ME C231A, EECS C220B Midterm pdf submission

Ziang Deng

TOTAL POINTS

43.5 / 60

QUESTION 1

1 Part A 8.5 / 14.5

✓ + 1 pts **A1.1 Ture**

✓ + 1 pts **A1.2 False**

✓ + 1 pts **A1.3 True**

+ 2 pts A2.1 Correct equilibrium point

+ 2 pts A2.2 Correct linearization

✓ + 3.5 pts **A3**

✓ + 2 pts **A4.1 - correct proportional controller**

+ 2 pts A4.2 - adding disturbance correctly

+ 1 pts A4 - partial credits

+ 0 pts Missing

QUESTION 2

2 Part B 8 / 11.5

✓ + 0.5 pts **B1.1**

✓ + 0.5 pts **B1.2**

✓ + 0.5 pts **B1.3**

✓ + 0.5 pts **B1.4**

+ 0.5 pts B1.5

✓ + 0.5 pts **B1.6**

+ 6 pts B2

✓ + 3 pts **B2 - partial credits**

✓ + 0.5 pts **B3 - Solver C**

✓ + 0.5 pts **B3 - Solver A**

✓ + 0.5 pts **B3 - Solver B**

✓ + 0.5 pts **B3 - Solver D**

✓ + 0.5 pts **B3 - Solver E**

+ 0 pts Missing

QUESTION 3

3 Part C 16 / 20

✓ + 2 pts **C1 - correct $$u_0^*, u_{18}^*, u_{49}^*, x_{19}^*, x_{49}^*$$ printed**

✓ + 6 pts **C1 - correct simulations (plots)**

+ 3 pts C1 - partial credits

✓ + 2 pts **C2 - disturbance is added correctly**

✓ + 6 pts **C2 - correct simulations**

+ 2 pts C2 - partial credits

+ 2 pts C3.1

+ 2 pts C3.2

+ 0 pts missing / incorrect

QUESTION 4

4 Part D 11 / 14

✓ + 5 pts **D1.1**

+ 3 pts D1.1 partial credits (minor mistake)

+ 6 pts D1.2

+ 4 pts D1.2 partial credits (numerical values are written instead of Inf)

✓ + 3 pts **D1.2 partial credits (numerical/minor mistakes)**

✓ + 3 pts **D2**

+ 0 pts missing / incorrect

1. 2.64

2. 1.64

ME C231A, EE C220B, Experiential Advanced Control I

Midterm Fall 2020

University of California Berkeley

Remember

1. The submission will be EXACTLY the same as your homework. Download the midterm from bCourses and upload with Gradescope.
2. Submit your file BEFORE 12.30PM. The online submission will close at 12.30PM and NO LATE submission will be accepted.
3. Multiple submissions (before 12.30) will be allowed.
4. **Your submission of your solutions will act as a confirmation that you have not discussed this exam with anyone during the exam period. We have different tools to identify cheating, and we will not tolerate any exception.**

A quick summary of the questions. "Minimal coding" means 1 or 2 lines of code.

- The questions are in blue, do not modify the cells containing the questions
- A1. No coding. Estimated time 2min
- A2 part 1 and 2. No coding, little pencil/paper work. Estimated time 4min
- A3. Minimal coding. Estimated time 1min
- A4 part 1 and 2. Minimal coding. Estimated time 2min
- B1. No coding. Estimated time 2min
- B2. Some coding (start from software in lab/homework and work will be minimal). Estimated time 4min
- B3. No coding. Estimated time 2min
- C1. Some coding (start from software in lab/homework and work will be minimal). Estimated time 10min
- C2. Minimal coding once C1 is available. Estimated time 4min
- C3. No coding. Estimated time 2min
- D1. No coding, some pencil/paper work. Estimated time 10min
- D2. No coding. Estimated time 3min

Total: expected 60 minutes (points assignment for each question approximately: 1point=1 expected minute)

In [17]:
```
# # Run this cell only if you are using Colab
# # install required dependencies
# import sys
# IN_COLAB = 'google.colab' in sys.modules
# if IN_COLAB:
#   !pip install -q pyomo
#   !apt-get install -y -qq glpk-utils
#   !apt-get install -y -qq coinor-cbc
#   !wget -N -q "https://ampl.com/dl/open/ipopt/ipopt-linux64.zip"
#   !unzip -o -q ipopt-linux64
```

Part A: Modeling

Question A1. System properties

Consider the system

$$x(t+1) = sin(t)x(t) + cos(t)u(t)$$

Answer the following questions in the Text Cell below with True or False

1. The system is linear
2. The system is time-invariant
3. The pair (0,0) is an equilibrium point of the system

ANSWER A1:

1. True
2. False
3. True

Question A2. Equilibrium Point and Linearization

Consider the discrete time nonlinear system

$$(A2) \qquad x(t+1) = x^2(t) + u_1(t)u_2(t)$$

Find an equilibrim point with $x_{eq} = 2$ and report the corresponding for $u_{1,eq}$ and $u_{2,eq}$ below

ANSWER A2 part 1:

An equilibrium point is:

- $x_{eq} = 2$
- $u_{1,eq} = -2$
- $u_{2,eq} = 2$

Report the A and B matrices of the linear system obtained by linearizing the system (A2) around the previous equilibrium point. You can use the notation [[1,1,3],[1,2,1]] to denote a matrix 2x3 with first row [1,1,3]

ANSWER A2 part 2:

1. A= [4]
2. B= [[-2,0],[0,1]]

Question A3. Euler Discretization of an ODE Model

Consider the ODE

$$\dot{T} = T(t)u(t)$$

Use the Euler discretization approximation to derive the equivalent discrete time model with sampling time $T_s = 0.1s$:

$$T_{k+1} = f(T_k, u_k)$$

Fill in the line 3 of the following code cell and run the cell to print the output

In [18]:
```
# ANSWER A3
def f(Tk, uk):
    Ts = 0.1
    T_KplusOne = Tk + Ts* (Tk * uk)
    return T_KplusOne

T_KplusOne = f(Tk=60, uk=1)
print(T_KplusOne)
```

```
66.0
```

Problem A4. Simulation of a Nonlinear Model

The following code is a simulator for the system

$$x_{k+1} = sin(x_k) + u_k$$

starting from initial condition $x0 = 0$ and applying no input. Look at the code, run it and go to the next Text cells which contain the questions.

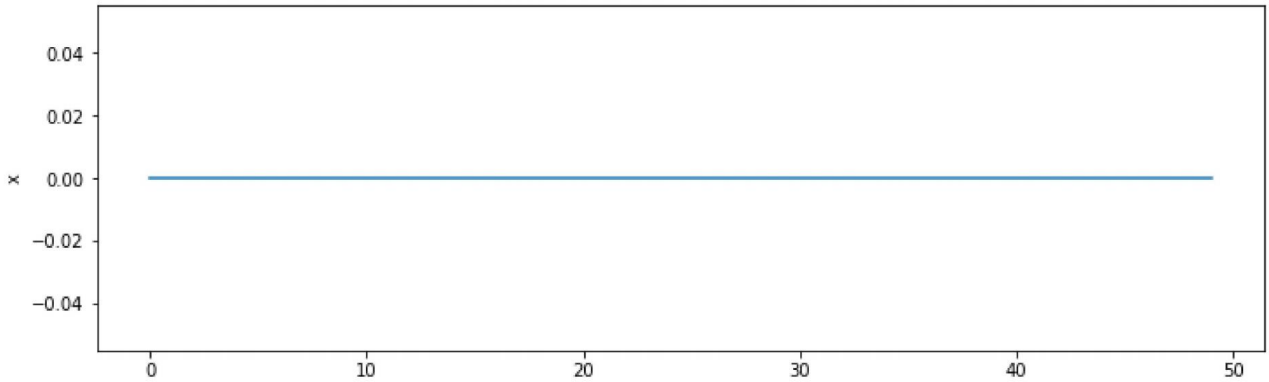
--

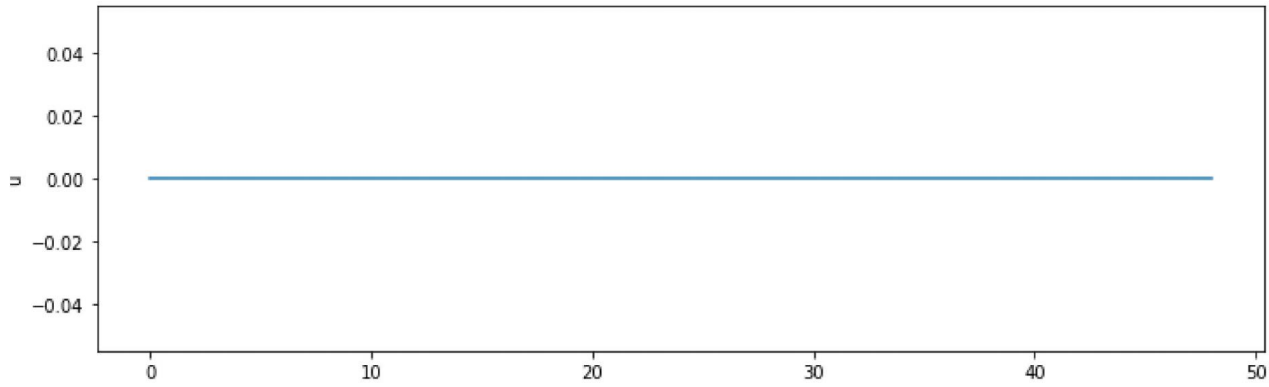
In [19]:

```
import matplotlib.pyplot as plt
import numpy as np
from numpy import sin, cos, tan, arcsin, exp,pi
def f(xk, uk, k):
    xkplusone= sin(xk) +uk
    return xkplusone

def sim(numSteps, x0):

    # Initialize trends
    xtrend = []
    utrend = []
    xtrend.append(x0)
    #set initail state
    xk = x0

    for k in range(0, numSteps-1):
        uk=0
        xkplusone = f(xk, uk,k)
        xtrend.append(xkplusone)
        utrend.append(uk)
        xk=xkplusone

    return np.asarray(xtrend, dtype=object), np.asarray(utrend, dtype=object)

# starting the simualtion
x0 = 0.0
numSteps = 50
xtrend, utrend = sim(numSteps, x0)


# plotting the results
fig=plt.figure(figsize=(12,8))
plt.subplot(2,1,1)
plt.plot(xtrend)
plt.ylabel('x')
plt.subplot(2,1,2)
plt.plot(utrend)
plt.ylabel('u')
plt.show()
```

1 Part A 8.5 / 14.5

- ✓ + 1 pts **A1.1 Ture**
- ✓ + 1 pts **A1.2 False**
- ✓ + 1 pts **A1.3 True**
- + 2 pts A2.1 Correct equilibrium point
- + 2 pts A2.2 Correct linearization
- ✓ + 3.5 pts **A3**
- ✓ + 2 pts **A4.1 - correct proportional controller**
- + 2 pts A4.2 - adding disturbance correctly
- + 1 pts A4 - partial credits
- + 0 pts Missing





Question A4 Part 1.

