Control Theory: Assignment 3

Mike Ivanov

Artem Bakhanov (B18-03)

Contents

1   Introduction

I created a GitHub repository here. All the problems are solved by me, Artem Bakhanov, a student of Innopolis University. My variant is **g**.

2   Python

For the first task I used Python to calculate everything. You can find the code in Appendix 1 in the end of the document.

2.1   A

In this subtask I tested 2 trajectories: $\overset{*}{x} = 1$ and $\overset{*}{x} = 4$. Zero-conditions are $x(0) = .28789452$ and $x'(0) = .72404173$
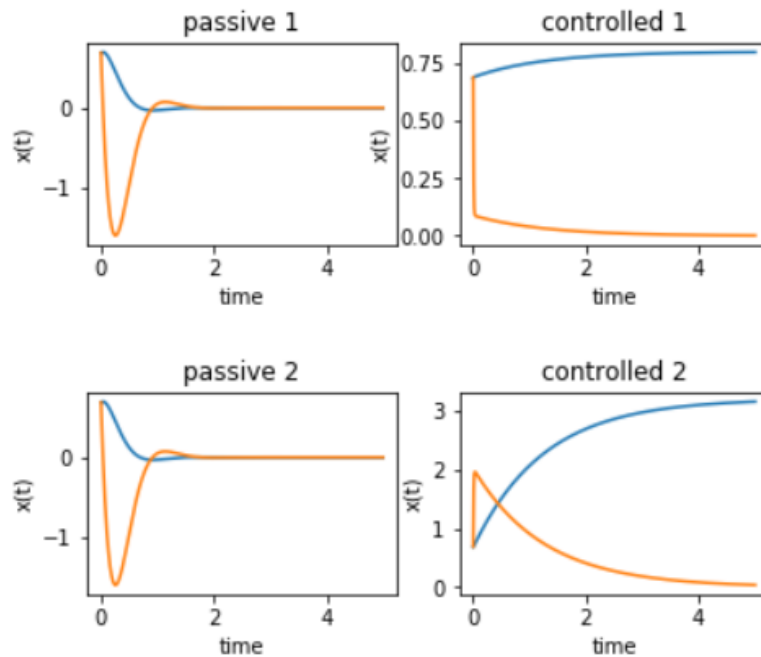

Figure 1: Two trajectories

2.2   B

For this part I used a process described on Wikipedia. I got parameters $k_p = 2000$ and $k_d = 300$. Trajectories are the same. I used step with amplitude 1 and starting time 0.5. On the plots below you can see that the PD controller works but not perfectly. I-component is missing.
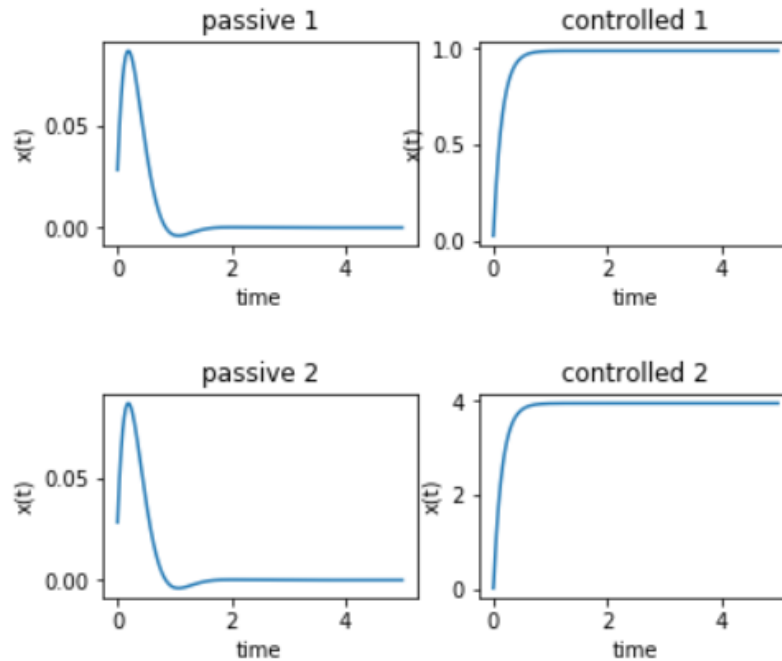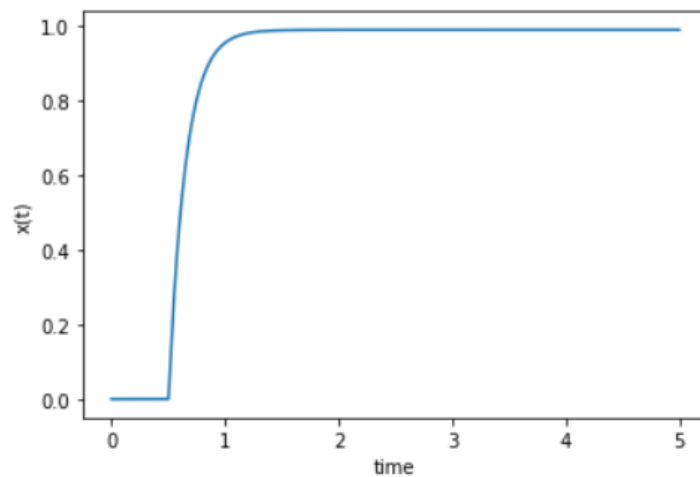
Figure 2: Tuned PD

