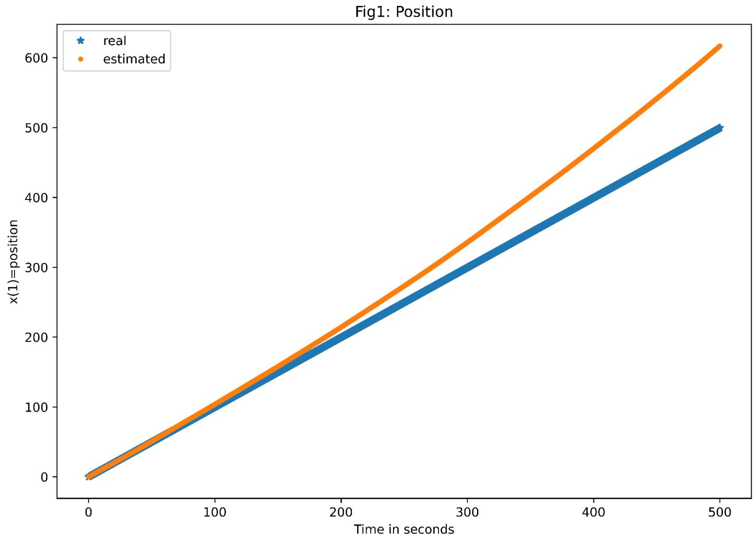
CS 498 Ex set 3

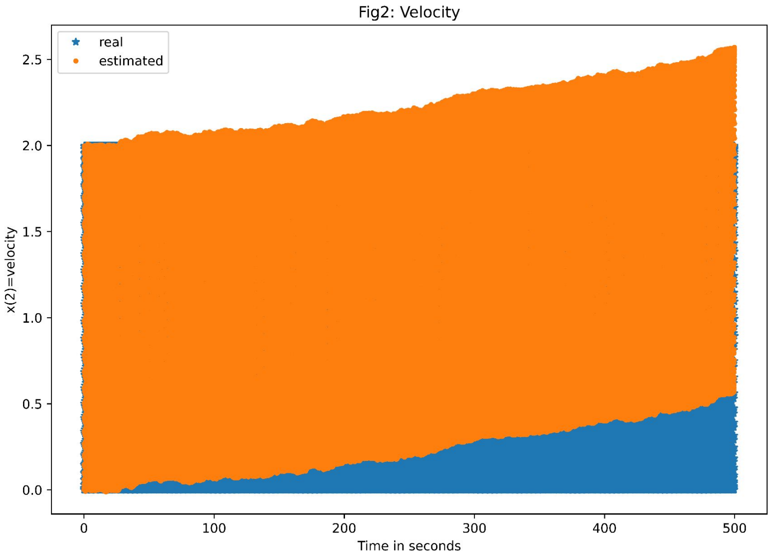
Kulbir

Qn 2

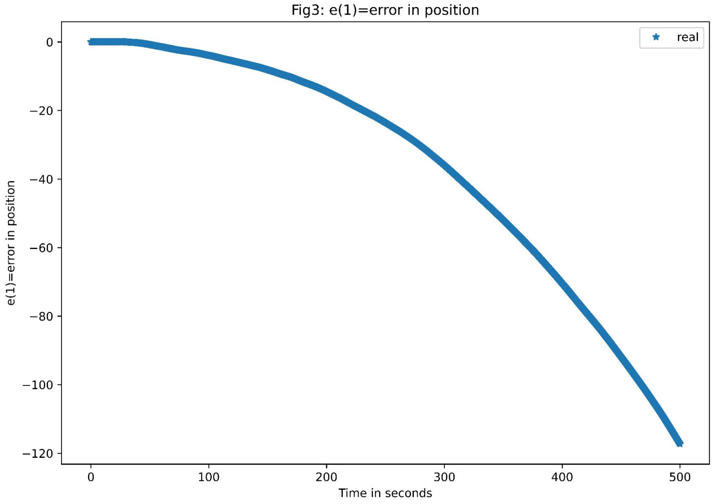
We see the divergence of estimated position