We copied the code provided above in the following cell. Design and tune a simple Proportional controller to bring the system state x_k as close as possible to $x_{ref} = 1$ without too many oscillations. (a rough tuning is ok... start from a controller gain of k=-0.5). Modify the code in the next cell to plot the simulations of the closed-loop system.

In [20]:
```
# ANSWER A4.1 (Please modify this code to answer Question A4 part 1)
import matplotlib.pyplot as plt
import numpy as np
from numpy import sin, cos, tan, arcsin, exp,pi
def f(xk, uk, k):
    xkplusone= sin(xk) +uk
    return xkplusone

def sim(numSteps, x0):

    # Initialize trends
    xtrend = []
    utrend = []
    xtrend.append(x0)
    #set initail state
    xk = x0

    for k in range(0, numSteps-1):
        uk= -1.5(xk -1)
        xkplusone = f(xk, uk,k)
        xtrend.append(xkplusone)
        utrend.append(uk)
        xk=xkplusone

    return np.asarray(xtrend, dtype=object), np.asarray(utrend, dtype=object)

# starting the simualtion
x0 = 0.0
numSteps = 50
xtrend, utrend = sim(numSteps, x0)


# plotting the results
fig=plt.figure(figsize=(12,8))
plt.subplot(2,1,1)
plt.plot(xtrend)
plt.ylabel('x')
plt.subplot(2,1,2)
plt.plot(utrend)
plt.ylabel('u')
plt.show()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-20-c46fe9c36531> in <module>()
     28 x0 = 0.0
     29 numSteps = 50
---> 30 xtrend, utrend = sim(numSteps, x0)
     31
     32

<ipython-input-20-c46fe9c36531> in sim(numSteps, x0)
     17
     18     for k in range(0, numSteps-1):
---> 19         uk= -1.5(xk -1)
     20         xkplusone = f(xk, uk,k)
     21         xtrend.append(xkplusone)

TypeError: 'float' object is not callable
```

Question A4 Part2.

Use the same controller and tuning as before and assume that the system is subject to an additive step disturbance.

$$x_{k+1} = sin(x_k) + u_k + d_k$$

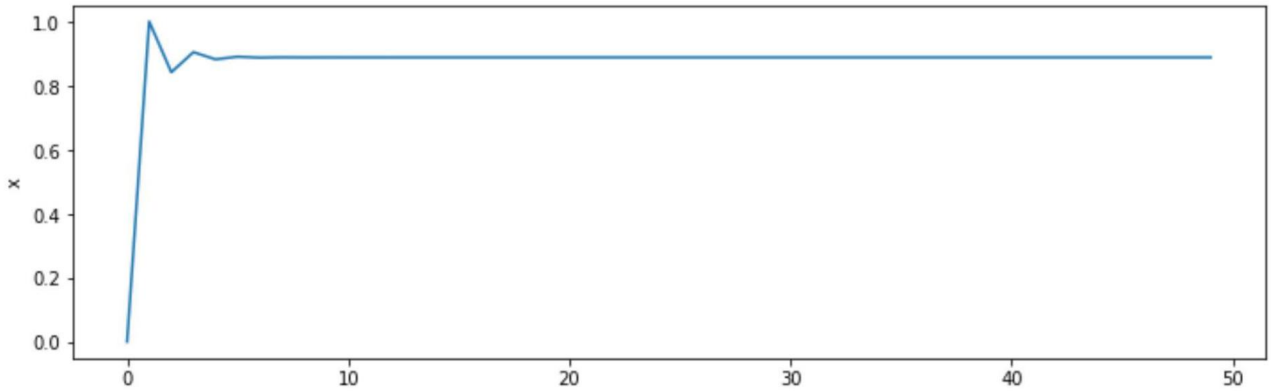
where

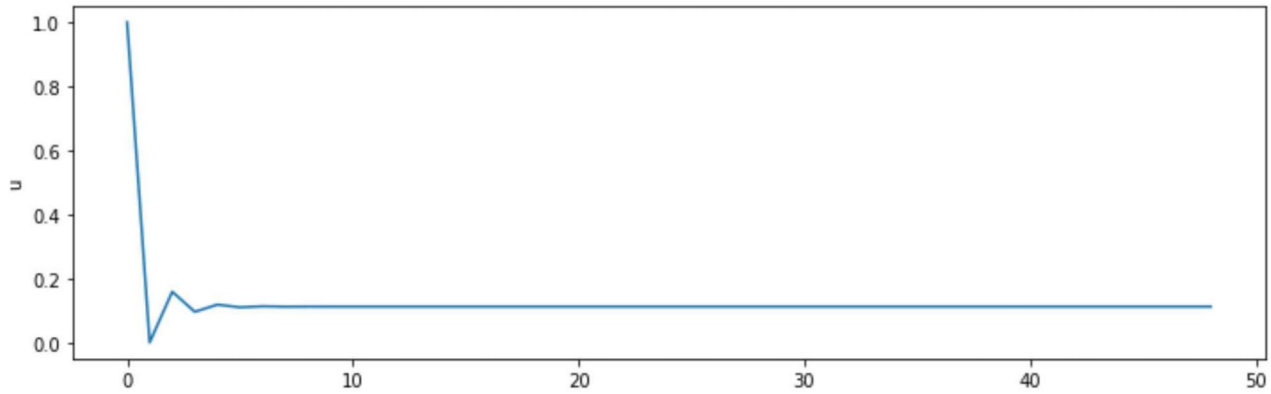
$$d_k = 0 \text{ if } k < 20 \quad \text{and} \quad d_k = 1 \text{ if } k >= 20$$

Copy the code of question A4 part 1 in the cell below and modify it to simulate the step disturbance effect. Report the plots of the closed-loop simulations.

