

# MEEN 644 - Homework 2

Logan Harbour  
February 17, 2019

## Problem statement

Consider one-dimensional heat conduction in a cylindrical copper rod of length 1.0 m long. The diameter of the rod is 0.05 m. The left end of the rod is held at 100 °C and the ambient temperature is 25 °C. Heat is transported from the surface of the rod and the right end of the rod through natural convection to the ambient. The natural convection heat transfer coefficient is 0.5 W/m<sup>2</sup> °C. Write a finite volume code to predict temperature distribution as a function of length. Use TDMA to solve a set of discretization equations. Make calculations using ITMAX: 6, 11, 21, 41, and 81 nodes. Plot your results.

## Preliminaries

### One-dimensional heat conduction

With one-dimensional heat conduction with convection and constant material properties, we have the ODE

$$\begin{cases} \frac{d^2 T}{dx^2} + \frac{h}{kd}(T - T_\infty) = 0, \\ T(0) = T_0, \\ \left. \frac{dT}{dx} \right|_{x=L} = -\frac{h}{k}(T - T_\infty), \end{cases} \quad (1)$$

where

$$\begin{array}{lll} k \equiv 400 \text{ W/m } ^\circ\text{C}, & h \equiv 0.5 \text{ W/m}^2 \text{ } ^\circ\text{C}, & d \equiv 0.05 \text{ m}, \\ L \equiv 1.0 \text{ m}, & T_0 \equiv 100 \text{ } ^\circ\text{C}, & T_\infty \equiv 25 \text{ } ^\circ\text{C}. \end{array}$$

We then make the substitutions  $\theta(x) = T(x) - T_\infty$  and  $m = 4h/kd$  to obtain the simplification

$$\begin{cases} \frac{d^2 \theta}{dx^2} + m\theta = 0, \\ \theta(0) = T_0 - T_\infty, \\ \left. \frac{d\theta}{dx} \right|_{x=L} = -\frac{h}{k}\theta. \end{cases} \quad (2)$$

## Grid generation

We discretize the region on  $x = [0, L]$  by  $N$  (also defined as ITMAX) nodes and  $N$  control volumes, as follows in Figure 1 with  $\Delta x = L/(N - 1)$ .

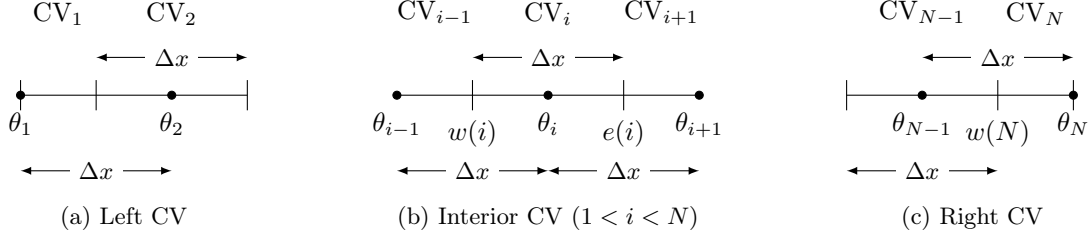


Figure 1: The control volumes defined for discretization of the problem.

## Equation discretization

### Internal control volume equation

We start with the integration over an interior control volume, as

$$\int_{CV_i} \left[ -\frac{d^2\theta}{dx^2} + m\theta \right] dx = 0, \quad 1 < i < N,$$

in which we know that the material properties are independent and we assume  $\theta_i$  to be constant over the cell for the second term to obtain

$$-\left( \frac{d\theta}{dx} \Big|_{e(i)} - \frac{d\theta}{dx} \Big|_{w(i)} \right) + m\Delta x \theta_i = 0, \quad 1 < i < N.$$

Use the two node formulation for the derivative terms and simplify as

$$\begin{aligned} & -\left( \frac{\theta_{i+1} - \theta_i}{\Delta x} - \frac{\theta_i - \theta_{i-1}}{\Delta x} \right) + m\Delta x \theta_i = 0, \quad 1 < i < N, \\ & -\frac{1}{\Delta x} \theta_{i-1} + \left( m\Delta x + \frac{2}{\Delta x} \right) \theta_i - \frac{1}{\Delta x} \theta_{i+1} = 0, \quad 1 < i < N. \end{aligned}$$