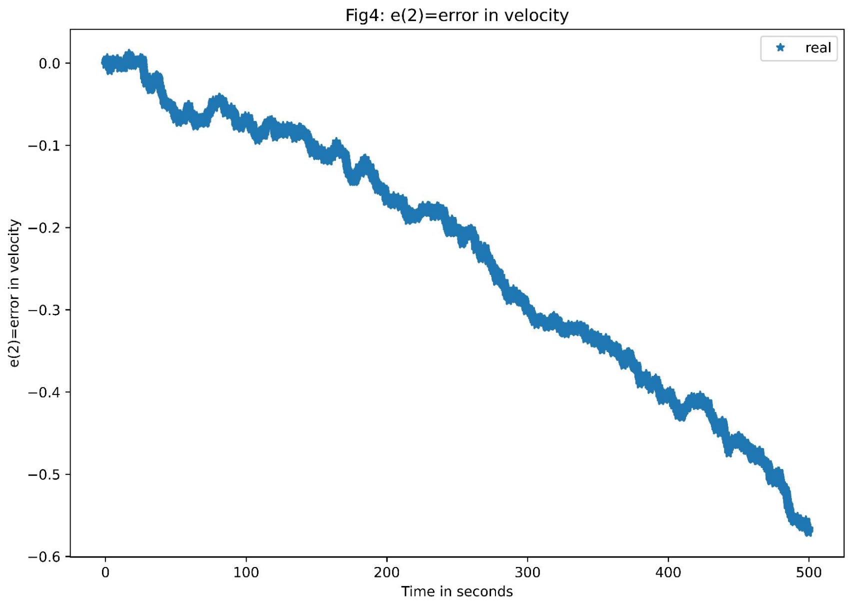
We see the divergence of estimated velocity



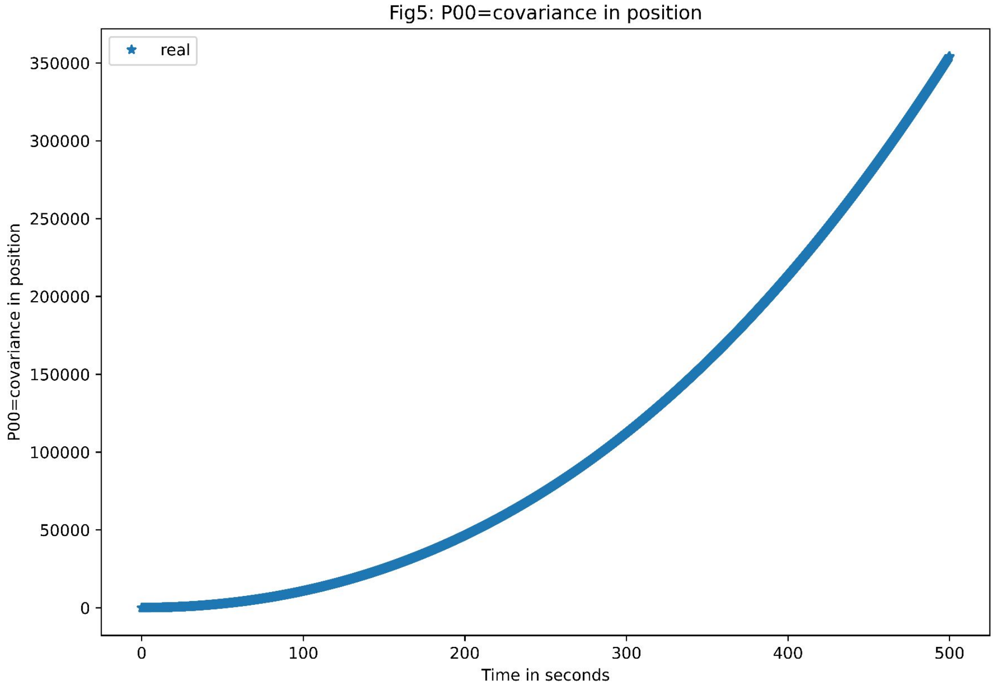
We see the divergence in error of estimated position



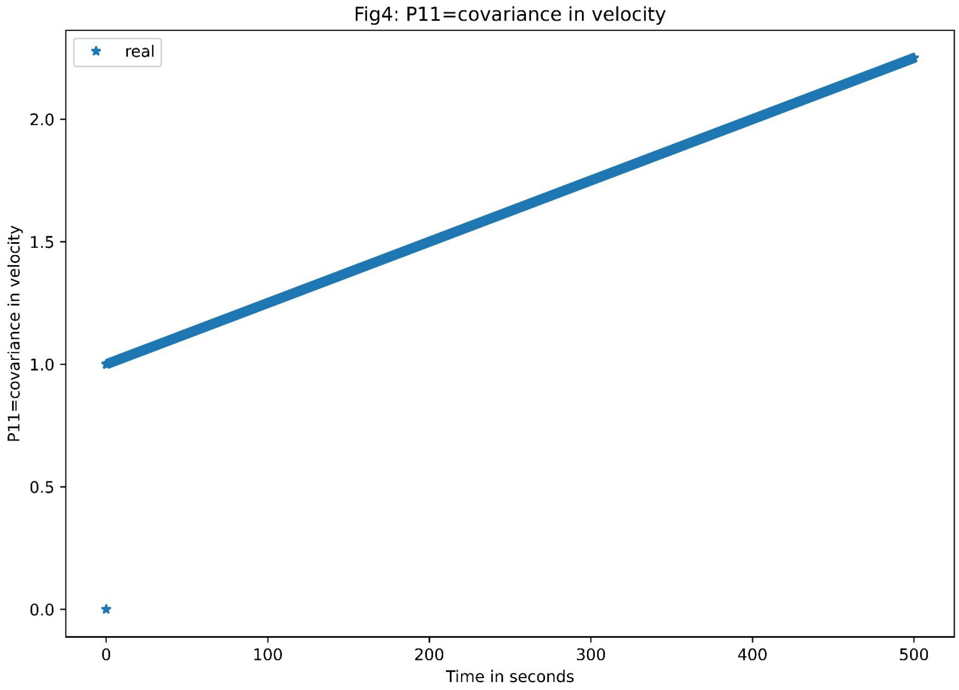
We see the divergence in error of estimated velocity