In [16]:

```
# ANSWER A4.2 Part 2

def f(xk, uk, k):
    xkplusone= sin(xk) +uk
    return xkplusone

def sim(numSteps, x0):

    # Initialize trends
    xtrend = []
    utrend = []
    xtrend.append(x0)
    #set initail state
    xk = x0

    for k in range(0, numSteps-1):
        uk= -1 *(xk -1)
        xkplusone = f(xk, uk,k)
        xtrend.append(xkplusone)
        utrend.append(uk)
        xk=xkplusone

    return np.asarray(xtrend, dtype=object), np.asarray(utrend, dtype=object)

# starting the simualtion
x0 = 0.0
numSteps = 50
xtrend, utrend = sim(numSteps, x0)


# plotting the results
fig=plt.figure(figsize=(12,8))
plt.subplot(2,1,1)
plt.plot(xtrend)
plt.ylabel('x')
plt.subplot(2,1,2)
plt.plot(utrend)
plt.ylabel('u')
plt.show()
```





Part B- Optimization

Question B1. Consider the following optimization problem:

$$\begin{aligned} \min_{x\in\mathbb{R}^n, \epsilon\in\mathbb{R}} \quad & x^T H x + f^T x + g \\ \text{s.t.} \quad & Ax \leq b - \epsilon^2 \mathbb{1}_m \end{aligned}$$

where $H \in \mathbb{R}^{n\times n}, f \in \mathbb{R}^n, g \in \mathbb{R}, A \in \mathbb{R}^{m\times n}, b \in \mathbb{R}^m, \epsilon \in \mathbb{R}$, and $\mathbb{1}_m$ is a column vector of length m containing ones.

1. Its convexity depends only on H
2. The problem is convex for $H > 0, f = \text{rows of zeros}, g = 0.$
3. Its convexity depends on A and b
4. Its feasibility depends on H and f
5. Its feasibility depends on A and b
6. It is a quadratic program

ANSWER B1:

1. True
2. True
3. False
4. False

2 Part B 8 / 11.5

- ✓ + 0.5 pts **B1.1**
- ✓ + 0.5 pts **B1.2**
- ✓ + 0.5 pts **B1.3**
- ✓ + 0.5 pts **B1.4**
  + 0.5 pts B1.5
- ✓ + 0.5 pts **B1.6**
  + 6 pts B2
- ✓ + 3 pts **B2 - partial credits**
- ✓ + 0.5 pts **B3 - Solver C**
- ✓ + 0.5 pts **B3 - Solver A**
- ✓ + 0.5 pts **B3 - Solver B**
- ✓ + 0.5 pts **B3 - Solver D**
- ✓ + 0.5 pts **B3 - Solver E**
  + 0 pts Missing

5. False
6. True

Question B2. Consider the following Linear Program:

$$p^* = \underset{x \in \mathbb{R}^3}{\text{minimize}} \quad \|Ax - b\|_\infty$$
$$\text{subject to} \quad Cx \leq d$$

where $A = \begin{bmatrix} 2 & 0 & -1 \\ 1 & -1 & 0 \end{bmatrix}$, $b = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$, $C = \begin{bmatrix} 1 & -1 & 1 \\ -1 & -1 & 0 \end{bmatrix}$, $d = \begin{bmatrix} -1 \\ -1 \end{bmatrix}$.

1. Transform it into a standard linear program (you can do this on a piece of paper, nothing to submit)
2. Solve the LP in the Code Cell below and print the optimal cost p^*, the optimizer x^*

