**Homework 01**

**AA203: Optimal and learning-based Control**

Ricardo Luiz Moreira Paschoeto

[rp304154@stanford.edu](mailto:rp304154@stanford.edu)

**Problem 1:** Direct Model-reference Adaptive Control (MRAC),

(I)

**a)** Control law:

(III)

(IV)

(VI).

Substituting (III) in (I),

Using (V) and (VI),

Finally True plant match the reference model,

,

**b)** Given,

Find Differential equation for *e*,

Manipulating the equation above,

(VII).

**c)** Show that given

(VIII)

Substituting (VII) in (VIII), and , .

Using the adaptation law,

We have,

Using the property,

We have,

Where , , so V is definite-positive in x for is a function only of and is negative semi-definite, *e(t)* is bounded.

**d)** Barbalat's Lemma

,

We have *r(t)* bounded, so are bounded, therefore, *e(t)* is bounded, , and the function is bounded, then from Barbalat’s Lemma V is differentiable and has bounded derivate, so is uniformly continuous.

**e)** Apply MRAC to unstable plant,

Reference model,

Initial conditions,

Equations,

,

,

,

Results,

Gráfico, Gráfico de linhas

Descrição gerada automaticamente

Figure

Gráfico

Descrição gerada automaticamente

Figure

Gráfico, Gráfico de linhas

Descrição gerada automaticamente

Figure

Gráfico, Gráfico de linhas

Descrição gerada automaticamente

Figure 4

The trend for signals and for *r(t)=4* (figure 2) has an error in steady state. In both simulations the error *e(t)* = = 0, and for the *r(t) = 4sin(3t)* the trend for the signals and has error zero in steady state, My conclusion is related with the Boundedness of *r(t)* (in the case of *r(t)=4* is unbounded), that has influence in the Boundedness of error functions and for and , respectively, when (Lyapunov and Barbalat's Lemma).

**Python Code:**

import numpy as np

import matplotlib.pyplot as plt

alpha = -1

beta = 3

alpha\_m = 4

beta\_m = 4

dt = 0.01

gamma = 2.0

tval = np.linspace(0,10,int((10/dt)))

y = 0

ym = 0

kr = 0

ky = 0

y\_history = []

ym\_history = []

kr\_history = []

ky\_history = []

error\_history = []

delta\_r\_hist = []

delta\_y\_hist = []

y\_history.append(y)

ym\_history.append(ym)

kr\_history.append(kr)

ky\_history.append(ky)

kr\_star = beta\_m/beta

ky\_star = (alpha - alpha\_m)/beta

kr\_star\_hist = kr\_star\*np.ones(len(tval))

ky\_star\_hist = ky\_star\*np.ones(len(tval))

error = 0

error\_history.append(error)

dr = 0 - kr\_star

delta\_r\_hist.append(dr)

dy = 0 - ky\_star

delta\_y\_hist.append(dy)

ref\_in = 1

def ref(t):

    if ref\_in == 0:

        r = 4

    else:

        r = 4\*np.sin(3\*t)

    return r

def y\_dot(y, r, ky, kr):

    return 3\*control(y, r, ky, kr) + y

def control(y, r, ky, kr):

    return kr\*r + ky\*y

def ym\_dot(ym, r):

    return 4\*(r - ym)

def kr\_dot(gamma, error, r):

    return -gamma\*error\*r

def ky\_dot(gamma, error, y):

    return -gamma\*error\*y

def plot\_outputs(y\_history, ym\_history):

    plt.figure

    plt.plot(tval, y\_history)

    plt.plot(tval, ym\_history)

    plt.title('True Model and Reference Model')

    plt.legend('y')

    plt.legend('ym')

    plt.xlabel('time')

    plt.ylabel('Outputs')

    plt.grid(True)

    plt.show()

def plot\_gains(kr\_history, ky\_history, kr\_star\_hist, ky\_star\_hist):

    plt.figure

    plt.plot(tval, kr\_history,'b', 'LineWidth',1)

    plt.plot(tval, kr\_star\_hist,'b--', 'LineWidth',2)

