

HW № 5

Matvey Plevako
m.plevako@innopolis.university
BS18-02
Variant (c)

Problem

Problem 1

Consider classical benchmark system in control theory - inverted pendulum on a cart (Figure 1). It is nonlinear under-actuated system that has the following dynamics.

$$(M + m)\ddot{x} - ml \cos(\theta)\ddot{\theta} + ml \sin(\theta)\dot{\theta}^2 = F$$

$$-\cos(\theta)\ddot{x} + l\ddot{\theta} - g \sin(\theta) = 0$$

where $g = 9.81$ is gravitational acceleration.

$$(c) M = 3.6, m = 3.6, l = 1.01$$

The system dynamics can be written in state space form:

$$\dot{z} = f(z) + g(z)u$$

$$y = h(z) = \begin{bmatrix} x & \theta \end{bmatrix}^T$$

where $z = \begin{bmatrix} x & \theta & \dot{x} & \dot{\theta} \end{bmatrix}^T$ is the state vector of the system, y is the output vector. The dynamics of the system around unstable equilibrium of the pendulum ($\bar{z} = \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}^T$) can be described by a linear system that is obtained from linearization of the nonlinear dynamics around \bar{z} .

$$\delta \dot{z} = A \delta z + B \delta u$$

$$\delta y = C \delta z$$

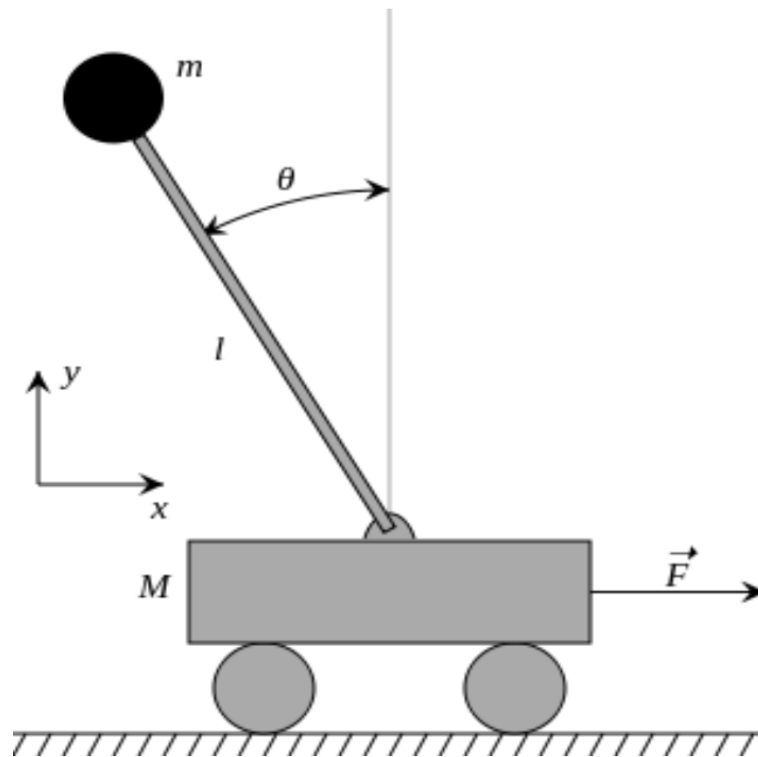


Figure 1: A schematic drawing of the inverted pendulum on a cart. The rod is considered massless. The mass of the cart and the point mass at the end of the rod are denoted by M and m . The rod has a length l .

part A

Description: prove that it is possible to design state observer of the linearized system

Solution: System is observable, if matrix $S = \begin{bmatrix} C \\ CA \\ CA^2 \\ CA^3 \end{bmatrix}$ has rank 4

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

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

$$CA^2 = \begin{bmatrix} 0 & \frac{mg}{M} & 0 & 0 \\ 0 & \frac{g(m+M)}{lM} & 0 & 0 \end{bmatrix}$$

$$CA^3 = \begin{bmatrix} 0 & 0 & 0 & \frac{mg}{M} \\ 0 & 0 & 0 & \frac{g(m+M)}{lM} \end{bmatrix}$$

$$S = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{mg}{M} & 0 & 0 \\ 0 & \frac{g(m+M)}{lM} & 0 & 0 \\ 0 & 0 & 0 & \frac{mg}{M} \\ 0 & 0 & 0 & \frac{g(m+M)}{lM} \end{bmatrix}$$

We can see the Identity matrix 4x4 at the upper part of the S matrix => its rank is 4

part B

Description: for open loop state observer, is the error dynamics stable?