We see the exponentially increasing covariance of estimated position. This means there is no confidence in estimated values.



We see the increasing covariance of estimated velocity. This means there is no confidence in estimated values.



Qn 3a

A = transition matrix = `A=np.array([[0, 1], [0,0]])`

Measurement matrix = C = `C = np.array([[1,0]])`

Qn 3a

Equations of Kalman filter:

```
bias_std_dev = math.sqrt(1e-6)
    #euler integ of bias to get bias, bias_dot=omega=drawn fron N(0,bias_std_dev^2)
    bias_dot = np.random.normal(loc=0, scale=bias_std_dev)
    bias = bias + bias_dot*dt

    zeta_noise = np.random.normal(loc=0, scale=math.sqrt(2.5e-3))

    acceleration_real_value = 10*(math.sin(0.1*k*dt))
    u = np.array([[acceleration_real_value,0]]).T
    # u = np.array([[acceleration_real_value,bias_dot]]).T

    #we have two state variables, i.e., position, velocity
    # for exact x, we use exact accn
    x_dot = A@x + B@u
    #euler integ
    x = x + x_dot*dt
    y_exact = C@x


    # print(x)
    accn_model = acceleration_real_value + bias + zeta_noise
    u_model = np.array([[accn_model,0]]).T
    # u_model = np.array([[accn_model,bias_dot]]).T

    #for x_hat we use accn_model
    x_hat_dot = A@x_hat + B@u_model
    x_hat = x_hat + x_hat_dot*dt
    y_hat = C@x_hat

    t=t+dt
```

```
# #filter
    # #prediction stage: This is where we make an a-priori prediction on what the system will do
    # #we predict assuming that we know the A and B matrices
    # x_tilde=Ad@x_hat+B*u
    # y_tilde=C@x_tilde

    #we predict the error covariance here,
    P=Ad@P@Ad.transpose()+dt*Q


    # #correction step, i.e. we use measurements to correct for any errors
    #we use GPS

    gps_noise = np.random.normal(0,3**0.5)

    if k%(1/dt)==0:

        position_from_GPS = y_exact[0,0] + gps_noise
        #P_tilde = Pk- = P
        #using eqn 78:
        #H = C
        K=P@C.T@np.linalg.inv(C@P@C.T+R)

        #x_tilde=x_hat, update x_hat
        y_hat = C@x_hat

        x_hat=x_hat+K*(position_from_GPS-y_hat[0,0]) #this is the "money" step, the actual correction to the estimate is applied here

        #correction of P
        P=(np.eye(2,2)-K@C)@P #error covariance correction
```

