MTH 510

Inverse Problems and Data Assimilation

Homework #4

Submitted by Andrew Huffman

November 27, 2019

- **Model Performance**

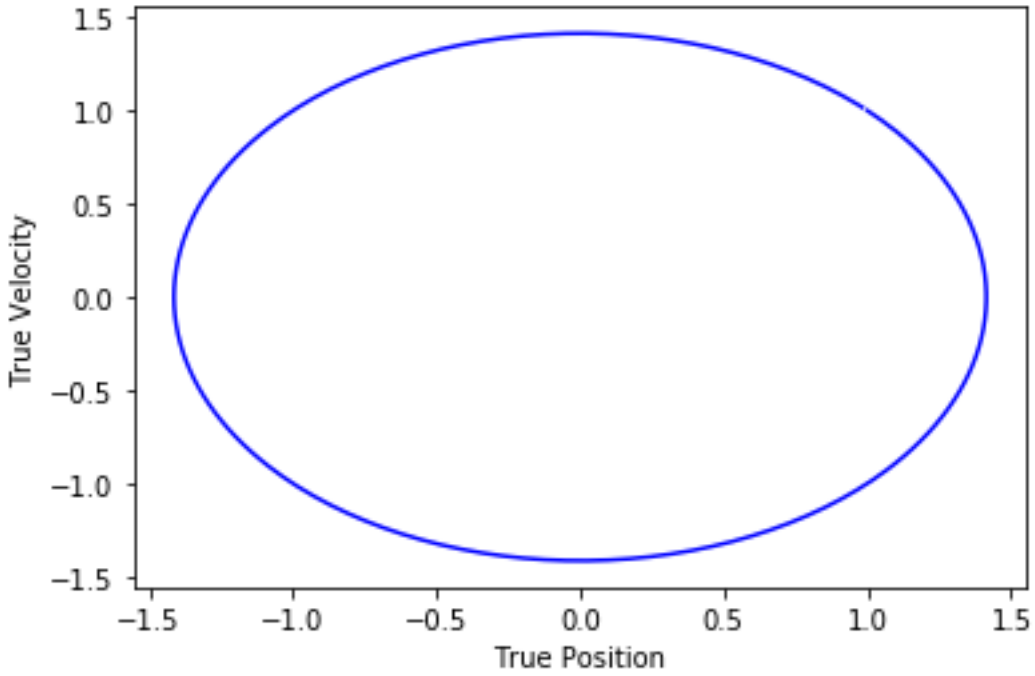

Figure 1: Plot of true position vs. true velocity

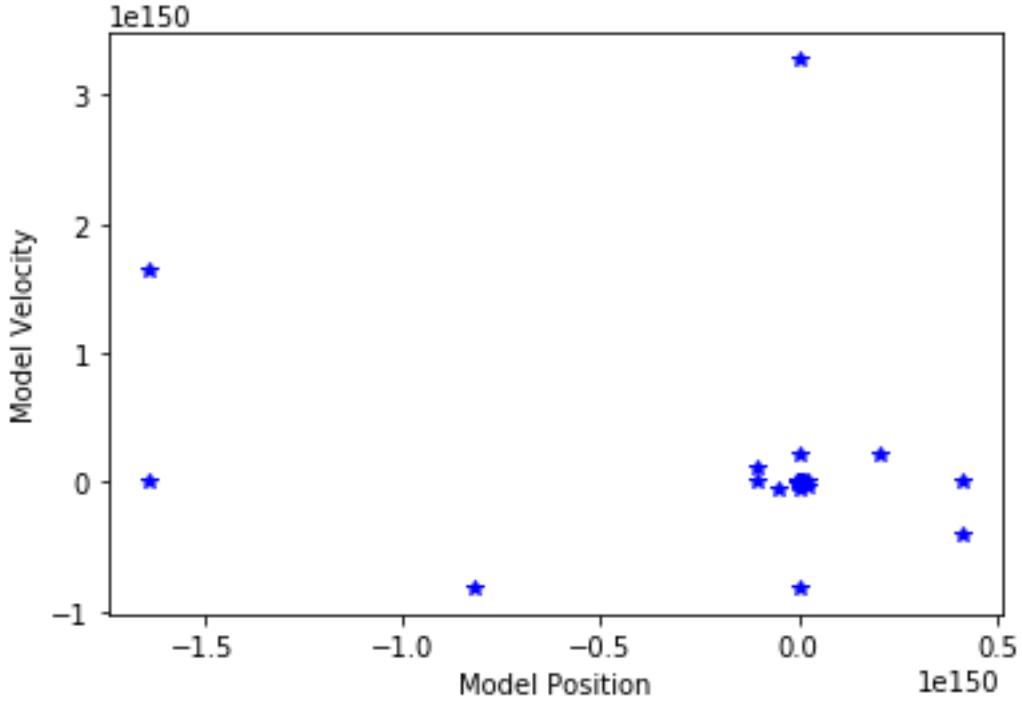

Figure 2: Plot of position vs. velocity from model

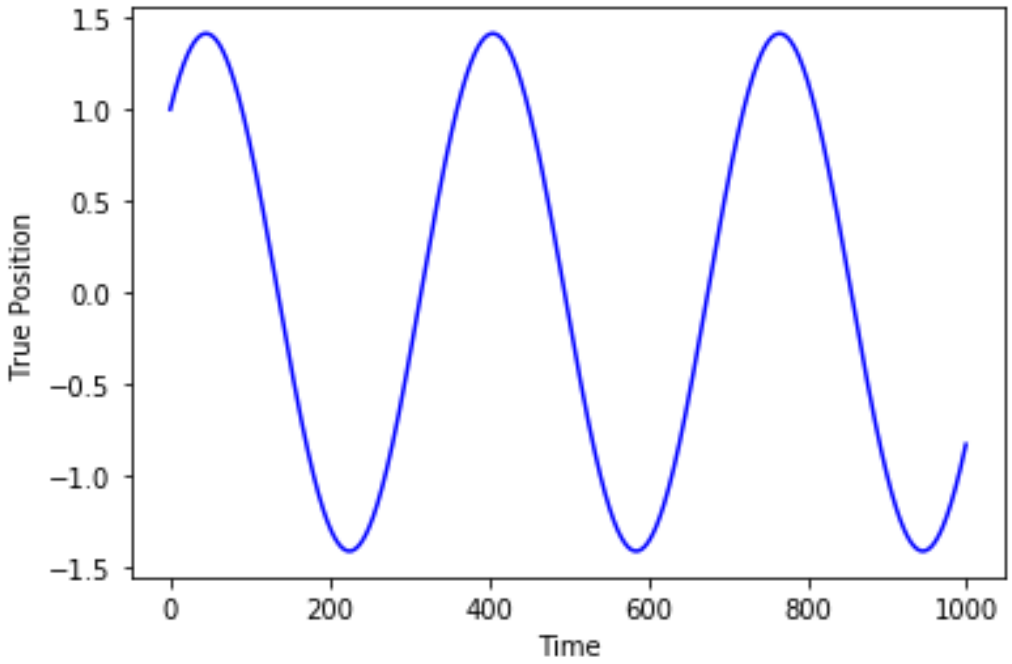

Figure 3: True position vs. time for 1000 time increments

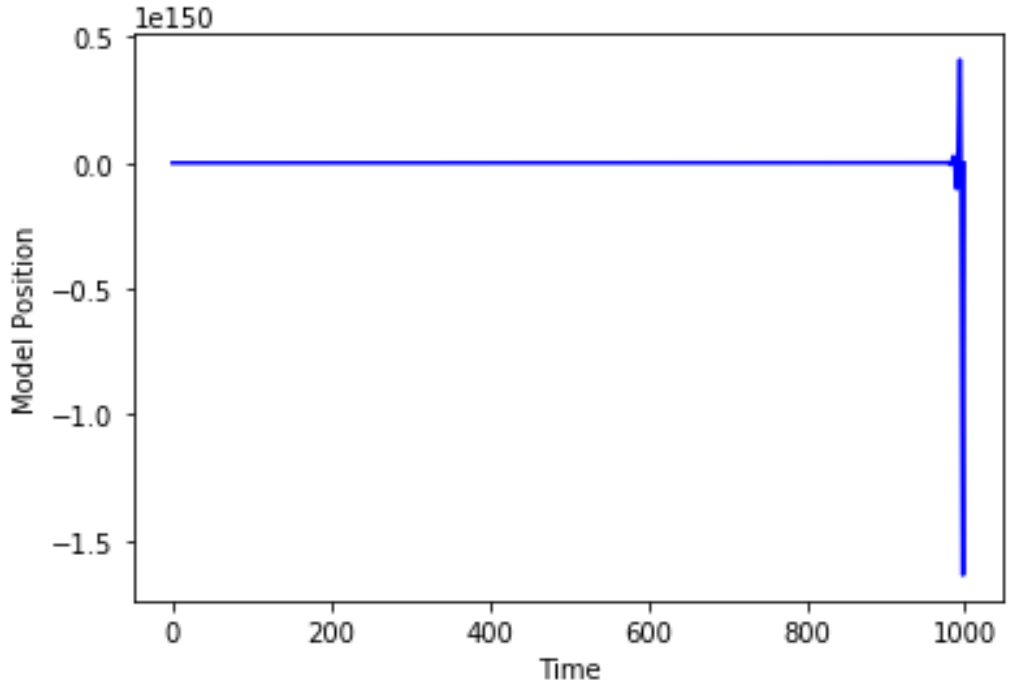

Figure 4: Model position vs. time for 1000 time increments (The takeaway from this plot is that the model position increases very rapidly. The position is not close to zero along the portion of the plot that appears flat. It is just an issue of scale as the model adds energy into the system with each time step.)

