CAPE Lab

Assignment 4

Name: Aditya Deep
Roll No: 22CH10003
Group No: 5

Date: 7 Feb 2025

Assignment 4

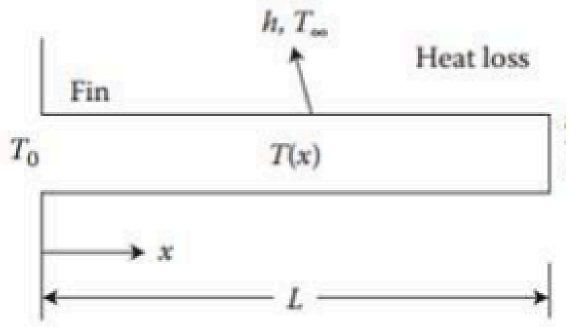
Objective: Numerical solution of Ordinary Differential Equation: Boundary Value Problem

Consider the steady-state heat transfer in a fin of uniform cross-section as shown below. The thermophysical properties of the fin material are constant. Find the temperature along the length of the fin $T(x)$ using

(a) Finite Difference Method (write your own code)

(b) Shooting Method (write your own code)

(c) MATLAB function `bvp4c`



The following BVP represents the governing equation for the fin.

$$\frac{d^2T}{dx^2} - \beta(T - T_\infty) = 0, \; T(x=0) = T_0, T(x=L) = T_L$$

Given: $T_0 = 100$, $T_L = 30$, $T_\infty = 30$, $L = 2$, $\beta = 1.5$ (in appropriate units)

1.Finite Difference Method

Algorithm:

1. **Discretization**:
    - Divide the interval $[0, L]$ into N equal steps with step size $\Delta x = \frac{L}{N}$.
    - Approximate derivatives using central difference formulas:
    $$\frac{d^2T}{dx^2} \approx \frac{T_{i+1} - 2T_i + T_{i-1}}{\Delta x^2}$$
    - Substituting this into the equation gives:
    $$\frac{T_{i+1} - 2T_i + T_{i-1}}{\Delta x^2} - \beta(T_i - T_\infty) = 0$$
2. **Set Up a Linear System**:
    - Rewrite as a system of equations for each grid point i (excluding boundary points).
    - Use **matrix methods (Ax = b)** to solve for T.
3. **Solve the System**:
    - Use MATLAB's `\` operator or `linsolve()` to find the temperature distribution.

MATLAB Code

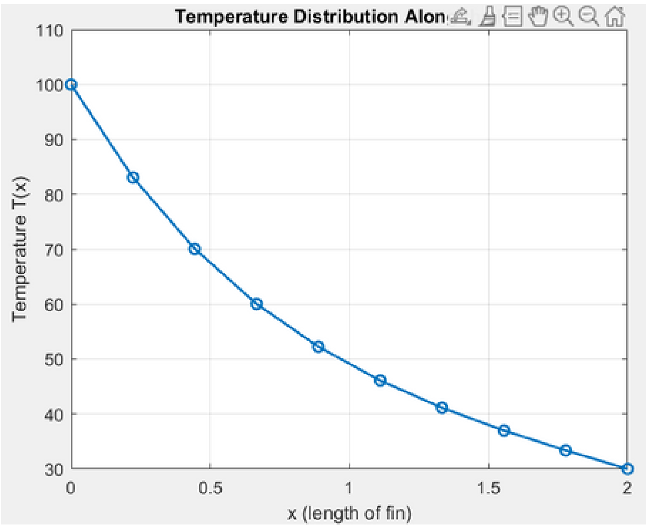
```
clc; clear; close all;

% Given parameters
T0 = 100;         % Temperature at x = 0
TL = 30;          % Temperature at x = L
T_inf = 30;       % Ambient temperature
L = 2;            % Length of the fin
beta = 1.5;       % Coefficient

N = 10;           % Number of grid points
dx = L / (N - 1); % Step size
x = linspace(0, L, N); % Discretized x domain
```

```
% Coefficients for finite difference scheme
A = zeros(N, N);
b = zeros(N, 1);

% Boundary conditions
A(1,1) = 1;
b(1) = T0; % T(0) = T0

A(N,N) = 1;
b(N) = TL; % T(L) = TL

% Finite Difference Method for interior points
for i = 2:N-1
    A(i, i-1) = 1/dx^2;
    A(i, i)   = -2/dx^2 - beta;
    A(i, i+1) = 1/dx^2;
    b(i) = -beta * T_inf;
end
```

```
% Solve the linear system
T = A \ b;

disp(table(x', T, 'VariableNames', {'x', 'T'}));

% Plot the results
figure;
plot(x, T, '-o', 'LineWidth', 1.5);
xlabel('x (length of fin)');
ylabel('Temperature T(x)');
title('Temperature Distribution Along the Fin');
grid on;
```