    plt.plot(tval, ky\_history,'r', 'LineWidth',1)

    plt.plot(tval, ky\_star\_hist,'r--', 'LineWidth',2)

    plt.title('Gains Time history')

    plt.xlabel('time')

    plt.ylabel('Gains')

    plt.legend('kr')

    plt.legend('kr\*')

    plt.legend('ky')

    plt.legend('ky\*')

    plt.grid(True)

    plt.show()

for t in tval[1:]:

    r = ref(t)

    y = y + np.dot(y\_dot(y, r, ky, kr), dt)

    y\_history.append(y)

    ym = ym + np.dot(ym\_dot(ym, r), dt)

    ym\_history.append(ym)

    error = y - ym

    error\_history.append(error)

    kr = kr + np.dot(kr\_dot(gamma, error, r), dt)

    kr\_history.append(kr)

    ky = ky + np.dot(ky\_dot(gamma, error, y), dt)

    ky\_history.append(ky)

    delta\_r = kr - kr\_star

    delta\_r\_hist.append(delta\_r)

    delta\_y = ky - ky\_star

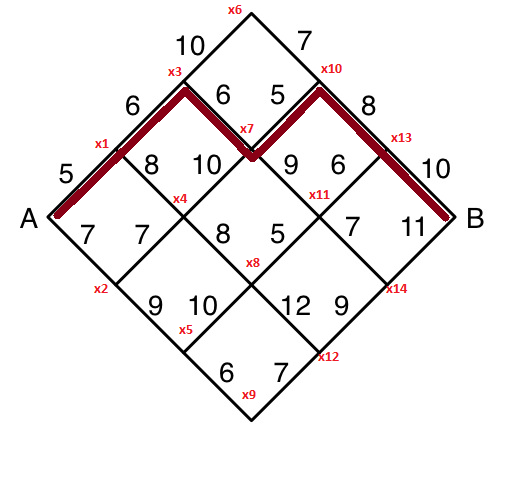
    delta\_y\_hist.append(delta\_y)

plot\_outputs(y\_history, ym\_history)

plot\_gains(kr\_history, ky\_history, kr\_star\_hist, ky\_star\_hist)

**Problem 2:** Shortest path through a grid

**a)** Dynamic Programming



Figure

Computation 1,

Computation 2,

Computation 3 and 4,

Computation 5,

Computation 6 and 7,

Computation 8,

Computation 9 and 10,

Computation 11,

Computation 12 and 13,

Computation 14 and 15,

Total Cost = 40

A - - B.

**b)** For the case of exhaustive search, we have *2n* and the order doesn’t matter. Each element from the total *n* can be chosen.

For *n=3*,

For the DP algorithm the relation is between the number of nodes. To compute the number of nodes by the number of segments (*n*) – nodes = . The number of computations in DP is equal to – *(nodes -1)* or .

**Problem 3: Machine maintenance**

In this problem we have two machine states – Running (R) ou Broke Down (B) – and for each state we have two actios – Running [Maintenance (m), Do Nothing (DN)] or Broke Down [Repair (rp), Replace (r)]. The costs and probabilities:

Maintenance - $20 – fail:04

Do Nothing - $0 – fail:0.7

Repair – $40 – fail:0.4

Replace - $150 – fail:0

Above we have the Table with set of possibilities for 4 weeks long, considering the new machine (Replaced) at the beginning of first week:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **W1** | **W2** | **W3** | **W4** |
| **1** | **R** | **R** | **R** | **R** |
| **2** | **R** | **R** | **R** | **B** |
| **3** | **R** | **R** | **B** | **R** |
| **4** | **R** | **R** | **B** | **B** |
| **5** | **R** | **B** | **R** | **R** |
| **6** | **R** | **B** | **R** | **B** |
| **7** | **R** | **B** | **B** | **R** |
| **8** | **R** | **B** | **B** | **B** |

1 – ()

-20

30 ()

20

70 ()

60

-

2 – Same as case 1, change in W04,

50 ()

=-20

-

3 – As case 2 change in W03,

10 ()

