$'), 'Interpreter', 'Latex')
312     set(get(gca, 'Title'), 'String', 'Actual Position vs. Estimated
        Position (Gauss-Markov Process)');
313
314     figure(5)
315     plot(ts, trueXs(2, :))
316     hold on
317     plot(ts, xHats(2, :))
318     hold off
319     xlabel('Time Since Launch (s)')
320     ylabel('Velocity (ft/s)')

```

```

321 set(legend('$v$', '$\hat{v}$'), 'Interpreter', 'Latex')
322 set(get(gca, 'Title'), 'String', 'Actual Velocity vs. Estimated
    Velocity (Gauss-Markov Process)');
323
324 figure(6)
325 plot(ts, trueXs(3, :))
326 hold on
327 plot(ts, xHats(3, :))
328 hold off
329 xlabel('Time Since Launch (s)')
330 ylabel('Acceleration (ft/s^2)')
331 set(legend('$a_{T}$', '$\hat{a}_{T}$'), 'Interpreter', 'Latex')
332 set(get(gca, 'Title'), 'String', 'Actual Acceleration vs.
    Estimated Acceleration (Gauss-Markov Process)');
333
334 figure(7)
335 P1 = sqrt(filterPs(:, 1, 1));
336 plot(ts, trueXs(1, :) - xHats(1, :))
337 hold on
338 plot(ts, P1, 'black')
339 plot(ts, -P1, 'black')
340 hold off
341 xlabel('Time Since Launch (s)')
342 ylabel('Error in Position (ft)')
343 set(get(gca, 'Title'), 'String', 'Error in Position Estimates
    with 1-\sigma (Gauss-Markov Process)');
344
345 figure(8)
346 P2 = sqrt(filterPs(:, 2, 2));
347 plot(ts, trueXs(2, :) - xHats(2, :))
348 hold on
349 plot(ts, P2, 'black')
350 plot(ts, -P2, 'black')
351 hold off
352 xlabel('Time Since Launch (s)')
353 ylabel('Error in Velocity (ft/s)')
354 set(get(gca, 'Title'), 'String', 'Error in Velocity Estimates
    with 1-\sigma (Gauss-Markov Process)');
355
356 figure(9)
357 P3 = sqrt(filterPs(:, 3, 3));
358 plot(ts, trueXs(3, :) - xHats(3, :))
359 hold on
360 plot(ts, P3, 'black')

```

```

361     plot(ts, - P3, 'black')
362     hold off
363     xlabel('Time Since Launch (s)')
364     ylabel('Error in Acceleration (ft/s^2)')
365     set(legend('$a_{T}$', '$\hat{a}_{T}$'), 'Interpreter', 'Latex')
366     set(get(gca, 'Title'), 'String', 'Error in Acceleration Estimates
        with 1-\sigma (Gauss-Markov Process)');
367
368 end
369
370 function plotResults3(ts, trueYs, trueXs, xHats, filterPs)
371     figure(10)
372     plot(ts, trueYs)
373     xlabel('Time Since Launch (s)')
374     ylabel('\theta (rad)')
375     % set(gca, 'YScale', 'log')
376     set(get(gca, 'Title'), 'String', 'Measurement \theta (Random
        Telegraph Signal)');
377
378     figure(11)
379     plot(ts, trueXs(1, :))
380     hold on
381     plot(ts, xHats(1, :))
382     hold off
383     xlabel('Time Since Launch (s)')
384     ylabel('Position (ft)')
385     set(legend('$y$', '$\hat{y}$'), 'Interpreter', 'Latex')
386     set(get(gca, 'Title'), 'String', 'Actual Position vs. Estimated
        Position (Random Telegraph Signal)');
387
388     figure(12)
389     plot(ts, trueXs(2, :))
390     hold on
391     plot(ts, xHats(2, :))
392     hold off
393     xlabel('Time Since Launch (s)')
394     ylabel('Velocity (ft/s)')
395     set(legend('$v$', '$\hat{v}$'), 'Interpreter', 'Latex')
396     set(get(gca, 'Title'), 'String', 'Actual Velocity vs. Estimated
        Velocity (Random Telegraph Signal)');
397
398     figure(13)
399     plot(ts, trueXs(3, :))
400     hold on

```

```

401 plot(ts , xHats(3, :))
402 hold off
403 xlabel('Time Since Launch (s)')
404 ylabel('Acceleration (ft/s^2)')
405 set(legend('$a_{T}$', '$\hat{a}_{T}$'), 'Interpreter', 'Latex')
406 set(get(gca, 'Title'), 'String', 'Actual Acceleration vs.
    Estimated Acceleration (Random Telegraph Signal)');