Qn 3

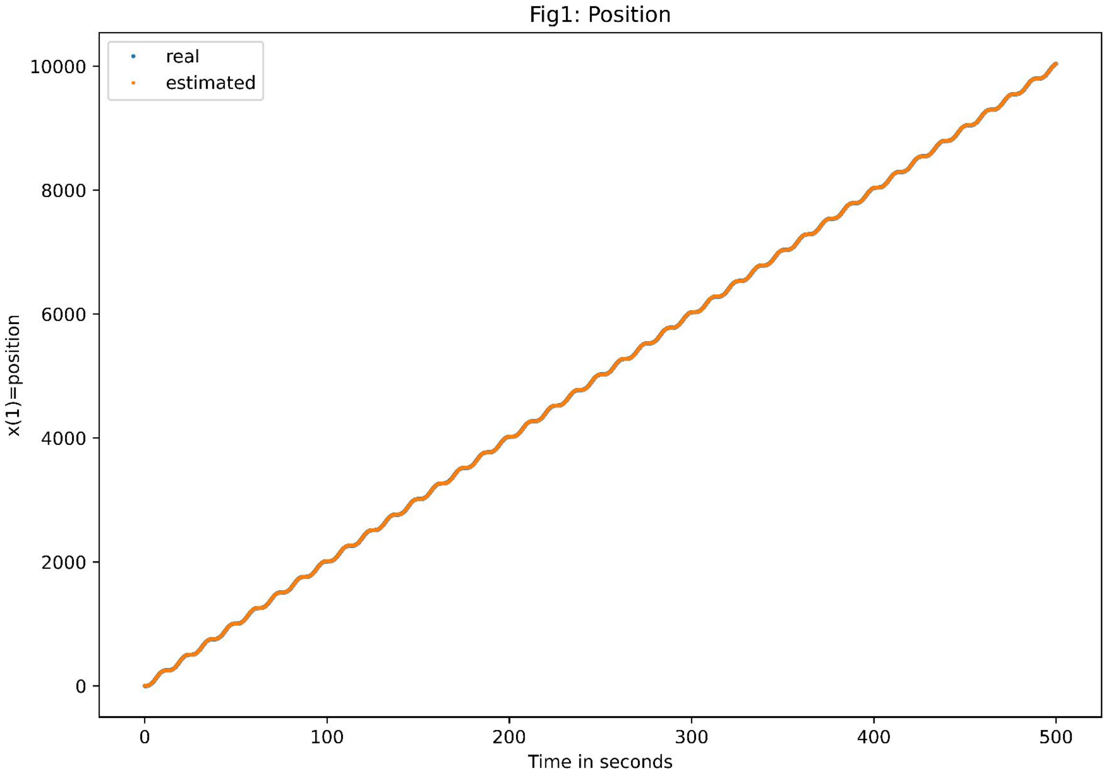


Fig1: Position
real
estimated
10000
8000
6000
4000
2000
0
x(1)=position
0
100
200
300
400
500
Time in seconds

