CL 244 Course Project

Om Mihani
Rohit Suryavanshi
Shreya Makkar
Sumit Kumar

November 24, 2021

Department of Chemical Engineering
Indian Institute of Technology, Bombay

Contents

1 Problem Statement

The following ODE-BVP describes heat transfer in a straight fin of uniform cross section:

$$\frac{d^2\theta}{dx^2} - m^2\theta = 0$$
$$x = 0, \theta = 1$$
$$x = 1, \frac{d\theta}{dx} = -Bi\theta$$

Where θ is the dimensionless temperature at any position in the fin, x is dimensionless location, Bi is the dimensionless Biot number, and m^2 is the product of Bi and a dimensionless group involving fin dimensions.

(A) Derive backward finite difference approximation of order $O(x^2)$.

(B) Obtain the set of algebraic equations to be solved using finite difference derived in (A) at boundary and central difference approximations.

(C) Obtain the temperature profile for Bi = 0.04737, m=2.1324 using the jacobi method.

(D) Solve for varying mesh sizes and demonstrate convergence.

2 Part-A

Taylor series of $f(x - \delta x)$:

$$f(x-\delta x) = f(x) - \delta x f'(x) + \frac{\delta x^2 f''(x)}{2!} - \frac{\delta x^3 f'''(x)}{3!} + \frac{\delta x^4 f^{(4)}(x)}{4!} - \frac{\delta x^5 f^{(5)(x)}}{5!} + \dots \quad (1)$$

Taylor series of $f(x - 2\delta x)$:

$$f(x-2\delta x) = f(x) - 2\delta x f'(x) + 4\frac{\delta x^2 f''(x)}{2!} - 8\frac{\delta x^3 f'''(x)}{3!} + 16\frac{\delta x^4 f^{(4)}(x)}{4!} - 64\frac{\delta x^5 f^{(5)}(x)}{5!} + \dots \quad (2)$$

Taylor series of $f(x - 3\delta x)$:

$$f(x-3\delta x) = f(x) - 3\delta x f'(x) + 9\frac{\delta x^2 f''(x)}{2!} - 27\frac{\delta x^3 f'''(x)}{3!} + 81\frac{\delta x^4 f^{(4)}(x)}{4!} - 243\frac{\delta x^5 f^{(5)}(x)}{5!} + \dots \quad (3)$$

-4eq. (1)+5eq. (2)-eq. (3) becomes:

$$-2f(x) + \delta x^2 f''(x) + O(\delta x^4) \quad (4)$$

(eq. (4) + 2f(x))/δx^2 gives

$$f''(x) + O(\delta x^2)$$

Thus,

$$f''(x) = \frac{2f(x) - 5f(x-\delta x) + 4f(x-2\delta x) - f(x-3\delta x)}{\delta x^2} + O(\delta x^2)$$

Similarly, -4eq. (1) + eq. (2) gives

$$2\delta x f'(x) - 3f(x) + O(\delta x^3) \tag{5}$$

Thus,

$$f'(x) = \frac{3f(x) - 4f(x-\delta x) + f(x-2\delta x)}{2\delta x} + O(\delta x^2) \tag{6}$$

3 Part-B

3.1 Backward Difference

The interval $[0,1]$ is divided into n intervals (where $n = 1/h$, h being the step size). The naming convention of the points is such that $x = 0$ is x_1, $x = 0 + h$ is x_2 and so on. We employ backward difference formula for all points except x_1, x_2 and x_3. For x_2 and x_3, we employ central difference formula:

$$f''(x_i) = \frac{f(x_{i+1}) - 2f(x_i) + f(x_{i-1})}{\delta x^2},$$

so that we have equal number of equations and variables.

3.1.1 Boundary conditions

We are given that at $x = 0$, $\theta = 1$ and at $x = 1$, $\frac{d\theta}{dx} = -Bi\theta$.
Employing backward difference approximation in the second boundary condition,

$$\frac{3\theta(n+1) - 4\theta(n) + \theta(n-1)}{2\delta x} + Bi\theta(n+1) = 0$$

i.e.

$$\theta(n+1)\left(\frac{3}{2\delta x} + Bi\right) + \theta(n)\left(\frac{-2}{\delta x}\right) + \theta(n-1)\left(\frac{1}{2\delta x}\right) \tag{7}$$

3.1.2 Interior points

For x_2, using central difference approximation,

$$\frac{\theta(x_1) - 2\theta(x_2) + \theta(x_3)}{\delta x^2} - m^2\theta(x_2) = 0$$

which gives,

$$\theta(x_1)\left(\frac{1}{\delta x^2}\right) + \theta(x_2)\left(\frac{-2}{\delta x^2} - m^2\right) + \theta(x_3)\left(\frac{1}{\delta x^2}\right) = 0 \tag{8}$$

Similarly for x_3, we get

$$\theta(x_2)\left(\frac{1}{\delta x^2}\right)+\theta(x_3)\left(\frac{-2}{\delta x^2}-m^2\right)+\theta(x_4)\left(\frac{1}{\delta x^2}\right)=0 \tag{9}$$

For $i=4$ to n, we apply backward difference approximation

$$\frac{2\theta(x_i)-5\theta(x_{i-1})+4\theta(x_{i-2})-\theta(x_{i-3})}{\delta x^2}-m^2\theta(x_i)=0$$

which gives,

$$\theta(x_{i-3})\left(\frac{-1}{\delta x^2}\right)+\theta(x_{i-2})\left(\frac{4}{\delta x^2}\right)+\theta(x_{i-1})\left(\frac{-5}{\delta x^2}\right)+\theta(x_i)\left(\frac{2}{\delta x^2}-m^2\right)=0 \tag{10}$$

