MEEN 644 - Homework 4

Logan Harbour

March 5, 2019

Problem statement

Consider a thin copper square plate of dimensions 0.5 m × 0.5 m. The temperature of the west and south edges are maintained at 50 °C and the north edge is maintained at 100 °C. The east edge is insulated. Using finite volume method, write a program to predict the steady-state temperature solution.

(a) **(35 points)** Set the over relaxation factor α from 1.00 to 1.40 in steps of 0.05 to identify α_{opt}. Plot the number of iterations required for convergence for each α.

(b) **(15 points)** Solve the same problem using $21^2, 25^2, 31^2$, and 41^2 CVs, respectively. Plot the temperature at the center of the plate (0.25 m, 0.25 m) vs CVs.

(c) **(10 points)** Plot the steady state temperature contour in the 2D domain with the 41^2 CV solution.

Preliminaries

Two-dimensional heat conduction

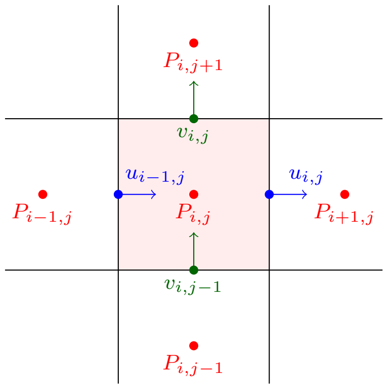
With two-dimensional heat conduction with constant material properties, insulation on the right and prescribed temperatures on all other sides, we have the PDE

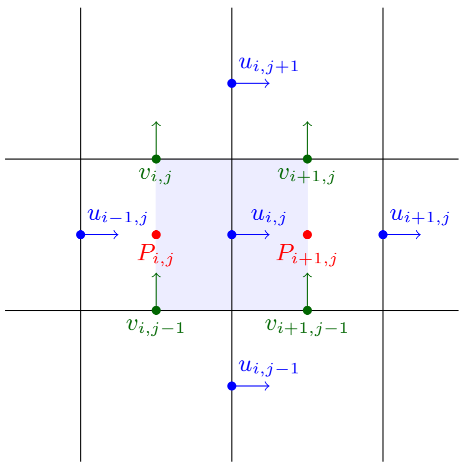
$$\begin{cases} k\frac{\partial^2 T}{\partial x^2} + k\frac{\partial^2 T}{\partial y^2} = 0\,, \\ T(x,0) = T_B\,, \\ T(0,y) = T_L\,, \\ T(0,L_y) = T_T\,, \\ -k\frac{\partial T}{\partial x}\Big|_{x=L_x} = 0\,, \end{cases} \tag{1}$$

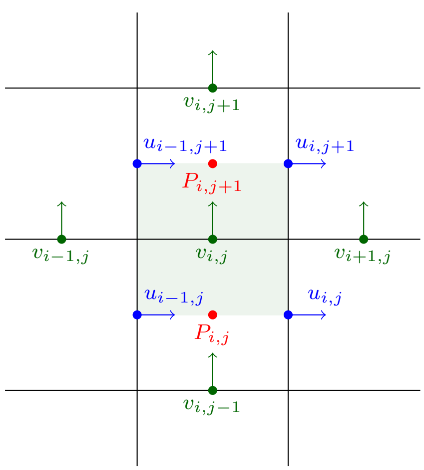
where

$$T_B \equiv 50\ ^\circ\text{C}\,, \qquad T_L \equiv 50\ ^\circ\text{C}\,, \qquad T_T \equiv 100\ ^\circ\text{C}\,.$$
$$k \equiv 386\ \text{W/m}\ ^\circ\text{C}\,, \qquad L_x \equiv 0.5\ \text{m}\,, \qquad L_y \equiv 0.5\ \text{m}\,.$$