In [13]:

```
#ANSWER B2

from pyomo.opt import SolverStatus, TerminationCondition
import numpy as np
import pyomo.environ as pyo
import numpy as np
import cvxopt


def reg1Inf(A1,b1,Ainf,binf,Ac,bc):
    # the question transformed to:
    # min(z1,z2,,zn,t1,t2,,t_nc,t_inf){t1,t2,,t_nc,t_inf}
    # t1,t2,,t_nc = |A1 *z -b1|
    # t_inf = max(|A_inf *z - b_inf|)

    n_1, nx = np.shape(A1)
    n_inf, _ = np.shape(Ainf)
    nc, _ = np.shape(Ac)

    c = np.concatenate(
        [
            # np.zeros(np.size(Ac,1)),np.ones(np.size(A1,0)),np.array((1,))
            np.zeros(nx),np.ones(n_1),np.array((1,))
        ],axis= 0
    )
    A = np.concatenate(
        [
            # np.concatenate([A1,-np.eye(np.size(A1,0)),np.zeros((np.size(A1,0),1))],axis=1),
            # np.concatenate([-A1,-np.eye(np.size(A1,0)),np.zeros((np.size(A1,0),1))],axis=1),
            # np.concatenate([Ainf,np.zeros((np.size(Ainf,1),np.size(A1,1))),np.ones((np.size(Ainf
            # np.concatenate([-Ainf,np.zeros((np.size(Ainf,1),np.size(A1,1))),-np.ones((np.size(Ai
            # np.concatenate([Ac,np.zeros((np.size(Ac,0),np.size(A1,0))),np.zeros((np.size(Ac,0),1
            np.concatenate([A1,-np.eye(n_1),np.zeros((n_1,1))],axis=1),
            np.concatenate([-A1,-np.eye(n_1),np.zeros((n_1,1))],axis=1),
            np.concatenate([Ainf,np.zeros((n_inf,nx)),np.ones((n_inf,1))],axis= 1),
            np.concatenate([-Ainf,np.zeros((n_inf,nx)),-np.ones((n_inf,1))],axis= 1),
            np.concatenate([Ac,np.zeros((nc,n_1)),np.zeros((nc,1))],axis=1)

        ],axis = 0
    )
    b = np.concatenate([b1,-b1,binf,-binf,bc],axis=0)

    c = cvxopt.matrix(c,tc='d')
    A = cvxopt.matrix(A,tc='d')
    b = cvxopt.matrix(b,tc='d')


    sol = cvxopt.solvers.lp(c,A,b)
    xOpt = sol['x']
    J = sol['primal objective']
    return xOpt[:3],J

a1 = np.zeros((3,3))
b1 = np.array([0,0,0]).reshape((3,1))
ainf = np.array([[2,0,-1],[1,-1,0]])
binf = np.array([1,2]).reshape((2,1))
ac = np.array([[1,-1,1],[-1,-1,0]])
bc = np.array([-1,-1]).reshape((2,1))
xOpt, J = reg1Inf(a1,b1,ainf,binf,ac,bc)
print('xOpt: ',xOpt)
print('J* = ',J)
```

```
     pcost       dcost       gap    pres   dres   k/t
 0:  5.0000e+00  9.0000e+00  3e+01  1e+00  4e+00  1e+00
 1:  4.7378e+00  5.3376e+00  7e-01  8e-02  3e-01  3e-01
 2:  4.9978e+00  5.0041e+00  7e-03  1e-03  4e-03  3e-03
 3:  5.0000e+00  5.0000e+00  7e-05  1e-05  4e-05  3e-05
 4:  5.0000e+00  5.0000e+00  7e-07  1e-07  4e-07  3e-07
 5:  5.0000e+00  5.0000e+00  7e-09  1e-09  4e-09  3e-09
Optimal solution found.
xOpt:  [-1.00e+00]
[ 2.00e+00]
[ 2.00e+00]

J* =  4.999999997783805
```

Question B3

You have 5 solvers which you use to solve the following problem:

$$\begin{aligned} \min_{z1,z2} \quad & -z_1 - 2z_2 \\ \text{s.t. } & z_1 + z_2 \leq 10 \\ & z_1 \geq 0 \\ & z_2 \geq 0 \end{aligned}$$

Their respective primal/dual optimal solutions are

- Solver A: $z_1^* = -10,\ z_2^* = 10$, and $u_1^* = 1,\ u_2^* = 1, u_3^* = 1,$
- Solver B: $z_1^* = 0,\ z_2^* = 10$, and $u_1^* = 2,\ u_2^* = 1, u_3^* = 1,$
- Solver C: $z_1^* = 0,\ z_2^* = 10$, and $u_1^* = 2,\ u_2^* = 1, u_3^* = 0,$
- Solver D: $z_1^* = 1,\ z_2^* = 10$, and $u_1^* = -2,\ u_2^* = 1, u_3^* = 1,$
- Solver E: $z_1^* = 0.5,\ z_2^* = 0.5$, and $u_1^* = 2,\ u_2^* = 1, u_3^* = 0,$

(where u_1^*, u_2^* and u_3^* are the Lagrange multipliers associated to the three inequality constraints in the order written above.) Only one solver is correct. The others have some bugs. Without using any coding and simply thinking of KKT conditions and LP properties can you quickly identify which solver is the correct one?

ANSWER B3:

Complete the dots "..." for each solver

- The correct sover is C
- Solver D is wrong because u1* < 0
- Solver A is wrong because Z1* <0
- Solver E is wrong because u1* * g(z1,z2) is not 0
- Solver B is wrong because u3* * g(z1,z2) is not 0

Part C- Optimal Control - Batch Approach

Question C.1 Consider the following finite-time optimal control problem

$$\min_{x_0,\dots,x_N,u_0,\dots,u_{N-1}} \sum_{k=0}^{k=N} \|x_k - \bar{x}_N\|_2^2$$

$$\begin{aligned}
x_{k+1} &= \sin(x_k) + u_k && \forall k = \{0,\dots,N-1\} \\
\text{-}0.2 &\leq u_k \leq 0.2 && \forall k = \{0,\dots,N-1\} \\
|x_N - \bar{x}_N| &\leq 0.1 \\
x_0 &= x(0)
\end{aligned}$$