Figure 3: Step response. $t_0 = 0.5$

2.3   C

For this part I used built-in Matlab function `isstable`. The function proved that the controlled oscillator dynamics is stable for $k_p = 2000$ and $k_d = 300$. You can find the code below.

2.4   D

The solution is pretty the same but the parameters are different. You can look at my code in the Appendix 1. I did not get how we can do something with the system where we do not know the B matrix, so I assumed

```
1  A = [0  1;  −25  −7];
2  B = [0;  1];
3  C = [1  0];
4  D = 1;
5  system = ss(A, B, C, D);
6  controller = pid(2000, 0, 300);
7  cl = system * controller;
8  disp(isstable(cl));
```
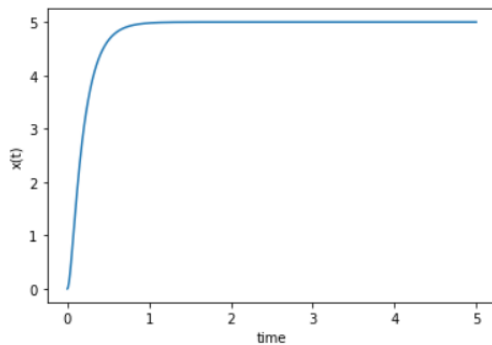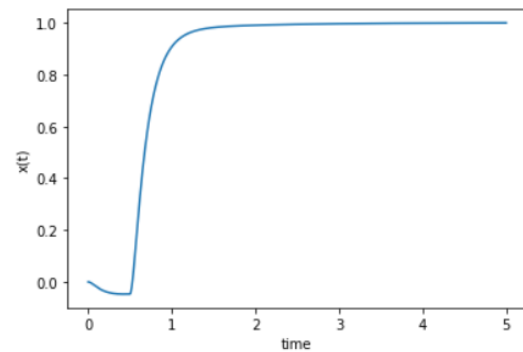
Listing 1: Stability proof

that $B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$. The problem is that I did not understand how to get the parameters, since it is now a numbers but matrices.

2.5   E

The PID controller is very similar to what was done in the task B. I found PID parameters by hand and got: 150, 130, and 25 for k_p, k_i, and k_d respectively.

(a) Desired x: 5

(b) Step response with amplitude 1 and time 0.5

Figure 4: Examples of tests.

3   PID Tuner

In this section I used Simulink for analyzing the system and PIDTuner to tune the PID controller in the system. The system looks like:
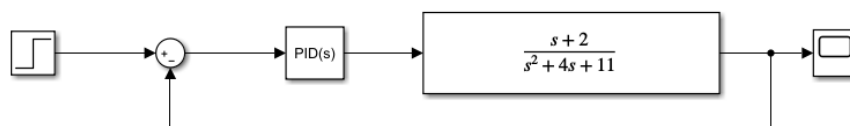