Control volume equations







Velocity update

Define the Pechlet number on each boundary of a control volume $c_{i,j}$ as

$$P_b^{c_{i,j}} = \frac{F_b^{c_{i,j}}}{D_b^{c_{i,j}}}\,, \quad \text{where} \quad b = [n, e, s, w] \quad \text{and} \quad c = [u, v]\,, \tag{2}$$

where

$$D_n^{c_{i,j}} = \frac{\Delta x \mu}{\Delta y}\,, \tag{3a}$$
$$D_e^{c_{i,j}} = \frac{\Delta y \mu}{\Delta x}\,, \tag{3b}$$
$$D_s^{c_{i,j}} = \frac{\Delta x \mu}{\Delta y}\,, \tag{3c}$$
$$D_w^{c_{i,j}} = \frac{\Delta y \mu}{\Delta x}\,. \tag{3d}$$

u-velocity update

Integrating the x-momentum equation (with the guessed variables and neglecting the $\frac{\partial v^*}{\partial x}$ term) an internal u-velocity control volume and using the power-law scheme, we obtain

$$a_p^{u_{i,j}} u_{i,j}^* = a_n^{u_{i,j}} u_{i,j+1}^* + a_e^{u_{i,j}} u_{i+1,j}^* + a_s^{u_{i,j}} u_{i,j-1}^* + a_w^{u_{i,j}} u_{i-1,j}^* + \Delta y^{u_{i,j}} (p_{i,j}^* - p_{i+1,j}^*)\,, \tag{4}$$

where

$$a_n^{u_{i,j}} = D_n^{u_{i,j}} \max\left[0, (1 - 0.1|P_n^{u_{i,j}}|)^5\right] + \max\left[-F_n^{u_{i,j}}, 0\right]\,, \tag{5a}$$
$$a_e^{u_{i,j}} = D_e^{u_{i,j}} \max\left[0, (1 - 0.1|P_e^{u_{i,j}}|)^5\right] + \max\left[-F_e^{u_{i,j}}, 0\right]\,, \tag{5b}$$
$$a_s^{u_{i,j}} = D_s^{u_{i,j}} \max\left[0, (1 - 0.1|P_s^{u_{i,j}}|)^5\right] + \max\left[F_s^{u_{i,j}}, 0\right]\,, \tag{5c}$$
$$a_w^{u_{i,j}} = D_w^{u_{i,j}} \max\left[0, (1 - 0.1|P_w^{u_{i,j}}|)^5\right] + \max\left[F_w^{u_{i,j}}, 0\right]\,, \tag{5d}$$
$$a_p^{u_{i,j}} = a_n^{u_{i,j}} + a_e^{u_{i,j}} + a_s^{u_{i,j}} + a_w^{u_{i,j}}\,, \tag{5e}$$

and

$$F_n^{u_{i,j}} = \frac{1}{2}\rho \Delta x^{u_{i,j}} \left(v_{i,j} + v_{i+1,j}\right)\,, \tag{6a}$$
$$F_e^{u_{i,j}} = \frac{1}{2}\rho \Delta y^{u_{i,j}} \left(u_{i,j} + u_{i+1,j}\right)\,, \tag{6b}$$
$$F_s^{u_{i,j}} = \frac{1}{2}\rho \Delta x^{u_{i,j}} \left(v_{i,j-1} + v_{i+1,j-1}\right)\,, \tag{6c}$$
$$F_w^{u_{i,j}} = \frac{1}{2}\rho \Delta y^{u_{i,j}} \left(u_{i-1,j} + u_{i,j}\right)\,. \tag{6d}$$

There exist the following manipulations for the boundary control volumes:

- On the left and right boundaries:

$$D_n^{u_{i,j}} = \frac{3\Delta x \mu}{2\Delta y}\,, \quad i = 1, M_x^u - 1\,, \quad 0 < j < M_y^u\,, \tag{7}$$
$$D_s^{u_{i,j}} = \frac{3\Delta x \mu}{2\Delta y}\,, \quad i = 1, M_x^u - 1\,, \quad 0 < j < M_y^u\,, \tag{8}$$