Total variables: n+1
Total equations: n+1 (first boundary condition, eq. (7), eq. (8), eq. (9) and $n-3$ equations from eq. (10))
Thus, a linear system of equations of the form $Ax=B$ can be constructed.
A will be written as

$$\begin{array}{ccccccc}
1 & 0 & 0 & 0 & 0 & 0 & \cdots \\
\frac{1}{h^2} & \frac{-2}{h^2}-m^2 & \frac{1}{h^2} & 0 & 0 & 0 & \cdots \\
0 & \frac{1}{h^2} & \frac{-2}{h^2}-m^2 & \frac{1}{h^2} & 0 & 0 & \cdots \\
\frac{-1}{h^2} & \frac{4}{h^2} & \frac{-5}{h^2} & \frac{-2}{h^2}-m^2 & 0 & 0 & \cdots \\
0 & \frac{-1}{h^2} & \frac{4}{h^2} & \frac{-5}{h^2} & \frac{-2}{h^2}-m^2 & 0 & \cdots \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
\cdots & \cdots & \cdots & \cdots & \frac{1}{2h} & \frac{-2}{h} & \frac{3}{2h}+Bi
\end{array}$$

x vector is

$$\begin{array}{c}
x_1 \\
x_2 \\
x_3 \\
\cdots \\
\cdots \\
x_{n+1}
\end{array}$$

B vector will be

$$\begin{array}{c}
1 \\
0 \\
0 \\
\cdots \\
\cdots \\
0
\end{array}$$

3.2 Central Difference

The interval [0,1] is divided into n intervals (where $n=1/h$, h being the step size). The naming convention of the points is such that $x=0$ is $x_1, x=0+h$

is x_2 and so on. We employ central difference formula for all points except the boundary points.

3.2.1 Boundary Conditions

We are given that at $x = 0$, $\theta = 1$ and at $x = 1$, $\frac{d\theta}{dx} = -Bi\theta$.
Employing backward difference approximation in the second boundary condition, we obtain eq. (7)

3.2.2 Interior Points

Taylor series of $f(x + \delta x)$:

$$f(x+\delta x) = f(x)+\delta x f'(x)+\frac{\delta x^2 f''(x)}{2!}+\frac{\delta x^3 f'''(x)}{3!}+\frac{\delta x^4 f^{(4)}(x)}{4!}+\frac{\delta x^5 f^{(5)(x)}}{5!}+\dots \tag{11}$$

$\frac{f(x-\delta x)+f(x+\delta x)}{2\delta x} + O(\delta x^2)$ gives $f'(x)$.
Thus, for the interior points, i.e. for $i = 2$ to n,

$$\frac{\theta(x_{i+1}) - 2\theta(x_i) + \theta(x_{i-1})}{h^2} - m^2\theta(x_i) = 0$$

which gives,

$$\theta(x_{i-1}) + \theta(x_i)\left(-2 - h^2m^2\right) + \theta(x_{i-1}) = 0 \tag{12}$$

Total variables: n+1
Total equations: n+1 (2 boundary conditions and $n-1$ equations from eq. (12))
Thus, a linear system of the form $Ax = B$ can be constructed.
A will be

$$\begin{array}{ccccccc}
1 & 0 & 0 & 0 & 0 & 0 & \cdots \\
1 & -(2+h^2m^2) & 1 & 0 & 0 & 0 & \cdots \\
0 & 1 & -(2+h^2m^2) & 1 & 0 & 0 & \cdots \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
\cdots & \cdots & \cdots & \cdots & 1 & -4 & 3+2hBi
\end{array}$$

x vector is

$$\begin{array}{c}
x_1 \\
x_2 \\
\cdots \\
\cdots \\
x_{n+1}
\end{array}$$

B vector will be

$$\begin{array}{c}
1 \\
0 \\
0 \\
\cdots \\
\cdots \\
0
\end{array}$$

4 Part-C

4.1 Jacobi Method

Jacobi iteration is an iterative method to solve a system of linear equations. We have the following set of equations:

$$\begin{aligned} a_{11} * x_1 + a_{12} * x_2 + \ldots + a_{1n} * x_n &= b_1 \\ a_{21} * x_1 + a_{22} * x_2 + \ldots + a_{2n} * x_n &= b_2 \\ &\ldots \\ a_{n1} * x_1 + a_{n2} * x_2 + \ldots + a_{nn} * x_n &= b_n \end{aligned}$$

Step 1:

$$x_i = \frac{b_i - \sum_{j=1,j}^{n} a_{ij} \times x_j}{a_{ii}} \text{ for } i = 1:n \tag{13}$$

Step 2:
Take a guess solution x_0 and find a new solution x_1 using the equation derived in Step1 Substitute x_j by x_j^0 in eq. (13) and get x_i^1, which is equal to x_i calculated i.e.

$$x_i^1 = \frac{b_i - \sum_{j=1,j}^{n} a_{ij} \times x_j^0}{a_{ii}} \text{ for } i = 1:n$$

Step3:
Repeat step 2 by using x^0 as newly calculated x^1, and calculate x^1 until you get convergence Convergence criteria:

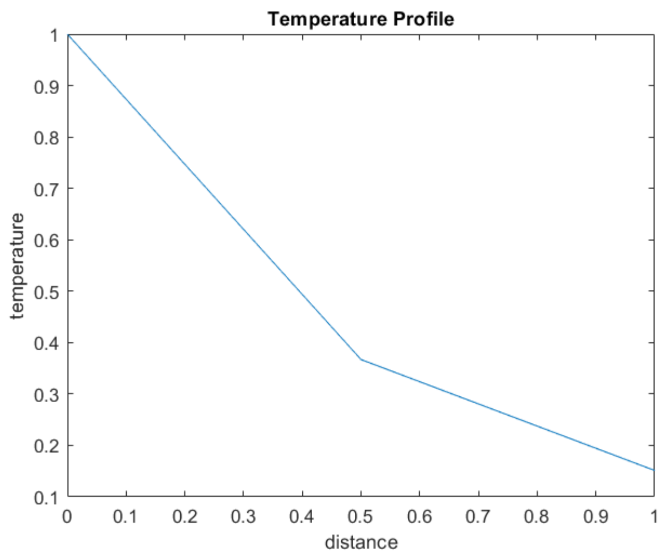
$$\frac{||x^1 - x^0||}{||x^1||} \leq tolerance$$