Solution: Open-loop state observer has a form: $\hat{\dot{z}} = A\hat{z} + Bu$ Error dynamics:

$$\epsilon = \hat{z} - z, \dot{\epsilon} = A\epsilon$$

Thus, open loop state observe is stable, when A is negative definite, which is not the case, because:

$$\text{Det}\left(\begin{bmatrix} -\lambda & 0 & 1 \\ 0 & -\lambda & 0 \\ 0 & \frac{mg}{M} & -\lambda \\ 0 & \frac{g(M+m)}{lM} & 0 & -\lambda \end{bmatrix}\right) = -\lambda(-\lambda^3 + \lambda(\frac{g(M+m)}{lM})) = \lambda^2(\lambda^2 - \frac{g(M+m)}{lM})$$

if $\lambda = 0$ or $\lambda = \pm\sqrt{\frac{g(M+m)}{lM}}$ $\lambda = \pm 4,4052174287$

part C

Description: design Luenberger observer for linearized system using both pole placement and LQR methods

Solution: Luenberger observer has form:

$$\hat{\dot{z}}_{k+1} = A\hat{z}_k + Bu_k + L(y_k - \hat{y}_k)$$

$$\hat{y}_k = C\hat{z}_k + Du_k$$

In the given case, $D = 0$ Pole placement method:

it is usually used in case: $A - BL \prec 0$,

now the system $A - LC \prec 0$ is given, if it is transposed: $A^T - C^T L^T \prec 0$

$$L^T = \text{poles}(A^T, C^T, \text{eigVals})$$

LQR method (possible, because, C is a part of Identity matrix):

$$L^T = \text{lqr}(A^T, C^T, Q, R)$$

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import scipy.signal as sig
4 from scipy.integrate import odeint
5 import scipy.linalg as lin
6
7 g = 9.81
8 M = 3.6
```

```

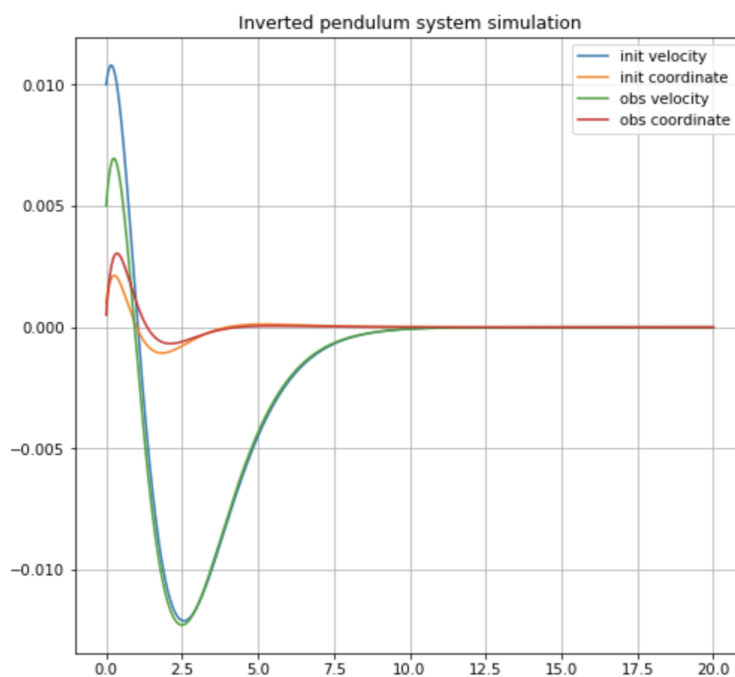
9  m = 3.6
10 l = 1.01
11
12 eig = [-1.1, -1.2, -1.3, -1.4]
13
14 A = np.array([[0, 0, 1, 0], [0, 0, 0, 1], [0, g*m/M, 0, 0], [0, g*(M+m)↵
        )/l/M, 0, 0]])
15 B = np.array([0, 0, 1/M, 1/l/M]).reshape(1, -1).T
16 C = np.array([[1, 0, 0, 0], [0, 1, 0, 0]])
17 # pole placement method
18 pole = sig.place_poles(A.T, C.T, eig)
19 L_pole = pole.gain_matrix.T
20
21 # lqr method
22 # Q, R - random, but appropriate
23 Q = np.array([[1, 0, 0, 0],
24               [0, 1, 0, 0],
25               [0, 0, 1, 0],
26               [0, 0, 0, 1]])
27
28 R = np.array([[4, 1], [1, 4]])
29
30 S = lin.solve_continuous_are(A.T, C.T, Q, R)
31 L_lqr = np.array(np.linalg.inv(R)).dot(C).dot(S).T
32
33 pole = sig.place_poles(A, B, eig)
34 P = -pole.gain_matrix
35
36 def usual(x, t, u):
37     n = np.dot(A, x) + np.dot(B, u)
38     return n
39
40 def observer(x_hat, t, u, x):
41     return np.dot(A, x_hat) + np.dot(B, u) + np.dot(L_lqr, C).dot(x - ↵
        x_hat)
42
43 dt = 1/10000
44 T = 20
45 time = np.linspace(0, T, dt**(-1))
46
47 x = [np.array([0.01, 0.001, 0.01, 0.01])]
48 x_hat = [np.array([0.02, 0.002, 0.02, 0.02])/4]
49
50 for i in range(1, len(time)):
51     # Use odeint between two dots,
52     # but u is fixed between two points
53     # P controller u = Px

```