= -60

50 ()

40

-

4 – Like in Case 3, change in W04,

30 ()

=-40

-

5 - =-30 ()

=-100

10 ()

0

50 ()

40

-

6- =-30 ()

= -100

10 ()

0

30 ()

=-40

-

7 – Like 6, change in W3,

-10 ()

=-80

30 ()

20

-

8 – Like 7, change only in W4,

10 ()

=-60

-

**Problem 4:** Markovian drone

**a)** Heatmap for *V(x)*

Forma, Retângulo

Descrição gerada automaticamente

Figure 6 – Cost function

**b.1)** Policy heatmap

Gráfico

Descrição gerada automaticamente

Figure 7 – Policy.

**b.2)** Drone Trajectory

Gráfico

Descrição gerada automaticamente

Figure 8 – Trajectory of drone (squares with blue edge).

**Policy task**

For each state, the permitted actions were evaluated by the cost function, the actions most valuable will be chosen for optimal policy.

**Python code: Grid**

import numpy as np

class Grid:

    # action (0=up, 1=down, 2=left, 3=right)

    def \_\_init\_\_(self, n):

        self.actions = [0, 1, 2, 3]

        self.n = n

        self.all\_states = self.grid()

        self.ns = n\*n

        self.na = len(self.actions)

        self.rewards = {}

        self.aS = {}

    def grid(self):

        states = []

        for x1 in range(self.n):

            for x2 in range(self.n):

                states.append((x1,x2))

        return states

    def actions\_space(self, x\_goal):

        for x in self.all\_states:

            act = ()

            for a in self.actions:

                x\_next = self.execute(a, x)

                if self.validate(x\_next):

                    act = act + (a,)

            self.aS[x] = act

    def set\_rewards(self, x\_goal):

        for x in self.all\_states:

            if x == x\_goal:

                self.rewards[x] = 1

            else:

                self.rewards[x] = 0

    def prob(self, a, x):

        x\_next = self.execute(a, x)

        if not(self.validate(x\_next)):

                x\_next = x

        return x\_next

    def validate(self, x):

        x1, x2 = x

        if (x1 >= self.n or x1 < 0) or (x2 >= self.n or x2 < 0):

            return False

        return True

    def execute(self, a,x):

        x1, x2 = x

        if a == 0:

            x\_next = (x1, x2 + 1)

        elif a == 1:

            x\_next = (x1, x2 - 1)

        elif a == 2:

            x\_next = (x1 - 1, x2)

        else:# a == 3

            x\_next = (x1 + 1, x2)

        return x\_next

**Python code: Value Iteration**

import numpy as np

import matplotlib.pyplot as plt

from grid import Grid as G

import seaborn as sb

import pandas as pd

import math as m

from matplotlib.patches import Rectangle

n=20

env = G(n)

x\_eye=(15,15)

x\_goal=(19,9)

env.actions\_space(x\_goal)

env.set\_rewards(x\_goal)

V = {}

for x in env.all\_states:

    V[x] = 0

V[x\_goal] = 0

policy = {}

for x in env.aS.keys():

    policy[x] = np.random.choice(env.aS[x])

def value\_iteration(env, epsilon=0.0001, discount=0.95, sigma=10):

    def next\_step(V, x):

        x1, x2 = x

        x1\_eye, x2\_eye = x\_eye

        w = np.exp(-((x1 - x1\_eye)\*\*2 + (x2 - x2\_eye)\*\*2)/(2\*(sigma\*\*2)))

        v = np.zeros(4)

        for a in env.aS[x]:

            x\_next = env.prob(a, x)

            for r\_act in env.aS[x]:

                if (r\_act == a):

                    p = (1 - w + w/len(env.aS[x]))

                    r = env.rewards[x\_next]

                    v\_next = V[x\_next]

                else:

                    x\_next\_rand = env.prob(r\_act, x)

                    p = w/len(env.aS[x])

                    r = env.rewards[x\_next\_rand]

                    v\_next = V[x\_next\_rand]

                v[a] += p\*(r + discount\*v\_next)

        return v

    iteration = 0

    while True :

        delta = 0

        for x in env.all\_states:

            Q = next\_step(V, x)

            best\_v\_action = max(Q)

            delta = max(delta, np.abs(V[x] - best\_v\_action))

            V[x] = best\_v\_action

            policy[x] = np.argmax(Q)

        if delta < epsilon:

            break

        iteration += 1

    print(iteration)

    return V, policy

def plot\_heatmap(data, ant):

    m = np.array([data[key] for key in data.keys()]).reshape((n, n)).T

    ax = sb.heatmap(np.round(m, 3), annot=ant)

    ax.invert\_yaxis()

    p = path(data, (0,19))

    for t in p:

        ax.add\_patch(Rectangle(t, 1, 1, fill=False, edgecolor='blue', lw=3))

    plt.show()

def path(policy, x\_start):

    p = []

    x = x\_start

    N = 100

    p.append(x\_start)

    iteration = 0

    while iteration <= N:

        if x == x\_goal:

            break

        x = env.prob(policy[x], x)

        p.append(x)

        iteration += 1

    return p

V, policy = value\_iteration(env)

plot\_heatmap(V, False)

plot\_heatmap(policy, True)

**Problem 5:** cart-Pole balance

**a)** Given , linearize about () and yields LTI system,

Where

A =

Finally,

**b)** LQR controller

results,

[[0. 0. 0. 0.]]

[[ 0. -0.01 -0.01 -0.01]]

[[-0. -0.04 -0.02 -0.02]]

[[-0. -0.09 -0.03 -0.04]]

[[-0.01 -0.18 -0.04 -0.06]]

[[-0.01 -0.33 -0.06 -0.11]]

[[-0.02 -0.59 -0.07 -0.18]]

[[-0.02 -1.05 -0.09 -0.32]]

[[-0.03 -1.86 -0.11 -0.55]]

[[-0.04 -3.3 -0.13 -0.97]]

[[-0.05 -5.82 -0.16 -1.7 ]]

[[ -0.06 -10.18 -0.19 -2.97]]

[[ -0.07 -17.54 -0.23 -5.11]]

[[ -0.08 -29.44 -0.26 -8.57]]

[[ -0.09 -47.39 -0.28 -13.79]]

[[ -0.09 -71.82 -0.29 -20.91]]

[[-8.0000e-02 -1.0082e+02 -2.7000e-01 -2.9360e+01]]

[[-6.0000e-02 -1.3014e+02 -2.2000e-01 -3.7910e+01]]

[[-4.0000e-02 -1.5536e+02 -1.5000e-01 -4.5270e+01]]

[[-1.0000e-02 -1.7425e+02 -7.0000e-02 -5.0780e+01]]

[[ 1.000e-02 -1.870e+02 -0.000e+00 -5.451e+01]]

[[ 3.0000e-02 -1.9504e+02 6.0000e-02 -5.6860e+01]]

[[ 4.0000e-02 -1.9994e+02 1.2000e-01 -5.8300e+01]]

[[ 6.0000e-02 -2.0288e+02 1.7000e-01 -5.9160e+01]]

[[ 7.0000e-02 -2.0467e+02 2.2000e-01 -5.9690e+01]]

[[ 9.0000e-02 -2.0579e+02 2.6000e-01 -6.0020e+01]]

[[ 1.0000e-01 -2.0653e+02 3.0000e-01 -6.0240e+01]]

[[ 1.1000e-01 -2.0706e+02 3.5000e-01 -6.0410e+01]]

[[ 1.3000e-01 -2.0748e+02 3.9000e-01 -6.0540e+01]]

[[ 1.4000e-01 -2.0784e+02 4.4000e-01 -6.0650e+01]]

[[ 1.5000e-01 -2.0818e+02 4.9000e-01 -6.0750e+01]]

[[ 1.700e-01 -2.085e+02 5.300e-01 -6.086e+01]]

[[ 1.8000e-01 -2.0883e+02 5.9000e-01 -6.0960e+01]]

