Homework #3

Temurbek Khujaev
Control Theory. Group 2
INNOPOLIS UNIVERSITY

April 2020

Intro and overview

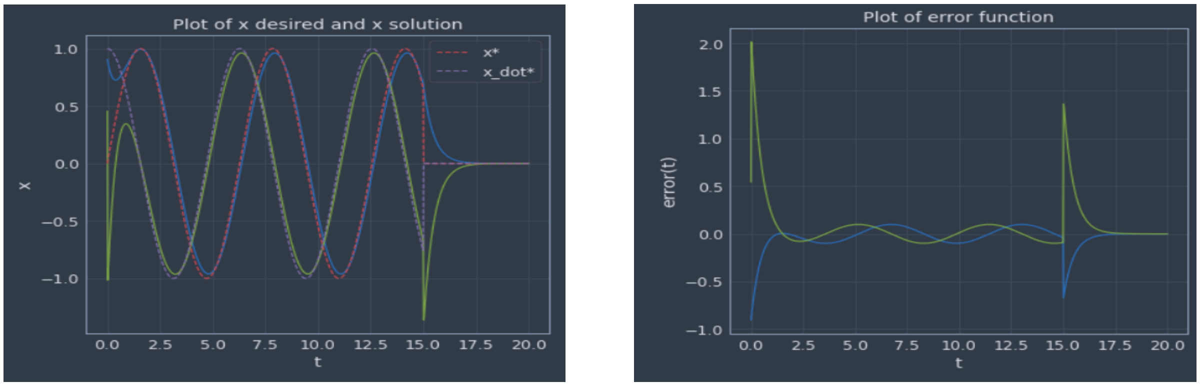
Temurbek Khujaev B18-02, t.xojayev@innopolis.university.
Generated variant is: **B**

Problem 2

(A) Design PD controller For given $\ddot{x} + 44\dot{x} + x = u$ I constructed 2 different types of trajectories and control you can see code below and results in the next pages

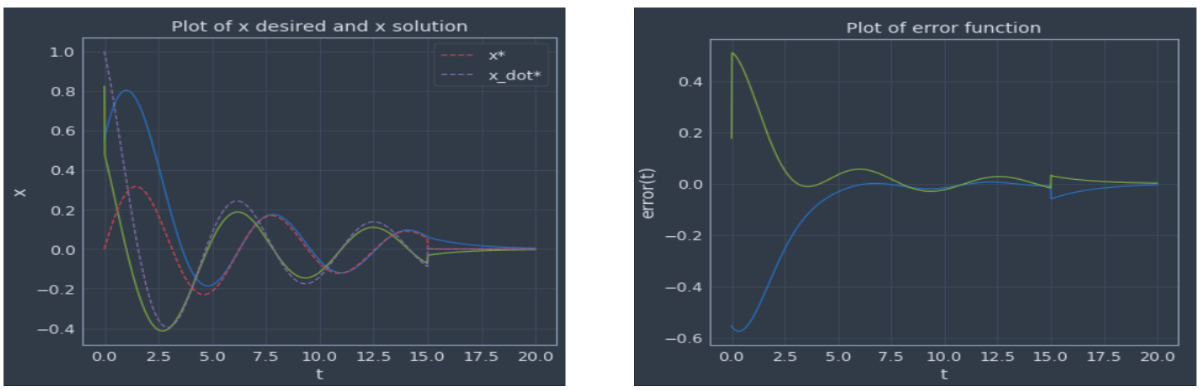
```
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt
from jupyterthemes import jtplot
from math import *
jtplot.style()
mu = 44;
k = 1;

def StepFunction(t):
    return 0 if t > 15 else 1
def x_desired_func(t):
    #return sin(t) *(t*t*3+5 + t*t+25) * StepFunction(t)
    return sin(t)*StepFunction(t)
def x_dot_desired_func(t):
    #return cos(t) *(t*t*t/10 -2*t*t  +3 *t + 10)*StepFunction(t)
    return cos(t)*StepFunction(t)

kp = 500
kd = 150

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

```
def Oscillator_Control(x, t):
    error     = x_desired_func(t)     - x[0]
    error_dot = x_dot_desired_func(t) - 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)
time = np.linspace(0, 20, 1000)
solution = odeint(Oscillator_Control, x0, time)
error = []
for t,x in zip(time,solution):
    error.append([ x_desired_func(t)-x[0],x_dot_desired_func(t) - x[1]])
x_desired = [x_desired_func(i) for i in time]
x_dot_desired = [x_dot_desired_func(i) for i in time]