407
408 figure(14)
409 P1 = sqrt(filterPs(:, 1, 1)');
410 plot(ts, trueXs(1, :) - xHats(1, :))
411 hold on
412 plot(ts, P1, 'black')
413 plot(ts, - P1, 'black')
414 hold off
415 xlabel('Time Since Launch (s)')
416 ylabel('Error in Position (ft)')
417 set(get(gca, 'Title'), 'String', 'Error in Position Estimates
    with  $1-\sigma$  (Random Telegraph Signal)');

418
419 figure(15)
420 P2 = sqrt(filterPs(:, 2, 2)');
421 plot(ts, trueXs(2, :) - xHats(2, :))
422 hold on
423 plot(ts, P2, 'black')
424 plot(ts, - P2, 'black')
425 hold off
426 xlabel('Time Since Launch (s)')
427 ylabel('Error in Velocity (ft/s)')
428 set(get(gca, 'Title'), 'String', 'Error in Velocity Estimates
    with  $1-\sigma$  (Random Telegraph Signal)');

429
430 figure(16)
431 P3 = sqrt(filterPs(:, 3, 3)');
432 plot(ts, trueXs(3, :) - xHats(3, :))
433 hold on
434 plot(ts, P3, 'black')
435 plot(ts, - P3, 'black')
436 hold off
437 xlabel('Time Since Launch (s)')
438 ylabel('Error in Acceleration (ft/s^2)')
439 set(legend('$a_{T}$', '$\hat{a}_{T}$'), 'Interpreter', 'Latex')
440 set(get(gca, 'Title'), 'String', 'Error in Acceleration Estimates
    with  $1-\sigma$  (Random Telegraph Signal)');

```

```

441
442 end
443
444 function plotResults4(ts, trueErrorVariance, trueErrorVarianceRTS,
    filterPs)
445     figure(17)
446     subplot(3, 3, 1)
447     hold on
448     plot(ts, trueErrorVariance(:, 1, 1))
449     plot(ts, filterPs(:, 1, 1))
450     hold off
451     xlabel('Time Since Launch (s)')
452     set(legend('True P', 'P'), 'Interpreter', 'Latex')
453     title('Covariance(1, 1)')
454
455     subplot(3, 3, 2)
456     hold on
457     plot(ts, trueErrorVariance(:, 1, 2))
458     plot(ts, filterPs(:, 1, 2))
459     hold off
460     xlabel('Time Since Launch (s)')
461     set(legend('True P', 'P'), 'Interpreter', 'Latex')
462     title('Covariance(1, 2)')
463
464     subplot(3, 3, 3)
465     hold on
466     plot(ts, trueErrorVariance(:, 1, 3))
467     plot(ts, filterPs(:, 1, 3))
468     hold off
469     xlabel('Time Since Launch (s)')
470     set(legend('True P', 'P'), 'Interpreter', 'Latex')
471     title('Covariance(1, 3)')
472
473     subplot(3, 3, 4)
474     hold on
475     plot(ts, trueErrorVariance(:, 2, 1))
476     plot(ts, filterPs(:, 2, 1))
477     hold off
478     xlabel('Time Since Launch (s)')
479     set(legend('True P', 'P'), 'Interpreter', 'Latex')
480     title('Covariance(2, 1)')
481
482     subplot(3, 3, 5)
483     hold on

```

```

484 plot(ts, trueErrorVariance(:, 2, 2))
485 plot(ts, filterPs(:, 2, 2))
486 hold off
487 xlabel('Time Since Launch (s)')
488 set(legend('True P', 'P'), 'Interpreter', 'Latex')
489 title('Covariance(2, 2)')
490
491 subplot(3, 3, 6)
492 hold on
493 plot(ts, trueErrorVariance(:, 2, 3))
494 plot(ts, filterPs(:, 2, 3))
495 hold off
496 xlabel('Time Since Launch (s)')
497 set(legend('True P', 'P'), 'Interpreter', 'Latex')
498 title('Covariance(2, 3)')
499
500 subplot(3, 3, 7)
501 hold on
502 plot(ts, trueErrorVariance(:, 3, 1))
503 plot(ts, filterPs(:, 3, 1))
504 hold off
505 xlabel('Time Since Launch (s)')
506 set(legend('True P', 'P'), 'Interpreter', 'Latex')
507 title('Covariance(3, 1)')
508
509 subplot(3, 3, 8)
510 hold on
511 plot(ts, trueErrorVariance(:, 3, 2))
512 plot(ts, filterPs(:, 3, 2))
513 hold off
514 xlabel('Time Since Launch (s)')
515 set(legend('True P', 'P'), 'Interpreter', 'Latex')
516 title('Covariance(3, 2)')
517
518 subplot(3, 3, 9)
519 hold on
520 plot(ts, trueErrorVariance(:, 3, 3))
521 plot(ts, filterPs(:, 3, 3))
522 hold off
523 xlabel('Time Since Launch (s)')
524 set(legend('True P', 'P'), 'Interpreter', 'Latex')
525 title('Covariance(3, 3)')
526

```