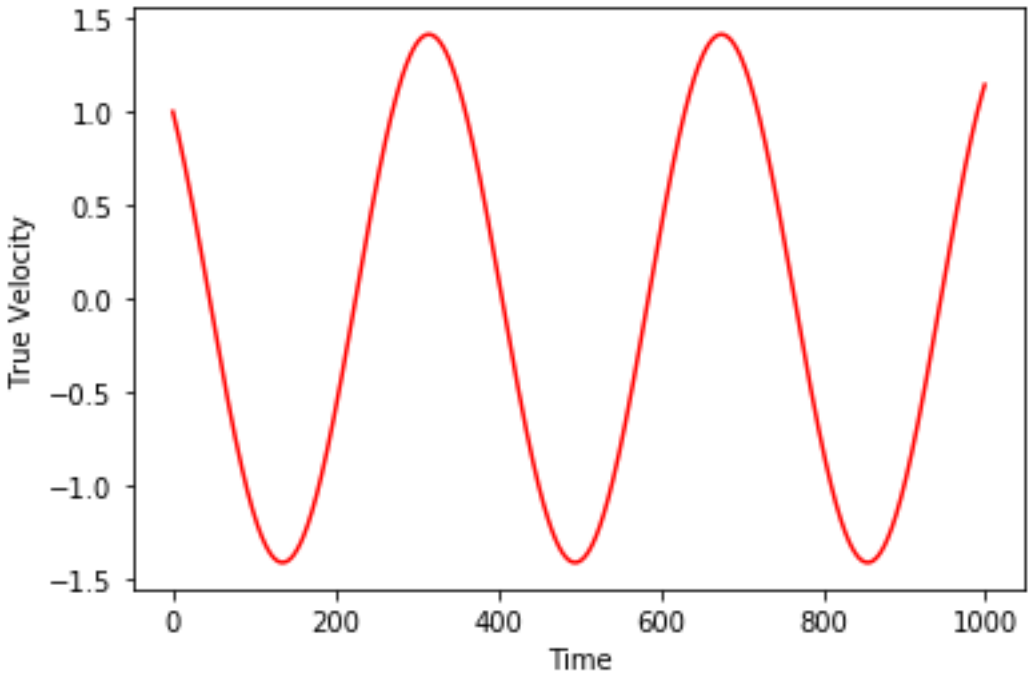

Figure 5: True velocity vs. time for 1000 time increments

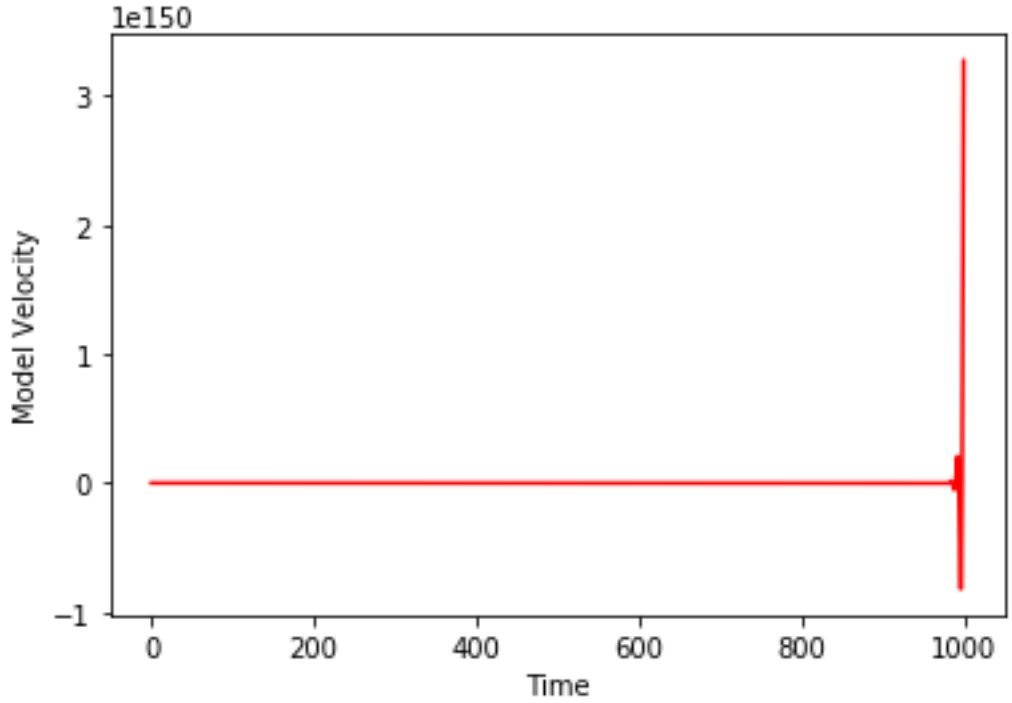

Figure 6: Model velocity vs. time for 1000 time increments (The takeaway from this plot is that the model velocity increases very rapidly. The velocity is not close to zero along the portion of the plot that appears flat. It is just an issue of scale as the model adds energy into the system with each time step.)

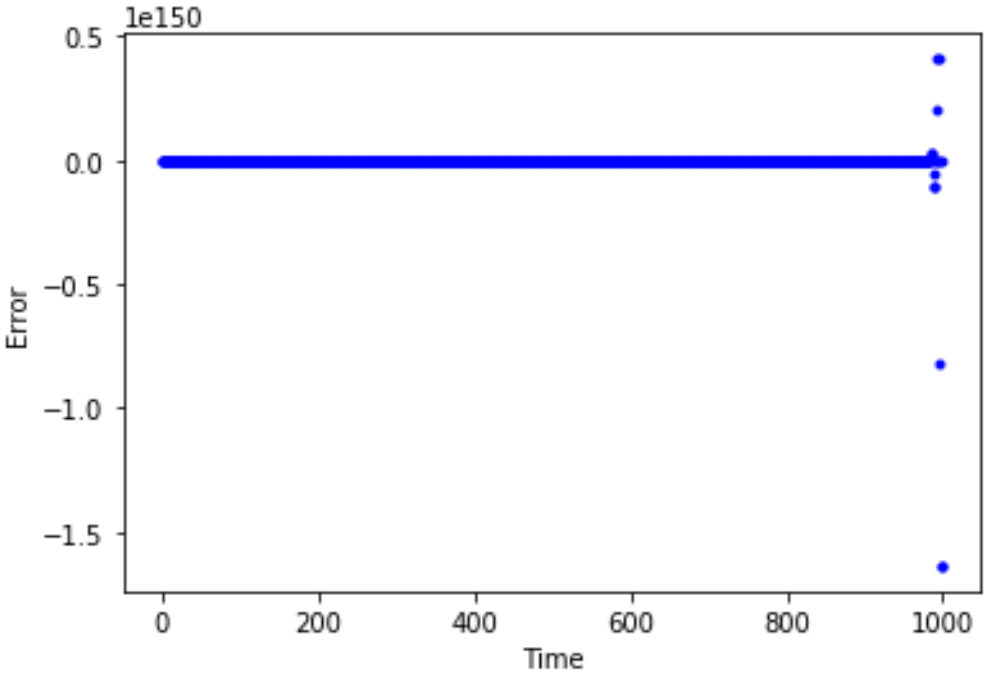

Figure 7: Model position error (model position-true position) for 100 time increments (The takeaway from this plot is that the model position