plt.plot(time, solution)
plt.xlabel('t')
plt.ylabel('x')
plt.plot(time, x_desired, linestyle="dashed", label = "x*")
plt.plot(time, x_dot_desired, linestyle="dashed", label= "x_dot*")
plt.title('Plot of x desired and x solution')
plt.legend()
plt.show()
plt.plot(time, error)
plt.xlabel('t')
plt.ylabel('error(t)')
plt.title('Plot of error function')
plt.show()
```



(a) Plot of $x, \dot{x}, x*, \dot{x}*$ (b) Error function

Figure 1: Plots for FIRST trajectory function



(a) Plot of $x, \dot{x}, x*, \dot{x}*$ (b) Error function

Figure 2: Plots for SECOND trajectory function

(B) Tune K_d and K_p

K_p and K_d are tuned according to following algorithm:

- Set paramaters to 0
- Increase the P gain until the response to a disturbance is steady oscillation.
- Increase the D gain until the the oscillations go away (i.e. it's critically damped).
- Repeat steps 2 and 3 until increasing the D gain does not stop the oscillations.
- Set P and D to the last stable values.

You can see results from plots that they have decreasing overshoot

(C) Prove that they are stable

We have given $k = 1$ and $\mu = 44$ which gives:

$$\begin{bmatrix} \ddot{x} \\ \dot{x} \end{bmatrix} = \begin{bmatrix} -\mu & -k \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \dot{x} \\ x \end{bmatrix} + \begin{bmatrix} K_d & K_p \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{x}^* - \dot{x} \\ x^* - x \end{bmatrix} \tag{1}$$

After assigning values and moving $\dot{x}$ and x to A matrix

$$\begin{bmatrix} \ddot{x} \\ \dot{x} \end{bmatrix} = \begin{bmatrix} -44 - K_d & -1 - K_p \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \dot{x} \\ x \end{bmatrix} + \begin{bmatrix} K_d & K_p \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{x}^* \\ x^* \end{bmatrix} \tag{2}$$

$$det \begin{pmatrix} -44 - K_d - \lambda & -1 - K_p \\ 1 & -\lambda \end{pmatrix} = 0 \tag{3}$$

by simple rule of determinant for 2x2 matrix we will get:

$$\lambda^2 + \lambda(44 + K_d) + K_p + 1 = 0$$

and by Vieta's :

$$-(\lambda_1 + \lambda_2) = 44 + K_d$$

$$\lambda_1 \lambda_2 = K_p + 1$$

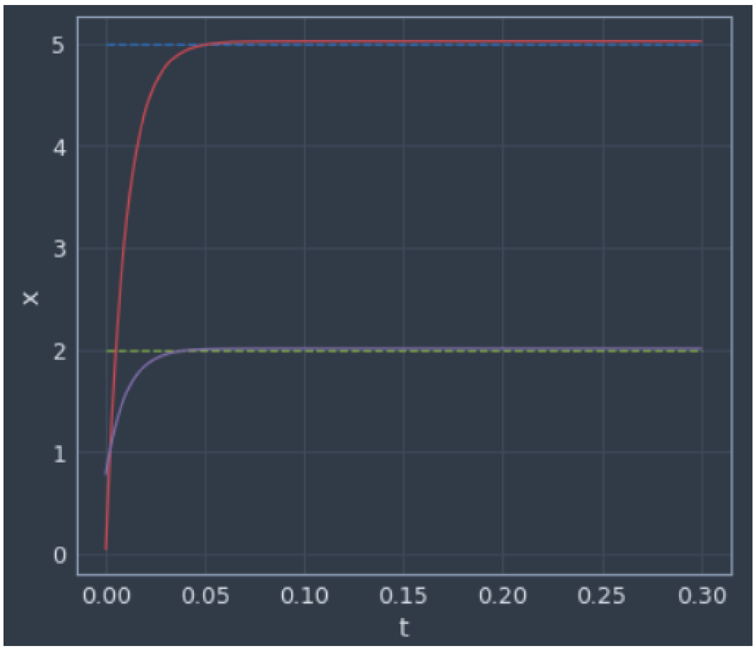
which gives: that we can choose any $K_d > -44$ and $K_p > -5$ that proves will be stable

(D) PD controller for linear system

It can be derived that system one of the eigen values is positive, so system is unstable and we need controller. Where, $u = K_p(x^* - x) + K_d(\dot{x}^* - \dot{x})$

```
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt
from jupyterthemes import jtplot
from math import *
jtplot.style()

A = np.array([[10,3],[5,-5]])
B = np.array([[1,0],[0,1]])
x_desired = np.array([5,2])
x_dot_desired = np.array([2,1])

kp = 100
kd = 1

def LTV(x,t):
    sys       = A.dot(x)
```

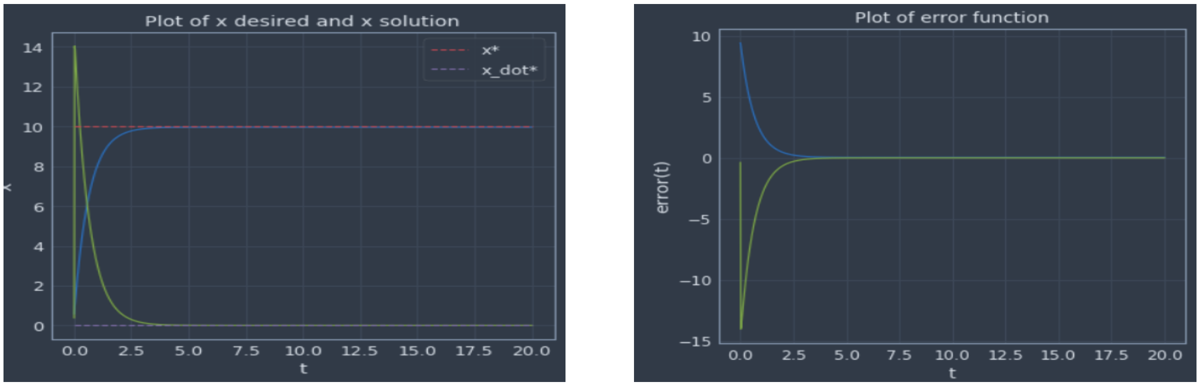
```
    error     = x_desired - x
    error_dot = x_dot_desired - sys
    u         = kp*error + kd*error_dot
    return sys + B.dot(u)