4.2 Results obtained

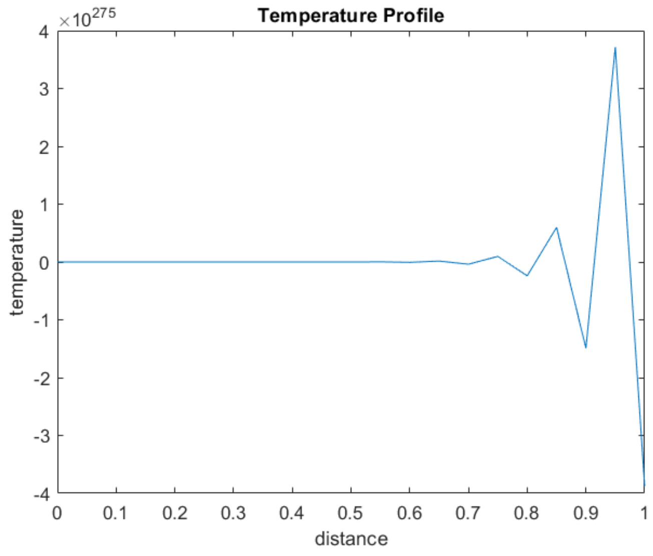
4.2.1 Backward Difference

Due to an ill-conditioned matrix obtained in section 3.1.2(of the order 10^8), incorrect results are obtained as we change the value of h.

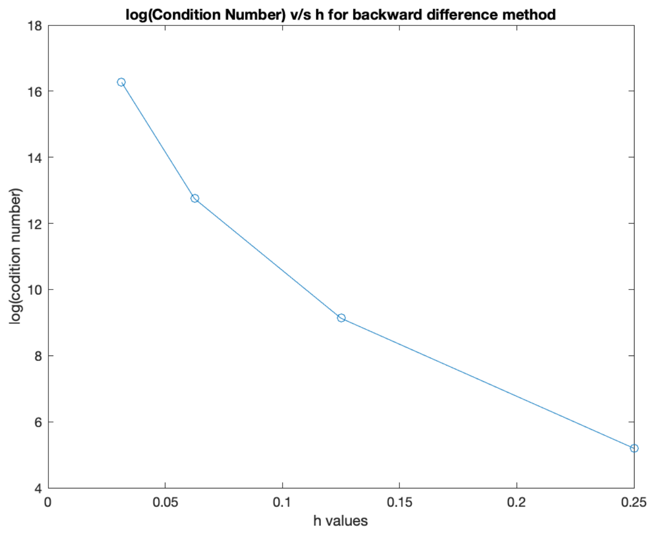
- For $h = 0.5$, the graph obtained is



- For $h = 0.05$, the graph obtained is



Remark. The upper bound of the condition number was obtained by evaluating $||A^{-1}||||A||$.



4.2.2 Central Difference

The condition number for the matrix obtained is small yet, the solution diverges for Jacobi method. Using the Jacobi method to solve the system with the value of h as 0.05, the solution vector i.e. the temperature profile obtained is
$1.0e + 52*$
0.0000
0.0000
-0.0000
0.0001
-0.0001
0.0001
-0.0002
0.0003
-0.0004
0.0006
-0.0008
0.0012
-0.0018
0.0026
-0.0038
0.0055
-0.0081
0.0118

-0.0172
0.0251
-0.0366
0.0533
-0.0778
0.1135
-0.1655
0.2415
-0.3522
0.5137
-0.7493
1.0930
-1.5943
2.3254
-3.3919
(for $h = 1/32$)

4.3 Inbuilt solvers

For the given problem we can use the following MATLAB inbuilt solvers:

1. BVP4c: This integrates a system of differential equations of the form y' = f(x,y), subject to the boundary conditions and the initial solution guess. It uses the implicit Runge-Kutta formula with a continuous extension interpolant. Known as the Lobatto IIIa method, it uses 3 stage formula.

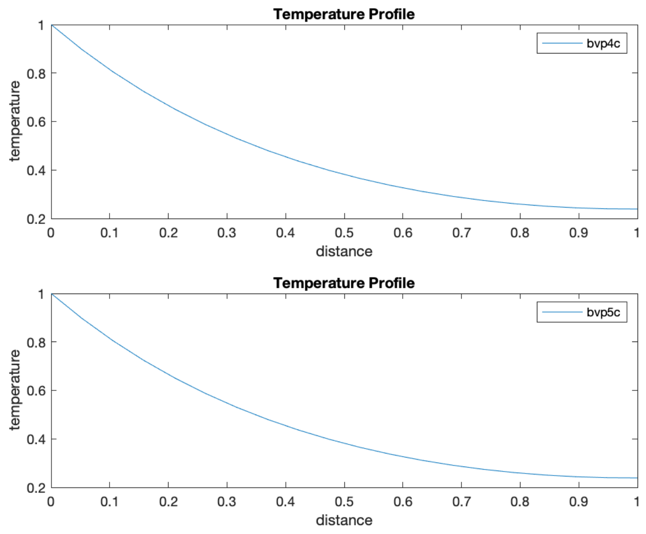
2. BVP5c: This integrates a system of differential equations of the form y' = f(x,y), subject to the boundary conditions and the initial solution guess. It uses the implicit Runge-Kutta formula with a continuous extension interpolant. Known as the Lobatto IIIa method, it uses 4 stage formula.

The formulation is done by first breaking the second order ODE to a system of first order ODEs.

$$y_1 = y(x)$$
$$y_2 = y'(x)$$

Thus, the system of equations become
$y_1' = y_2$
$y_2' = m^2 y_1$
Then boundary values are stored in another vector
$y_{1,L} - 1$
$y_{2,R} - Biy_{1,R}$
where y_L and y_R are the values at the left and right boundary.
After this, we need an initial guess vector which we can get from `bvpinit` which takes the mesh vector and a guess value. Then we can find the solution by `bvp4c` by supplying the system of odes, boundary conditions and initial guess vector.

Temperature Profiles obtained:



5 Part-D

5.1 Varying h

We now vary the mesh size, i.e. the step size h, and determine at what value of h we achieve mesh independence.

- For mesh independence we need to compare temperature profile vectors obtained from different h. Temperature at only common x values can be compared.