- On the right boundary:

$$F_n^{u_{M_x^u-1,j}} = \frac{\rho\Delta x}{4}\left(2v_{M_y^v-2,j} + 3v_{M_y^v-1,j} + v_{M_y^v,j}\right) , \quad 0 < j < M_y^u , \tag{9}$$

$$F_s^{u_{M_x^u-1,j}} = \frac{\rho\Delta x}{4}\left(2v_{M_y^v-2,j-1} + 3v_{M_y^v-1,j-1} + v_{M_y^v,j-1}\right) , \quad 0 < j < M_y^u , \tag{10}$$

$$F_e^{u_{M_x^u-1,1}} = \frac{\rho\Delta y}{2}\left(u_{M_x^u,0} + u_{M_x^u,1makmak}\right) , \tag{11}$$

$$F_e^{u_{M_x^u-1,M_y^u-1}} = \frac{\rho\Delta y}{2}\left(u_{M_x^u,M_y^u} + u_{M_x^u,M_y^u-1}\right) , \tag{12}$$

$$F_e^{u_{M_x^u-1,j}} = \rho\Delta y u_{M_x^u,j} , \quad 1 < j < M_y^u - 1 , \tag{13}$$

- On the left boundary:

$$F_n^{u_{1,j}} = \frac{\rho\Delta x}{4}\left(v_{0,j} + 2v_{1,j} + 3v_{2,j}\right) , \quad 0 < j < M_y^u , \tag{14}$$

$$F_s^{u_{1,j}} = \frac{\rho\Delta x}{4}\left(v_{0,j-1} + 2v_{1,j-1} + 3v_{2,j-1}\right) , \quad 0 < j < M_y^u , \tag{15}$$

$$F_w^{u_{1,1}} = \frac{\rho\Delta y}{2}\left(u_{0,0} + u_{0,1}\right) , \tag{16}$$

$$F_w^{u_{1,M_y^u-1}} = \frac{\rho\Delta y}{2}\left(u_{0,M_y^u-1} + u_{0,M_y^u}\right) , \tag{17}$$

$$F_w^{u_{1,j}} = \rho\Delta y u_{0,j} , \quad 1 < M_y^u - 1 , \tag{18}$$

v-velocity update

Integrating the x-momentum equation (with the guessed variables and neglecting the $\frac{\partial v u^*}{\partial y}$ term) an internal v-velocity control volume and using the power-law scheme, we obtain

$$a_p^{v_{i,j}} v_{i,j}^* = a_n^{v_{i,j}} v_{i,j+1}^* + a_e^{v_{i,j}} v_{i+1,j}^* + a_s^{v_{i,j}} v_{i,j-1}^* + a_w^{v_{i,j}} v_{i-1,j}^* + \Delta x^{u_{i,j}} (p_{i,j}^* - p_{i,j+1}^*) , \tag{19}$$

where

$$a_n^{v_{i,j}} = D_n^{v_{i,j}} \max\left[0, (1 - 0.1|P_n^{v_{i,j}}|)^5\right] + \max\left[-F_n^{v_{i,j}}, 0\right] , \tag{20a}$$

$$a_e^{v_{i,j}} = D_e^{v_{i,j}} \max\left[0, (1 - 0.1|P_e^{v_{i,j}}|)^5\right] + \max\left[-F_e^{v_{i,j}}, 0\right] , \tag{20b}$$

$$a_s^{v_{i,j}} = D_s^{v_{i,j}} \max\left[0, (1 - 0.1|P_s^{v_{i,j}}|)^5\right] + \max\left[F_s^{v_{i,j}}, 0\right] , \tag{20c}$$

$$a_w^{v_{i,j}} = D_w^{v_{i,j}} \max\left[0, (1 - 0.1|P_w^{v_{i,j}}|)^5\right] + \max\left[F_w^{v_{i,j}}, 0\right] , \tag{20d}$$

$$a_p^{v_{i,j}} = a_n^{v_{i,j}} + a_e^{v_{i,j}} + a_s^{v_{i,j}} + a_w^{v_{i,j}} , \tag{20e}$$

and

$$F_n^{v_{i,j}} = \frac{1}{2}\rho\Delta x^{v_{i,j}} \left(v_{i,j+1} + v_{i,j}\right) , \tag{21a}$$

$$F_e^{v_{i,j}} = \frac{1}{2}\rho\Delta y^{v_{i,j}} \left(u_{i,j} + u_{i,j+1}\right) , \tag{21b}$$

$$F_s^{v_{i,j}} = \frac{1}{2}\rho\Delta x^{v_{i,j}} \left(v_{i,j-1} + v_{i,j}\right) , \tag{21c}$$

$$F_w^{v_{i,j}} = \frac{1}{2}\rho\Delta y^{v_{i,j}} \left(u_{i-1,j} + u_{i-1,j+1}\right) . \tag{21d}$$