Figure 5: The system

The parameters that PIDTuner gave me: $k_p = 10.826321698108$, $k_i = 63.4181190369928$, and $k_d = -0.0076494311467772$. The step input (amplitude = 5) was used and the trajectory of the system is on the picture below.
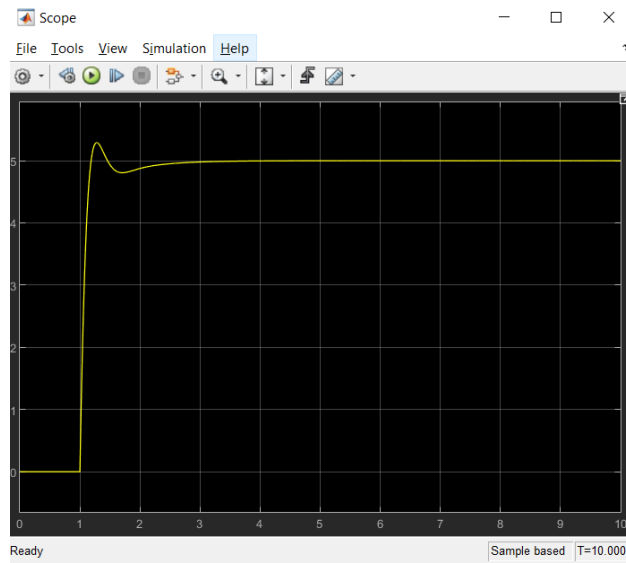
Figure 6: The system trajectory

Appendices

A   Python Code

<div align="center">

assignment3

April 1, 2020

</div>

```
[94]: import numpy as np
      from numpy.linalg import eig
      from scipy.integrate import odeint
      import matplotlib.pyplot as plt

      time = np.linspace(0, 5, 1000)      # interval from 0 to 5
      x0 = np.array([0.28789452, 0.72404173]) #np.random.rand(2)           # initial␣
       ↪state
```

1   Proportional-derivative (PD) control

Consider a second order linear ODE:

$$\ddot{x} + \mu\dot{x} + kx = u$$

We can use P control:

$$u = k_p(x^* - x)$$

or we can add a derivative term:

$$u = k_d(\dot{x}^* - \dot{x}) + k_p(x^* - x)$$

This is a **PD controller**.

Let us introduce $e = x^* - x$. Then we have:

$$u = k_d\dot{e} + k_p e$$

Variable e here is a **control error**.

```
[95]: mu = 7
      k = 25

      x_desired = 1
      x_dot_desired = 0
```

