# Math 4553 Final Project (II): Optimal Control for 1D Convection-Diffusion Equation

Name: Harsha Vaddireddy
UID : A20172633

**Abstract**

The CD equation describes the ow of heat, particles, or other physical quantities in situations where there is both diffusion and convection. In this report, we use optimal control input ($u$) to steer the temperature of a thin rod undergoing convection-diffusion (CD) process along with external forcing ($f$). Hence the cost function to be minimized consists of con icting objectives that is temperature distribution ($\theta$) and amount of control input ($u$). This formulation is equivalent to linear quadratic regulator (LQR) problem. The objective function is reformulated as quadratic programming problem which has unique analytical solution. Furthermore, we also use gradient descent algorithm to arrive at approximated solution which is compared to analytical solution.

## 1 Problems

Thin rod undergoing both convection and diffusion process with an external body forcing ($f$) is is modeled using 1D convection-diffusion equation. The control input ($u$) is added to the equation to control the temperature to a desired one given by $\theta_d$ temperature distribution. Hence we seek to minimize the difference between original temperature distribution $\theta$ and desired temperature distribution $\theta_d$. Additionally, we also impose minimal control input required to steer the temperature of the rod. The two conflicting objectives namely keeping the temperature difference ($\theta - \theta_d$) small while keeping the control input ($u$) effort to be minimal is modeled using weighted sum of both the objective functions resulting in minimizing:

$$\text{minimize} \qquad J_h(\theta, U) = \frac{1}{2}|||\theta - \theta_d||_2^2 + \frac{\gamma}{2}||U||_2^2 \tag{1}$$

where $\gamma$ is control parameter.

The temperature ($\theta$) distribution of the rod is given by the convection diffusion equation given by:

$$\theta_{xx} + \theta_x = f(x) + u(x), \qquad x \in (0, 1) \tag{2}$$

where $f(x)$ and $u(x)$ are external body force and control input respectively. The boundary conditions (B.C) to solve Eq. 2 is given by:

$$\theta(0) = \theta(1) = 0 \tag{3}$$

In this report, we specifically address the following questions:

1. Derive the analytical optimal solution of $\theta$ distribution and control input $u$ by applying Lagrange's Theorem to cost function $J_h(\theta, U)$.
2. Gradient descent algorithm is used to solve the quadratic optimization problem with different control weight $\gamma = 0.1, 1, \&10$.
3. We place additional constraint on control input $-1 \leq u(x) \leq 1$ where $x \in (0, 1)$ ans solve numerically with control weights $\gamma = 0.1, 1, \&10$.

# 2 Methodology

We shall apply second order central finite difference for discretizing the diffusion term $\theta_{xx}$ and forward Euler's method for discretizing the advection term $\theta_x$. To this end, we first divide the interval $[0, 1]$ into 100 equal subintervals $[x_{i-1}, x_i]$ where i = 1, 2, ..., 100. We use step size h = 0.01 resulting in spatial points as:

$$0 = x_0 < x_1 < \ldots < x_i < \ldots < x_{100} = 1. \tag{4}$$

Let $\theta_i = \theta(xi), u_i = u(x_i), f_i = f(x_i)$, and $\theta_i^d = \theta^d(x_i), i = 0, 1, \ldots, 100$. The discretized system for CD equation can be written as:

$$\frac{\theta_{i+1} - 2\theta_i + \theta_{i-1}}{h^2} + + \frac{\theta_{i+1} - \theta_i}{h} = f_i + u_i \tag{5}$$

where $i = 1, 2, \ldots, 99$ with boundaries $\theta_0 = \theta_{100} = 0$. We set external body forcing $f(x) = sin(\pi x)$ and desired temperature as $\theta_d = cos(\pi x)$.