There exist the following manipulations for the boundary control volumes:

- On the top and bottom boundaries:

$$D_e^{u_{i,j}} = \frac{3\Delta y\mu}{2\Delta x}\,, \quad 0 < j < M_x^u\,, \quad j = 1, M_y^u - 1\,, \tag{22}$$

$$D_w^{u_{i,j}} = \frac{3\Delta y\mu}{2\Delta x}\,, \quad 0 < j < M_x^u\,, \quad j = 1, M_y^u - 1\,, \tag{23}$$

- On the top boundary:

$$F_w^{v_{i,M_y^v-1}} = \frac{\rho\Delta y}{4}\left(u_{i-1,M_y^u} + 2u_{i-1,M_y^u-1} + 3u_{i-1,M_y^u-2}\right)\,, \quad 0 < i < M_x^v\,, \tag{24}$$

$$F_e^{v_{i,M_y^v-1}} = \frac{\rho\Delta y}{4}\left(u_{i,M_y^u} + 2u_{i,M_y^u-1} + 3u_{i,M_y^u-2}\right)\,, \quad 0 < i < M_x^v\,, \tag{25}$$

$$F_n^{v_{0,M_y^v-1}} = \frac{\rho\Delta x}{2}\left(v_{0,M_y^v} + u_{1,M_y^v}\right)\,, \tag{26}$$

$$F_n^{v_{M_x^v-1,M_y^v-1}} = \frac{\rho\Delta x}{2}\left(v_{M_x^v-1,M_y^v} + v_{M_x^v,M_y^v}\right)\,, \tag{27}$$

$$F_n^{v_{i,M_y^v-1}} = \rho\Delta x v_{i,M_y^v}\,, \quad 1 < i < M_x^v - 1\,, \tag{28}$$

- On the bottom boundary:

$$F_w^{v_{i,1}} = \frac{\rho\Delta y}{4}\left(u_{i-1,0} + 2u_{i-1,1} + 3u_{i-1,2}\right)\,, \quad 0 < i < M_x^v\,, \tag{29}$$

$$F_e^{v_{i,1}} = \frac{\rho\Delta y}{4}\left(u_{i,0} + 2u_{i,1} + 3u_{i,2}\right)\,, \quad 0 < i < M_x^v\,, \tag{30}$$

$$F_s^{v_{0,1}} = \frac{\rho\Delta x}{2}\left(v_{0,0} + u_{1,0}\right)\,, \tag{31}$$

$$F_s^{v_{M_x^v-1,1}} = \frac{\rho\Delta x}{2}\left(v_{M_x^v-1,0} + v_{M_x^v,0}\right)\,, \tag{32}$$

$$F_s^{v_{i,1}} = \rho\Delta x v_{i,0}\,, \quad 1 < i < M_x^v - 1\,, \tag{33}$$

Solving methodology

Results

Table 1: The solution for ITMAX = 6.

	1	2	3	4	5	6
1	0.00000E0	0.00000E0	0.00000E0	0.00000E0	0.00000E0	0.00000E0
2	0.00000E0	−3.38119E−4	−2.16978E−4	2.16978E−4	3.38119E−4	0.00000E0
3	0.00000E0	−2.54416E−4	−1.25749E−4	1.25749E−4	2.54416E−4	0.00000E0
4	0.00000E0	−1.03429E−4	−4.49112E−5	4.49109E−5	1.03429E−4	0.00000E0
5	0.00000E0	1.46223E−4	5.66644E−5	−5.66643E−5	−1.46223E−4	0.00000E0
6	0.00000E0	5.49741E−4	3.30974E−4	−3.30973E−4	−5.49741E−4	0.00000E0
7	0.00000E0	0.00000E0	0.00000E0	0.00000E0	0.00000E0	0.00000E0

Table 2: The solution for ITMAX = 6.