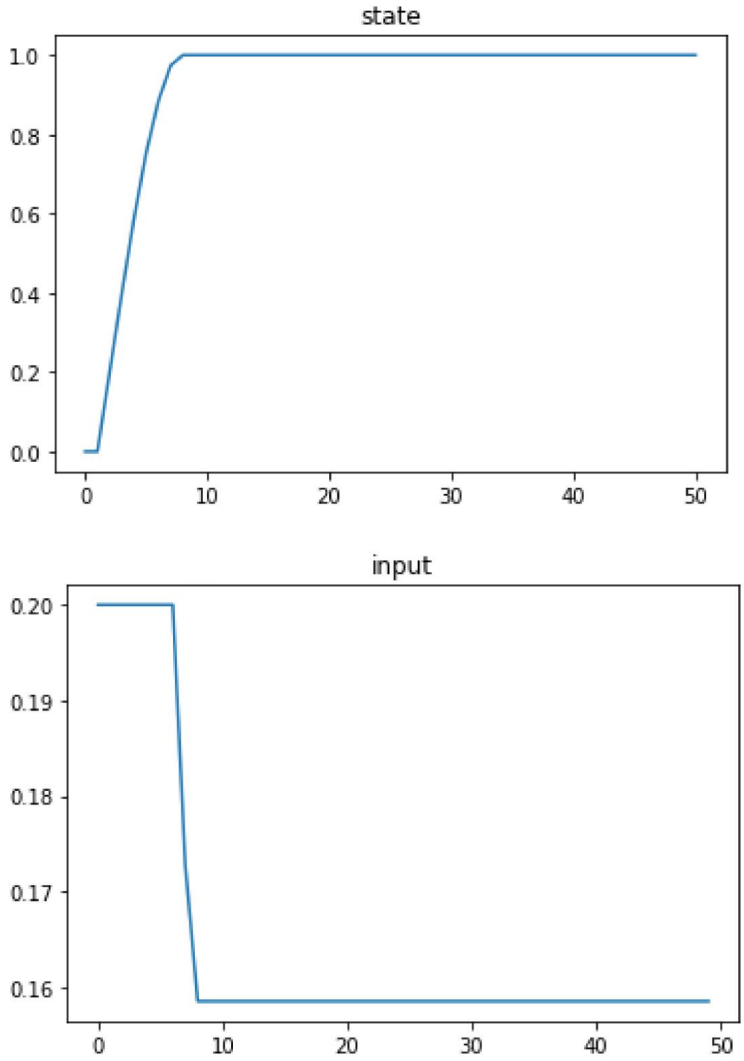
where $N = 50$ and $\bar{x}_N = 1$. Use the Batch approach to compute the *open-loop optimal* solution $U_0^*(x(0))$ where $U_0^* = [u_0^*, \dots, u_{49}^*]$ for the initial conditions $x(0) = 0.0$.

1. Type your code in the next Cell.
2. Print u_0^*, u_{18}^*, u_{49}^*, x_{19}^*, x_{49}^*
3. Plot the *optimal* state and input trajectory as a function of time

In [28]:

```
# ANSWER C.1

import matplotlib.pyplot as plt
import numpy as np
import pyomo.environ as pyo

N = 50
xNbar = 1
x0 =0
m = pyo.ConcreteModel()
m.tidx = pyo.Set(initialize = range(N+1))
m.u = pyo.Var(m.tidx)
m.x = pyo.Var(m.tidx)

m.cost = pyo.Objective(
    expr = sum((m.x[t]-xNbar) **2 for t in m.tidx),
    sense= pyo.minimize

)

m.c1 = pyo.Constraint(
    m.tidx, rule = lambda m,t:
    m.x[t+1] == pyo.sin(m.x[t]) + m.u[t]
    if t < N else pyo.Constraint.Skip
)
m.c21 = pyo.Constraint(
    m.tidx, rule = lambda m,t:
    m.u[t] <= 0.2
    if t<N else pyo.Constraint.Skip
)
m.c22 = pyo.Constraint(
    m.tidx, rule = lambda m,t:
    m.u[t] >= -0.2
    if t<N else pyo.Constraint.Skip
)
m.c31 = pyo.Constraint(
    expr = m.x[N] -xNbar >= -0.1
)
m.c32 = pyo.Constraint(
    expr = m.x[N] -xNbar <= 0.1
)
m.c4 = pyo.Constraint(expr = m.x[0] == x0)

results = pyo.SolverFactory('ipopt').solve(m)

x= [m.x[0]()]
u= [m.u[0]()]
for t in m.tidx:
    if t< N:
        x.append(m.x[t]())

    if t< N-1:
        u.append(m.u[t]())

print(u[0],u[18],u[49],x[19],x[49])
plt.plot(x)
plt.title('state')
plt.show()
plt.plot(u)
plt.title('input')
plt.show()
```

0.20000000977468013 0.1585290131438107 0.15852901314382012 0.9999999955442613 0.999999995544260
4



Question C.2. Solve the same problem of Question C.1 with just one change.

Assume that the system is subject to an additive disturbance which is completely known during the control design:

$$x_{k+1} = sin(x_k) + u_k + d_k$$

where

$$d_k = 0 \text{ if } k < 20 \quad \text{and} \quad d_k = 0.3 \text{ if } k >= 20$$

Here "the disturbance is completely known during the control design" means: add it in your model of the optimization problem when you are building the Pyomo model. Use the Batch approach to compute the optimal solution $U_0^*(x(0))$ where $U_0^* = [u_0^*, \dots, u_{49}^*]$ for the initial conditions $x(0) = 0.0$