error is extremely large. The error is not close to zero along the portion of the plot that appears flat. It is just an issue of scale as the error increases rapidly with each time step.)

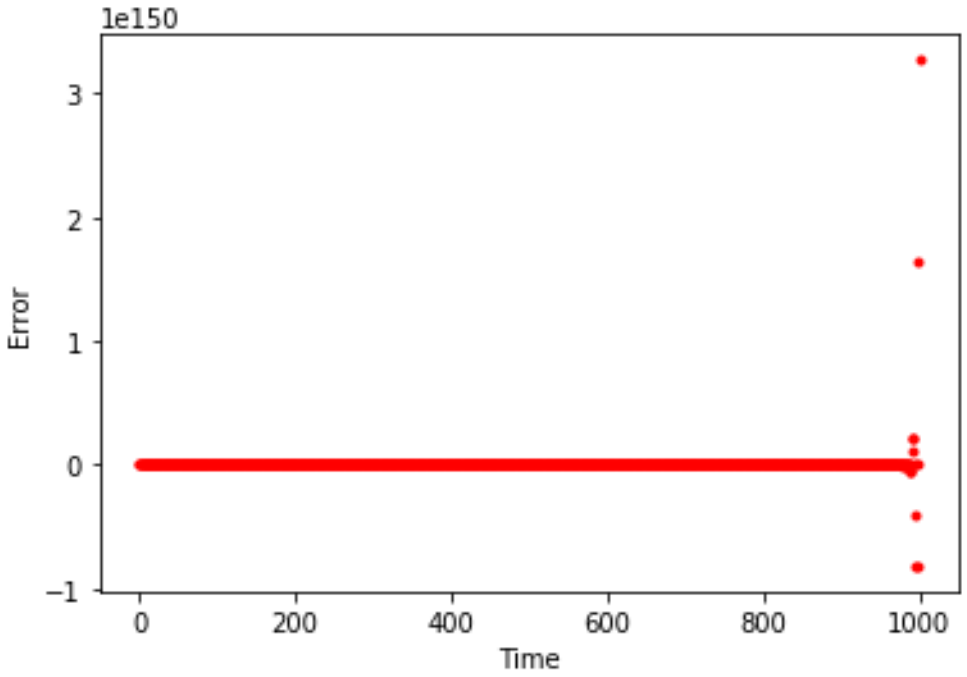

Figure 8: Model velocity error (model velocity-true velocity) for 1000 time increments (The takeaway from this plot is that the model velocity error is extremely large. The error is not close to zero along the portion of the plot that appears flat. It is just an issue of scale as the error increases rapidly with each time step.)

Clearly, the model does not perform well because it produces very large errors.