The objective function in Eq. 1 is re written as :

$$\text{minimize} \qquad J_h(\Theta, U) = \frac{1}{2}|||\Theta - \Theta_d||_2^2 + \frac{\gamma}{2}||U||_2^2 \tag{6}$$

where $\gamma$ is control weight parameter. Hence the discretized temperature distribution

and control input vector is written as:

$$\Theta = [\theta_1, \theta_2, \ldots, \theta_{99}], \tag{7}$$

$$\Theta^d = [\theta_1^d, \theta_2^d, \ldots, \theta_{99}^d], \tag{8}$$

$$U = [u_1, u_2, \ldots, u_{99}] \tag{9}$$

## 2.1 Linear Quadratic Programming (LQR)

We reformulate the Eq. 6 in to quadratic objective function so as to get the solution analytically by applying Lagrange's theorem. The reformulated cost function is given as:

$$\text{minimize} \qquad J_h(y, U) = \frac{1}{2}|||y||_2^2 + \frac{\gamma}{2}||U||_2^2 \tag{10}$$

where $y = \theta - \theta_d$. The Eq. 10 can be written in the form of quadratic programming problem given by,

$$\begin{aligned} &\text{minimize} \qquad F = \frac{1}{2}\boldsymbol{z^T Q z} \\ &\text{subject to} \quad \boldsymbol{Az = b} \end{aligned}$$

where $N = 99$, $\boldsymbol{Q} \in \mathbb{R}^{2N \times 2N}$, $\boldsymbol{A} \in \mathbb{R}^{N \times 2N}$, $\boldsymbol{z} \in \mathbb{R}^{2N}$ and $\boldsymbol{b} \in \mathbb{R}^N$. The definitions of the above matrices are given below.

The matrix $\boldsymbol{Q} \in \mathbb{R}^{2N \times 2N}$ is defined as:

$$\boldsymbol{Q} = \begin{bmatrix} \boldsymbol{I_N} & 0 \\ 0 & \gamma \boldsymbol{I_N} \end{bmatrix}$$

The matrix $\boldsymbol{A} \in \mathbb{R}^{N \times 2N}$ is given as:

$$\boldsymbol{A} = \begin{bmatrix} b & c & 0 & 0 & \ldots & 0 & -1 & 0 & 0 & \ldots & 0 \\ a & b & c & 0 & \ldots & 0 & 0 & -1 & 0 & \ldots & 0 \\ 0 & a & b & c & \ldots & 0 & 0 & 0 & -1 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 0 & a & b & 0 & 0 & \ldots & 0 & -1 \end{bmatrix}$$

where a, b, c are given as,
$a = \dfrac{1}{h^2}$,
$b = \dfrac{-2}{h^2} - \dfrac{1}{h}$,
$c = \dfrac{1}{h^2} + \dfrac{1}{h}$.

The matrix $\boldsymbol{b} \in \mathbb{R}^N$ is defined as:

$$\boldsymbol{b} = \begin{bmatrix} f_1 - a\theta_{d_0} - b\theta_{d_1} - c\theta_{d_2} - ay_0 \\ f_2 - a\theta_{d_1} - b\theta_{d_2} - c\theta_{d_3} \\ f_3 - a\theta_{d_2} - b\theta_{d_3} - c\theta_{d_4} \\ \vdots \\ f_{99} - a\theta_{d_{98}} - b\theta_{d_{99}} - c\theta_{d_{100}} - cy_{100} \end{bmatrix}$$

The unknown vector $\boldsymbol{z} \in \mathbb{R}^{2N}$ is given by:

$$\boldsymbol{b} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{99} \\ u_1 \\ u_2 \\ \vdots \\ u_{99} \end{bmatrix}$$

The analytical solution ($\boldsymbol{z^*}$) is derived using Lagrangian theorem. The Lagrangian function is defined as,

$$l(\boldsymbol{x}, \boldsymbol{\lambda}) = f(\boldsymbol{x}) + \boldsymbol{\lambda} h(\boldsymbol{x})$$
$$l(\boldsymbol{z}, \boldsymbol{\lambda}) = \frac{1}{2}(\boldsymbol{z^T Q z}) + \boldsymbol{\lambda^T}(\boldsymbol{b - Ax})$$

The Lagrangian conditions are given by,

$$Df(\boldsymbol{x^*}) + \boldsymbol{\lambda^{*T}} Dh(\boldsymbol{x^*}) = \boldsymbol{0^T}$$
$$h(\boldsymbol{x^*}) = \boldsymbol{0}$$

Therefore solving Lagrangian conditions we get,

$$(\boldsymbol{z^{*T} Q}) - \boldsymbol{\lambda^{*T}}(\boldsymbol{A}) = \boldsymbol{0^T}$$
$$\boldsymbol{b - Az^*} = \boldsymbol{0}$$