```

54     local_time = np.linspace(time[i-1], time[i])
55     u = np.dot(P, x[-1])
56
57     x_dot = odeint(usual, x[-1], local_time, args=tuple([u]))
58     x.append(x_dot[-1])
59
60     x_hat_dot = odeint(observer, x_hat[-1], local_time, args=tuple([u, ↵
        x[-1]]))
61     x_hat.append(x_hat_dot[-1])
62
63 def plot_sim(x, x_hat, time):
64     x = np.array(x)
65     y = np.dot(C, x.T)
66     x_hat = np.array(x_hat)
67     y_hat = np.dot(C, x_hat.T)
68
69     plt.figure(figsize=(8, 8))
70     plt.title("Inverted pendulum system simulation")
71     plt.plot(time, y[0], label="init velocity")
72     plt.plot(time, y[1], label="init coordinate")
73
74     plt.plot(time, y_hat[0], label="obs velocity")
75     plt.plot(time, y_hat[1], label="obs coordinate")
76     plt.grid()
77     plt.legend()
78
79 plot_sim(x, x_hat, time)

```

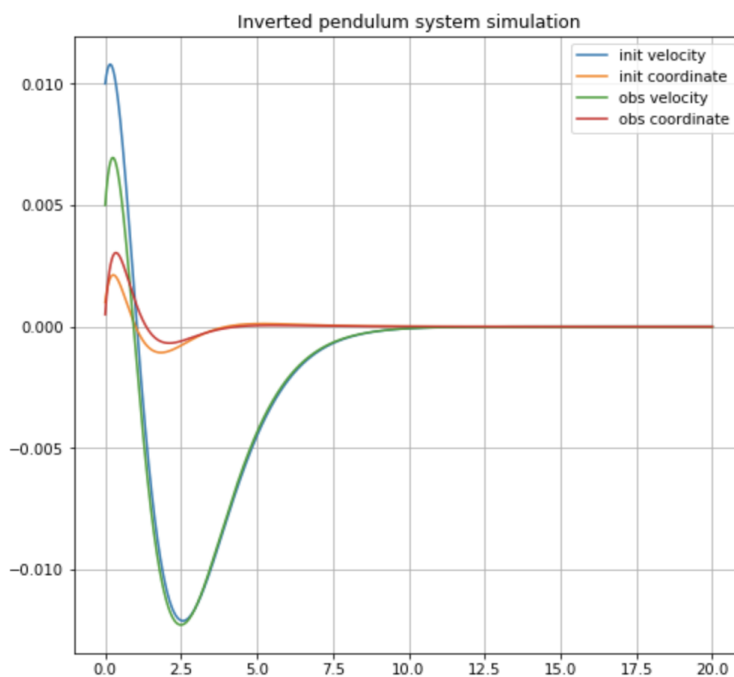


part D

Description: design state feedback controller for linearized system

Solution:

```
1 res_pole = sig.place_poles(A, B, eig)
2 K = res_pole.gain_matrix
3
4 # visualization
5 def control(x, t):
6     return np.dot(A - np.dot(B, K), x)
7
8 time = np.linspace(0, 20, 1000)
9 x0 = x[0]
10 res = odeint(control, x0, time).T
11
12 fig = plt.figure(figsize=(8, 8))
13 plt.title("Stabilisation of the system")
14 plt.xlabel("time")
15 plt.plot(time, res[0], "r-", label="x")
16 plt.plot(time, res[1], "b-", label="$\\theta$")
17 plt.plot(time, res[2], "k-", label="$\\dot{x}$")
18 plt.plot(time, res[3], "g-", label="$\\dot{\\theta}$")
19 plt.grid()
20 plt.legend(shadow=True)
21 plt.show()
```



part E

Description: Simulate nonlinear system with Luenberger observer and state feedback controller that uses estimated states ($u = K\hat{x}$). Make sure that the system is stabilized for various initial conditions around \bar{z} .

Solution:

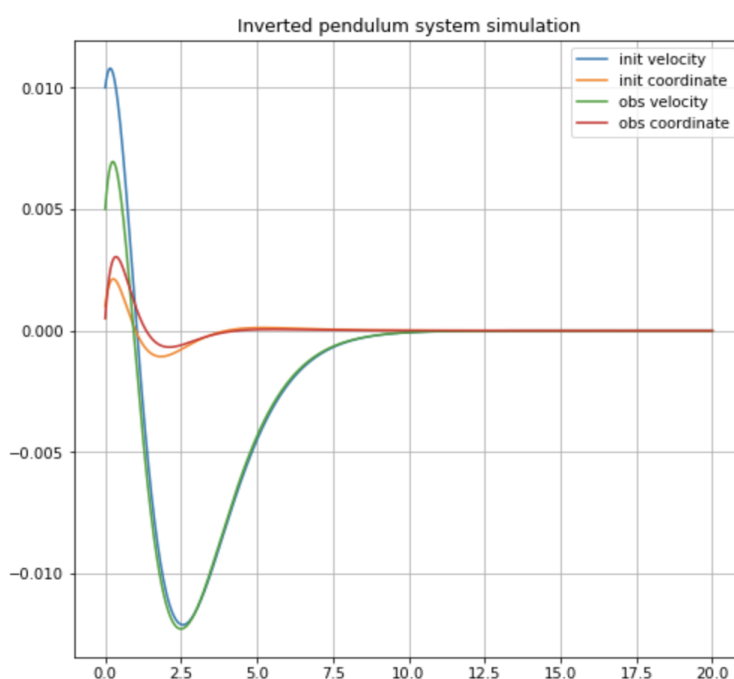
```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import scipy.signal as sig
4 from scipy.integrate import odeint
5 import scipy.linalg as lin
6
7 g = 9.81
8 M = 3.6
9 m = 3.6
10 l = 1.01
11
12 eig = [-1.1, -1.2, -1.3, -1.4]
13
14 A = np.array([[0, 0, 1, 0], [0, 0, 0, 1], [0, g*m/M, 0, 0], [0, g*(M+m)/l/M, 0, 0]])
15 B = np.array([0, 0, 1/M, 1/l/M]).reshape(1, -1).T
16 C = np.array([[1, 0, 0, 0], [0, 1, 0, 0]])
17 # pole placement method
18 pole = sig.place_poles(A.T, C.T, eig)
19 L_pole = pole.gain_matrix.T
20
21 # lqr method
22 # Q, R - random, but appropriate
23 Q = np.array([[1, 0, 0, 0],
24               [0, 1, 0, 0],
25               [0, 0, 1, 0],
26               [0, 0, 0, 1]])
27
28 R = np.array([[4, 1], [1, 4]])
29
30 S = lin.solve_continuous_are(A.T, C.T, Q, R)
31 L_lqr = np.array(lin.linalg.inv(R)).dot(C).dot(S).T
32
33
34 def usual(x, t, u):
35     n = np.dot(A, x) + np.dot(B, u)
36     return n
37
```



```

38
39 def observer(x_hat, t, u, x):
40     #     TRY WITH BOTH: L_lqr and L_pole
41     return np.dot(A, x_hat) + np.dot(B, u) + np.dot(L_lqr, C).dot(x - x_hat)
42
43 dt = 1/10000
44 T = 20
45 time = np.linspace(0, T, dt**(-1))
46
47 x = [np.array([0.01, 0.001, 0.01, 0.01])]
48 x_hat = [np.array([0.02, 0.002, 0.02, 0.02])/4]
49
50 for i in range(1, len(time)):
51     #     Use odeint between two dots,
52     #     but u is fixed between two points
53     #     P controller u = Px
54     local_time = np.linspace(time[i-1], time[i])
55     u = np.dot(P, x[-1])
56
57     x_dot = odeint(usual, x[-1], local_time, args=tuple([u]))
58     x.append(x_dot[-1])
59
60     x_hat_dot = odeint(observer, x_hat[-1], local_time, args=tuple([u, x[-1]]))
61     x_hat.append(x_hat_dot[-1])
62
63 plot_sim(x, x_hat, time)

```

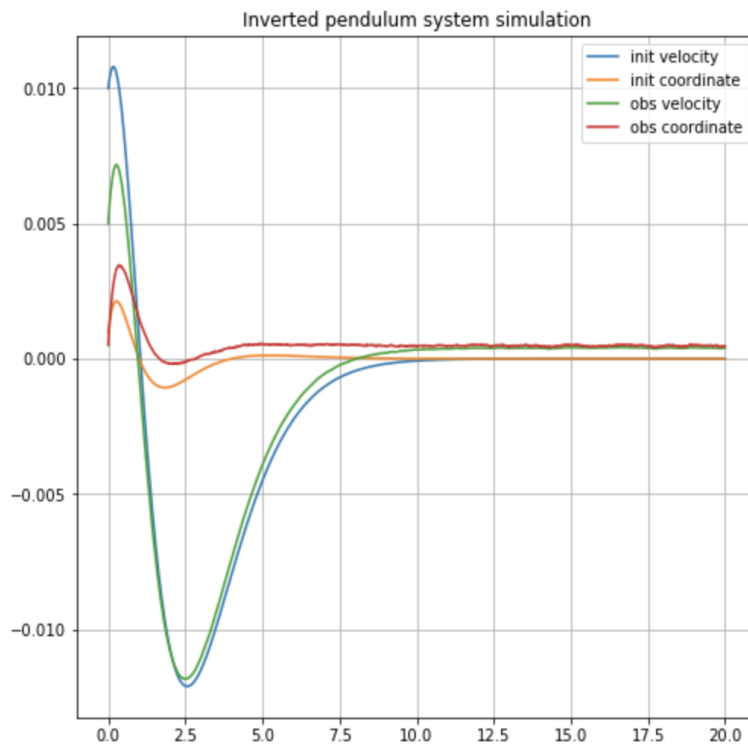


part F

Description: Add white gaussian noise to the output ($\delta y = C\delta z + v$).

Solution:

```
1 def usual(x, t, u):
2     n = np.dot(A, x) + np.dot(B, u)
3     return n
4
5
6 def observer(x_hat, t, u, dy):
7     return np.dot(A, x_hat) + np.dot(B, u) + np.dot(L_lqr, dy - np.dot(C, x_hat))
8
9 dt = 1/10000
10 T = 20
11 time = np.linspace(0, T, dt**(-1))
12
13 x = [np.array([0.01, 0.001, 0.01, 0.01])]
14 x_hat = [np.array([0.02, 0.002, 0.02, 0.02])/4]
15
16
17 for i in range(1, len(time)):
18     # BUT u is fixed between two points
19     # P controller u = P
20     local_time = np.linspace(time[i-1], time[i])
21     u = np.dot(P, x[-1])
22
23     x_dot = odeint(usual, x[-1], local_time, args=tuple([u]))
24     x.append(x_dot[-1])
25
26     dy = np.dot(C, x[-1])
27     dy += np.random.random(2) * 0.0005
28     x_hat_dot = odeint(observer, x_hat[-1], local_time, args=tuple([u, dy]))
29     x_hat.append(x_hat_dot[-1])
30
31 plot_sim(x, x_hat, time)
32 plt.show()
```



part G

Description: Add white gaussian noise to the dynamics ($\delta\dot{z} = A\delta z + B\delta u + w$). What happens to the state estimation and control system?

Solution:

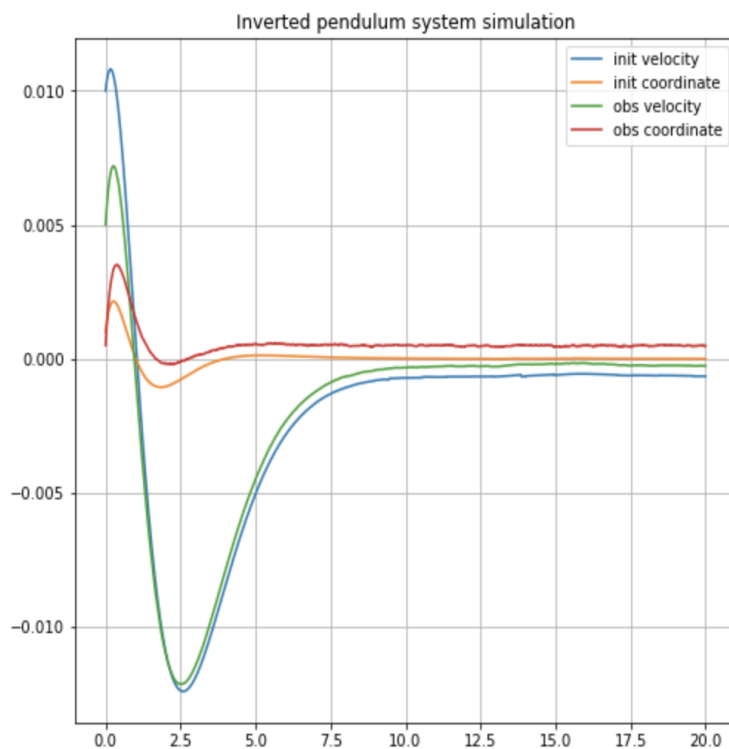
```

1  def usual(x, t, u):
2      n = np.dot(A, x) + np.dot(B, u) + np.random.random(4) * 0.00005
3      return n
4
5
6  def observer(x_hat, t, u, dy):
7      return np.dot(A, x_hat) + np.dot(B, u) + np.dot(L_lqr, dy - np.dot(C, x_hat))
8
9  dt = 1/10000
10 T = 20
11 time = np.linspace(0, T, dt**(-1))
12
13 x = [np.array([0.01, 0.001, 0.01, 0.01])]
14 x_hat = [np.array([0.02, 0.002, 0.02, 0.02])/4]
15
16
17 for i in range(1, len(time)):
18     # use odeint between two dots,
```

