

# Assignment 2

ME6151

AE21M007

Nachiketa Narayan Kurhade

Department of Aerospace Engineering

IIT Madras


Q1. a.

It is required to write a subroutine to solve a tri-diagonal set of linear algebraic equations using TDMA  
The required subroutine that is able to perform this function is “onetdma” in module “tdma”.

Subroutine “onetdma” requires:

- number of nodes, n.
- the a,b,c,d coefficient vectors of TDMA of length n and,
- a vector of n nodes it is solving for.

As its inputs, and it modifies the node vector “line” as result.


$$a_i T_i = b_i T_{i+1} + c_i T_{i-1} + d_i$$

Problem 1 is set up so that it feeds “onetdma” the question 3 solved in assignment 1 in the form of:

- Number of nodes to solve for
- Coefficient vectors a,b,c,d calculated by hand.
- Vector of nodes it is solving for with Dirichlet boundaries set up.

The output matches the hand calculations performed earlier and is shown in the following tests.

Problem 3 is set up so that it feeds only the environment the problem 1 is set up with.

By environment I mean the parameters such as:

- the diffusion coefficients,
- the source term,
- the Dirichlet boundary and,
- the geometry of the cells.

Subroutine “onedtdma” takes all these parameters as an input, builds coefficients vectors a,b,c,d based on the environment provided and uses subroutine “onetdma” to solve the node vector provided. Test case, exactly same as the question 3 solved using hand calculations in the assignment 2 has been shown to be executed successfully in the following pages to give the desired output.

Q1. b.

TDMA solver is extended to a line-by-line TDMA solver here in subroutine “twodtdma” of module “tdma” to solve the two dimensional systems resulting out of discretization of the structured meshes.

Four kinds of sweeps can be performed:

+x,-x,+y,-y

Just as done with the subroutine “onedtdma”, subroutine “twodtdma” takes environment of the problem set up as input, where environment means

- source terms
- diffusion coefficients
- Cell dimensions

Which are identified as a defining property for every cell/node in this algorithm.

- It then takes in another input in the form of a matrix of nodes it is solving for already equipped with the Dirichlet boundary conditions. Breaks the problem into individual lines, builds the respective coefficient vectors and then solves for lines using subroutine “onedtdma”. This action is then performed for every “line” which we call “sweep”. Sweeping is done repeatedly with updated matrices till we reach convergence.

Note: subroutine “twodtdma” is limited to solving the problems set up with Dirichlet boundary conditions only.

Advantage of environment based solver is that the sheer flexibility it provides in terms of setting up the problem. It does not require any sort of hand calculations to build up the coefficients and thus saves valuable time.

However the solver is limited to solve for 3\*3 matrix as the smallest matrix with the limiting factor being the very nature of problem it is designed to solve for.

While the subroutine is not in any ways restricted by “uniformity” of the grid, the grid still has to be an orthogonal structured grid with quadrilateral cells.

A sample problem is solved in following pages where we try to solve for an equation  $x^2 - y^2$  that satisfies the laplace equation over a rectangular grid. We set up the Dirichlet boundary conditions and find that the values for the unknown nodes of interior found using subroutine “twodtdma” match the analytical solution in any sweep direction within the convergence error.

Q2. b.

The discretized energy equation is solved for temperature  $T(y, z)$  using a new subroutine “newtdma2d”.

The subroutine “newtdma2d” is exactly same as subroutine “twotdma” except that it is designed to handle neuman boundary conditions.

Subroutine “newtdma2d”:

- assumes the values initialized at the boundary are the Dirichlet boundary conditions at first and finds intermediate values for them.
- Based on these intermediate values and fluxes available at wall, it then updates the boundary conditions.
- However the solution reached this way is then specific to the initialized boundary values, to fix the solution, we have to fix an interior point
- If a fixed interior point is encountered in a line, the “onetdma” has been modified such that it recognises it as a boundary and solves the line upto the fixed point, then solves the line after the fixed point.
- Ideally this process should require a recursion capability but since implementing recursive abilities in fortran is difficult and we will not encounter more than one iterations(remember only one fixed point in a line), we get away with redefining a mirror subroutine for “onetdma” that performs exactly the same task.
- If fixed interior point is not identified, the code runs the same way as if it has Dirichlet boundary conditions, except the boundary points will get updated at the end based on their nearest neighbours and fluxes available at the boundary which implements the neuman boundary.
- Corner points are handled by taking arithmetic average of their immediate neighbors.
- Important point to consider is that TDMA by its very nature, recognises the two end points of a line as fixed boundary conditions and thus assumes the coefficient matrix is not a singular matrix. It thus will never update the end points on its own. This task, as required for neuman boundary conditions has to be handled separately as described above.
- While except boundary and corner point handling, subroutine “newtdma2d” is identical to subroutine “twotdma” in terms of workflow, since it will be requiring those boundary fluxes as an additional input, I thought it would be better to define the function entirely from scratch.