Simplifying further we get the optimal solution $\boldsymbol{z^*}$ as,

$$\boldsymbol{z^* = Q^{-1} A^T (AQ^{-1} A^T)^{-1} b} \tag{11}$$

## 2.2 Numerical Algorithm

The Eq. 11 can be solved analytically. But the inverse of the matrices are computationally expensive and almost impossible for large systems. Hence numerical methods to approximate the solutions is sought out. We use gradient descent algorithm for finding rte optimal solution $\boldsymbol{z^*}$. We iteratively solve the equations simultaneously as given below,

$$\boldsymbol{z^{k+1}} = \boldsymbol{z^k} - \alpha(\boldsymbol{Q z^k} - \boldsymbol{A^T \lambda^k})$$
$$\boldsymbol{\lambda^{k+1}} = \boldsymbol{\lambda^k} - \beta(\boldsymbol{b} - \boldsymbol{A z^{k+1}})$$

where $\alpha$ and $\beta$ are learning rates.

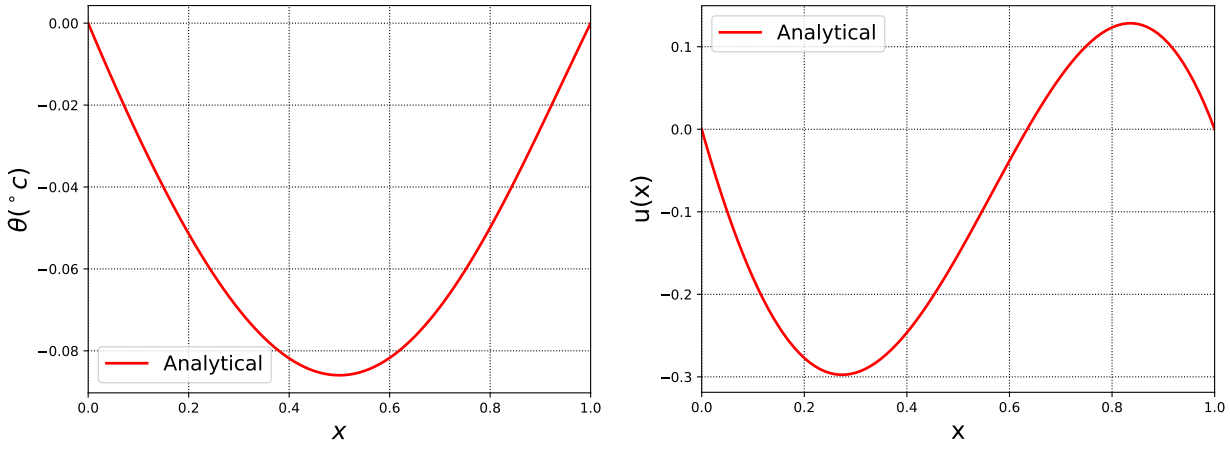
**Algorithm 1:** Gradient descent algorithm

**Input:** $\boldsymbol{Q}$, $\boldsymbol{A}$, $\boldsymbol{b}$
**Output:** $\boldsymbol{z}^*$ `// optimal vector`
`// Set the initial conditions`
$\boldsymbol{\lambda}$ = 0.2*ones(1, N) `//` $N$ `= 99.`
`// Learning rates.`
$\alpha$ = 1e-4
$\beta$ = 1e-5
`// Gradient descent()`
$\boldsymbol{z}$ = zeros(1, 2N)
**while** *tol >= 10e-6* **do**
  $\boldsymbol{z_{temp}} = \boldsymbol{z^k}$
  $\boldsymbol{z^{k+1}} = \boldsymbol{z^k}$ - $\alpha(\boldsymbol{Q z^k}$ - $\boldsymbol{A^T \lambda^k})$
  $\boldsymbol{\lambda^{k+1}} = \boldsymbol{\lambda^k} - \beta(\boldsymbol{b} - \boldsymbol{A z^{k+1}})$
  tol =norm$(\boldsymbol{z^{k+1}} - \boldsymbol{z_{temp}})$
**end while**
return $\boldsymbol{z}^*$

# 3 Results

In this section results are presented for analytical solution and later compared with numerical solution with different control weights $\gamma$ =0.1,1.0 & 10.0. We solve the same set up with additional constraint on control input $-1.0 \leq u(x) \leq 1.0$.