RESULT

x	T
0	100
0.22222	83.072
0.44444	70.076
0.66667	60.048
0.88889	52.246
1.1111	46.091
1.3333	41.129
1.5556	36.991
1.7778	33.371
2	30



2.Shooting Method

Algorithm:

1. **Convert to First-Order ODEs**: Define variables:

$$y_1 = T, \quad y_2 = \frac{dT}{dx}$$

   Rewrite as a system:

$$\frac{dy_1}{dx} = y_2$$

$$\frac{dy_2}{dx} = \beta(y_1 - T_\infty)$$

2. **Guess Initial Slope $y_2(0)$ (Shooting Parameter)**:
   - We know $T(0) = T_0$, but $y_2(0)$ is unknown.
   - Solve the IVP using an ODE solver (`ode45`) with an initial guess for $y_2(0)$.
3. **Adjust $y_2(0)$ Using Root-Finding**:
   - Compute $T(L)$ from the solution.
   - Adjust $y_2(0)$ using **Newton's method** (or `fsolve`) to satisfy $T(L) = T_L$.

MATLAB Code

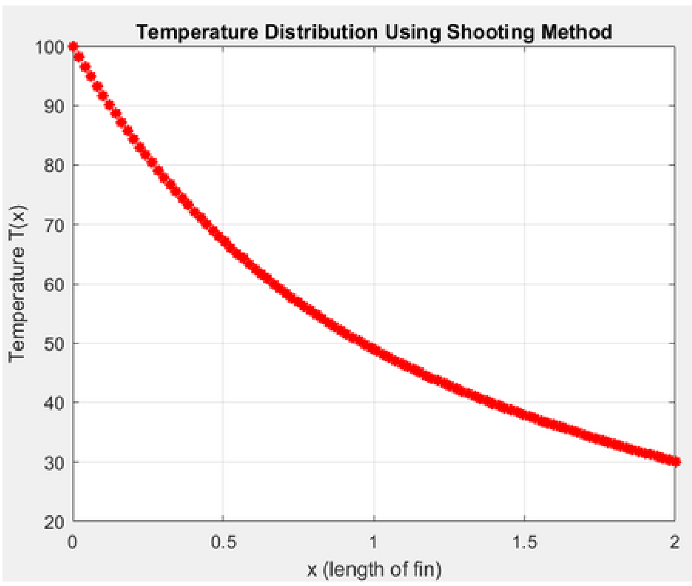
```
clc; clear; close all;

% constants
T0 = 100;
TL = 30;
T_inf = 30;
L = 2;
beta = 1.5;

% Define ODE as a system of first-order equations
odefun = @(x, y) [y(2); beta * (y(1) - T_inf)];

% Define shooting function to match boundary condition at x = L
shootingFunc = @(s) ode45(odefun, [0, L], [T0; s]); % Solve ODE
residualFunc = @(s) shootingFunc(s).y(1, end) - TL; % Residual at x = L
```

```
% Solve for correct initial slope using fzero
s_correct = fzero(residualFunc, 0);

% Solve ODE with correct slope
[x, y] = ode45(odefun, linspace(0, L, 100), [T0; s_correct]);

disp(table(x, y(:,1), 'VariableNames', {'x', 'T'}));


% Plot the results
figure;
plot(x, y(:,1), 'r*', 'LineWidth', 1.5);
xlabel('x (length of fin)');
ylabel('Temperature T(x)');
title('Temperature Distribution Using Shooting Method');
grid on;

% Display results
disp('Temperature distribution along the fin:');
disp(table(x, y(:,1), 'VariableNames', {'x', 'T'}));
```