- **Implement Kalman Filter for Case 1 (both position and velocity are observed)**

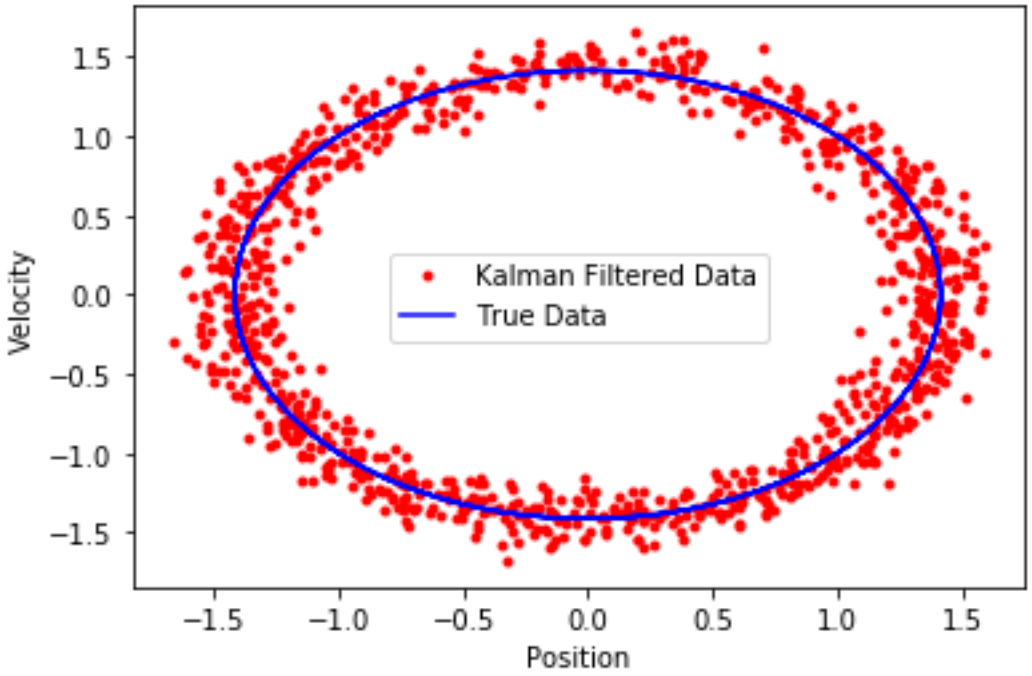

Figure 9: Velocity vs. position

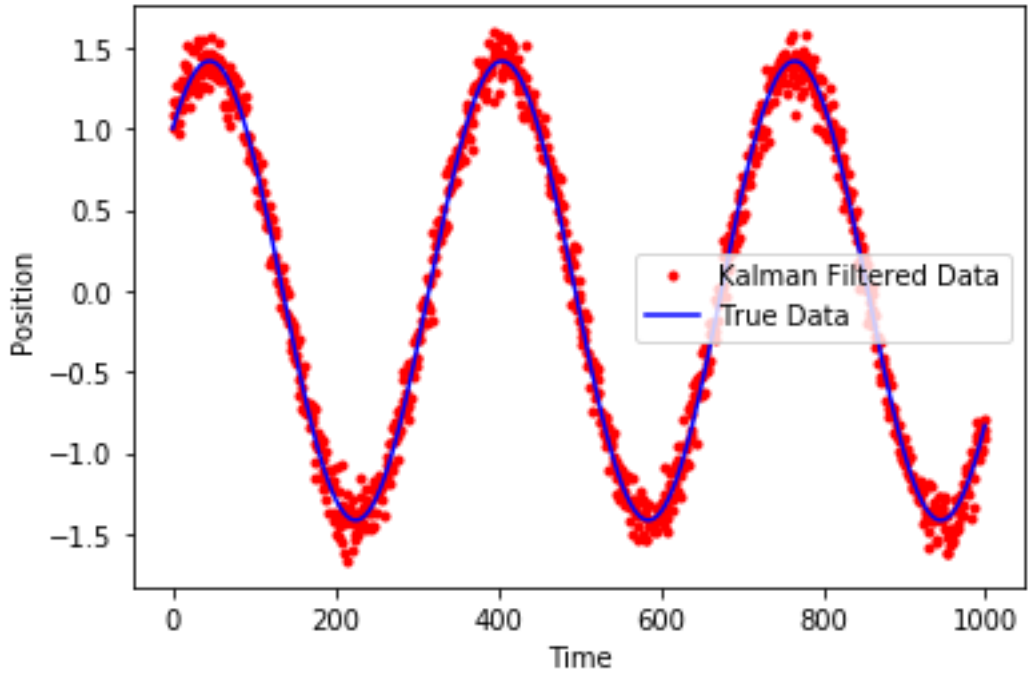

Figure 10: Position vs. time

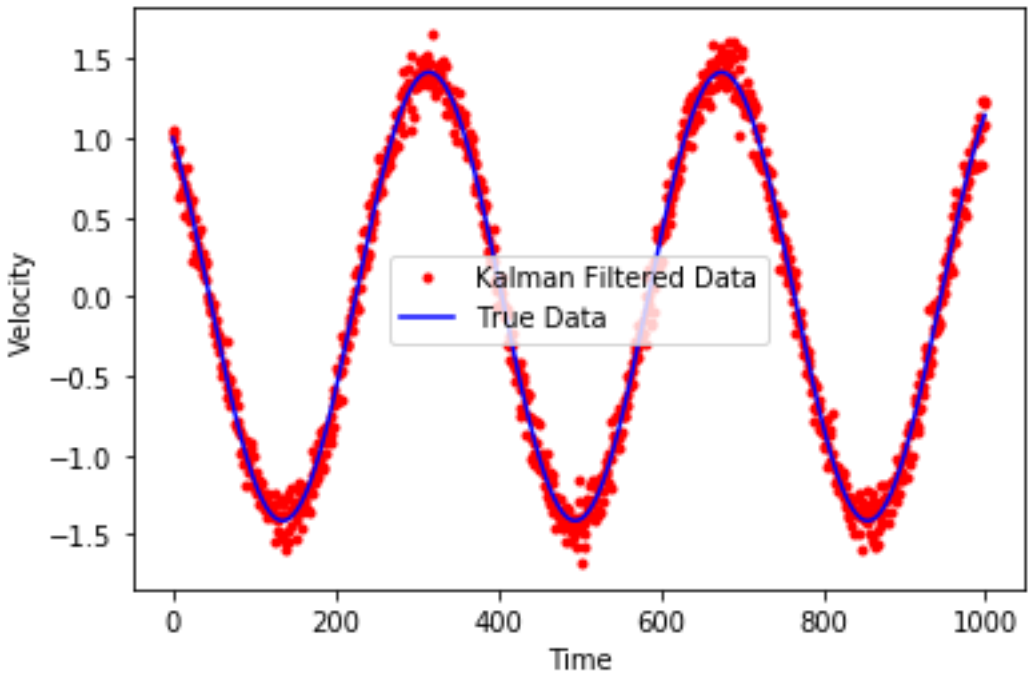

Figure 11: Velocity vs. time

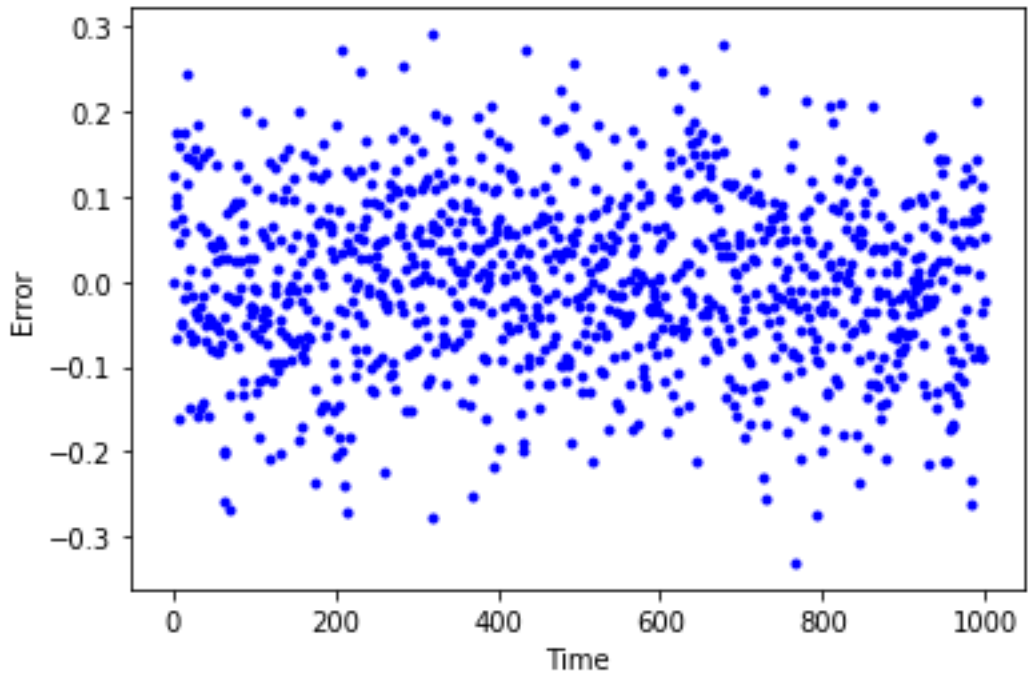

Figure 12: Kalman position error (Kalman position-true position) over 1000 time increments

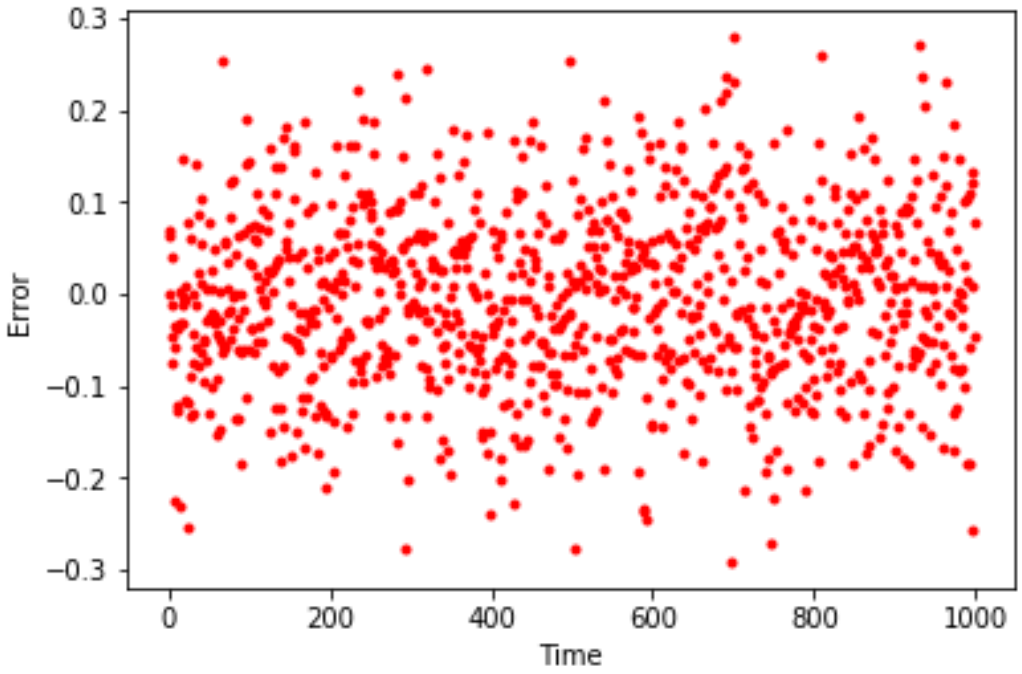