	1	2	3	4	5	6	7
1	4.01483E−3	0.00000E0	0.00000E0	0.00000E0	0.00000E0	0.00000E0	4.01483E−3
2	4.01483E−3	3.38119E−4	−1.21141E−4	−4.33956E−4	−1.21141E−4	3.38119E−4	4.01483E−3
3	4.01483E−3	5.92535E−4	−2.49808E−4	−6.85454E−4	−2.49809E−4	5.92535E−4	4.01483E−3
4	4.01483E−3	6.95964E−4	−3.08325E−4	−7.75276E−4	−3.08326E−4	6.95964E−4	4.01483E−3
5	4.01483E−3	5.49741E−4	−2.18767E−4	−6.61947E−4	−2.18768E−4	5.49741E−4	4.01483E−3
6	4.01483E−3	0.00000E0	0.00000E0	0.00000E0	0.00000E0	0.00000E0	4.01483E−3

Table 3: The solution for ITMAX = 6.

	1	2	3	4	5	6	7
1	−4.02266E−4	−4.02266E−4	−1.26481E−5	5.60848E−5	−1.26481E−5	−4.02266E−4	−4.02266E−4
2	−4.02266E−4	−4.02266E−4	−1.26481E−5	5.60848E−5	−1.26481E−5	−4.02266E−4	−4.02266E−4
3	−4.22010E−5	−4.22010E−5	−1.04441E−4	−5.47805E−5	−1.04441E−4	−4.22011E−5	−4.22011E−5
4	2.74130E−5	2.74130E−5	−1.61202E−4	−1.03080E−4	−1.61202E−4	2.74129E−5	2.74129E−5
5	2.52977E−4	2.52977E−4	−9.76194E−5	2.78124E−5	−9.76191E−5	2.52977E−4	2.52977E−4
6	8.36831E−4	8.36831E−4	2.82597E−4	3.12442E−4	2.82598E−4	8.36831E−4	8.36831E−4
7	8.36831E−4	8.36831E−4	2.82597E−4	3.12442E−4	2.82598E−4	8.36831E−4	8.36831E−4

Code listing

For the implementation, we have the following files:

- `Makefile` – Allows for compiling the c++ project with `make`.
- `hwk4.cpp` – Contains the `main()` function that is required by C that runs the cases requested in this problem set.
- `Flow2D.h` / `Flow2D.cpp` – Contains the `Flow2D` class which is the solver for the 2D problem required in this homework.
- `Matrix.h` – Contains the `Matrix` class which provides storage for a matrix with various standard matrix operations.
- `TriDiagonal.h` – Contains the `TriDiagonal` class which provides storage for a tri-diagonal matrix including the TDMA solver found in the member function `solveTDMA()`.
- `plots.py` - Produces the plots in this report.

Makefile

```
src = $(wildcard *.cpp)
obj = $(src:.cpp=.o)
CXXFLAGS = -std=c++14
CCFLAGS = $(CXXFLAGS)

hwk-opt: $(obj)
        clang++ -o $@ $^

.PHONY: clean
clean:
        rm -f $(obj) hwk-opt
```

hwk4.cpp

```
#include "Problem.h"

using namespace Flow2D;

int
main()
{
  // Problem wide constants
  double L = 0.1;
  double Re = 400;
```

```
  double rho = 998.3;
  double mu = 1.002e-3;
  double bc_val = Re * mu / (rho * 0.1);

  // Standard inputs
  InputArguments input;
  input.Lx = L;
  input.Ly = L;
  input.mu = mu;
  input.rho = rho;
  input.u_ref = bc_val;
  input.L_ref = L;

  // Problem 1: check symmetry
  input.u_bc = BoundaryCondition(0, 0, 0, 0);
  input.v_bc = BoundaryCondition(0, bc_val, 0, bc_val);
  std::cout << "Problem 1, check symmetry" << std::endl;
  {
    Problem problem(5, 5, input);
    problem.run();
    problem.print(Variables::u, "u =");
    problem.print(Variables::v, "v =");
    problem.print(Variables::p, "p =");
    std::cout << std::endl;
  }

  // Problem 2: change to top plate BC
  input.u_bc = BoundaryCondition(bc_val, 0, 0, 0);
  input.v_bc = BoundaryCondition(0, 0, 0, 0);
  std::cout << "Problem 2, top plate BC" << std::endl;
  for (unsigned int N : {8, 16, 32, 64, 128})
  {
    std::cout << "N = " << N << "x" << N << " - ";
    Problem problem(N, N, input);
    problem.run();
    problem.save(Variables::u, "results/p2/" + to_string(N) + "_u.csv");
    problem.save(Variables::v, "results/p2/" + to_string(N) + "_v.csv");
  }
}
```