```

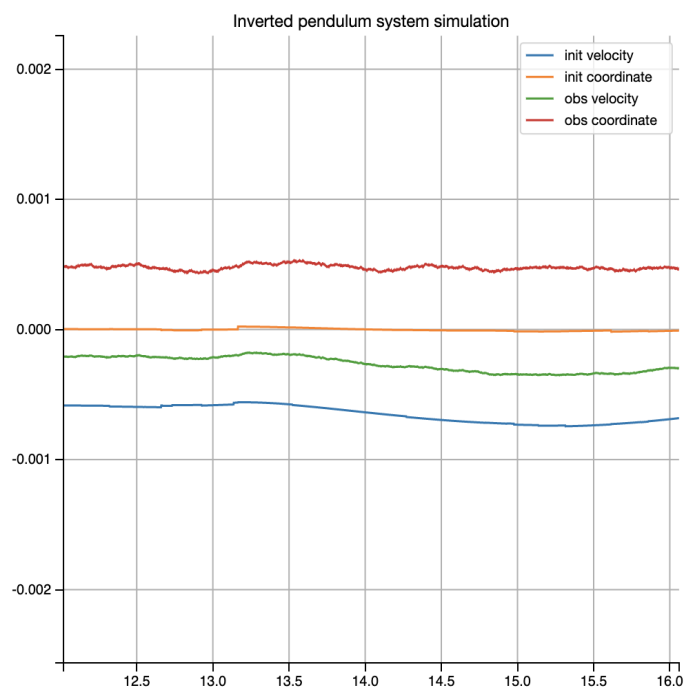
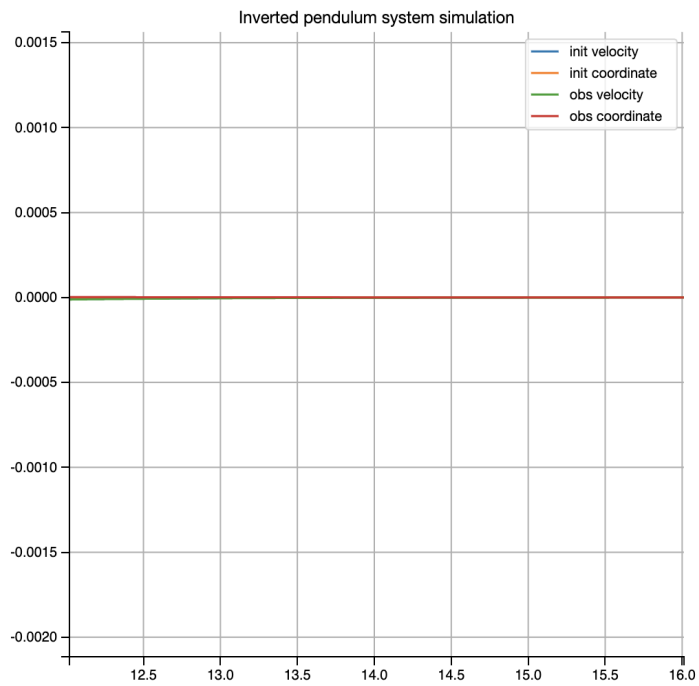
19 # but u is fixed between two poins
20 # P controller u = Px
21 local_time = np.linspace(time[i-1], time[i])
22 u = np.dot(P, x[-1])
23
24 x_dot = odeint(usual, x[-1], local_time, args=tuple([u]))
25 x.append(x_dot[-1])
26
27 dy = np.dot(C, x[-1])
28 dy += np.random.random(2) * 0.0005
29 x_hat_dot = odeint(observer, x_hat[-1], local_time, args=tuple([u, ↵
    dy]))
30 x_hat.append(x_hat_dot[-1])
31
32 plot_sim(x, x_hat, time)
33 plt.show()

```



Description: What happens to the state estimation and control system?

Solution: If we crop section from 12 to 14 we can see that noise does not allow to converge to 0, but it almost does not affect time to converge to "near zero" values it is 10.0 for both plots



part H

Description: implement Kalman Filter

Solution: Prediction

$$X_k^- = A_{k-1}X_{k-1} + B_kU_k$$

$$P_k^- = A_{k-1}P_{k-1}A_{k-1}^T + Q_{k-1}$$

Update

$$V_k = Y_k - H_kX_k^-$$

$$S_k = H_kP_k^-H_k^T + Q_{k-1}$$

$$K_k = P_k^-H_k^TS_k^{-1}$$

$$X_k = X_k^- + K_kV_k$$

$$P_k = P_k^- + K_kS_kK_k^T$$