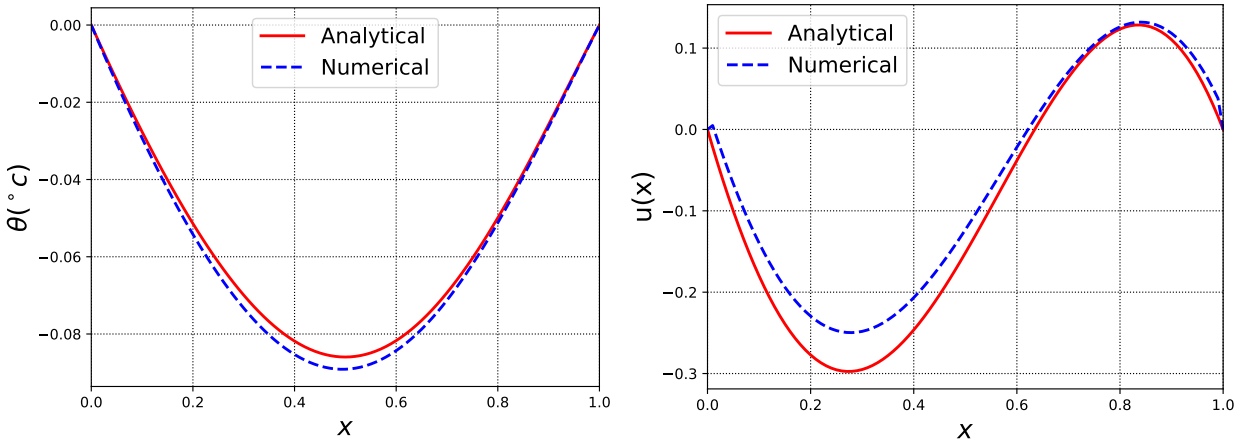
The analytical solution of the temperature distribution ($\theta$) and control input $u(x)$ are plotted as shown in Fig. 1. The numerical solutions with control weight $\gamma = 0.1$ varies from anaytical solution as seen in Fig. 2 where as with increased control weights $\gamma = 1.0$ & 10.0, the both solutions matches (Fig. 3 & Fig. 4).




(a) Optimum temperature ($\theta(x)$) distribution.

(b) Optimum control input $u(x)$.

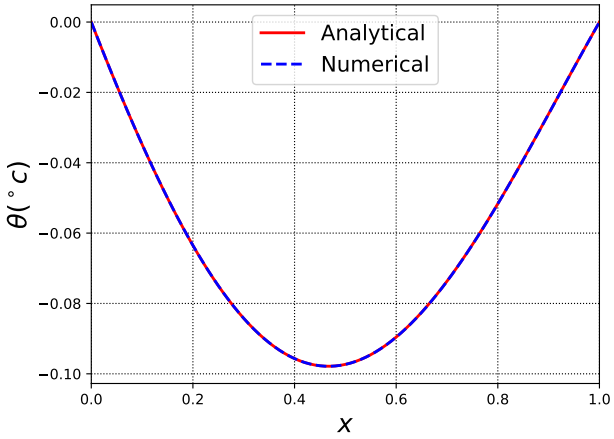
Figure 1: Analytical solution derived from Lagrange's theorem.



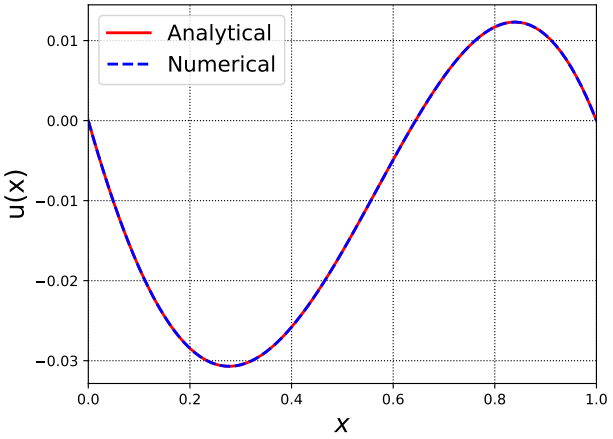

(a) Optimum temperature ($\theta(x)$) distribution.