Figure 13: Kalman velocity error (Kalman velocity-true velocity) over 1000 time increments

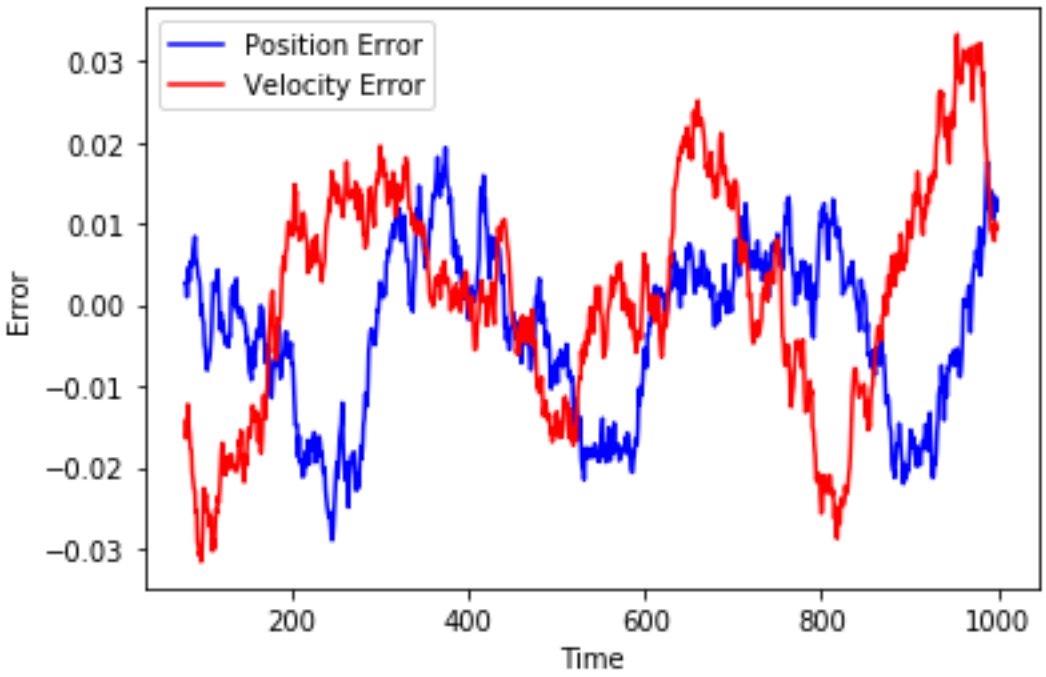

Figure 14: Moving average of errors in position and velocity from Kalman analysis over 1000 time increments

The Kalman filtering algorithm (with both state variables observed) performs quite well. The error in the Kalman predicted state is vastly less than the error in the original model's predicted state. The Kalman predicted state matches the true state well both for position and velocity.