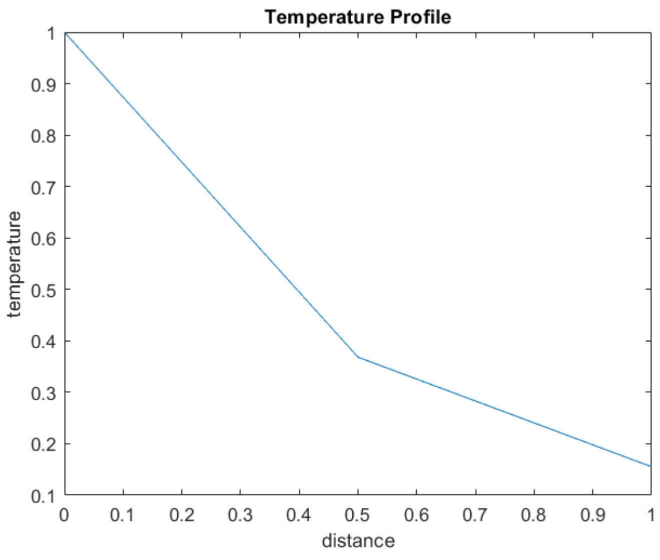
  For example, for $h = 1$, $\theta_{vector1} = [\theta_0, \theta_1]^T$, for $h = 0.5$, $\theta_{vector2} = [\theta_0, \theta_{0.5}, \theta_1]^T$. So for comparing $\theta_{vector1}$ and $\theta_{vector2}$, we will compare only θ_0 and θ_1 in both.

  Thus, if

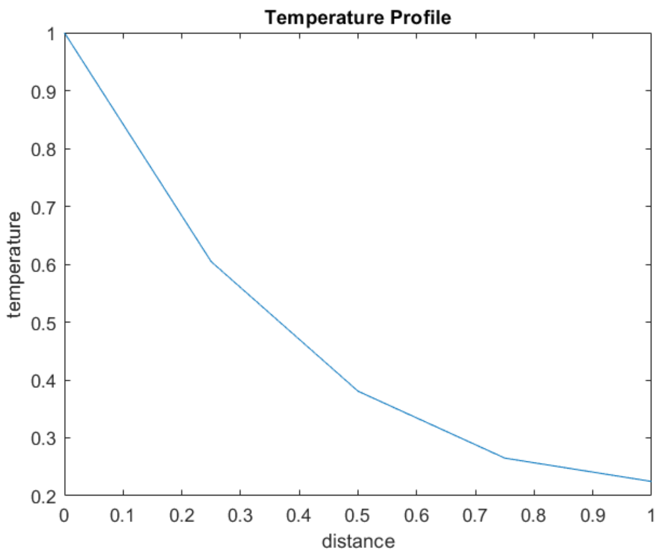
  $$\left|\frac{(\theta_{vector2}(2j-1) - \theta_{vector1}(j))}{(\theta_{vector2}(2j-1))}\right| < tolerance \; \forall j = 1 : length(\theta_{vector1})$$

  then we will have achieved mesh independence, i.e. the result will be independent to the value of h.