```

527     sgtitle('True Error Covariance vs. Estimated Error Covariance for
528             1000 Monte Carlo Runs (Gauss–Markov Process)')
529
529     figure(18)
530     subplot(3, 3, 1)
531     hold on
532     plot(ts, trueErrorVarianceRTS(:, 1, 1))
533     plot(ts, filterPs(:, 1, 1))
534     hold off
535     xlabel('Time Since Launch (s)')
536     set(legend('True P', 'P'), 'Interpreter', 'Latex')
537     title('Covariance(1, 1)')
538
539     subplot(3, 3, 2)
540     hold on
541     plot(ts, trueErrorVarianceRTS(:, 1, 2))
542     plot(ts, filterPs(:, 1, 2))
543     hold off
544     xlabel('Time Since Launch (s)')
545     set(legend('True P', 'P'), 'Interpreter', 'Latex')
546     title('Covariance(1, 2)')
547
548     subplot(3, 3, 3)
549     hold on
550     plot(ts, trueErrorVarianceRTS(:, 1, 3))
551     plot(ts, filterPs(:, 1, 3))
552     hold off
553     xlabel('Time Since Launch (s)')
554     set(legend('True P', 'P'), 'Interpreter', 'Latex')
555     title('Covariance(1, 3)')
556
557     subplot(3, 3, 4)
558     hold on
559     plot(ts, trueErrorVarianceRTS(:, 2, 1))
560     plot(ts, filterPs(:, 2, 1))
561     hold off
562     xlabel('Time Since Launch (s)')
563     set(legend('True P', 'P'), 'Interpreter', 'Latex')
564     title('Covariance(2, 1)')
565
566     subplot(3, 3, 5)
567     hold on
568     plot(ts, trueErrorVarianceRTS(:, 2, 2))
569     plot(ts, filterPs(:, 2, 2))

```

```

570 hold off
571 xlabel('Time Since Launch (s)')
572 set(legend('True P', 'P'),'Interpreter','Latex')
573 title('Covariance(2, 2)')
574
575 subplot(3, 3, 6)
576 hold on
577 plot(ts, trueErrorVarianceRTS(:, 2, 3))
578 plot(ts, filterPs(:, 2, 3))
579 hold off
580 xlabel('Time Since Launch (s)')
581 set(legend('True P', 'P'),'Interpreter','Latex')
582 title('Covariance(2, 3)')
583
584 subplot(3, 3, 7)
585 hold on
586 plot(ts, trueErrorVarianceRTS(:, 3, 1))
587 plot(ts, filterPs(:, 3, 1))
588 hold off
589 xlabel('Time Since Launch (s)')
590 set(legend('True P', 'P'),'Interpreter','Latex')
591 title('Covariance(3, 1)')
592
593 subplot(3, 3, 8)
594 hold on
595 plot(ts, trueErrorVarianceRTS(:, 3, 2))
596 plot(ts, filterPs(:, 3, 2))
597 hold off
598 xlabel('Time Since Launch (s)')
599 set(legend('True P', 'P'),'Interpreter','Latex')
600 title('Covariance(3, 2)')
601
602 subplot(3, 3, 9)
603 hold on
604 plot(ts, trueErrorVarianceRTS(:, 3, 3))
605 plot(ts, filterPs(:, 3, 3))
606 hold off
607 xlabel('Time Since Launch (s)')
608 set(legend('True P', 'P'),'Interpreter','Latex')
609 title('Covariance(3, 3)')
610
611 sgtitle('True Error Covariance vs. Estimated Error Covariance for
        1000 Monte Carlo Runs (Random Telegraph Signal)')
612 end

```



```

613
614 function plotResults5(ts , dt , residualHistory)
615     residualCorrelations = residualAnalysis(7, residualHistory , dt);
616
617     figure (19)
618     plot(ts , residualCorrelations)
619     xlabel('Time Since Launch (s)')
620     ylabel('Correlation')
621     title('Correlation of Residual at 7th second with Time')
622     Tf = 10;
623     R1 = 15e-6;
624     R2 = 1.67e-3;
625     V = R1 + R2./((Tf-ts).^2);
626
627     figure (20)
628     periodogram(squeeze(mean(residualHistory , 1))./V')
629     title('Power Spectral Density of Residual Process')
630
631 end

```