- **Implement Kalman Filter for Case 2 (only position is observed)**

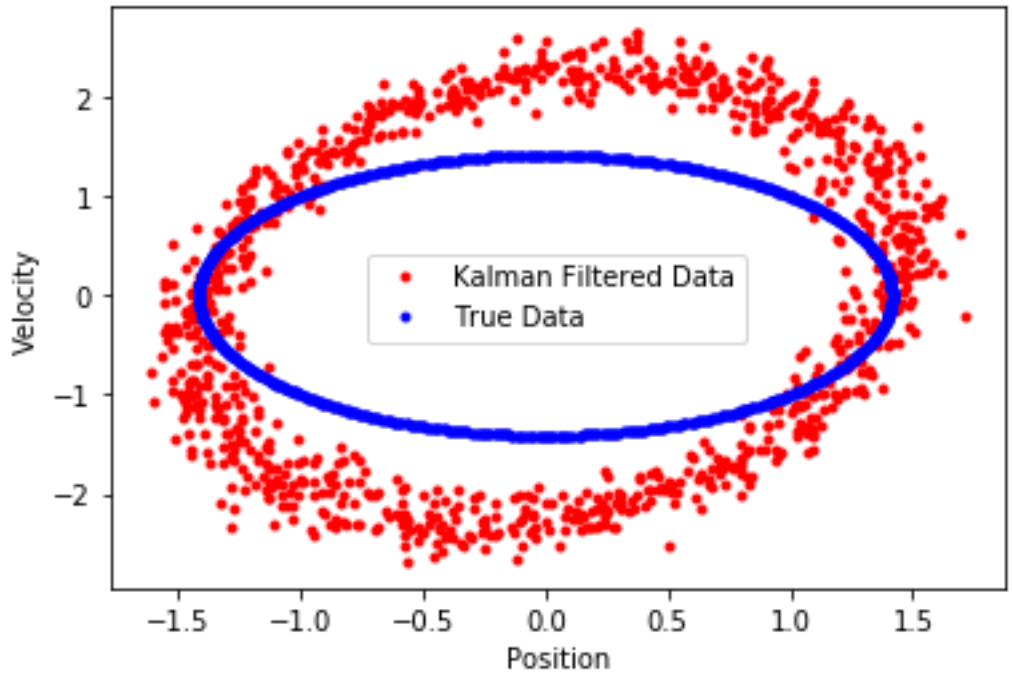

Figure 15: Velocity vs. position

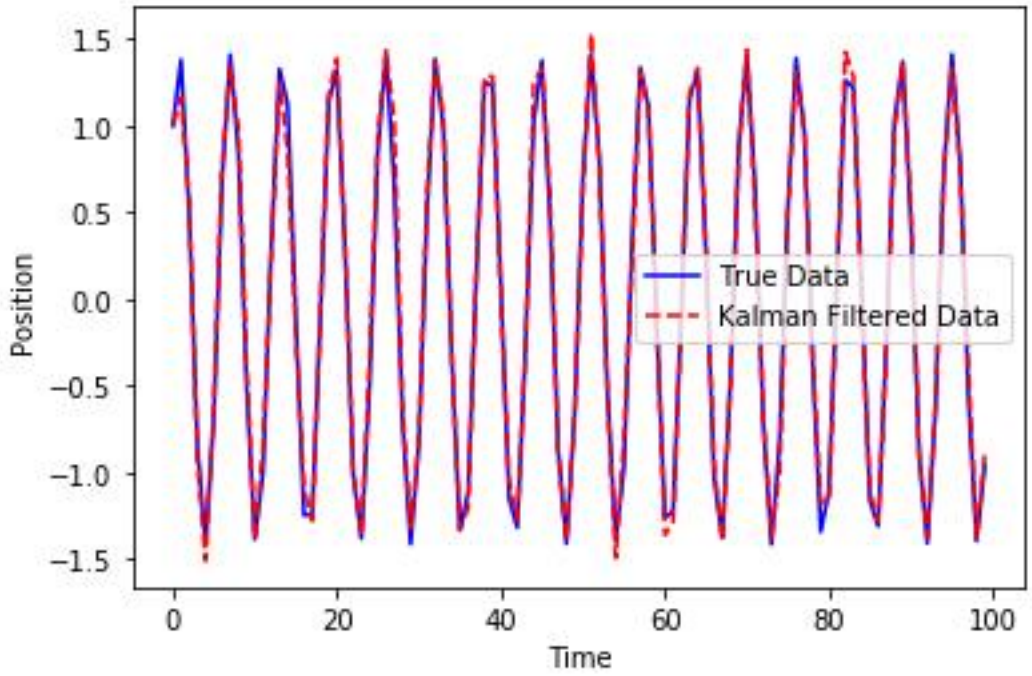

Figure 16: Position vs. time

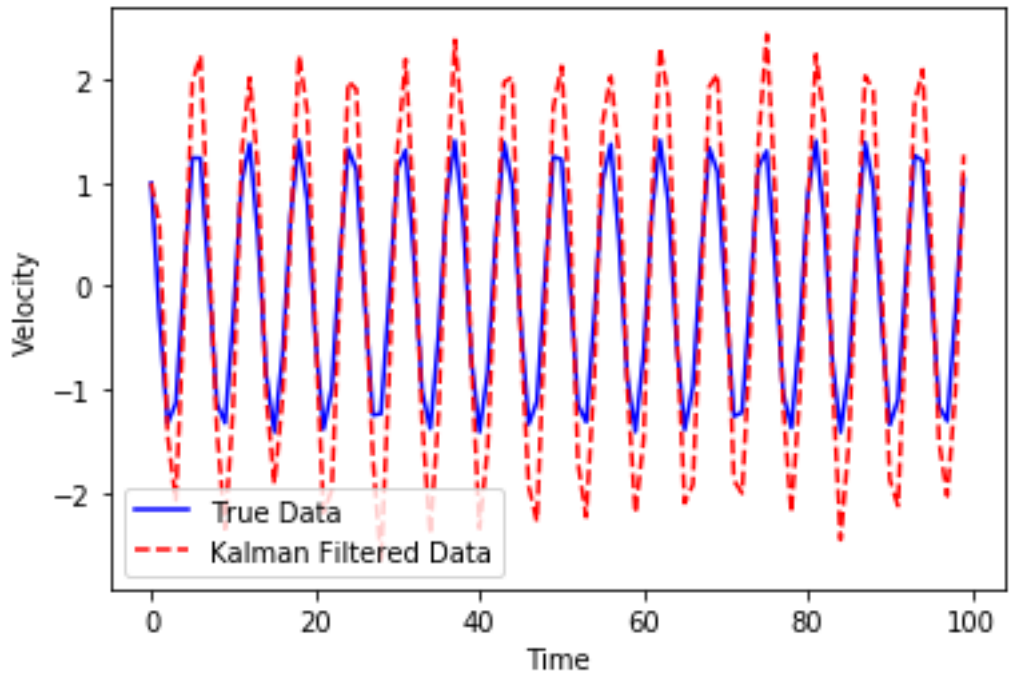

Figure 17: Velocity vs. time

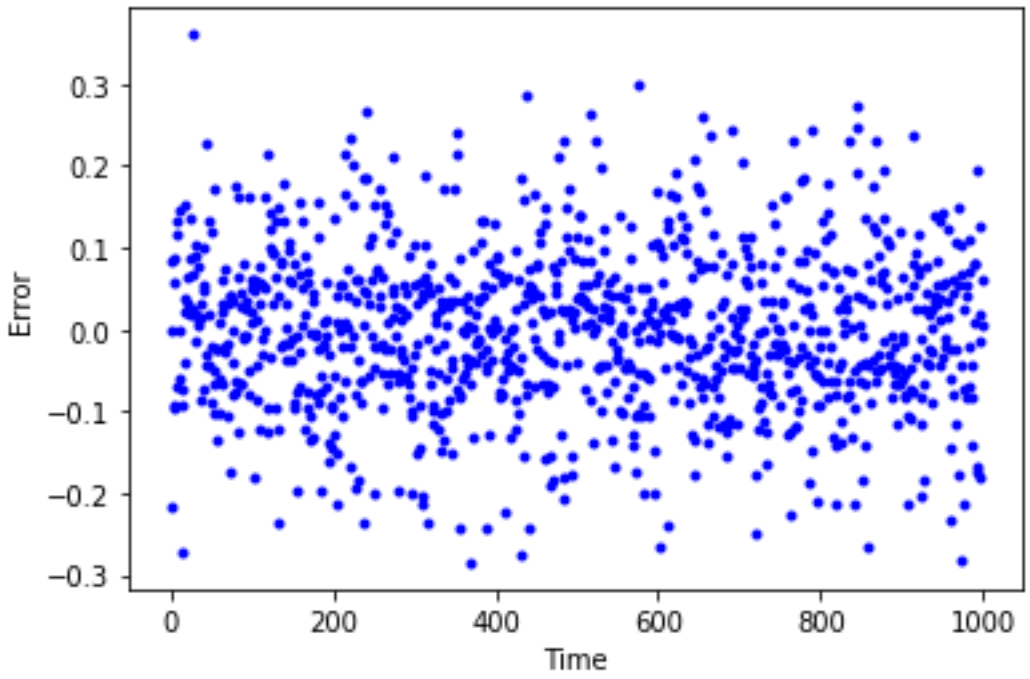

Figure 18: Kalman position error (Kalman position-true position) over 1000 time increments

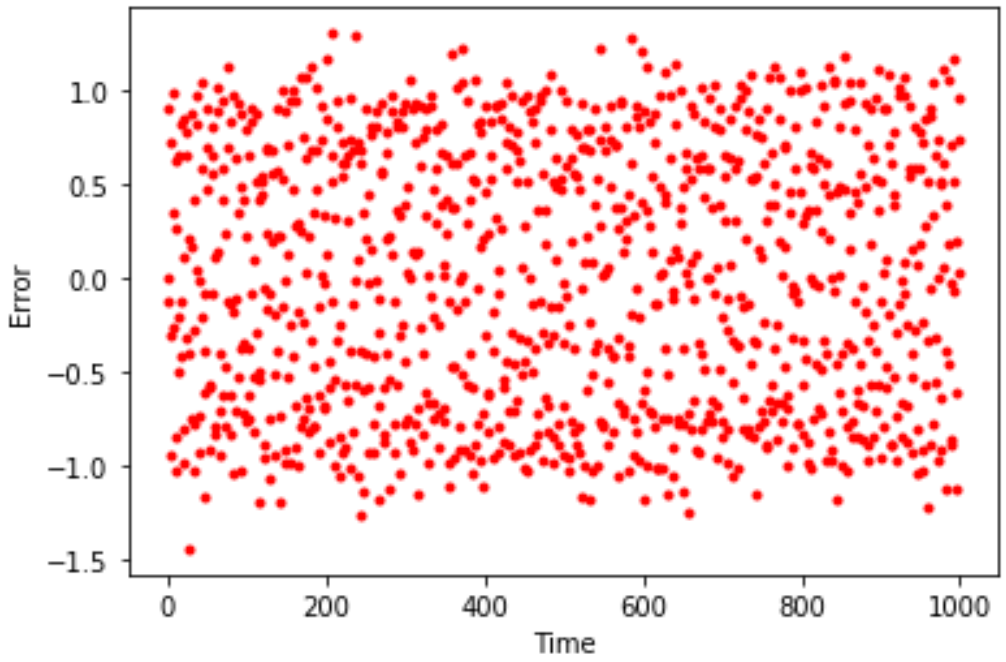