Qn 3

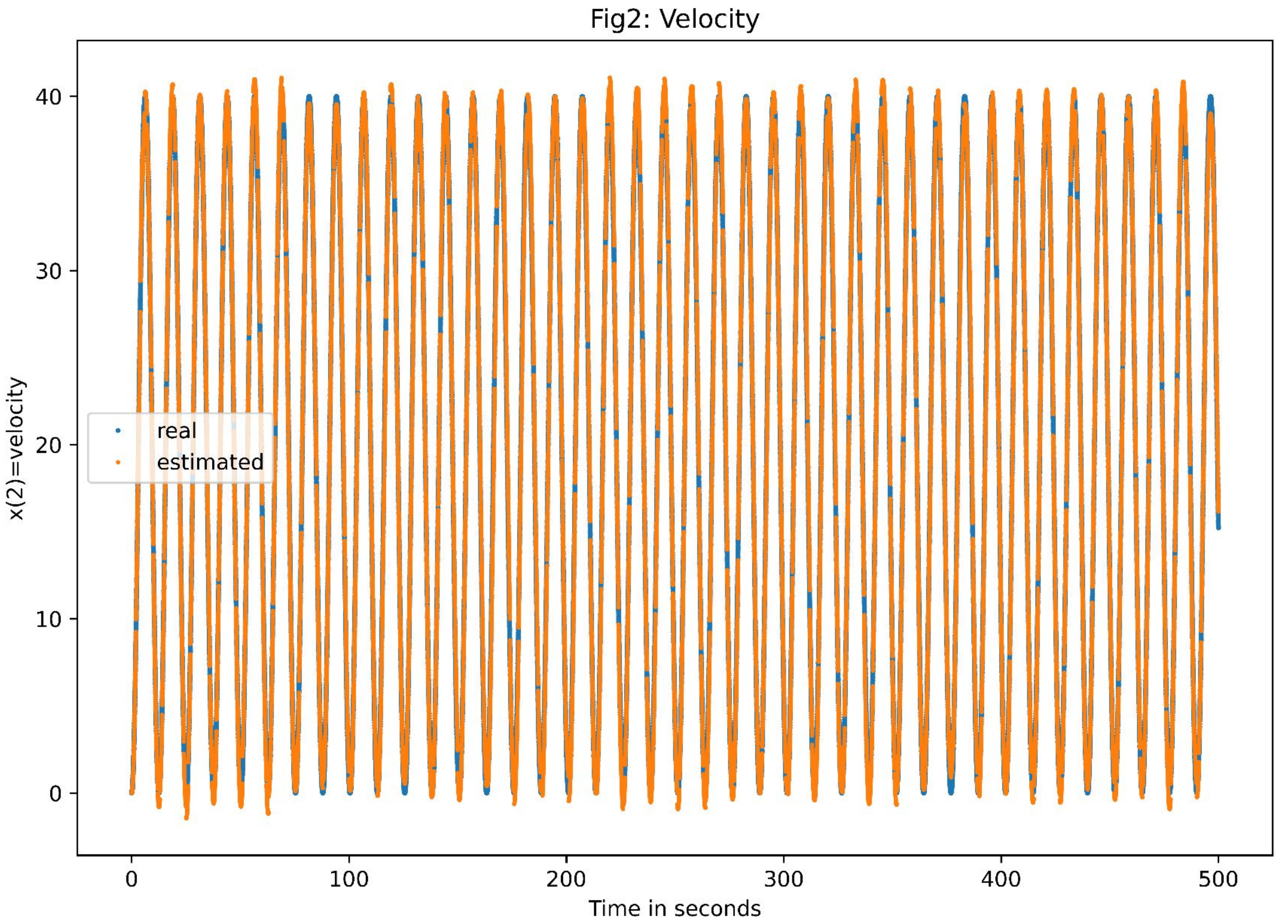


Fig2: Velocity
x(2)=velocity
Time in seconds
real
estimated
0
10
20
30
40
0
100
200
300
400
500

Qn 3

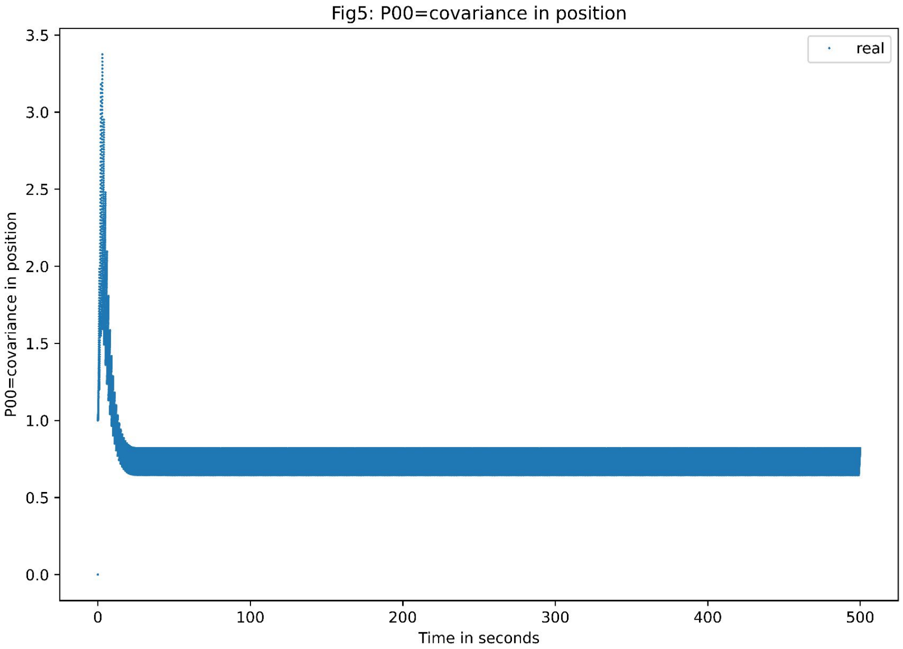


Fig5: P00=covariance in position
real
P00=covariance in position
Time in seconds
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
100
200
300
400
500