(b) Optimum control input $u(x)$.

Figure 2: Comparison of numerical & analytical solutions with control weight $\gamma = 0.1$

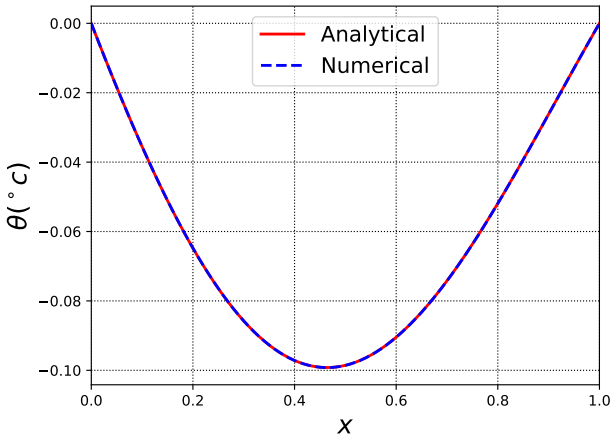



(a) Optimum temperature ($\theta(x)$) distribution.

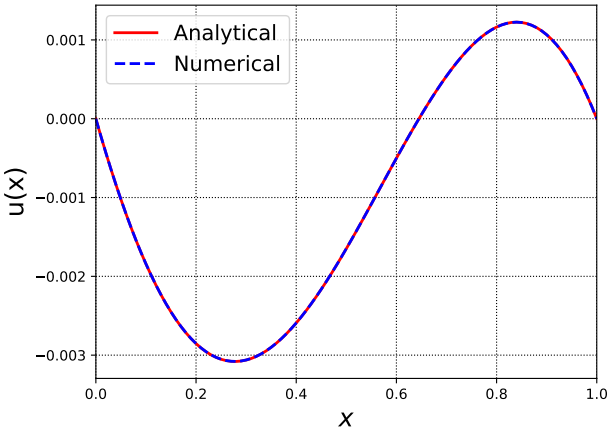



(b) Optimum control input $u(x)$.

Figure 3: Comparison of numerical & analytical solutions with control weight $\gamma = 1.0$



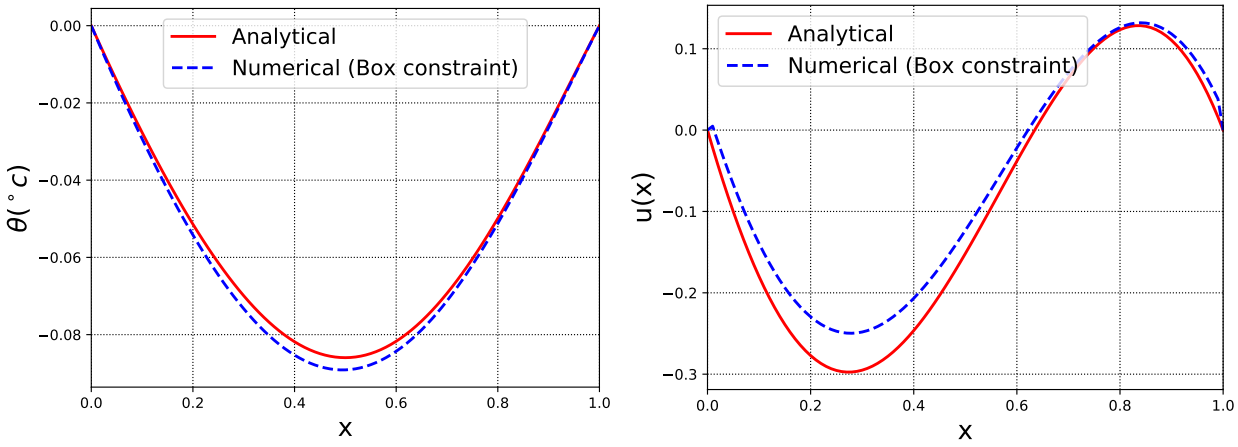

(a) Optimum temperature ($\theta(x)$) distribution.




(b) Optimum control input $u(x)$.