[[ 2.0000e-01 -2.0917e+02 6.4000e-01 -6.1070e+01]]

[[ 0.21 -209.52 0.7 -61.18]]

[[ 0.23 -209.88 0.76 -61.29]]

[[ 0.24 -210.25 0.82 -61.41]]

[[ 0.26 -210.64 0.88 -61.53]]

[[ 0.27 -211.05 0.95 -61.66]]

[[ 0.29 -211.47 1.02 -61.79]]

[[ 0.31 -211.9 1.09 -61.93]]

[[ 0.32 -212.35 1.16 -62.07]]

[[ 0.34 -212.82 1.23 -62.22]]

[[ 0.35 -213.29 1.31 -62.37]]

[[ 0.37 -213.78 1.39 -62.53]]

[[ 0.39 -214.28 1.47 -62.68]]

[[ 0.4 -214.79 1.55 -62.84]]

[[ 0.42 -215.3 1.64 -63.01]]

[[ 0.44 -215.82 1.72 -63.17]]

[[ 0.45 -216.35 1.8 -63.34]]

[[ 0.47 -216.89 1.89 -63.51]]

[[ 0.48 -217.42 1.98 -63.68]]

[[ 0.5 -217.96 2.06 -63.85]]

[[ 0.51 -218.5 2.15 -64.02]]

[[ 0.53 -219.03 2.23 -64.19]]

[[ 0.54 -219.56 2.32 -64.36]]

[[ 0.56 -220.09 2.4 -64.52]]

[[ 0.57 -220.61 2.48 -64.69]]

[[ 0.58 -221.13 2.56 -64.85]]

[[ 0.59 -221.63 2.64 -65.01]]

[[ 0.6 -222.13 2.72 -65.17]]

[[ 0.61 -222.61 2.8 -65.32]]

[[ 0.62 -223.09 2.87 -65.47]]

[[ 0.63 -223.55 2.94 -65.62]]

[[ 0.64 -224. 3.01 -65.76]]

[[ 0.65 -224.43 3.08 -65.9 ]]

[[ 0.66 -224.85 3.15 -66.03]]

[[ 0.67 -225.25 3.21 -66.16]]

[[ 0.67 -225.64 3.27 -66.28]]

[[ 0.68 -226.02 3.33 -66.4 ]]

[[ 0.68 -226.37 3.38 -66.51]]

[[ 0.69 -226.71 3.44 -66.62]]

[[ 0.69 -227.04 3.49 -66.72]]

[[ 0.7 -227.35 3.54 -66.82]]

[[ 0.7 -227.65 3.58 -66.92]]

[[ 0.71 -227.92 3.62 -67. ]]

[[ 0.71 -228.19 3.66 -67.09]]

[[ 0.71 -228.44 3.7 -67.17]]

[[ 0.71 -228.67 3.74 -67.24]]

[[ 0.72 -228.89 3.77 -67.31]]

[[ 0.72 -229.1 3.8 -67.38]]

[[ 0.72 -229.29 3.83 -67.44]]

[[ 0.72 -229.47 3.86 -67.49]]

[[ 0.72 -229.64 3.89 -67.55]]

[[ 0.72 -229.8 3.91 -67.6 ]]

[[ 0.72 -229.95 3.93 -67.64]]

[[ 0.72 -230.08 3.95 -67.69]]

[[ 0.72 -230.21 3.97 -67.73]]

[[ 0.72 -230.32 3.99 -67.76]]

[[ 0.72 -230.43 4. -67.8 ]]

[[ 0.72 -230.53 4.02 -67.83]]

[[ 0.72 -230.62 4.03 -67.86]]

[[ 0.72 -230.7 4.04 -67.88]]

[[ 0.72 -230.78 4.06 -67.91]]

[[ 0.72 -230.85 4.07 -67.93]]

[[ 0.72 -230.91 4.08 -67.95]]

[[ 0.72 -230.97 4.08 -67.97]]

[[ 0.72 -231.02 4.09 -67.98]]

[[ 0.72 -231.07 4.1 -68. ]]