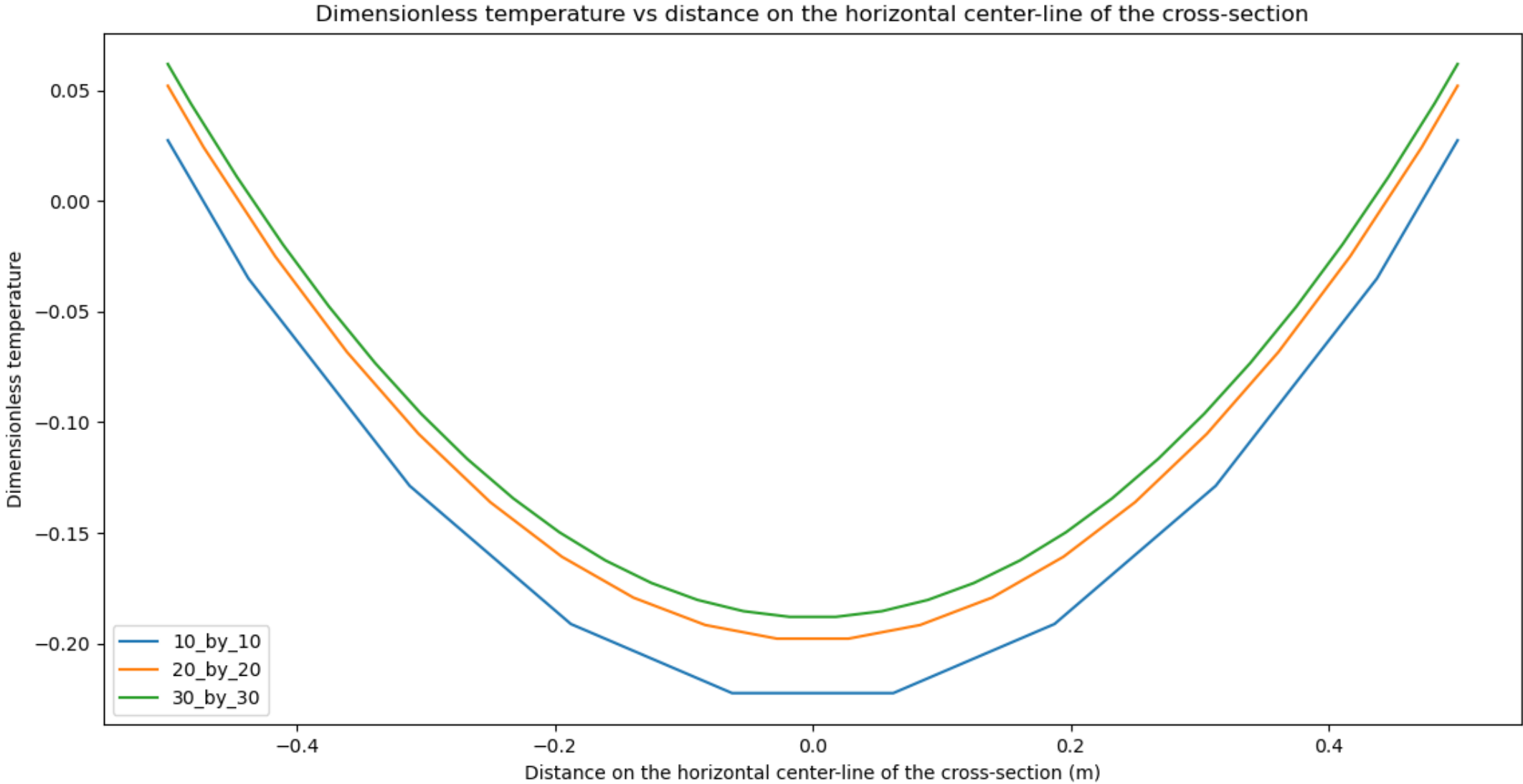
Q2. c. we can clearly see the temperature profile converging with finer grid. Looks like the more coarse the grid is the more error solution would have, although the curve will look just like an analytical solution.

10\_by\_10 : The bulk temperature for the given problem is 222.249999997921 K

20\_by\_20 : The bulk temperature for the given problem is 207.441358018501 K

30\_by\_30 : The bulk temperature for the given problem is 195.842403616001 K

Bulk temperature is decided by the position of fixed point in the provided grid. The required plot is shown below.