Figure 4: Comparison of numerical & analytical solutions with control weight $\gamma = 10.0$

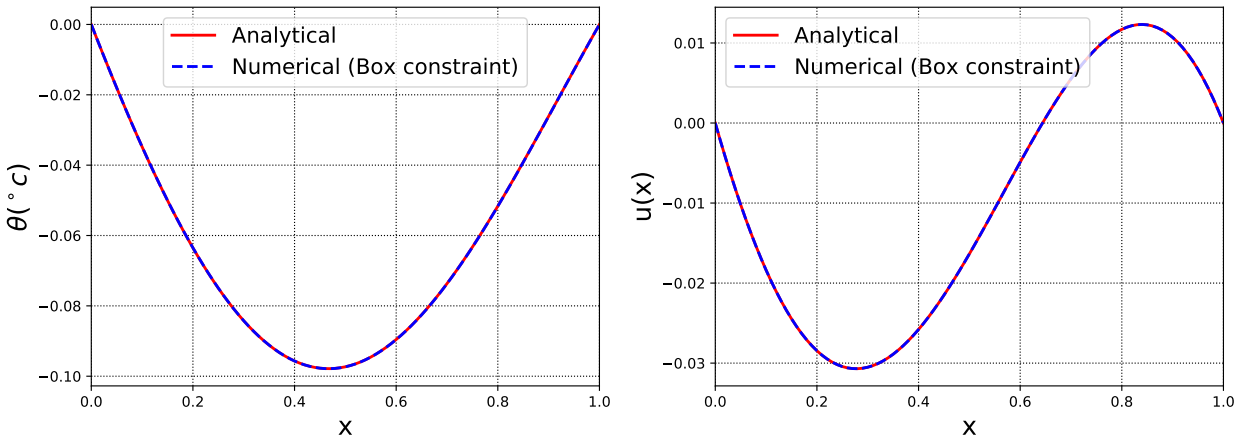
The above mentioned behavior is repeated for box constraint on control input $u(x)$. The numerical solutions with control weight $\gamma = 0.1$ varies from anaytical solution as seen in Fig. 5 where as with increased control weights $\gamma = 1.0$ & $10.0$, the both solutions matches (Fig. 6 & Fig. 7).




(a) Optimum temperature ($\theta(x)$) distribution.

(b) Optimum control input $u(x)$.

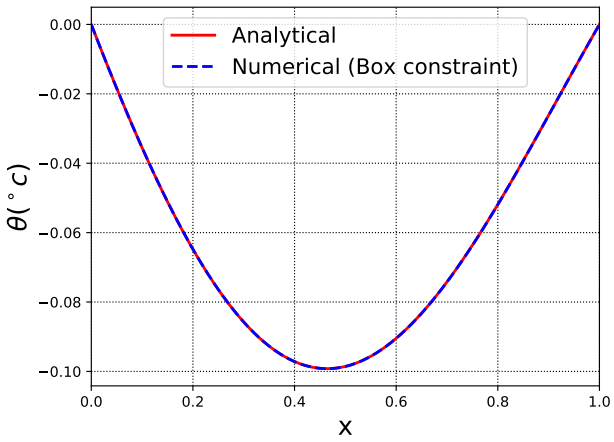
Figure 5: Comparison of numerical with box constraint & analytical solutions with control weight $\gamma = 0.1$



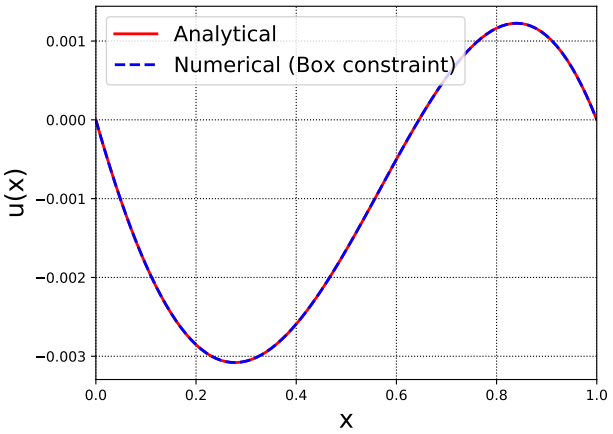

(a) Optimum temperature ($\theta(x)$) distribution.

(b) Optimum control input $u(x)$.