[[ 0.72 -231.11 4.1 -68.01]]

[[ 0.72 -231.15 4.11 -68.02]]

[[ 0.72 -231.18 4.11 -68.04]]

[[ 0.72 -231.21 4.12 -68.04]]

[[ 0.71 -231.24 4.12 -68.05]]

[[ 0.71 -231.27 4.13 -68.06]]

[[ 0.71 -231.29 4.13 -68.07]]

[[ 0.71 -231.31 4.13 -68.07]]

[[ 0.71 -231.33 4.13 -68.08]]

[[ 0.71 -231.34 4.14 -68.08]]

[[ 0.71 -231.35 4.14 -68.09]]

[[ 0.71 -231.37 4.14 -68.09]]

[[ 0.71 -231.38 4.14 -68.1 ]]

[[ 0.71 -231.38 4.14 -68.1 ]]

[[ 0.71 -231.39 4.14 -68.1 ]]

[[ 0.71 -231.4 4.14 -68.1 ]]

[[ 0.71 -231.41 4.15 -68.11]]

[[ 0.71 -231.41 4.15 -68.11]]

[[ 0.71 -231.42 4.15 -68.11]]

[[ 0.71 -231.42 4.15 -68.11]]

[[ 0.71 -231.42 4.15 -68.11]]

[[ 0.71 -231.43 4.15 -68.11]]

[[ 0.71 -231.43 4.15 -68.11]]

[[ 0.71 -231.43 4.15 -68.11]]

[[ 0.71 -231.43 4.15 -68.11]]

[[ 0.71 -231.44 4.15 -68.12]]

[[ 0.71 -231.44 4.15 -68.12]]

[[ 0.71 -231.44 4.15 -68.12]]

[[ 0.71 -231.44 4.15 -68.12]]

[[ 0.71 -231.45 4.15 -68.12]]

[[ 0.71 -231.45 4.15 -68.12]]

[[ 0.71 -231.45 4.15 -68.12]]

[[ 0.71 -231.46 4.15 -68.12]]

[[ 0.71 -231.46 4.15 -68.12]]

[[ 0.71 -231.46 4.15 -68.12]]

[[ 0.71 -231.47 4.15 -68.12]]

[[ 0.71 -231.47 4.16 -68.13]]

[[ 0.71 -231.47 4.16 -68.13]]

[[ 0.71 -231.48 4.16 -68.13]]

[[ 0.71 -231.48 4.16 -68.13]]

[[ 0.71 -231.49 4.16 -68.13]]

[[ 0.71 -231.49 4.16 -68.13]]

[[ 0.71 -231.5 4.16 -68.13]]

[[ 0.71 -231.5 4.16 -68.14]]

[[ 0.71 -231.51 4.16 -68.14]]

[[ 0.71 -231.51 4.16 -68.14]]

[[ 0.71 -231.52 4.16 -68.14]]

[[ 0.71 -231.52 4.16 -68.14]]

[[ 0.71 -231.53 4.17 -68.14]]

[[ 0.72 -231.53 4.17 -68.15]]

[[ 0.72 -231.54 4.17 -68.15]]

[[ 0.72 -231.55 4.17 -68.15]]

[[ 0.72 -231.55 4.17 -68.15]]

[[ 0.72 -231.56 4.17 -68.15]]

[[ 0.72 -231.56 4.17 -68.16]]

[[ 0.72 -231.57 4.17 -68.16]]

[[ 0.72 -231.58 4.17 -68.16]]

[[ 0.72 -231.58 4.18 -68.16]]

[[ 0.72 -231.59 4.18 -68.16]]

[[ 0.72 -231.6 4.18 -68.17]]

[[ 0.72 -231.6 4.18 -68.17]]

[[ 0.72 -231.61 4.18 -68.17]]

[[ 0.72 -231.62 4.18 -68.17]]

[[ 0.72 -231.62 4.18 -68.17]]

[[ 0.72 -231.63 4.18 -68.18]]

[[ 0.72 -231.64 4.18 -68.18]]

[[ 0.72 -231.64 4.19 -68.18]]