Q2. d. The Nusselt number has not been found to be the same for all grid sizes, however with finer mesh, the Nusselt number appears to be converging towards a constant value.

For 10\_by\_10 grid:

The mean wall temperature is: 234.027777775507 K

The nusselt number is: 8.49056603787452

For 20\_by\_20 grid:

The mean wall temperature is: 221.621182579767 K

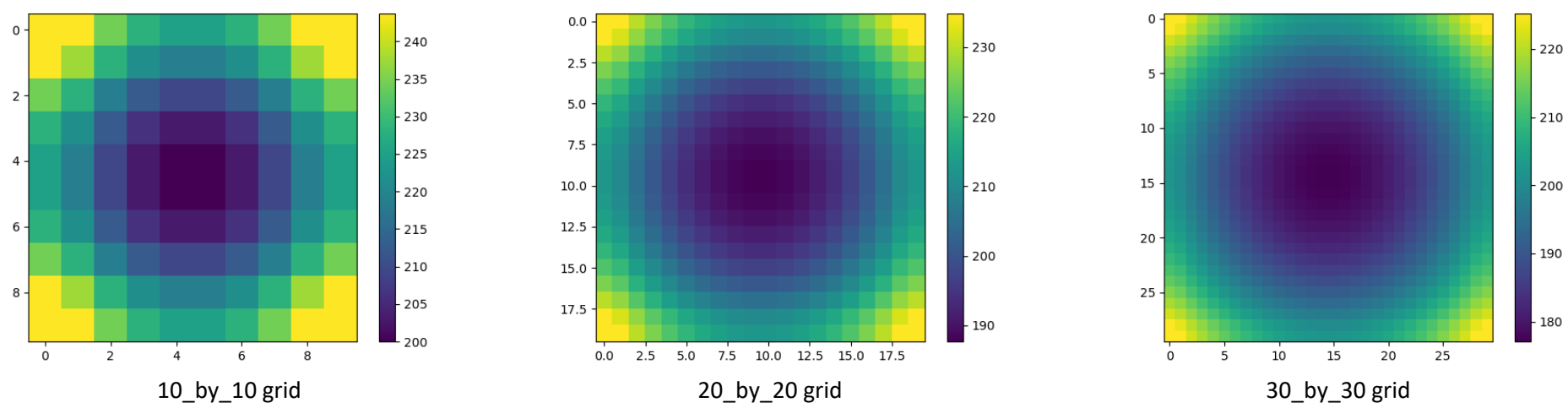
The nusselt number is: 7.05227343031855

For 30\_by\_30 grid:

The mean wall temperature is: 210.846235033762 K

The nusselt number is: 6.66496424917313

As clearly observed from above values, drop in the Nusselt number and the mean wall temperature diminishes with finer grid size, indicating convergence.



Q2. e.

Inspired by the idea, I developed subroutines “inmatsolve”, “inmatsolverec” and “newtdma2dinmat” which work similar to the TDMA subroutines we have been going through so far except they use direct matrix inversion to solve for the “line”. Subroutine “newtdma2dinmat” is basically a line-by-line matrix inversion solver that works just like line-by-line TDMA.

While the matrix inversion methods have been successful in achieving the same solutions found by TDMA solvers they still suffer from the same discretization error.

Matrix inversion methods have to save the whole coefficient matrix which then has to be inverted. Although the coefficient matrix is a sparse matrix, or basically a tridiagonal matrix, constructing such matrix and then performing matrix inversion and subsequent matrix multiplication to get the solution is a computationally heavy task.

Both TDMA and Matrix inversion methods take the exact same number of iterations to converge to the solution however the cpu time taken for every iteration by Matrix inversion methods is considerably larger than TDMA method. The difference is very subtle and not even distinctly noticeable at smaller grid sizes like  $11 \times 11$  but for larger grid sizes such as  $51 \times 51$  matrix inversion method takes considerable amount of time per iteration as compared to the TDMA method.

One has to keep in mind that one cannot simultaneously include the equations imposed by newman boundary conditions in coefficient matrix while calculating a solution based on the matrix inversion methods. The equations simply lead to a singular matrix that cannot be solved. One has to implement the neuman boundary handling functionality separately in order to be able to solve using direct methods.

Q2. f.

Exactly same solution has been found irrespective of the initialization used. (100K, 200K, 300K)

Iterations taken to reach the solution however have been found to be varying greatly, with initializations that have values closer to the solutions take less iterations to reach convergence.

For 100K

6410 iterations

For 200K

5986 iterations

For 300K

6281 iterations