Figure 6: Comparison of numerical with box constraint & analytical solutions with control weight $\gamma = 1.0$




(a) Optimum temperature ($\theta(x)$) distribution.




(b) Optimum control input $u(x)$.

Figure 7: Comparison of numerical with box constraint & analytical solutions with control weight $\gamma = 10.0$

| Control weight | Analytical | Numerical | Numerical with box constraint |
|---|---|---|---|
| $\gamma = 0.1$ | 24.8794 | 24.8837 | 24.8837 |
| $\gamma = 1.0$ | 25.0244 | 25.0254 | 25.0254 |
| $\gamma = 10$ | 25.0394 | 25.0404 | 25.0404 |

Table 1: Objective function value $\frac{1}{2}\boldsymbol{z}^T\boldsymbol{Q}\boldsymbol{z}$ for different control weights.

# 4 Observation and Conclusions

The CD equation with external body force and control input is solved using linear quadratic programming problem to achieving the desire temperature with minimal control effort. The analytical solution is derived using Lagrange theorem. The numerical solution is computed using gradient descent algorithm. As discussed in Section. 3, with lower control input $\gamma = 0.1$, results slightly vary from the analytical solution. With increase in control input $\gamma = 1.0$ & 10.0, the numerical solution approximated the analytical counterpart very well. The box constraint on the control input $u(x)$ didn't change the results and pattern repeated the same.

The number of iteration for gradient descent is is high in order of 400000 (0.4 million) steps to converge to tolerance 10e-6. This aspect can be improved if we use steepest descent algorithm and conjugate gradient algorithm. For this report, we restrict to gradient descent algorithm. The lower control wight is also less robust to initial condition $\boldsymbol{z}$ to start the gradient decent algorithm. While with increase in $\gamma$ the robustness increases with little to no change in solution with drastic initial condition change. Overall, in this report we steered the temperature of the 1D rod to desired

temperature woith minimal input effort using linear quadratic optimization problem.

# Reference


1. E.Chong and S.Zak An Introduction to Optimization, 3rd Ed., Wiley, 2008.
2. Jones, Eric, Travis ,and Peterson."SciPy: Open source scientific tools for Python."(2014).