<div align="center">

1

</div>

```python
kp = 2000
kd = 300
# kp = 100
# kd = 150

def Oscillator(x, t):
    return np.array([x[1], (- mu*x[1] - k*x[0])])

def Oscillator_Control(x, t):
    error     = x_desired     - x[0]
    error_dot = x_dot_desired - x[1]
    u = kp*error + kd*error_dot
    return np.array([x[1], (u - mu*x[1] - k*x[0])])

def StepFunction(t):
    return 0 if t < 0.5 else 1

def Oscillator_StepFunction(x, t):
    x_desired = StepFunction(t)
    error     = x_desired     - x[0]
    error_dot = x_dot_desired - x[1]
    u = kp*error + kd*error_dot
    return np.array([x[1], (u - mu*x[1] - k*x[0])])

x0 = np.random.rand(2)

solution = {"Oscillator1": odeint(Oscillator, x0, time), "Oscillator_Control1":␣
 ↪odeint(Oscillator_Control, x0, time)}
x_desired = 4
x_dot_desired = 0
solution["Oscillator2"] =  odeint(Oscillator, x0, time)
solution["Oscillator_Control2"] =  odeint(Oscillator_Control, x0, time)


plt.subplot(221)
plt.plot(time, solution["Oscillator1"].T[0].T)
plt.xlabel('time')
plt.ylabel('x(t)')
plt.title('passive 1')

plt.subplot(222)
plt.plot(time, solution["Oscillator_Control1"].T[0].T)
plt.xlabel('time')
plt.ylabel('x(t)')
plt.title('controlled 1')
```
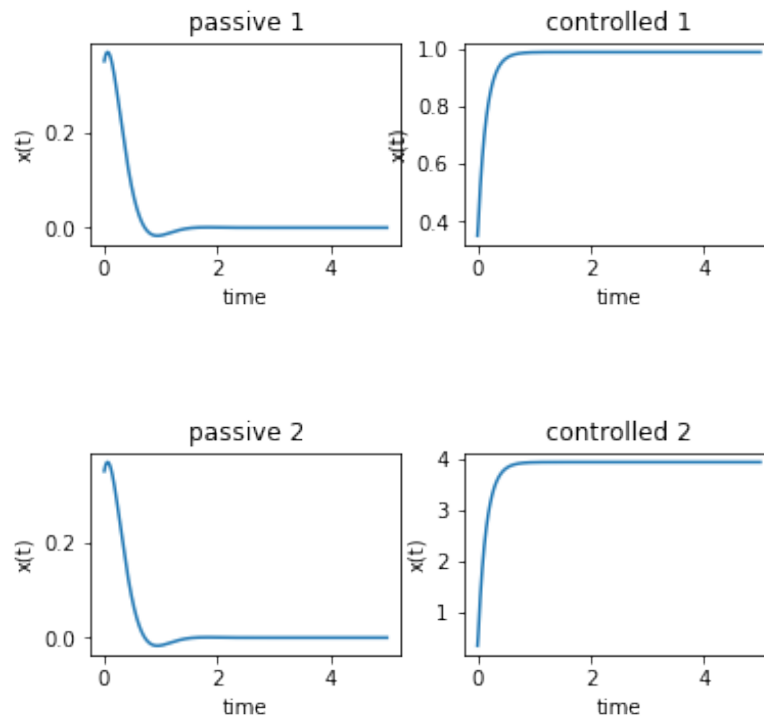
2

```python
plt.show()

plt.subplot(223)
plt.plot(time, solution["Oscillator2"].T[0].T)
plt.xlabel('time')
plt.ylabel('x(t)')
plt.title('passive 2')

plt.subplot(224)
plt.plot(time, solution["Oscillator_Control2"].T[0].T)
plt.xlabel('time')
plt.ylabel('x(t)')
plt.title('controlled 2')
plt.show()
```
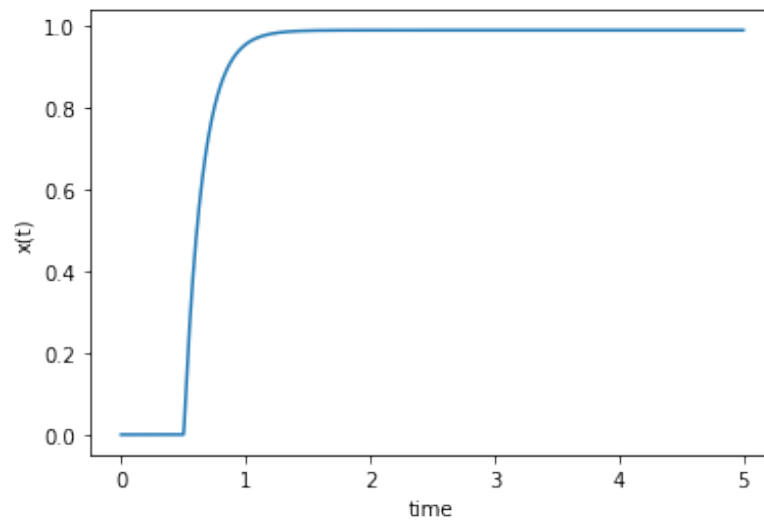

[96]:
```python
x0 = np.zeros((2))
```

3

```
solution = odeint(Oscillator_StepFunction, x0, time)

plt.plot(time, solution.T[0].T)
plt.xlabel('time')
plt.ylabel('x(t)')
plt.show()
```
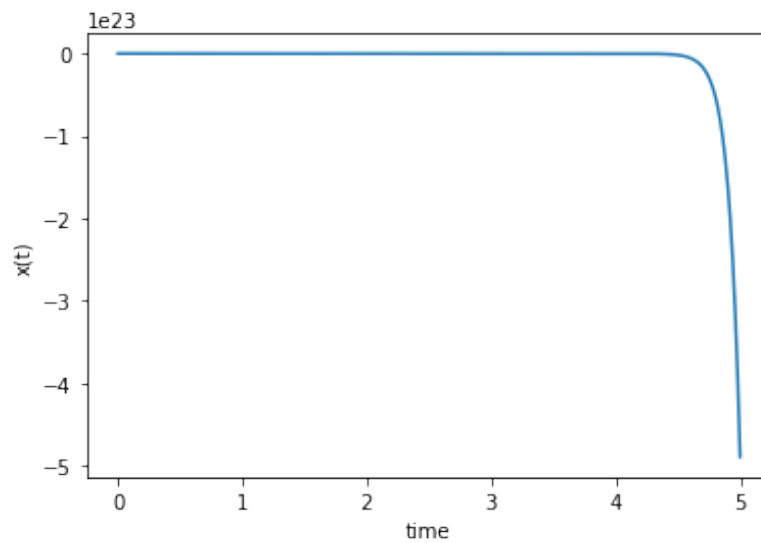