Qn 3

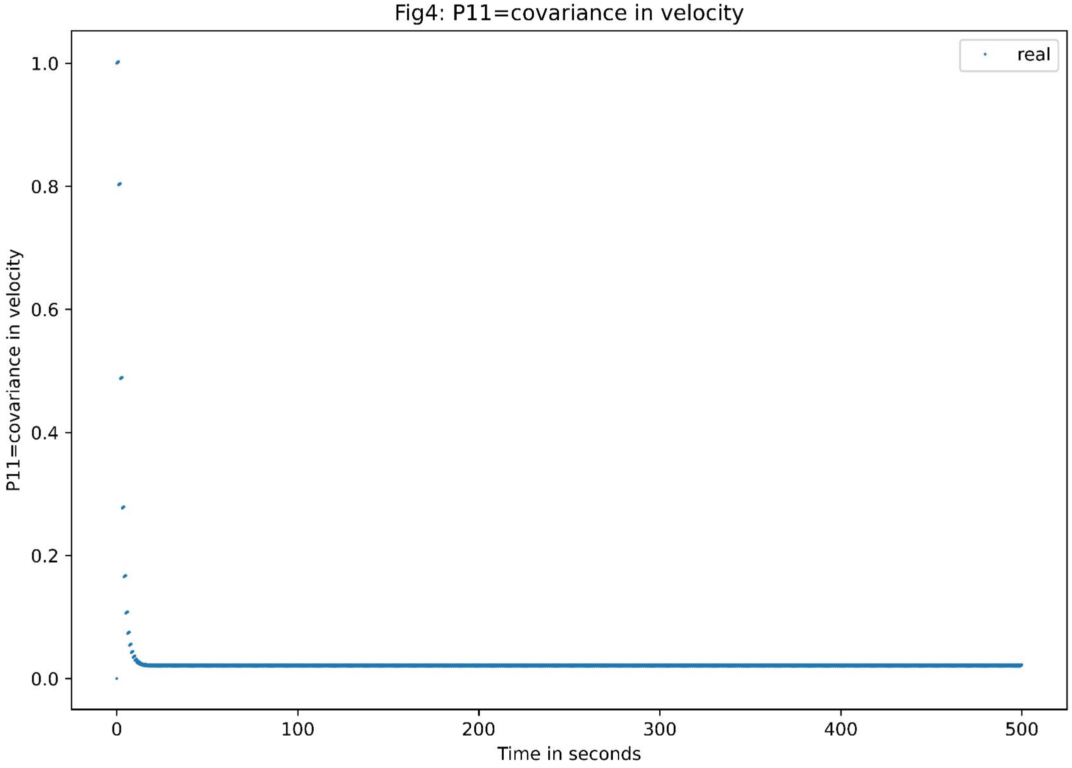


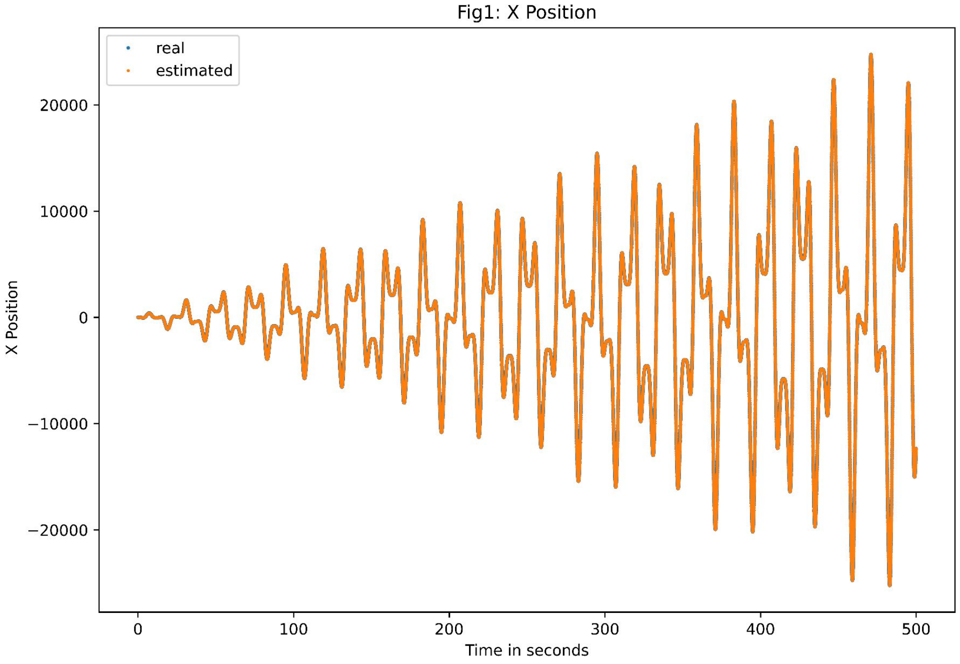
Fig4: P11=covariance in velocity
P11=covariance in velocity
Time in seconds
real
1.0
0.8
0.6
0.4
0.2
0.0
0
100
200
300
400
500

Qn 4

$$A = \begin{bmatrix} 0 & 0 & -v\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & v\cos(\theta) & \sin(\theta) & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Here v=0 and theta = pi/4