# Appendix

## Python code

```
#%%
# The CD equation with extrenal forcing(f(x)) and control input(u(x)
# is solved using LQR and gradeint descent alogrithm.
import numpy as np
from matplotlib import pyplot as plt

## Constants
n = 101
gamma = 0.1
ic = 0.1
h = 0.01
#### Construct Q matrix of 198 x 198 (2N x 2N)
q11 = np.eye(n-2,n-2)
q12 = np.zeros([n-2,n-2])
q13 = np.zeros([n-2,n-2])
q14 = np.eye(n-2,n-2)*gamma
q = np.block([[q11,q12],[q13,q14]])
##### Construct A matrix of 99 x 198 (N x 2N)
c1 = 1.0/h**2
c2 = -2.0/h**2 - 1.0/h
c3 = 1.0/h**2 + 1.0/h
c11 = np.full(n-3,c1)
c12 = np.full(n-2,c2)
c13 = np.full(n-3,c3)
a11 = np.diag(c11,-1)
a12 = np.diag(c12, 0)
a13 = np.diag(c13, 1)
a1 = a11 + a12 + a13
a2  = np.eye(n-2,n-2)*-1.0
a  =  np.concatenate((a1, a2), axis=1)
#%% Construct b matirx
```

```
x = np.linspace(0,1,n)
f  = np.sin(np.pi*x)
th_d = np.cos(np.pi*x)
b = np.zeros([n-2,1])
for i in range(n-2):
    b[i] = f[i+1] - c1*th_d[i] -c2*th_d[i+1] -c3*th_d[i+2]
b[0] = b[0] + c1*th_d[0]
b[n-3] = b[n-3] + c3*th_d[100]
```

```
#%%
## Construct the solution
qinv = np.linalg.inv(q)
atrp = np.transpose(a)

t1 = np.matmul(qinv, atrp)

t2 = np.matmul(a, qinv)

t2 = np.linalg.inv(np.matmul(t2, atrp))

z = np.matmul(t1,t2)
z = np.matmul(z,b)

u = z[n-2:,0]
u = np.append(u,[0])
u = np.append([0],u)

theta = z[0:99,] + th_d.reshape(101,1)[1:100,]
theta = np.append(theta,[0]).reshape(100,1)
theta = np.append([0],theta).reshape(101,1)
#%% Numerical solution
z_num = np.full((2*n-4,1),ic)
lam = np.full((n-2,1),ic)

alpha = 0.0001
beta = 0.00001
tol = 5
i = 0
#for i in range(1):
while tol >= 10e-6:
      z_temp = z_num
      z_num = z_num - alpha*(np.matmul(q,z_num) - np.matmul(atrp,
   lam))
      lam = lam + beta*(b - np.matmul(a,z_num))
```

```
        i = i+1
        print(i)
        tol  = np.linalg.norm((z_temp - z_num))

u_num = z_num[n-2:,0]
u_num = np.append(u_num,[0])
u_num = np.append([0],u_num)
```

```
th_num = np.append([0],th_num).reshape(101,1)

#%% Numerical solution with box constraint
z_num1 = np.full((2*n-4,1),ic)
lam1 = np.full((n-2,1),ic)

alpha = 0.0001
beta = 0.00001
tol1 = 5

k = 0
#for i in range(1):
while tol1 >= 10e-6:
        z_temp1 = z_num1
        z_num1 = z_num1 - alpha*(np.matmul(q,z_num1) - np.matmul(atrp,
    lam1))
        lam1 = lam1 + beta*(b - np.matmul(a,z_num1))
        for j in range(n-2):
            if z_num1[n-2+j,0] < -1:
                z_num1[n-2+j,0] == -1
            elif z_num1[n-2+j,0] > 1:
                    z_num1[n-2+j,0] == 1
        tol1  = np.linalg.norm((z_temp1 - z_num1))
        k = k+1
        print(k)

u_num1 = z_num1[n-2:,0]
u_num1 = np.append(u_num1,[0])
u_num1 = np.append([0],u_num1)

th_num1 = z_num1[0:n-2,]+ th_d.reshape(101,1)[1:100,]
th_num1 = np.append(th_num1,[0]).reshape(100,1)
th_num1 = np.append([0],th_num1).reshape(101,1)

#%% Post Processing 1
plt.figure()
```

```
plt.plot(x, theta, 'r' , linewidth=2.0, label='Analytical')
plt.plot(x, th_num, '--b' , linewidth=2.0, label='Numerical')
plt.xlabel('$x$',fontsize=18)
plt.ylabel(r'$\theta(^\circ c)$',fontsize=18)
plt.legend( prop={'size': 15})
plt.grid(color='black', linestyle='dotted')
```

```
th_num = np.append([0],th_num).reshape(101,1)

#%%  Numerical solution with box constraint
z_num1 = np.full((2*n-4,1),ic)
lam1 = np.full((n-2,1),ic)

alpha = 0.0001
beta = 0.00001
tol1 = 5

k = 0
#for i in range(1):
while tol1 >= 10e-6:
      z_temp1 = z_num1
      z_num1 = z_num1 - alpha*(np.matmul(q,z_num1) - np.matmul(atrp,
    lam1))
      lam1 = lam1 + beta*(b - np.matmul(a,z_num1))
      for j in range(n-2):
          if z_num1[n-2+j,0] < -1:
             z_num1[n-2+j,0] == -1
          elif z_num1[n-2+j,0] > 1:
               z_num1[n-2+j,0] == 1
      tol1  = np.linalg.norm((z_temp1 - z_num1))
      k = k+1
      print(k)

u_num1 = z_num1[n-2:,0]
u_num1 = np.append(u_num1,[0])
u_num1 = np.append([0],u_num1)

th_num1 = z_num1[0:n-2,]+ th_d.reshape(101,1)[1:100,]
th_num1 = np.append(th_num1,[0]).reshape(100,1)
th_num1 = np.append([0],th_num1).reshape(101,1)

#%% Post Processing 1
plt.figure()
plt.plot(x, theta, 'r' , linewidth=2.0, label='Analytical')
```

```
plt.plot(x, th_num, '--b' , linewidth=2.0, label='Numerical')
plt.xlabel('$x$',fontsize=18)
plt.ylabel(r'$\theta(^\circ c)$',fontsize=18)
plt.legend( prop={'size': 15})
plt.grid(color='black', linestyle='dotted')
```