Figure 19: Kalman velocity error (Kalman velocity-true velocity) over 1000 time increments

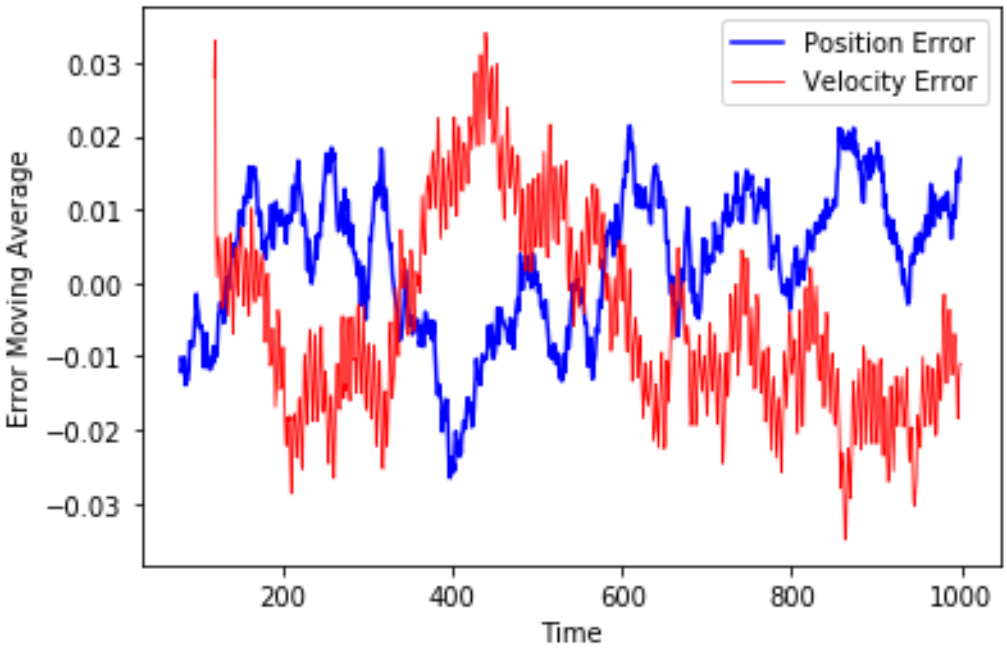

Figure 20: Moving average of errors in position and velocity from Kalman analysis over 1000 time increments

For Case 2 the Kalman Filter still vastly improves the estimate of the state variables compared to the original model. The position estimate is essentially the same for Case 2 as for Case 1. However, the estimate of the velocity variable is worse than in Case 1 because velocity is not observed in Case 2.