Flow2D.h

```
#include "Problem.h"

using namespace Flow2D;

int
main()
{
  // Problem wide constants
  double L = 0.1;
  double Re = 400;
  double rho = 998.3;
  double mu = 1.002e-3;
  double bc_val = Re * mu / (rho * 0.1);

  // Standard inputs
  InputArguments input;
  input.Lx = L;
  input.Ly = L;
  input.mu = mu;
  input.rho = rho;
  input.u_ref = bc_val;
  input.L_ref = L;
```

```
  // Problem 1: check symmetry
  input.u_bc = BoundaryCondition(0, 0, 0, 0);
  input.v_bc = BoundaryCondition(0, bc_val, 0, bc_val);
  std::cout << "Problem 1, check symmetry" << std::endl;
  {
    Problem problem(5, 5, input);
    problem.run();
    problem.print(Variables::u, "u =");
    problem.print(Variables::v, "v =");
    problem.print(Variables::p, "p =");
    std::cout << std::endl;
  }

  // Problem 2: change to top plate BC
  input.u_bc = BoundaryCondition(bc_val, 0, 0, 0);
  input.v_bc = BoundaryCondition(0, 0, 0, 0);
  std::cout << "Problem 2, top plate BC" << std::endl;
  for (unsigned int N : {8, 16, 32, 64, 128})
  {
    std::cout << "N = " << N << "x" << N << " - ";
    Problem problem(N, N, input);
    problem.run();
    problem.save(Variables::u, "results/p2/" + to_string(N) + "_u.csv");
    problem.save(Variables::v, "results/p2/" + to_string(N) + "_v.csv");
  }
}
```

Flow2D.cpp

```
#include "Problem.h"

using namespace Flow2D;

int
main()
{
  // Problem wide constants
  double L = 0.1;
  double Re = 400;
  double rho = 998.3;
  double mu = 1.002e-3;
  double bc_val = Re * mu / (rho * 0.1);

  // Standard inputs
  InputArguments input;
  input.Lx = L;
  input.Ly = L;
  input.mu = mu;
  input.rho = rho;
  input.u_ref = bc_val;
  input.L_ref = L;

  // Problem 1: check symmetry
  input.u_bc = BoundaryCondition(0, 0, 0, 0);
  input.v_bc = BoundaryCondition(0, bc_val, 0, bc_val);
  std::cout << "Problem 1, check symmetry" << std::endl;
  {
    Problem problem(5, 5, input);
    problem.run();
    problem.print(Variables::u, "u =");
    problem.print(Variables::v, "v =");
    problem.print(Variables::p, "p =");
    std::cout << std::endl;
  }
```

```
  // Problem 2: change to top plate BC
  input.u_bc = BoundaryCondition(bc_val, 0, 0, 0);
  input.v_bc = BoundaryCondition(0, 0, 0, 0);
  std::cout << "Problem 2, top plate BC" << std::endl;
  for (unsigned int N : {8, 16, 32, 64, 128})
  {
    std::cout << "N = " << N << "x" << N << " - ";
    Problem problem(N, N, input);
    problem.run();
    problem.save(Variables::u, "results/p2/" + to_string(N) + "_u.csv");
    problem.save(Variables::v, "results/p2/" + to_string(N) + "_v.csv");
  }
}
```