x0 = np.random.rand(2)
time = np.linspace(0,0.3,1000)

solution = odeint(LTV, x0, time)
plt.plot(time, [x_desired for i in time], linestyle="dashed")

plt.plot(time,solution)

plt.xlabel('t')
plt.ylabel('x')
plt.show()
```



(E) PI/PID controller for $\ddot{x} + 44\dot{x} + x + 9.8 = u$

```
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt
from jupyterthemes import jtplot
from math import *
jtplot.style()
mu = 44;
k = 1;


def StepFunction(t):
    return 0 if t > 15 else 1

def x_desired_func(t):
#     return sin(t)*exp(-t/10-1) * StepFunction(t)
#     return sin(t)*StepFunction(t)
    return 10
def x_dot_desired_func(t):
#     return cos(t) *(1/(t/2+1))*StepFunction(t)
#     return cos(t)*StepFunction(t)
    return 0

kp = 300
kd = 150
ki = 10

error_cumulative_i = 0

last_time = 0
def Oscillator_Control_PID(x, t):
    error     = x_desired_func(t)     - x[0]
    error_dot = x_dot_desired_func(t) - x[1]
    global error_cumulative_i, last_time
    current_time = t
    dt = current_time-last_time
    last_time = current_time
    error_cumulative_i += error*dt

    u = kp*error + kd*error_dot + ki*error_cumulative_i
    return np.array([x[1], (u - mu*x[1] - k*x[0] - 9.8)])

x0 = np.random.rand(2)
```

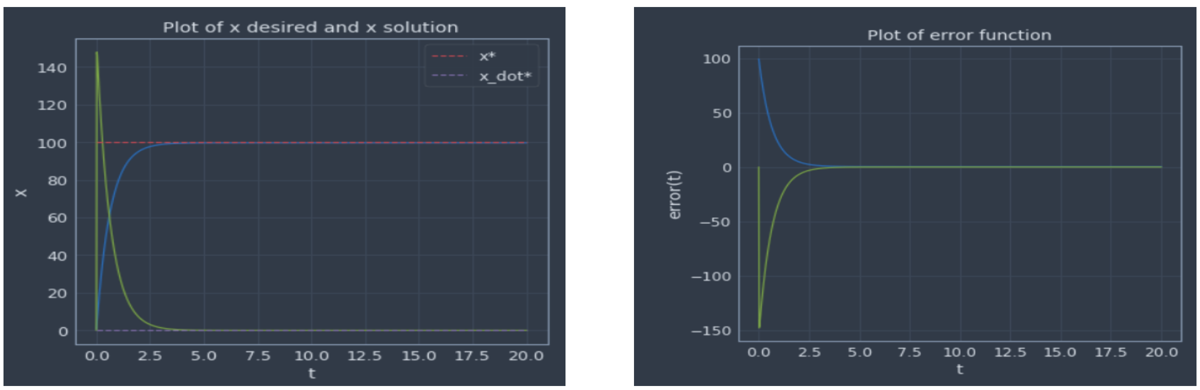
```
time = np.linspace(0, 20, 1000)

solution = odeint(Oscillator_Control, x0, time)
error = []
for t,x in zip(time,solution):
    error.append([ x_desired_func(t)-x[0],x_dot_desired_func(t) - x[1]])
x_desired = [x_desired_func(i) for i in time]
x_dot_desired = [x_dot_desired_func(i) for i in time]


plt.plot(time, solution)
plt.xlabel('t')
plt.ylabel('x')
plt.plot(time, x_desired, linestyle="dashed", label = "x*")
plt.plot(time, x_dot_desired, linestyle="dashed", label= "x_dot*")
plt.title('Plot of x desired and x solution')
plt.legend()
plt.show()

plt.plot(time, error)
plt.xlabel('t')
plt.ylabel('error(t)')
plt.title('Plot of error function')
plt.show()
```



(a) Plot of $x, \dot{x}, x*, \dot{x}*$

(b) Error function

Figure 3: Plots for SECOND trajectory function



(a) Plot of $x, \dot{x}, x*, \dot{x}*$

(b) Error function

Figure 4: Plots for SECOND trajectory function

Problem 3. Design a PID controller

had problems with matlab:(