Appendix: Script used to generate solutions

```
# -*- coding: utf-8 -*-
"""
Created on Sun Nov 24 17:24:34 2019

@author: Andrew
"""
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
import time

x_0 = np.ones((2,1)) #vector storing initial position and velocity, both
equal to
eps_b = np.random.normal(0,1,(2,1)) #noise for initial state estimate
x_a = x_0+eps_b
P_a = np.identity(2) #initial analysis error covariance matrix
sigma_q = 1 #forecast error parameter
sigma_o = 0.1
dt = 1
M = np.ones((2,2))
M[0][1] = dt
M[1][0] = dt*(-1)
#m = 1
m = 2
k = 1000
case = 1


### True state ###
t = np.zeros((k,1))
x_true = np.zeros((k,2))
for i in range(0,k):
    t_i = dt*i
    x = (x_0[1][0]*math.sin((t_i)))+(x_0[0][0]*math.cos((t_i)))
    v = (x_0[1][0]*math.cos((t_i)))-(x_0[0][0]*math.sin((t_i)))
    x_true[i][0] = x
    x_true[i][1] = v
    t[i][0] = i*dt

fig_true_xv = plt.figure()
plt.plot(x_true[:,0],x_true[:,1],"-b")
```

```
plt.xlabel("True Position")
plt.ylabel("True Velocity")

fig_true_xt = plt.figure()
plt.plot(t,x_true[:,0],"-b")
plt.xlabel("Time")
plt.ylabel("True Position")

fig_true_vt = plt.figure()
plt.plot(t,x_true[:,1],"-r")
plt.xlabel("Time")
plt.ylabel("True Velocity")



### Pure forecasting ###
x_k = x_0
t_forecast = np.zeros((k,1))
x_forecast = np.zeros((k,2))
for i in range(0,k):
    if i == 0:
        x_k = x_0
    else:
        x_k =(M@x_k)
    x_forecast[i][0] = x_k[0][0]
    x_forecast[i][1] = x_k[1][0]

error_model = x_forecast-x_true

fig_forecast_xv = plt.figure()
plt.plot(x_forecast[:,0],x_forecast[:,1],"*b")
plt.xlabel("Model Position")
plt.ylabel("Model Velocity")

fig_forecast_xt = plt.figure()
plt.plot(t,x_forecast[:,0],"-b")
plt.xlabel("Time")
plt.ylabel("Model Position")

fig_forecast_vt = plt.figure()
plt.plot(t,x_forecast[:,1],"-r")
plt.xlabel("Time")
plt.ylabel("Model Velocity")
```

```
### Kalman Filter ###
t_kalman = np.zeros((k,1))
x_kalman = np.zeros((k,2))
if case == 1:
    H = np.zeros((1,2))
    H[0][0] = 1
    R = sigma_o**2
elif case == 2:
    H = np.identity(2)
    R = (sigma_o**2)*H
Q = (sigma_q**2)*np.identity(2)
M_T = np.transpose(M)
H_T = np.transpose(H)
x_kalman[0][0] = x_0[0][0]
x_kalman[0][1] = x_0[1][0]

for i in range(1,k):
    x_f = M@x_a
    P_f = (M@P_a@M_T)+Q
    Z = (H@P_f@H_T)+R
    Z_I = np.linalg.inv(Z)
    K = P_f@H_T@Z_I
    x_t = x_true[i][:].reshape(2,1)
    y = H@x_t+np.random.normal(0,sigma_o,(case,1))
    x_a = x_f+(K@(y-(H@x_f)))
    P_a = (np.identity(m)-(K@H))@P_f
    x_kalman[i][0] = x_a[0][0]
    x_kalman[i][1] = x_a[1][0]

error_kalman = x_kalman-x_true
error_kalman_df = pd.DataFrame(error_kalman)
if case == 1:
    interval_x = 80
    interval_v = 120
elif case == 2:
    interval_x = 80
    interval_v = 80
moving_ave_error_kalman_x =
error_kalman_df[0].rolling(window=interval_x).mean()
moving_ave_error_kalman_v =
error_kalman_df[1].rolling(window=interval_v).mean()
```

```
### Generate additional plots ###
fig_kalman_xv = plt.figure()
plt.plot(x_kalman[:,0],x_kalman[:,1],".r",label="Kalman Filtered Data")
plt.plot(x_true[:,0],x_true[:,1],".b",label="True Data")
plt.xlabel("Position")
plt.ylabel("Velocity")
plt.legend()

fig_kalman_xt = plt.figure()
plt.plot(t[0:100],x_true[0:100,0],"-b",label="True Data")
plt.plot(t[0:100],x_kalman[0:100,0],"--r",label="Kalman Filtered Data")
plt.xlabel("Time")
plt.ylabel("Position")
plt.legend()

fig_kalman_vt = plt.figure()
plt.plot(t[0:100],x_true[0:100,1],"-b",label="True Data")
plt.plot(t[0:100],x_kalman[0:100,1],"--r",label="Kalman Filtered Data")
plt.xlabel("Time")
plt.ylabel("Velocity")
plt.legend()

fig_error_model_x = plt.figure()
plt.plot(t,error_model[:,0],".b",label="Position Error")
plt.xlabel("Time")
plt.ylabel("Error")

fig_error_model_v = plt.figure()
plt.plot(t,error_model[:,1],".r",label="Velocity Error")
plt.xlabel("Time")
plt.ylabel("Error")

fig_error_kalman_x = plt.figure()
plt.plot(t,error_kalman[:,0],".b",label="Position Error")
plt.xlabel("Time")
plt.ylabel("Error")

fig_error_kalman_v = plt.figure()
plt.plot(t,error_kalman[:,1],".r",label="Velocity Error")
plt.xlabel("Time")
plt.ylabel("Error")

fig_error_kalman_moving_ave = plt.figure()
plt.plot(t,moving_ave_error_kalman_x,"-b",label="Position Error")
```

```
plt.plot(t,moving_ave_error_kalman_v,"-r",label="Velocity
Error",linewidth=0.75)
plt.xlabel("Time")
plt.ylabel("Error Moving Average")
plt.legend()
```