Matrix.h

```
#ifndef MATRIX_H
#define MATRIX_H

// #define NDEBUG
#include <cassert>
#include <fstream>
#include <vector>

using namespace std;

/**
 * Class that holds a N x M matrix with common matrix operations.
 */
template <typename T>
class Matrix
{
public:
  Matrix() {}
  Matrix(const unsigned int N, const unsigned int M) : N(N), M(M), A(N, vector<T>(M)) {}

  // Const operator for getting the (i, j) element
  const T & operator()(const unsigned int i, const unsigned int j) const
  {
    assert(i < N && j < M);
    return A[i][j];
  }
  // Operator for getting the (i, j) element
  T & operator()(const unsigned int i, const unsigned int j)
  {
    assert(i < N && j < M);
    return A[i][j];
  }
  // Operator for setting the entire matrix to a value
  void operator=(const T v)
  {
    for (unsigned int j = 0; j < M; ++j)
      setRow(j, v);
  }

  // Prints the matrix
  void print(const string prefix = "", const bool newline = false, const unsigned int pr = 5) const
  {
    if (prefix.length() != 0)
      cout << prefix << endl;
    for (unsigned int j = 0; j < M; ++j)
    {
      for (unsigned int i = 0; i < N; ++i)
        cout << showpos << scientific << setprecision(pr) << A[i][j] << " ";
      cout << endl;
```

```
    }
    if (newline)
      cout << endl;
  }
  // Saves the matrix in csv format
  void save(const string filename, const unsigned int pr = 12) const
  {
    ofstream f;
    f.open(filename);
    for (unsigned int j = 0; j < M; ++j)
    {
      for (unsigned int i = 0; i < N; ++i)
      {
        if (i > 0)
          f << ",";
        f << setprecision(pr) << A[i][j];
      }
      f << endl;
    }
    f.close();
  }

  // Set the j-th row to v
  void setRow(unsigned int j, T v)
  {
    assert(j < M);
    for (unsigned int i = 0; i < N; ++i)
      A[i][j] = v;
  }
  // Set the i-th column to v
  void setColumn(unsigned int i, T v)
  {
    assert(i < N);
    for (unsigned int j = 0; j < M; ++j)
      A[i][j] = v;
  }

private:
  // The size of this matrix
  const unsigned int N = 0, M = 0;

  // Matrix storage
  vector<vector<T>> A;
};

#endif /* MATRIX_H */
```

TriDiagonal.h

```
#ifndef TRIDIAGONAL_H
#define TRIDIAGONAL_H

#define NDEBUG
#include <cassert>
#include <fstream>
#include "Vector.h"

using namespace std;

/**
 * Class that holds a tri-diagonal matrix and is able to perform TDMA in place
 * with a given RHS.
 */
template <typename T>
class TriDiagonal
```

```
{
public:
  TriDiagonal() {}
  TriDiagonal(unsigned int N, T v = 0) : N(N), A(N, v), B(N, v), C(N - 1, v) {}

  // Setters for the top, middle, and bottom rows
  void setTopRow(T b, T c)
  {
    B[0] = b;
    C[0] = c;
  }
  void setMiddleRow(unsigned int i, T a, T b, T c)
  {
    assert(i < N - 1 && i != 0);
    A[i] = a;
    B[i] = b;
    C[i] = c;
  }
  void setBottomRow(T a, T b)
  {
    A[N - 1] = a;
    B[N - 1] = b;
  }

  // Prints the matrix
  void print(const string prefix = "", const bool newline = false, const unsigned int pr = 6) const
  {
    if (prefix.length() != 0)
      cout << prefix << endl;
    for (unsigned int i = 0; i < N; ++i)
      cout << showpos << scientific << setprecision(pr) << (i > 0 ? A[i] : 0) << " " << B[i] << " "
           << (i < N - 1 ? C[i] : 0) << endl;
    if (newline)
      cout << endl;
  }
  // Saves the matrix in csv format
  void save(const string filename, const unsigned int pr = 12) const
  {
    ofstream f;
    f.open(filename);
    for (unsigned int i = 0; i < N; ++i)
    {
      if (i > 0)
        f << setprecision(pr) << A[i] << ",";
      else
        f << "0"
          << ",";
      f << setprecision(pr) << B[i] << ",";
      if (i != N - 1)
        f << setprecision(pr) << C[i] << endl;
      else
        f << 0 << endl;
    }
    f.close();
  }

  // Solves the system Ax = d in place where d eventually stores the solution
  void solveTDMA(Vector<T> & d)
  {
    // Forward sweep
    T 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 != numeric_limits<unsigned int>::max(); --i)
    {
      d[i] -= C[i] * d[i + 1];
      d[i] /= B[i];
    }
  }

protected:
  // Matrix size (N x N)
  unsigned int N = 0;

  // Left/main/right diagonal storage
  vector<T> A, B, C;
};

#endif /* TRIDIAGONAL_H */
```

plots.py