Qn3.3 Posn, vel, angle plots



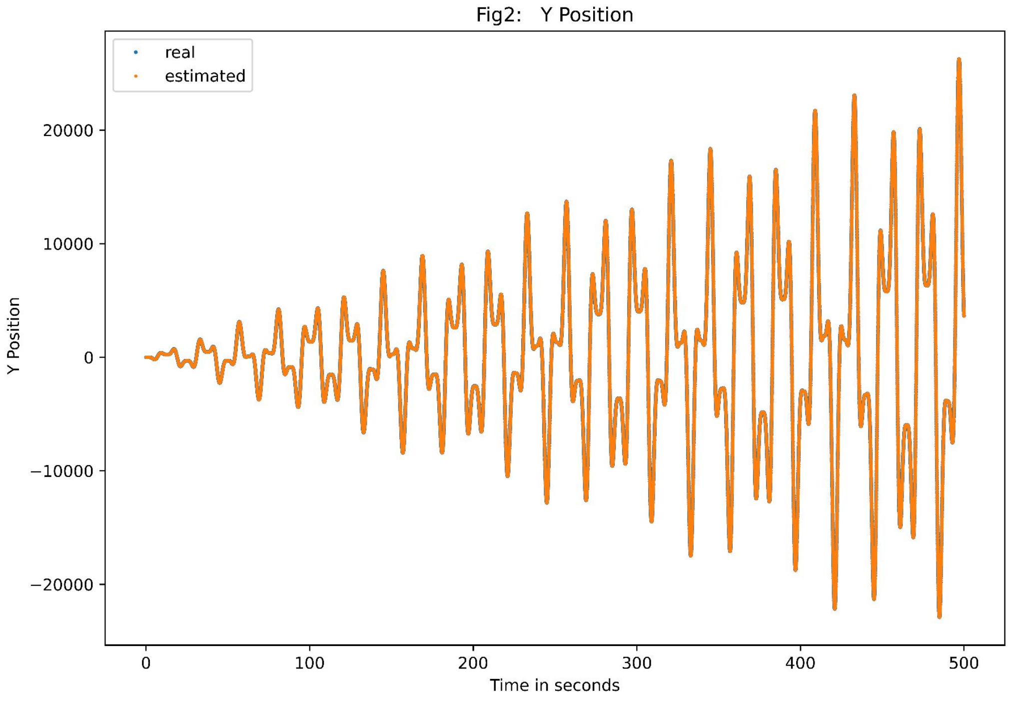


Fig2:   Y Position
real
estimated
20000
10000
0
−10000
−20000
Y Position
0
100
200
300
400
500
Time in seconds

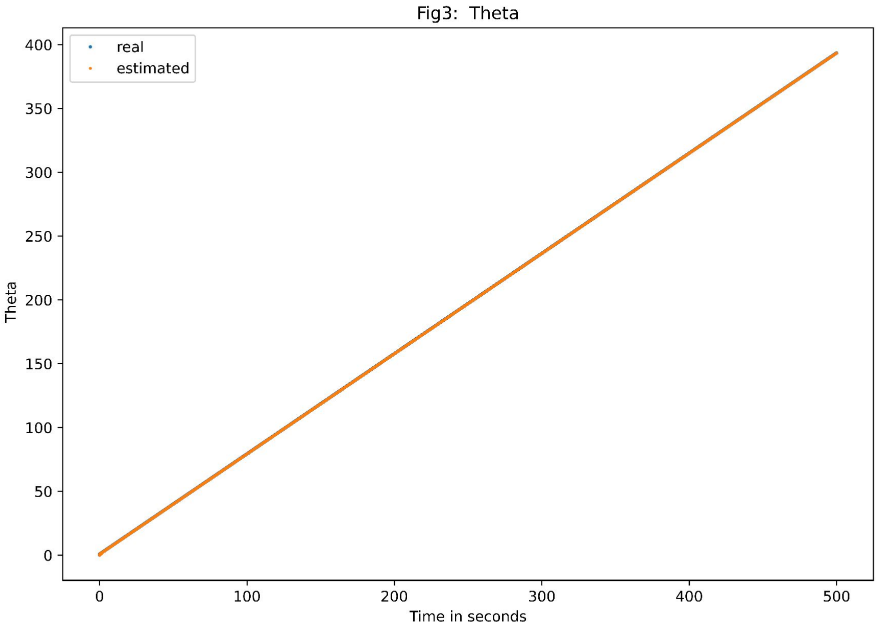


Fig3: Theta
real
estimated
Theta
Time in seconds
0
50
100
150
200
250
300
350
400
0
100
200
300
400
500