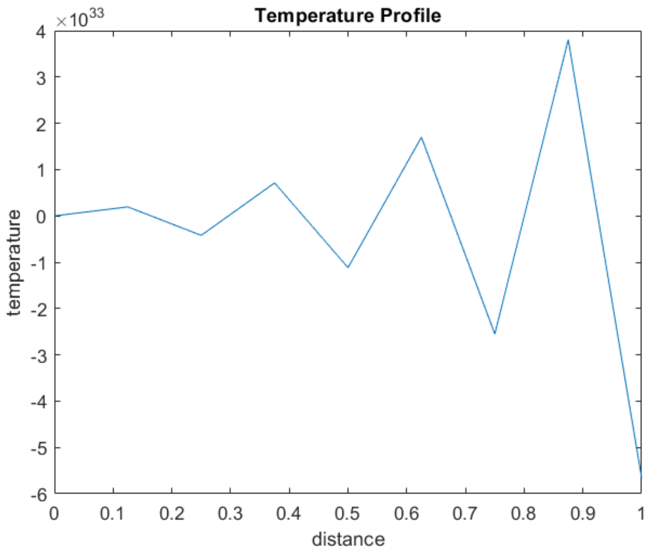
1. $h = 0.5$



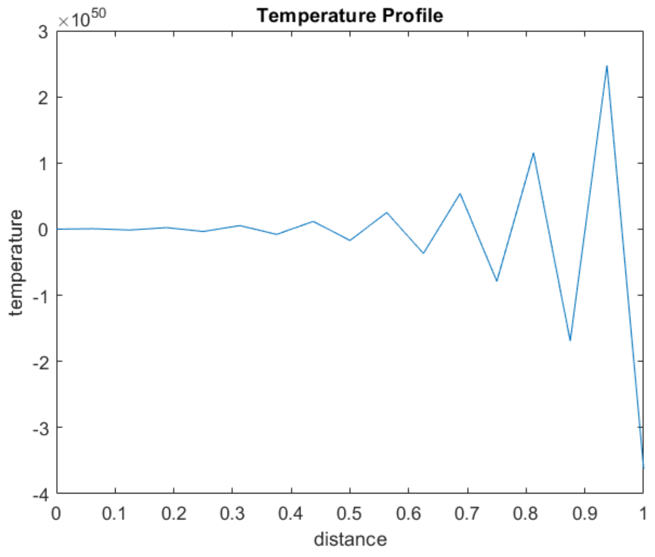
2. $h = 0.25$



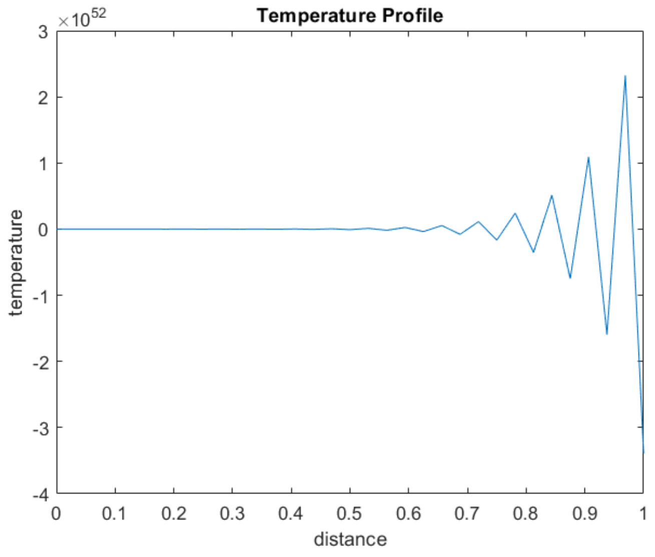
3. $h = 0.125$



4. $h = 0.0625$

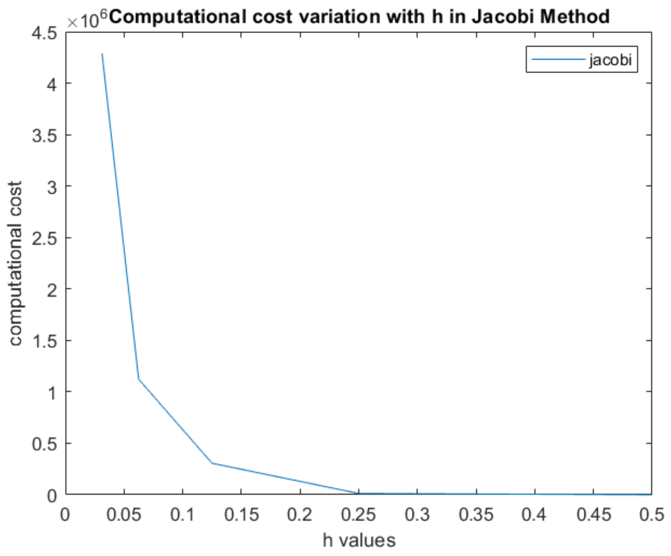


5. $h = 0.03125$



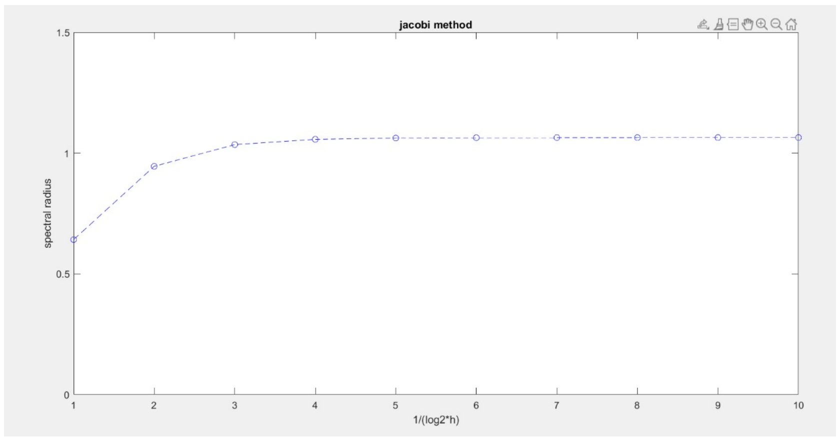
Analysis: We obtain a decreasing profile for initial values of h, but upon decreasing h to smaller values, the values start to diverge and the system becomes unstable.

Variation of computational cost v/s h:
(for jacobi method, the computational cost is MN^2, where M is the number of iterations and N is the size of coefficient matrix.)



5.2 Convergence

Convergence of Jacobi method can be determined by calculating the spectral radius of $S^{-1}T$.
Matrix representation of jacobi method: $x^k = -x^{k-1}D^{-1}(L+U)+D^{-1}b$. Thus $S^{-1} = D^{-1}$ and $T = -(L+U)$. (L contains entries below diagonal, U above diagonal and D contains just the diagonal entries.)
Spectral radius $\rho(S^{-1}T)$ is the maximum eigenvalue of $S^{-1}T$. If $\rho(S^{-1}T) < 1$, then convergence of jacobi method is guaranteed.
$\rho(S^{-1}T)$ for central difference method: 1.0668
$\rho(S^{-1}T)$ for backward difference method: 1.3644
($h = 0.05$ and $w = 1.2$)

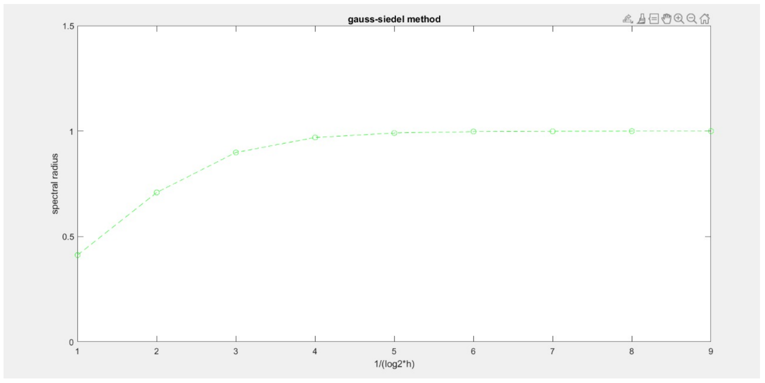


6 Additional Analysis

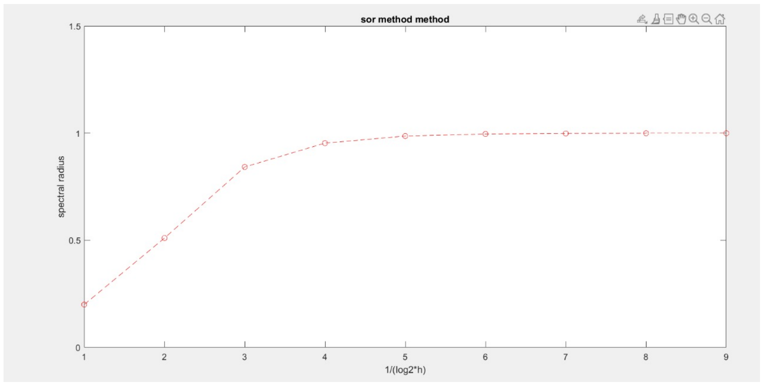
6.1 Other Iterative Methods

We can solve the system of equations using other iterative methods such as:

1. Gauss Seidel Method
   Spectral radius: Backward difference: 1.0001
   Central Difference: 0.9833

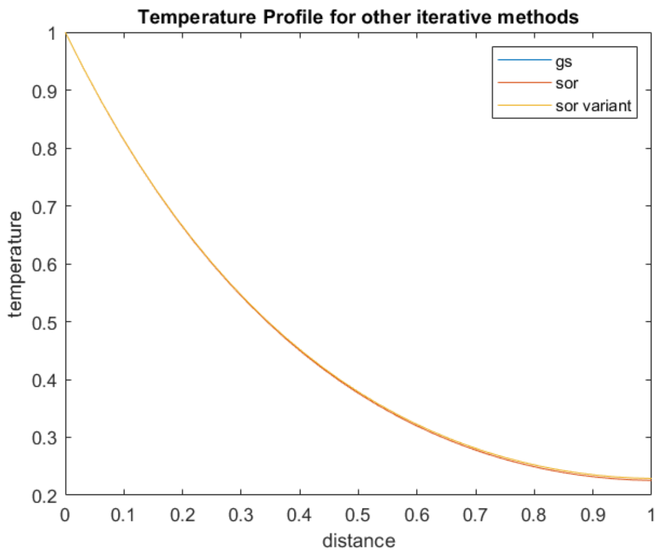


2. Successive Over relaxation Method
   Spectral radius: Backward difference: 1.0001
   Central Difference: 0.9745

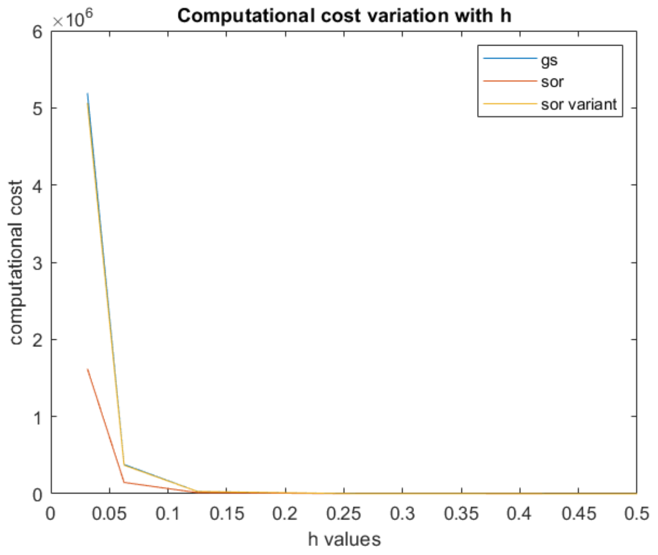


3. Successive Over relaxation Variant Method
   Spectral radius: Backward difference: 1.0001
   Central Difference: 0.9833

Temperature Profiles



Computational cost variation with h



Temperature vector of Gauss Seidel, SOR and SOR variant respectively:

```
1.0000  1.0000  1.0000
0.9373  0.9371  0.9373
0.8788  0.8784  0.8788
0.8242  0.8236  0.8242
0.7732  0.7725  0.7732
0.7257  0.7248  0.7257
0.6814  0.6804  0.6814
0.6401  0.6389  0.6401
0.6017  0.6003  0.6017
0.5659  0.5644  0.5659
0.5327  0.5310  0.5327
0.5018  0.5000  0.5018
0.4731  0.4712  0.4731
0.4466  0.4445  0.4466
0.4220  0.4198  0.4220
0.3993  0.3970  0.3993
0.3784  0.3759  0.3784
0.3591  0.3566  0.3591
0.3415  0.3388  0.3415
0.3253  0.3226  0.3253
0.3106  0.3078  0.3106
0.2973  0.2944  0.2973
0.2853  0.2823  0.2853
0.2746  0.2715  0.2746
0.2651  0.2620  0.2651
0.2568  0.2536  0.2568
0.2496  0.2463  0.2496
0.2435  0.2402  0.2435
0.2385  0.2352  0.2385
0.2346  0.2312  0.2346
0.2317  0.2283  0.2317
0.2298  0.2265  0.2298
0.2289  0.2256  0.2289
```

6.2 Physical significance of problem

The equation in the problem statement results when we solve the heat transfer equation for a static straight fin with a uniform cross-section and total length L, which is at a higher temperature than its surroundings at a steady state. Also, the heat transfer coefficient at the interface is 'h' and the surrounding temperature is T_0. Heat transfer equation:

$$\rho c_p \frac{DT}{Dt} = k\nabla^2 T - \zeta : \nabla v - \frac{\partial ln\rho}{\partial lnt}\frac{D\rho}{DT} - h(T - T_0)$$

$$\implies \rho c_p \left(\frac{\partial T}{\partial t} + v \cdot \nabla T\right) = k\nabla^2 T - \zeta : \nabla v - \frac{\partial ln\rho}{\partial lnt}\frac{D\rho}{DT} - h(T - T_0)$$

$\frac{\partial T}{\partial t} = 0$ at steady state, $v \cdot \nabla T = 0$ and $\zeta : \nabla v = 0$ as $v = 0$, $\frac{\partial ln\rho}{\partial lnt}\frac{D\rho}{DT} = 0$ as ρ is constant.

$$\implies k\nabla^2 T - h(T - T_0) = 0$$

$$k\frac{d^2T}{dx^2} - h(T - T_0) = 0$$

non dimensionalising the equation,

$$\theta = \frac{T - T_0}{T_0} \text{ and } x^* = \frac{x}{L}$$

so the equation becomes

$$\frac{T_0 \times k}{L^2} \times \frac{d^2\theta}{dx^{*2}} - h \times T_0\theta = 0$$

$$\implies \frac{k}{L^2}\frac{d^2\theta}{dx^{*2}} - h\theta = 0$$

$$\implies \frac{d^2\theta}{dx^{*2}} - \frac{L^2 h}{k}\theta = 0$$

replace $\frac{L^2h}{k}$ by m^2

$$\implies \frac{d^2\theta}{dx^{*2}} - m^2\theta = 0$$

which is the desired equation.