[[ 0.72 -231.65 4.19 -68.18]]

[[ 0.72 -231.66 4.19 -68.18]]

[[ 0.72 -231.66 4.19 -68.19]]

[[ 0.72 -231.67 4.19 -68.19]]

[[ 0.72 -231.67 4.19 -68.19]]

[[ 0.72 -231.68 4.19 -68.19]]

[[ 0.72 -231.69 4.19 -68.19]]

[[ 0.72 -231.69 4.19 -68.2 ]]

[[ 0.72 -231.7 4.19 -68.2 ]]

[[ 0.72 -231.7 4.2 -68.2 ]]

[[ 0.72 -231.71 4.2 -68.2 ]]

[[ 0.72 -231.71 4.2 -68.2 ]]

[[ 0.72 -231.72 4.2 -68.2 ]]

[[ 0.73 -231.72 4.2 -68.21]]

[[ 0.73 -231.73 4.2 -68.21]]

[[ 0.73 -231.73 4.2 -68.21]]

[[ 0.73 -231.74 4.2 -68.21]]

[[ 0.73 -231.74 4.2 -68.21]]

[[ 0.73 -231.75 4.2 -68.21]]

[[ 0.73 -231.75 4.2 -68.22]]

[[ 0.73 -231.76 4.2 -68.22]]

[[ 0.73 -231.76 4.2 -68.22]]

[[ 0.73 -231.77 4.21 -68.22]]

[[ 0.73 -231.77 4.21 -68.22]]

[[ 0.73 -231.77 4.21 -68.22]]

[[ 0.73 -231.78 4.21 -68.22]]

[[ 0.73 -231.78 4.21 -68.22]]

[[ 0.73 -231.78 4.21 -68.23]]

[[ 0.73 -231.79 4.21 -68.23]]

[[ 0.73 -231.79 4.21 -68.23]]

[[ 0.73 -231.79 4.21 -68.23]]

[[ 0.73 -231.8 4.21 -68.23]]

[[ 0.73 -231.8 4.21 -68.23]]

[[ 0.73 -231.8 4.21 -68.23]]

[[ 0.73 -231.81 4.21 -68.23]]

[[ 0.73 -231.81 4.21 -68.23]]

[[ 0.73 -231.81 4.21 -68.23]]

[[ 0.73 -231.81 4.21 -68.23]]

[[ 0.73 -231.82 4.21 -68.24]]

[[ 0.73 -231.82 4.21 -68.24]]

[[ 0.73 -231.82 4.21 -68.24]]

[[ 0.73 -231.82 4.21 -68.24]]

[[ 0.73 -231.82 4.21 -68.24]]

[[ 0.73 -231.82 4.22 -68.24]]

[[ 0.73 -231.83 4.22 -68.24]]

[[ 0.73 -231.83 4.22 -68.24]]

[[ 0.73 -231.83 4.22 -68.24]]

[[ 0.73 -231.83 4.22 -68.24]]

[[ 0.73 -231.83 4.22 -68.24]]

[[ 0.73 -231.83 4.22 -68.24]]

[[ 0.73 -231.84 4.22 -68.24]]

[[ 0.73 -231.84 4.22 -68.24]]

[[ 0.73 -231.84 4.22 -68.24]]

[[ 0.73 -231.84 4.22 -68.24]]

[[ 0.73 -231.84 4.22 -68.24]]

[[ 0.73 -231.84 4.22 -68.24]]

[[ 0.73 -231.84 4.22 -68.24]]

[[ 0.73 -231.84 4.22 -68.24]]

[[ 0.73 -231.84 4.22 -68.24]]

[[ 0.73 -231.84 4.22 -68.24]]

[[ 0.73 -231.84 4.22 -68.24]]

[[ 0.73 -231.85 4.22 -68.24]]

[[ 0.73 -231.85 4.22 -68.24]]

[[ 0.73 -231.85 4.22 -68.24]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

[[ 0.73 -231.85 4.22 -68.25]]