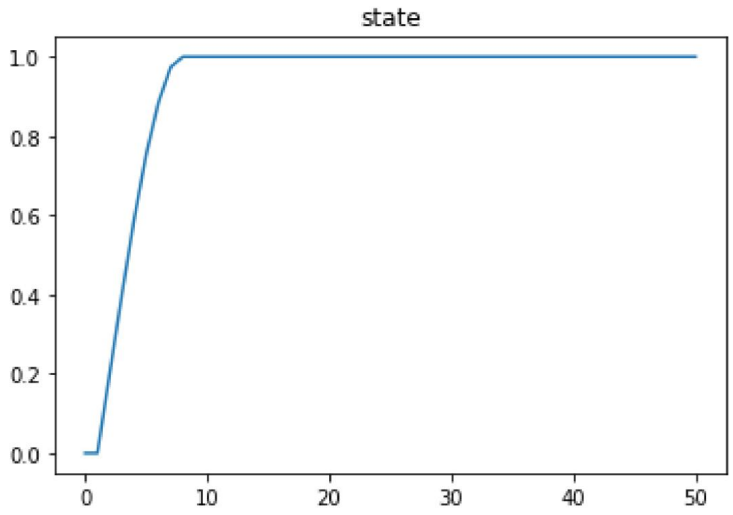
1. Type your code in the next Cell.
2. Print u_0^*, u_{18}^*, u_{49}^*, x_{19}^*, x_{49}^*
3. Plot the optimal state and input trajectory as a function of time

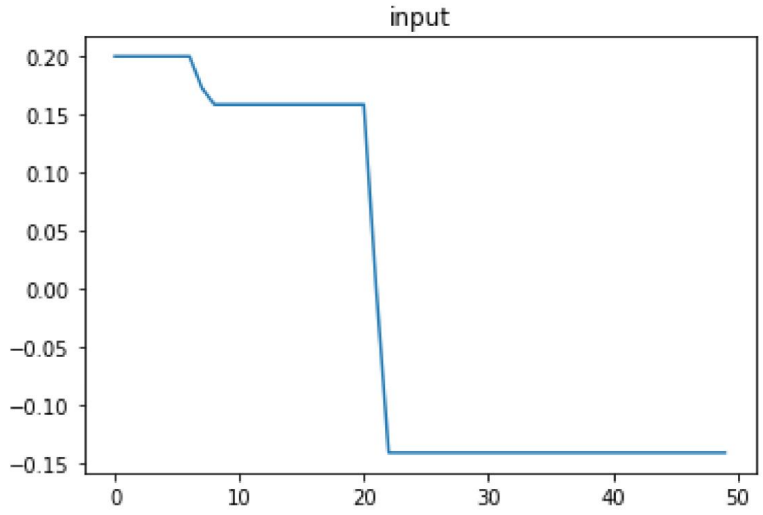
In [29]:

```
#  ANSWER C.2

import matplotlib.pyplot as plt
import numpy as np
import pyomo.environ as pyo

N = 50
xNbar = 1
x0 =0
m = pyo.ConcreteModel()
m.tidx = pyo.Set(initialize = range(N+1))
m.u = pyo.Var(m.tidx)
m.x = pyo.Var(m.tidx)
m.d = pyo.Var(m.tidx)

m.cost = pyo.Objective(
    expr = sum((m.x[t]-xNbar) **2 for t in m.tidx),
    sense= pyo.minimize

)

m.c1 = pyo.Constraint(
    m.tidx, rule = lambda m,t:
    m.x[t+1] == pyo.sin(m.x[t]) + m.u[t] + m.d[t]
    if t < N else pyo.Constraint.Skip
)
m.c21 = pyo.Constraint(
    m.tidx, rule = lambda m,t:
    m.u[t] <= 0.2
    if t<N else pyo.Constraint.Skip
)
m.c22 = pyo.Constraint(
    m.tidx, rule = lambda m,t:
    m.u[t] >= -0.2
    if t<N else pyo.Constraint.Skip
)
m.c31 = pyo.Constraint(
    expr = m.x[N] -xNbar >= -0.1
)
m.c32 = pyo.Constraint(
    expr = m.x[N] -xNbar <= 0.1
)
m.c4 = pyo.Constraint(expr = m.x[0] == x0)
m.c5 = pyo.Constraint(m.tidx, rule = lambda m,t: m.d[t] ==0 if t < 20 else pyo.Constraint.Skip)
m.c6 = pyo.Constraint(m.tidx, rule = lambda m,t: m.d[t] ==0.3 if t > 20 else pyo.Constraint.Skip

results = pyo.SolverFactory('ipopt').solve(m)

x= [m.x[0]()]
u= [m.u[0]()]
for t in m.tidx:
    if t< N:
        x.append(m.x[t]())

    if t< N-1:
        u.append(m.u[t]())

print(u[0],u[18],u[49],x[19],x[49])
plt.plot(x)
plt.title('state')
plt.show()
plt.plot(u)
plt.title('input')
```

```
plt.show()
```

```
0.2000000093789072 0.15852900954599444 -0.14147098105945097 0.9999999877177835 1.000000008154145
```





Question C.3 Compare the results of Question C1 and C2 and answer the following questions (just text , no Code).

1. Why the two u^*_{19} are different, even if the disturbance happens at step k=20?

2. If you increase the disturbance amplitude d_k from 0.3 to 1 the problem becomes infeasible.What is the reason of this infeasibility?

ANSWER C.3

1. ...
2. ...