7 Appendix

1. Code of Main file (for varying h):

```
%Main file(varies h for mesh independence)
h(1) = 0.5;
[T,~,~,~] = central_project_solver(0.5);
itermax = 3;
tol = 10^-1;
for i = 2:itermax
    T_old = T;
    h(i) = h(i-1)/2;
    [T,~,~,c] = project_solver(h(i));
    for j = 1:length(T_old)
        disp(j)
        e(j) = abs((T(2*j-1)-T_old(j))/T(2*j-1));
    end
    cost(i) = c;
    err(i) = max(e);
    if err(i) < tol
        disp(i)
        break;
        end
end
```

2. Code of Jacobi Method:

```
function [x,k] = jacobi_om(A,b)
n = length(b);
x = rand(n,1);
xguess = zeros(n,1);
ops = 0;
for i = 1:2000
    for j = 1:n
        x(j) = (b(j)-(A(j,[1:j-1,j+1:n])*xguess([1:j-1,j+1:n])))/A(j,j);
        ops = ops+1+2*(n-1);
    end
    error(i) =  max(abs((x-xguess)./x));
        if error(i) < 1e-6
            break
        end
        xguess = x;
end
k = ops;
end
```

3. Code for Backward difference formulation:

```
function [Tgs,Tj,Ts,Tsv,A,b,c] = project_solver(h)
n = (1/h)+1;
msq = 2.1324^2;
Bi = 0.04737;
% p,q,r are numerical factors
p = -1*(2 + (h^2)*(msq));
q = 2 - (h^2)*msq;
r = 3 + 2*Bi;
%Generating A & b
A = zeros(n);
A(1,1) = 1;
for i = 2:3
    A(i,i-1) = 1;
    A(i,i) = p;
    A(i,i+1) = 1;
end
```

```
for i = 4:n-1
    A(i,i) = q;
    A(i,i-1) = -5;
    A(i,i-2) = 4;
    A(i,i-3) = -1;
end
A(n,n-2) = 1;
A(n,n-1) = -4;
A(n,n) = r;
b = zeros(n,1);
b(1) = 1;
cond_A = max(max(A))*max(max(inv(A)));

%solving the system of equations
% (Kindly take care of the naming)
Tgs = gausssiedel(A,b);
[Tj,c] = jacobi(A,b);
Ts = SOR(A,b);
Tsv = Sor_variant(A,b);
Ts = Tsv;
end
```

4. Code for Central Difference Formulation:

```
function [Tj, Tgs, A,b] = central_project_solver(h)
n = (1/h)+1;
msq = 2.1324^2;
Bi = 0.04737;
% p,q,r are numerical factors
p = -1*(2 + (h^2)*(msq));
r = 3 + 2*h*Bi;
%Generating A & b
A = zeros(n);
A(1,1) = 1;
for i = 2:n-1
    A(i,i-1) = 1;
    A(i,i) = p;
    A(i,i+1) = 1;
end
A(n,n-2) = 1;
A(n,n-1) = -4;
A(n,n) = r;
b = zeros(n,1);
b(1) = 1;
%solving the system of equations
% (Kindly take care of the naming)
```

```
Tgs = Gauss_siedal(A,b);
Tj = jacobi_om(A,b);
Ts = SOR(A,b);
Tsv = Sor_variant(A,b);
end
```

5. Code of Gauss Seidel algorithm:

```
function [x_new] = gaussseidel(A, B, x0)
format short
n = length(A);
%gauss seidel
tol = 10^(-6);
itermax = 100000;
iter = 0;
err = 0;
x_guess = x0;
x_new = x_guess;
while iter<itermax
    for i = 1:n
        sum1=0;
        sum2=0;
        for j = 1:i-1
            sum1 = A(i,j)*x_new(j);
        end
        for j = i+1:n
            sum2 = A(i,j)*x_guess(j);
        end
        x_new(i) = (B(i) - sum1 - sum2)/A(i,i);
    end
    iter = iter +1;
    err1 = sqrt(sum((x_new - x_guess).^2));
    err2 = sqrt(sum(x_new.^2));
    err = abs(err1/err2);
    if err<=tol
        break
    end
    %update x_guess
    x_guess = x_new;
end
end
```

6. Code of SOR variant algorithm:

```
function x = Sor_variant(A,b)
```

```
n = length(b);
x = rand(n,1);
xguess = zeros(n,1);
w = 1.5;
%ops = 0;
y = rand(n,1);
for i = 1:10000
    for j = 1:n
        x(j) = (b(j)/A(j,j))-(A(j,[1:j-1,j+1:n])*xguess([1:j-1,j+1:n]))/A(j,j);
        xguess(j)=x(j);
        %ops = ops + 2*(n-1)+1;
    end
    error(i) = max(abs((y-x)./x));
    if error(i) < 10^(-9)
       break
    end
    x = w*x+(1-w)*y;
    y = x;
end
%disp(ops)
```

7. Code for finding spectral radii:

```
        %% checking convergence theorem
function[rho_j,rho_g,rho_sor]=sumit_con(A,w) %rho_j-for jacobi,A=matrix ,w-sor
method
%% initialising variables
[m,n]=size(A);
L=zeros(m,n);  %% lower traingular matrix
U=zeros(m,n);  %% upper triangular matrix
D=zeros(m,n);  %% diagonal matrix
%% creating L,U,D
for i=1:m
    for j=1:n
        if(j<i)
            L(i,j)=A(i,j);
        elseif(i==j)
            D(i,j)=A(i,j);
        else
            U(i,j)=A(i,j);
        end
    end
end
%% jacobian
```

```
inv_s_T=-inv(D)*(L+U);
rho_j=abs(eigs(inv_s_T,1));
%% gauss_siedel
inv_s_T=-inv(L+D)*U;
rho_g=abs(eigs(inv_s_T,1));
%% sor
inv_s_T=(D+w*L)\((1-w)*D-(w*U));
rho_sor=abs(eigs(inv_s_T,1));
disp(rho_j)
disp(rho_g)
disp(rho_sor)
end
```