**Python Code:**

import numpy as np

def riccati():

    dt = 0.1

    mp = 2.0

    mc = 10.0

    l = 1.0

    g = 9.81

    Pk = np.zeros((4,4))

    Kk = np.zeros((1,4))

    Q = np.eye(4,4)

    R = 1

    A = np.matrix([[1,0,dt,0],[0,1,0,dt],[0,dt\*mp\*g/mc,1,0],[0,dt\*(mc+mp)\*g/(mc\*l),0,1]])

    B = np.array([0,0,dt/mc,dt/(mc\*l)]).reshape(4,1)

    while True:

        Pk\_adv = Pk

        Kk = (np.linalg.inv(-(R + (B.T)@Pk\_adv@B))\*(B.T)@Pk\_adv@A)

        Pk = (Q + A.transpose()@Pk\_adv@(A + B@Kk))

        print(Kk.round(2))

        if np.linalg.norm(Pk\_adv - Pk) < 10\*\*(-4):

            break

    return Kk

K = riccati()

**c)** Simulate the system,

Gráfico, Gráfico de linhas

Descrição gerada automaticamente

Figure 9 -

**d)** Noise added,

Gráfico, Gráfico de linhas

Descrição gerada automaticamente

Figure 10 -

**Python Code:**

from scipy.integrate import odeint

import numpy as np

import matplotlib.pyplot as plt

import animations as an

def cartpole(s,t,u):

    mp = 2.0

    mc = 10.0

    l = 1.0

    g = 9.81

    \_, teta, x\_dot, teta\_dot = s

    x\_ddot = (mp\*np.sin(teta)\*(l\*teta\_dot\*\*2 + g\*np.cos(teta)) + u)/(mc + mp\*(np.sin(teta))\*\*2)

    teta\_ddot = -((mc + mp)\*g\*np.sin(teta) + mp\*l\*(teta\_dot\*\*2)\*np.sin(teta)\*np.cos(teta) + u\*np.cos(teta))/((mc + mp\*(np.sin(teta))\*\*2)\*l)

    dsdt = [x\_dot, teta\_dot, x\_ddot, teta\_ddot]

    return dsdt

def noise(mean, cov):

    cov\_matrix = np.diag(cov)

    w = np.random.multivariate\_normal(mean, cov\_matrix, 1)

    return w

def simulate(s0, animated, add\_noise):

    tf = 30.0

    dt = 0.1

    t = np.linspace(0, tf, int(tf/dt) + 1)

    kinf = [0.7291397,  -231.85419281,    4.21967188,  -68.24742825]

    w =  noise(np.array([0,0,0,0]), np.array([0, 0 ,10\*\*(-4),10\*\*(-4)]))

    s\_star = np.array([0, np.pi, 0, 0])

    if add\_noise:

        s = [s0 + w[0]]

    else:

        s = [s0]

    u = [kinf @ (s[0] - s\_star)]

    for k in range(len(t)-1):

        if add\_noise:

            w =  noise(np.array([0,0,0,0]), np.array([0, 0 ,10\*\*(-4),10\*\*(-4)]))

            s.append((odeint(cartpole, s[k], t[k:k+2],(u[k],))[1] + w[0]))

            u.append(kinf @ (s[k] - s\_star))

        else:

            s.append(odeint(cartpole, s[k], t[k:k+2],(u[k],))[1])

            u.append(kinf @ (s[k] - s\_star))

    \_, ax = plt.subplots()

    ax.plot(t, np.asanyarray(s)[:,0], 'k--', label='Displacement')

    ax.plot(t, np.asanyarray(s)[:,1], 'g:', label='Angle')

    ax.plot(t, np.asanyarray(s)[:,2], 'b', label='Velocity')

    ax.plot(t, np.asanyarray(s)[:,3], 'r', label='Agular Velocity')

    plt.grid(True)

    ax.legend()

    if animated:

        \_, \_ = an.animate\_cartpole(t, np.asanyarray(s)[:,0], np.asanyarray(s)[:,1])

    plt.show()

simulate(np.array([0, 3\*np.pi/4, 0, 0]), True, False)