```
1 class KalmanFilter(object):
2     def __init__(self, F = None, B = None, H = None, Q = None, R = None, P = None, x0 = None):
3         self.n = F.shape[1]
4         self.m = H.shape[1]
5
6         self.F = F
7         self.H = H
8         self.B = 0 if B is None else B
9         self.Q = np.eye(self.n) if Q is None else Q
10        self.R = np.eye(self.m) if R is None else R
11        self.P = np.eye(self.n) if P is None else P
12        self.x = np.zeros((self.n, 1)) if x0 is None else x0
13
14        def predict(self, u = 0):
15            self.x = np.dot(self.F, self.x) + np.dot(self.B, u)
16            self.P = np.dot(np.dot(self.F, self.P), self.F.T) + self.Q
17            return self.x
18
19        def update(self, z):
20            y = z - np.dot(self.H, self.x)
21            S = self.R + np.dot(self.H, np.dot(self.P, self.H.T))
22            K = np.dot(np.dot(self.P, self.H.T), np.linalg.inv(S))
23            self.x = self.x + np.dot(K, y)
24            I = np.eye(self.n)
25            self.P = np.dot(np.dot(I - np.dot(K, self.H), self.P),
26                            (I - np.dot(K, self.H)).T) + np.dot(np.dot(K, self.H.T), self.P)
```

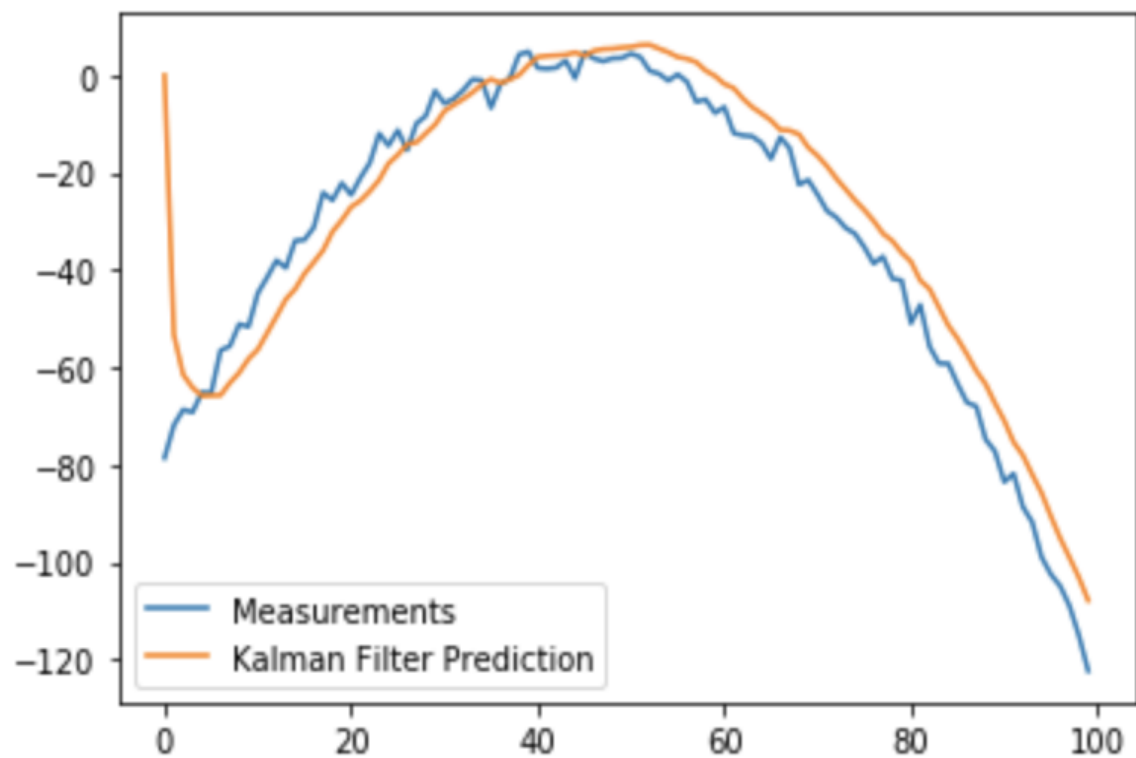
```
self.R), K.T)
```

part I

Description: generate some data and show that your implementation of KF is correct

Solution:

```
1 dt = 1.0/60
2 F = np.array([[1, dt, 0], [0, 1, dt], [0, 0, 1]])
3 H = np.array([1, 0, 0]).reshape(1, 3)
4 Q = np.array([[0.05, 0.05, 0.0], [0.05, 0.05, 0.0], [0.0, 0.0, 0.0]])
5 R = np.array([0.5]).reshape(1, 1)
6
7 x = np.linspace(-10, 10, 100)
8 measurements = - (x**2 + 2*x - 2) + np.random.normal(0, 2, 100)
9
10 kf = KalmanFilter(F = F, H = H, Q = Q, R = R)
11 predictions = []
12
13 for z in measurements:
14     predictions.append(np.dot(H, kf.predict())[0])
15     kf.update(z)
16
17
18 plt.plot(range(len(measurements)), measurements, label = 'Measurements↵
19 ')
20 plt.plot(range(len(predictions)), np.array(predictions), label = '↵
21     Kalman Filter Prediction')
```



part J

Description: using KF function implement LQG controller

Solution:

```

1  M = 1000;
2  m1 = 100;
3  m2 = 100;
4  l1 = 20;
5  l2 = 10;
6  g = 9.80;
7  x0= [ 5 ; 0 ; 0.1 ; 0 ; 0.2 ; 0 ; 0 ; 0 ; 0 ; 0 ; 0 ; 0 ; 0]
8  A= [0 1 0 0 0 0 ; 0 0 -(m1*g/M) 0 -(m2*g/M) 0 ; 0 0 0 1 0 0 ; 0 0 -(g*(M+m1))/(M*l1) 0 -(m2*g)/(M*l1) 0 ; 0 0 0 0 0 1; 0 0 -(m1*g)/(M*l2) 0 -(g*(M+m2))/(M*l2) 0]
9  B= [ 0; 1/M ; 0; 1/(M*l1) ; 0 ; 1/(M*l2)];
10 C = [1 0 0 0 0 0];
11 D = 0;
12 Q=[1 0 0 0 0 0; 0 1 0 0 0 0; 0 0 100 0 0 0; 0 0 0 1000 0 0; 0 0 0 0 1500 0; 0 0 0 0 0 1500]
13 R=0.0001
14 K = lqr(A,B,Q,R);
15 sys_1 = ss(A,[B B],C,[zeros(1,1) zeros(1,1)]);
16 vd = 0.3;
17 vn = 1;

```



```

18 sen = [1];
19 known = [1];
20 [~,L,~] = kalman(sys_1,vd,vn,[],sen,known)
21 Ac = [A-B*K B*K;zeros(size(A)) A-L*C];
22 Bc = zeros(12,1);
23 Cc = [C zeros(size(C))];
24 sys_cl_lqg = ss(Ac,Bc,Cc,D );
25
26 t = 0:0.01:100;
27 F = zeros(size(t));
28 [Y,~,X] = lsim(sys_cl_lqg,F,t,x0);
29 figure
30 plot(t,Y(:,1),'b');
31 u = zeros(size(t));
32 for i = 1:size(X,1)
33 u(i) = K * (X(i,1:6))';
34 end
35 Xhat = X(:,1) - X(:,6);
36 figure(2);
37 hold on
38 plot(t,Xhat)
39
40 plot(t,X(:,1),'r')
41 legend('X__hat','X')
42 hold off
43
44
45
46 x0 =
47
48     5.0000
49         0
50     0.1000
51         0
52     0.2000
53         0
54         0
55         0
56         0
57         0
58         0
59         0
60
61
62 A =
63
64         0     1.0000         0         0         0         0

```

| | | | | | | | |
|----|------------|---|---------|--------|---------|--------|---|
| 65 | 0 | 0 | -0.9800 | 0 | -0.9800 | 0 | |
| 66 | 0 | 0 | 0 | 1.0000 | 0 | 0 | |
| 67 | 0 | 0 | -0.5390 | 0 | -0.0490 | 0 | |
| 68 | 0 | 0 | 0 | 0 | 0 | 1.0000 | |
| 69 | 0 | 0 | -0.0980 | 0 | -1.0780 | 0 | |
| 70 | | | | | | | |
| 71 | | | | | | | |
| 72 | Q = | | | | | | |
| 73 | | | | | | | |
| 74 | 1 | 0 | 0 | 0 | 0 | 0 | ↔ |
| | | 0 | | | | | |
| 75 | 0 | 1 | 0 | 0 | 0 | 0 | ↔ |
| | | 0 | | | | | |
| 76 | 0 | 0 | 100 | 0 | 0 | 0 | ↔ |
| | | 0 | | | | | |
| 77 | 0 | 0 | 0 | 1000 | 0 | 0 | ↔ |
| | | 0 | | | | | |
| 78 | 0 | 0 | 0 | 0 | 0 | 150 | ↔ |
| | | 0 | | | | | |
| 79 | 0 | 0 | 0 | 0 | 0 | 0 | ↔ |
| | 1500 | | | | | | |
| 80 | | | | | | | |
| 81 | | | | | | | |
| 82 | R = | | | | | | |
| 83 | | | | | | | |
| 84 | 1.0000e-04 | | | | | | |
| 85 | | | | | | | |
| 86 | | | | | | | |
| 87 | L = | | | | | | |
| 88 | | | | | | | |
| 89 | 0.0303 | | | | | | |
| 90 | 0.0005 | | | | | | |
| 91 | 0.0000 | | | | | | |
| 92 | 0.0000 | | | | | | |
| 93 | 0.0001 | | | | | | |
| 94 | 0.0000 | | | | | | |