RESULT

x	T
0	100
0.020202	98.263
0.040404	96.568
0.060606	94.914
0.080808	93.3
0.10101	91.724
0.12121	90.186
0.14141	88.685
0.16162	87.22
0.18182	85.79
0.20202	84.394
0.22222	83.031
0.24242	81.701
0.26263	80.403
0.28283	79.135
0.30303	77.897
0.32323	76.689
0.34343	75.509
0.36364	74.358
0.38384	73.233
0.40404	72.135
0.42424	71.062
0.44444	70.015
0.46465	68.992
0.48485	67.993
0.50505	67.018
0.52525	66.065
0.54545	65.134
0.56566	64.225
0.58586	63.336
0.60606	62.468
0.62626	61.62
0.64646	60.791
0.66667	59.981
0.68687	59.19
0.70707	58.416
0.72727	57.66

0.74747	56.92
0.76768	56.197
0.78788	55.491
0.80808	54.799
0.82828	54.123
0.84848	53.462
0.86869	52.815
0.88889	52.182
0.90909	51.563
0.92929	50.957
0.94949	50.363
0.9697	49.782
0.9899	49.214
1.0101	48.657
1.0303	48.111
1.0505	47.576
1.0707	47.053
1.0909	46.539
1.1111	46.036
1.1313	45.543
1.1515	45.059
1.1717	44.584
1.1919	44.119
1.2121	43.662
1.2323	43.213
1.2525	42.772
1.2727	42.34
1.2929	41.914
1.3131	41.496
1.3333	41.086
1.3535	40.681
1.3737	40.284
1.3939	39.893
1.4141	39.507
1.4343	39.128
1.4545	38.754
1.4747	38.386
1.4949	38.022
1.5152	37.664
1.5354	37.31
1.5354	37.31
1.5556	36.961
1.5758	36.616
1.596	36.275
1.6162	35.938
1.6364	35.605
1.6566	35.275
1.6768	34.948
1.697	34.624
1.7172	34.303
1.7374	33.985
1.7576	33.669
1.7778	33.356
1.798	33.044
1.8182	32.734
1.8384	32.426
1.8586	32.12
1.8788	31.815
1.899	31.51
1.9192	31.207
1.9394	30.905
1.9596	30.603
1.9798	30.301
2	30



2.MATLAB function ‘bvp4c’

Algorithm:

1. **Convert to First-Order ODEs**:
   - Same transformation as in the **Shooting Method**:

$$\frac{dy_1}{dx} = y_2, \quad \frac{dy_2}{dx} = \beta(y_1 - T_\infty)$$

2. **Define the Boundary Conditions**:
   - $y_1(0) = T_0, y_1(L) = T_L.$
3. **Provide an Initial Guess**:
   - Define a rough estimate for $y_1(x)$ and $y_2(x)$, often a linear function.
4. **Solve Using** `bvp4c`:
   - MATLAB refines the solution iteratively.

MATLAB Code

```
clc; clear; close all;

%constants
T0 = 100;
TL = 30;
T_inf = 30;
L = 2;
beta = 1.5;

% Define the system of first-order ODEs
odefun = @(x, y) [y(2); beta * (y(1) - T_inf)];

% Define boundary conditions
bcfun = @(ya, yb) [ya(1) - T0; yb(1) - TL];

% Initial mesh points
xmesh = linspace(0, L, 10);
```

```
% Initial guess for the solution as a function handle
y_guess = @(x) [T0 + (TL - T0) * x / L; 0]; % Linear guess for T(x) and 0 slope

% Generate initial solution structure using bvpinit
solinit = bvpinit(xmesh, y_guess);
```