3 Part C 16 / 20

✓ **+ 2 pts** **C1 - correct $$u_0^*, u_{18}^*, u_{49}^*, x_{19}^*, x_{49}^*$$ printed**

✓ **+ 6 pts** **C1 - correct simulations (plots)**

+ 3 pts C1 - partial credits

✓ **+ 2 pts** **C2 - disturbance is added correctly**

✓ **+ 6 pts** **C2 - correct simulations**

+ 2 pts C2 - partial credits

+ 2 pts C3.1

+ 2 pts C3.2

+ 0 pts missing / incorrect

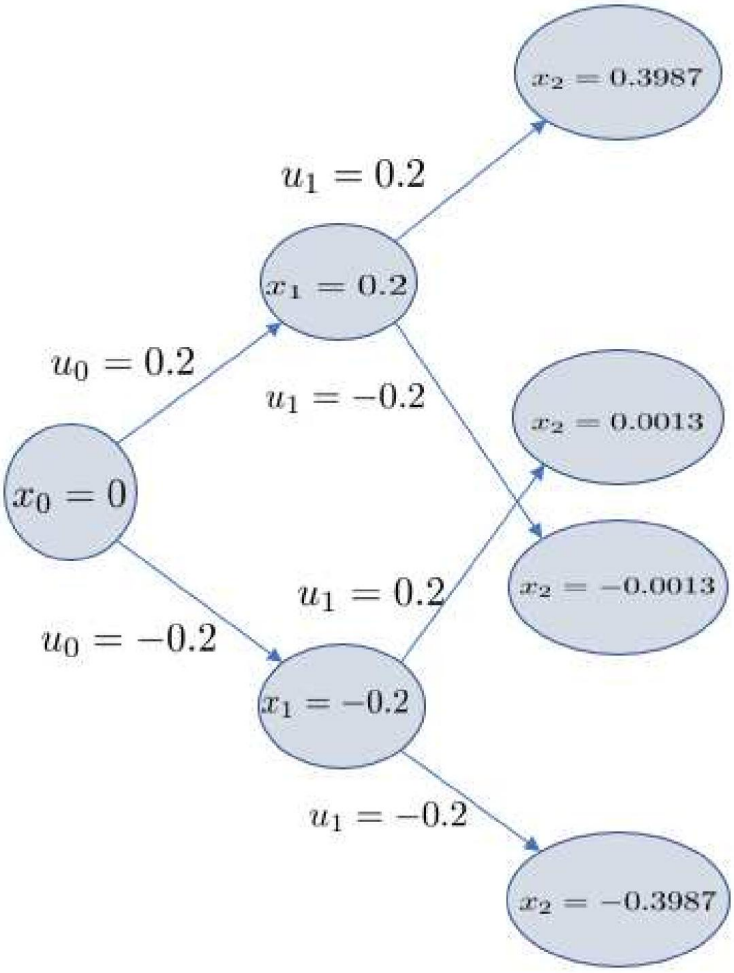
Part D- Optimal Control - DP Approach

Question D.1 Consider the following finite-time optimal control problem

$$\min_{x_0,\dots,x_N,u_0,\dots,u_{N-1}} \sum_{k=0}^{k=N} \|x_k - \bar{x}_N\|_2^2$$

$$\begin{aligned}
x_{k+1} &= \sin(x_k) + u_k && \forall k = \{0,\dots,N-1\} \\
\text{-}0.2 &\le u_k \le 0.2 && \forall k = \{0,\dots,N-1\} \\
|x_N| &\le 0.1 \\
\bar{x}_N &= 1 \\
x_0 &= 0
\end{aligned}$$

where $N = 2$. Use the Dynamic Programming approach to compute the closed-loop optimal solution of the problem only for the gridded states and gridded inputs shown in the next figure



In [ ]:
```
answer: -0.2; 2.44; -0.2 0.2; 2.443 2.437;  NaN, 0.9974, 1.0026,  NaN
```

Question D.2 The approach to solve the CFTOC problem in question D1 relies on state and input discretization. Assume you solved the same problem in question D1 with a batch approach. If you compared the cost $J^*_{0\to 2}(x_0)$ of the Batch solution with the same cost of the DP solution you would obtain

1. The Batch Approach cost is lower since the DP approach uses discretization.
2. The DP Approach cost might be still the same, since we were lucky and discretized along the optimal state and input trajectory.
3. You cannot make any statements since the Batch solver (IPOPT) might get stuck in a local optima and the DP approach uses a discretization and thus is suboptimal. Anything can happen: $J^*_{0\to 2}(x_0)$ Batch <= $J^*_{0\to 2}(x_0)$ DP or $J^*_{0\to 2}(x_0)$ Batch > $J^*_{0\to 2}(x_0)$ DP.

```
Only one is True, which one?

ANSWER D2: 1,2, or 3?

3
```

In [ ]:
```
# pdf conversion steps:

# If you are using docker, to make a pdf from your .ipynb file follow these steps:

# Inside your docker container open a New Launcher by tapping + icon at the top left and then a Te
# cd to the directory that your Midterm_2020.ipynb file is located.
# Run jupyter nbconvert --to html Midterm_2020.ipynb to make html version. (converting directly to
# Navigate to your mounted folder on your local machine and open the html file (If you open html f
# Now you can print (Ctrl+p) and save it as a pdf file.

# If you are using Colab, you just need to print and save as a pdf file.
```

4 Part D 11 / 14

✓ **+ 5 pts** **D1.1**

+ 3 pts D1.1 partial credits (minor mistake)

+ 6 pts D1.2

+ 4 pts D1.2 partial credits (numerical values are written instead of Inf)

✓ **+ 3 pts** **D1.2 partial credits (numerical/minor mistakes)**

✓ **+ 3 pts** **D2**

+ 0 pts missing / incorrect

1. 2.64
2. 1.64