8. Helper function for matrix generation:

```
function [C,b] = A_b_generator(h)
n = (1/h)+1;
msq = 2.1324^2;
Bi = 0.04737;
% p,q,r are numerical factors
p = -1*(2 + (h^2)*(msq));
r = 3 + 2*Bi;
%Generating A & b
A = zeros(n);
A(1,1) = 1;
for i = 2:n-1
    A(i,i-1) = 1;
    A(i,i) = p;
    A(i,i+1) = 1;
end
A(n,n-2) = 1;
A(n,n-1) = -4;
A(n,n) = r;
b = zeros(n,1);
b(1) = 1;
C=A;
end
```

9. Stability:

```
%% stability checker
%% defining variables
```

```
h=zeros(10);%% stepsize
s=1;
for i=1:10
    h(i)=s/power(2,i);
end
[~,m]=size(h);
rho_j=zeros(1,m);
rho_g=zeros(1,m);
rho_sor=zeros(1,m);
w=1.2;
for j=1:m
    [A,~]=A_b_generator(h(j));
    [rho_j(j),rho_g(j),rho_sor(j)] = sumit_con(A,w);
    prompt=":jacobi not converging";
    if rho_j(j)>=1
        prompt2="for h value:";
        disp(prompt2);
        disp(h(j));
        disp(prompt);
    end
     prompt="gauss siedel not converging";
    if rho_g(j)>=1
        prompt2="for h value:";
        disp(prompt2);
        disp(h(j));
        disp(prompt);
    end
     prompt="sor not converging";
    if rho_sor(j)>=1
        prompt2="for h value:";
        disp(prompt2);
        disp(h(j));
        disp(prompt);
end end
 %% ploting graphs
 figure(1)
 plot(rho_j,"b--o");
 title("jacobi method");
 xlabel("1/(log2*h)");
 ylabel("spectral radius");
 ylim([0 1.5]);
 figure(2)
 plot(rho_g,"g--o");
 title("gauss-siedel method");
 xlabel("1/(log2*h)");
 ylabel("spectral radius");
 ylim([0 1.5]);
 figure(3)
 plot(rho_sor,"r--o");
 title("sor method method");
 xlabel("1/(log2*h)");
 ylabel("spectral radius");
 ylim([0 1.5]);
```

10. Code for SOR algorithm:

```
function [x,c,k]= sor(a,b)
n=length(b);
w=(1:0.1:2);
h=length(w);
temp=zeros(n,1);
x=zeros(n,1);
y=zeros(h,1);
c_add=0;
c_sub=0;
c_div=0;
c_mult=0;
D=zeros(n);
for i=1:n
    D(i,i)=a(i,i);
end
L=zeros(n);
for i=2:n
    for j=1:i-1
        L(i,j)=a(i,j);
    end
end
U=zeros(n);
for i=2:n
    for j=i+1:n
        U(i,j)=a(j,i);
    end
end
for u=1:length(w)
    C=(inv(D+w(u)*L))*((1-w(u))*D-w(u)*U);
    e=eig(C);
    t=max(abs(e));
    y(u)=sqrt(t);
end
[~,idx]=min(y);
```

```
t=w(idx);
k=0;
while 1>0
    for i=1:n
        s=0;
        p=0;
        for j=1:i-1  %%%%Loop through each row and multiply
            if i~=j  %%%%element with x(i) of latest iteration
                s=s+a(i,j)*temp(j);
                c_add=c_add + 1;
                c_mult=c_mult+1;
            end
        end
        for j=i+1:n  %%%%Loop through each row and multiply
            if i~=n  %%%%element with x(i) of previous iteration
                p=p+a(i,j)*x(j);
                c_add=c_add + 1;
                c_mult=c_mult+1;
            end
        end
        temp(i)=(b(i)-s-p)/a(i,i);%%%Updating value
        temp(i)= x(i) + t*(temp(i) - x(i));
        c_sub=c_sub+1+2;
        c_div = c_div + 1;
        c_mult = c_mult + 1;
    end
    if max(abs(temp-x)./abs(temp))<10^-4 %%%%Convergence
        x=temp;
        k=k+1;
        break
    end
    x=temp;
    k=k+1;
end
c=c_add+c_sub+c_div+c_mult;
% % [~,idx]=min(y);
% % x=x(:,idx);
% % x
end
```

11. Code for solving by inbuilt solvers:

```
%break y'' - ky= 0 into y' = y(2) so that y'' = y'(2)
```

```
%boundary conditions
%y(x=0) = 1 i.e. y(1)(x=1) = 1
%y'(x=1) - Biy(x=1)=0 i.e. y(2)(x=1) - Bi*y(1)(x=1) = 0
h = 0.05;
Bi =  0.04737;
k = 2.1324^2;

x = linspace(0,1,1/h);
guess = [0,0];
init = bvpinit(x, guess);

sol1 = bvp4c(@bvp_rhs, @bvp_bc, init);
sol2 = bvp5c(@bvp_rhs, @bvp_bc, init);

%plot(sol.x, sol.y, '-o')
BS1=deval(sol1,x);
BS2=deval(sol2,x);
subplot(2,1,1)
plot(x,BS1(1,:))
legend('bvp4c');
xlabel('distance');
ylabel('temperature');
title('Temperature Profile');

subplot(2,1,2)
plot(x, BS2(1,:))
legend('bvp5c');
xlabel('distance');
ylabel('temperature');
title('Temperature Profile');

function rhs=bvp_rhs(x,y)
rhs=[y(2);   2.1324^2*y(1)];
end

function bc=bvp_bc(yL,yR)
bc=[yL(1)-1; yR(2)- 0.04737*yR(1)];
end
```

8 Contributions

Om Mihani: Comparison of convergence and computational costs, contributed in report writing, code for SOR variant, code for mesh independence, code for central and backward difference formulation
Rohit Suryavanshi: Derived central, backward and forward difference conditions, contributed in report writing, code for SOR, graphs for temperature profiles, computational costs and convergence
Shreya Makkar: Derived central, backward and forward difference conditions, comparison of convergence and computational costs, contributed in report writing, calculating $\rho(S^{-1}T)$, code for inbuilt solvers
Sumit Kumar: Physical explanation of problem, contributed in report writing, code for jacobi and explananation, code for $\rho(S^{-1}T)$ and check convergence theorems, stability with respect to h value, explanation of inbuilt solvers