Take note that at the  $i = 2$  equation,  $\theta_1$  is known therefore we have

$$\boxed{\left( m\Delta x + \frac{2}{\Delta x} \right) \theta_2 - \frac{1}{\Delta x} \theta_3 = \frac{T_0 - T_\infty}{\Delta x}}, \quad (3)$$

$$\boxed{-\frac{1}{\Delta x} \theta_{i-1} + \left( m\Delta x + \frac{2}{\Delta x} \right) \theta_i - \frac{1}{\Delta x} \theta_{i+1} = 0, \quad 2 < i < N.} \quad (4)$$

### Right control volume equation

We start with the integration over the right control volume,  $CV_N$ , as

$$\int_{CV_N} \left[ -\frac{d^2\theta}{dx^2} + m\theta \right] dx = 0,$$

in which for the second term we will assume  $\theta_N$  to be constant over  $CV_N$  to obtain

$$-\left( \frac{d\theta}{dx} \Big|_{x=L} - \frac{d\theta}{dx} \Big|_{w(N)} \right) + \frac{1}{2} m\Delta x \theta_N = 0.$$

Use the two node formulation for the derivative term at  $w(N)$  and the right boundary condition for the derivative term at  $x = L$  m to obtain

$$\begin{aligned} \frac{h}{k}\theta_N + \frac{\theta_N - \theta_{N-1}}{\Delta x} + \frac{1}{2}m\Delta x\theta_N &= 0, \\ -\frac{1}{\Delta x}\theta_{N-1} + \left(\frac{1}{2}m\Delta x + \frac{h}{k} + \frac{1}{\Delta x}\right)\theta_N &= 0. \end{aligned} \tag{5}$$

## TDMA

The system we are solving is of the form

$$\begin{bmatrix} b_1 & c_1 & & & 0 \\ a_2 & b_2 & c_2 & & \\ & a_3 & b_3 & \ddots & \\ & & \ddots & \ddots & c_{n-1} \\ 0 & & & a_n & b_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_n \end{bmatrix}.$$

We will solve this system using the tridiagonal matrix algorithm (TDMA) given the fact that our system is positive definite. With our matrix in the form above, the algorithm follows in Algorithm 1. Note that for the algorithm below, we are solving the solution vector  $\vec{x}$  in-place with the vector  $\vec{d}$ . This algorithm is in the member function `solveTDMA()` in the `TriDiagonal` class, which stores the left hand side system in the vectors  $\vec{a}$ ,  $\vec{b}$ , and  $\vec{c}$ .

```

for  $i = 2, 3, \dots, n$  do
     $w = a_i/b_{i-1}$ ;
     $b_i = b_i - wc_{i-1}$ ;
     $d_i = d_i - wd_{i-1}$ ;
end
 $d_n = d_n/b_n$ ;
for  $i = n-1, n-2, \dots, 1$  do
     $d_i = d_i - c_i d_{i+1}$ ;
     $d_i = d_i/b_i$ ;
end
```

**Algorithm 1:** The tridiagonal matrix algorithm (TDMA).

## Results

The plotted results as requested follow below in Figure 2. In addition, the solution for  $ITMAX = 6$  is tabulated in Table 1.

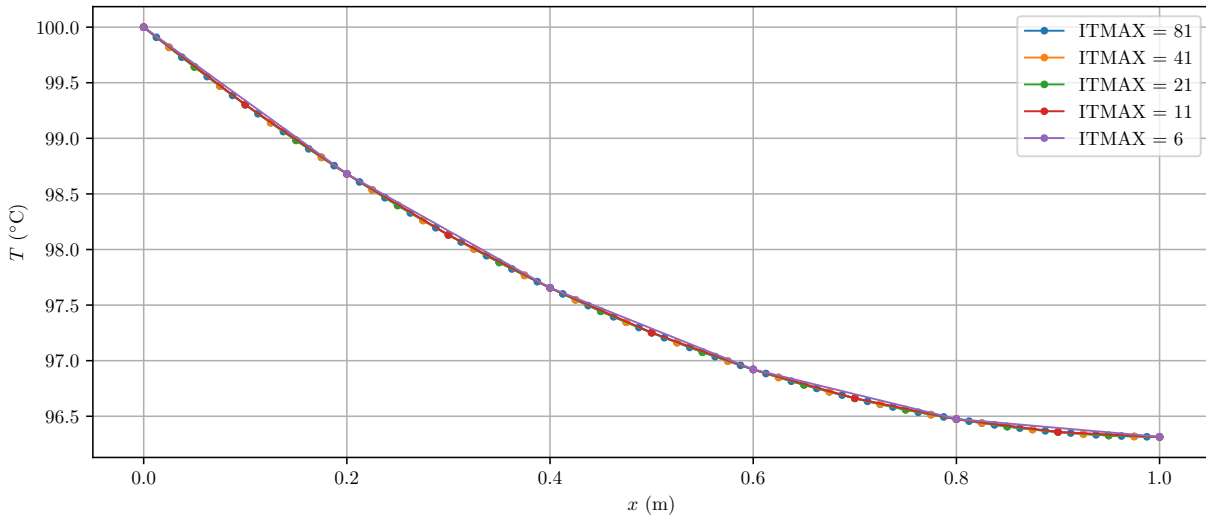


Figure 2: The plotted solution.