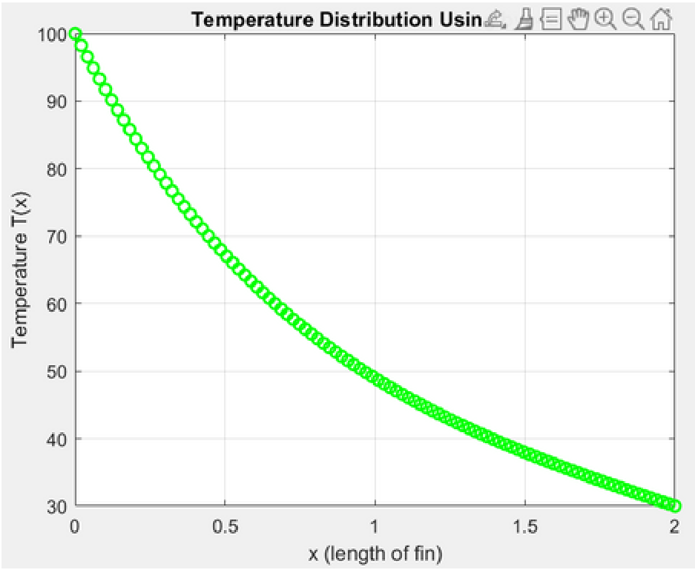
```
% Solve the boundary value problem
sol = bvp4c(odefun, bcfun, solinit);

% Extract refined solution
x = linspace(0, L, 100); % Refined x-mesh
y = deval(sol, x); % Evaluate the solution

% Plot the results
figure;
plot(x, y(1,:), 'go', 'LineWidth', 1.5);
xlabel('x (length of fin)');
ylabel('Temperature T(x)');
title('Temperature Distribution Using bvp4c');
grid on;

% Display results
disp('Temperature distribution along the fin:');
disp(table(x', y(1,:)', 'VariableNames', {'x', 'T'}));
```

RESULT

x	T
0	100
0.020202	98.263
0.040404	96.568
0.060606	94.914
0.080808	93.299
0.10101	91.723
0.12121	90.185
0.14141	88.684
0.16162	87.219
0.18182	85.79
0.20202	84.394
0.22222	83.031
0.24242	81.701
0.26263	80.402
0.28283	79.135
0.30303	77.897
0.32323	76.689

0.32323	76.689
0.34343	75.509
0.36364	74.357
0.38384	73.233
0.40404	72.135
0.42424	71.062
0.44444	70.015
0.46465	68.992
0.48485	67.993
0.50505	67.018
0.52525	66.065
0.54545	65.134
0.56566	64.224
0.58586	63.336
0.60606	62.468
0.62626	61.62
0.64646	60.791
0.66667	59.981
0.68687	59.19

0.70707	58.416
0.72727	57.66
0.74747	56.92
0.76768	56.197
0.78788	55.49
0.80808	54.799
0.82828	54.123
0.84848	53.462
0.86869	52.815
0.88889	52.182
0.90909	51.563
0.92929	50.957
0.94949	50.363
0.9697	49.782
0.9899	49.213
1.0101	48.656
1.0303	48.111
1.0505	47.576
1.0707	47.053
1.0909	46.54
1.1111	46.036
1.1313	45.543
1.1515	45.059
1.1717	44.584
1.1919	44.119
1.2121	43.662
1.2323	43.213
1.2525	42.772
1.2727	42.34
1.2929	41.914
1.3131	41.497
1.3333	41.086
1.3535	40.682
1.3737	40.284
1.3939	39.893
1.4141	39.507
1.4343	39.128
1.4545	38.754
1.4747	38.386
1.4949	38.022
1.5152	37.664
1.5354	37.31
1.5556	36.961
1.5758	36.616
1.596	36.275
1.6162	35.938
1.6364	35.605
1.6566	35.275
1.6768	34.948
1.697	34.624
1.7172	34.303
1.7374	33.985
1.7576	33.669
1.7778	33.356
1.798	33.044
1.8182	32.734
1.8384	32.426
1.8586	32.12
1.8788	31.815
1.899	31.51
1.9192	31.207
1.9394	30.905
1.9596	30.603
1.9798	30.301
2	30



Finite Difference	Shooting	bvp4c
Converts BVP into algebraic equations	Converts BVP to IVP, solves using ode45	Uses collocation and adaptive refinement
Simple and direct	Works well for simple cases	Most efficient and stable
Requires fine grid for accuracy	Needs iterative root-finding	More complex to set up
time taken = 0.0019	time taken = 0.0171	time taken = 0.0061