2 Task 2d

```
[97]: A = np.array([[10, 3], [5, -5]]) # state matrix

x_desired = 10
x_dot_desired = 0
kp = 1;
kd = 2;
def LTV(x, t):
    error     = x_desired     - x[0]
    error_dot = x_dot_desired - x[1]
    u = kp*error + kd*error_dot
    x_  = A.dot(x)
    x_[1] -= u
    return x_
```

4

```
[98]: solution = odeint(LTV, x0, time)

      plt.plot(time, solution.T[0].T)
      plt.xlabel('time')
      plt.ylabel('x(t)')
      plt.show()
```
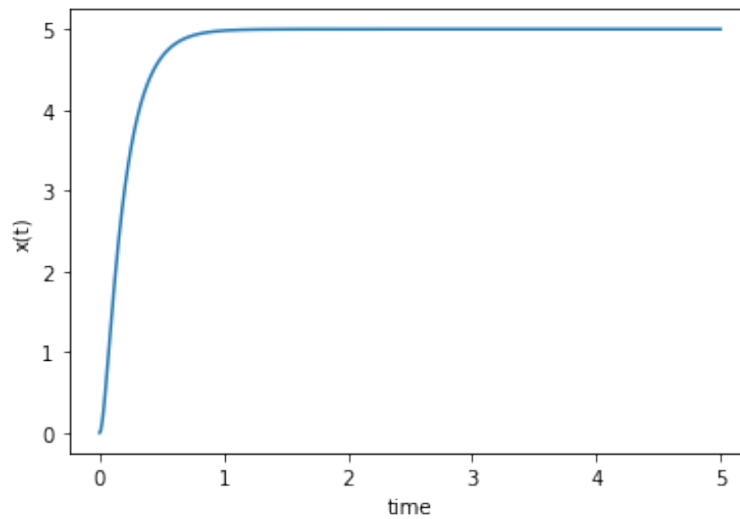

3   Task 2e

Here the PID controller is implemented

```
[239]: x_desired = 5
       x_dot_desired = 0
       kp = 150
       kd = 25
       ki = 130.0
       error_i = 0
       errors = []
       t_prev = 0
       def Oscillator_Control(x, t):
           global error_i, ki, kd, kp, t_prev
           error     = x_desired     - x[0]
           error_dot = x_dot_desired - x[1]
```

5

```
        dt = t - t_prev
        t_prev = t
        error_i = error_i + dt * error
        errors.append(error_i)
        u = kp*error + kd*error_dot + ki*error_i
        return np.array([x[1], (u - mu*x[1] - k*x[0] - 9.8)])
```

```
[240]: error_i = 0
       errors = []
       t_prev = 0
       x_desired = 5
       x_dot_desired = 0
       solution = odeint(Oscillator_Control, x0, time,)
       plt.plot(time, solution.T[0].T)
       plt.xlabel('time')
       plt.ylabel('x(t)')
       plt.show()
```



```
[241]: x_desired = 5
       x_dot_desired = 0
       kp = 150
       kd = 25
       ki = 130.0
       error_i = 0
```

6

```
errors = []
t_prev = 0
def Oscillator_Control_Step(x, t):
    global error_i, ki, kd, kp, t_prev
    x_desired = StepFunction(t)
    error     = x_desired     - x[0]
    error_dot = x_dot_desired - x[1]
    dt = t - t_prev
    t_prev = t
    error_i = error_i + dt * error
    errors.append(error_i)
    u = kp*error + kd*error_dot + ki*error_i
    return np.array([x[1], (u - mu*x[1] - k*x[0] - 9.8)])
```

```
[242]: error_i = 0
       errors = []
       t_prev = 0
       solution = odeint(Oscillator_Control_Step, x0, time,)
       plt.plot(time, solution.T[0].T)
       plt.xlabel('time')
       plt.ylabel('x(t)')
       plt.show()
```


7