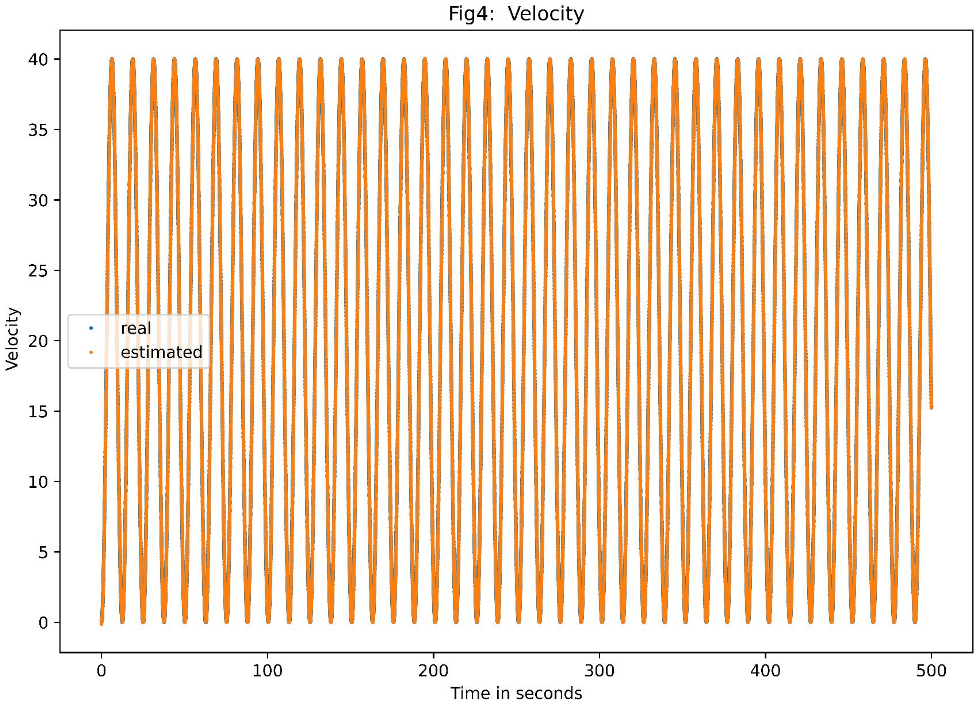


Fig4: Velocity
Velocity
Time in seconds
real
estimated
0
5
10
15
20
25
30
35
40
0
100
200
300
400
500

3.4

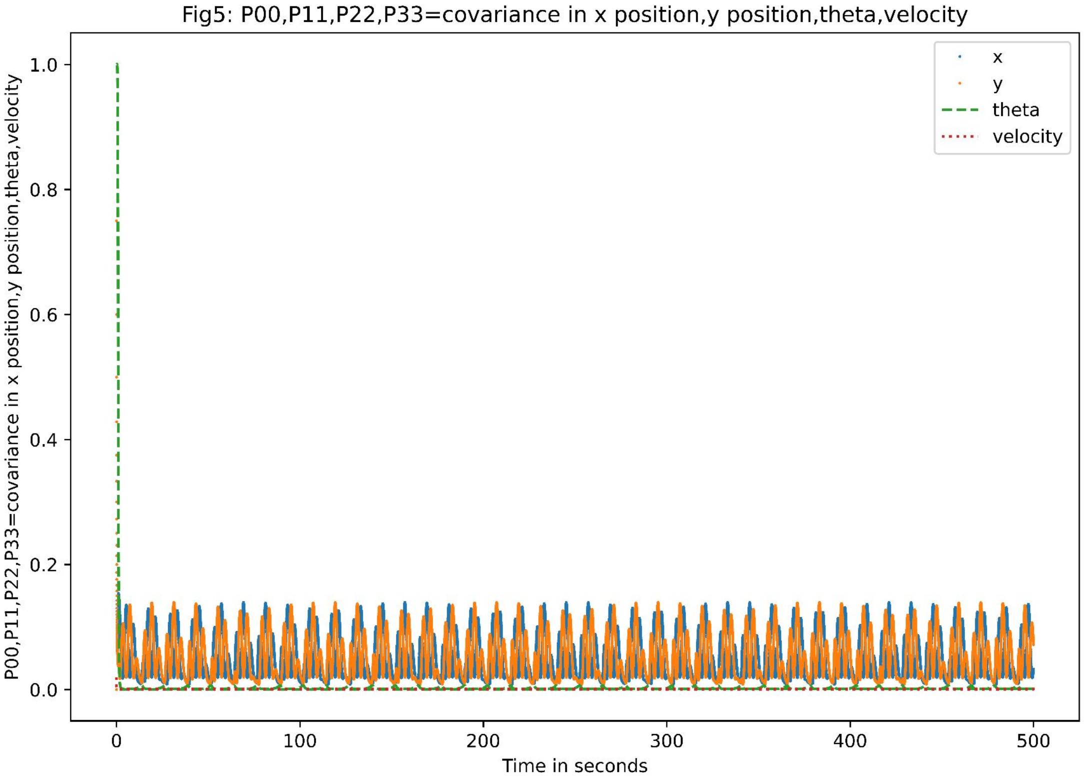


Fig5: P00,P11,P22,P33=covariance in x position,y position,theta,velocity
x
y
theta
velocity
1.0
0.8
0.6
0.4
0.2
0.0
P00,P11,P22,P33=covariance in x position,y position,theta,velocity
0
100
200
300
400
500
Time in seconds