Table 1: The solution for  $ITMAX = 6$ .

$x$ (m)	$\theta$ (°C)	$T$ (°C)
0.0	75.00000	100.00000
0.2	73.68060	98.68060
0.4	72.65593	97.65593
0.6	71.92188	96.92188
0.8	71.47552	96.47552
1.0	71.31506	96.31506

## Code listing

### Makefile

```
all: hwk2

hwk2: TriDiagonal.h hwk2.cpp
    clang++ -std=c++14 hwk2.cpp -o $@
```

### hwk2.cpp

```
#include <fstream>
#include <iomanip>
#include <string>

#include "TriDiagonal.h"
```

```

/**
 * Saves a vector to a csv.
 */
void saveVectorCsv(const std::vector<double> x, const std::string filename) {
    std::ofstream f;
    f.open(filename);
    for (unsigned int i = 0; i < x.size(); ++i)
        f << std::setprecision(12) << x[i] << std::endl;
    f.close();
}

/**
 * Solves the 1D heat-conduction (with convection) problem with N nodes.
 */
void solveRod(unsigned int N) {
    // Initialize geometry, material properties, and constant coefficients
    double L = 1,           // [m]
           dx = L / (N - 1), // [m]
           d = 0.05,        // [m]
           k = 400.0,        // [W/m-C]
           h = 0.5,         // [W/m^2-C]
           T0 = 100.0,       // [C]
           Tinf = 25.0,      // [C]
           m = 4.0 * h / (k * d),
           a_p = m * dx + 2.0 / dx,
           a_w = 1.0 / dx,
           a_e = 1.0 / dx;

    // Initialize system A theta = b
    TriDiagonal A(N - 1);
    std::vector<double> b(N - 1);

    // Fill system
    A.addTopRow(a_p, -a_e);
    b[0] = a_w * (T0 - Tinf);
    for (unsigned int i = 1; i < N - 2; ++i)
        A.addMiddleRow(i, -a_w, a_p, -a_e);
    A.addBottomRow(-a_w, m * dx / 2.0 + h / k + a_e);

    // Solve system in place and save
    A.solveTDMA(b);
    saveVectorCsv(b, "../results/theta_" + std::to_string(N) + ".csv");
}

int main() {
    // Run each requested case
    for (unsigned int N : {6, 11, 21, 41, 81})
        solveRod(N);

    return 0;
}

```

## TriDiagonal.h

```

#define NDEBUG
#include <cassert>

```

```

#include <vector>

/**
 * Class that holds a tri-diagonal matrix and is able to perform TDMA in place
 * with a given RHS and solution vector.
 */
class TriDiagonal {
public:
    TriDiagonal(unsigned int N) : N(N), A(N), B(N), C(N - 1) {}

    // Adders for the top, interior, and bottom rows
    void addTopRow(double b, double c) {
        B[0] += b;
        C[0] += c;
    }
    void addMiddleRow(unsigned int i, double a, double b, double c) {
        assert(i < N - 1 && i != 0);
        A[i] += a;
        B[i] += b;
        C[i] += c;
    }
    void addBottomRow(double a, double b) {
        A[N - 1] += a;
        B[N - 1] += b;
    }

    // Solves the system  $Ax = d$  in place where  $d$  eventually stores the solution
    void solveTDMA(std::vector<double> &d) {
        // Forward sweep
        double tmp = 0;
        for (unsigned int i = 1; i < N; ++i) {
            tmp = A[i] / B[i - 1];
            B[i] -= tmp * C[i - 1];
            d[i] -= tmp * d[i - 1];
        }

        // Backward sweep
        d[N - 1] /= B[N - 1];
        for (unsigned int i = N - 2; i != std::numeric_limits<unsigned int>::max();
            --i) {
            d[i] -= C[i] * d[i + 1];
            d[i] /= B[i];
        }
    }

protected:
    // Matrix size ( $N \times N$ )
    unsigned int N;
    // Left/main/right diagonal storage
    std::vector<double> A, B, C;
};

```