MIE 1210: Computational Fluid Mechanics and Heat Transfer, Assignment 2: 2D Diffusion Problem

Declan Bracken[1, *]

[1] *Department of Mechanical and Industrial Engineering, University of Toronto, Toronto, ON M5R 0A3, Canada*

(Dated: October 30th, 2023)

I. INTRODUCTION

A. Finite Volume Discretization

The purpose of this assignment was the construction of a 2-dimensional finite volume computational fluid dynamics solver. The following derivation of the finite volume method applied to a 2D diffusion equation follows the works at [1]. As the second assignment in the course, this project has been constrained to solving for a state-variable, like temperature, using only the diffusion equation:

$$\Gamma \nabla \cdot \nabla \varphi = \Gamma \nabla^2 \varphi = S_{\varphi}. \tag{1}$$

Where φ is the state variable representing temperature in a 2D, rectangular domain of size L_x, L_y, and Γ is a constant coefficient describing the thermal conductance of the domain's material. Using the finite volume method, a control volume integration is performed on equation (1), yielding;

$$\begin{aligned}\int_{CV} \nabla \cdot (\Gamma \nabla \varphi)\, dV + \int_{CV} S_{\varphi}\, dV = \\ \int_{A} \mathbf{n} \cdot (\Gamma \nabla \varphi)\, dA + \int_{CV} S_{\varphi}\, dV = 0,\end{aligned} \tag{2}$$

which, for a 2D problem, yields the partial differential equation:

$$\begin{aligned}\Gamma_e A_e \frac{\partial \varphi}{\partial x_e} - \Gamma_w A_w \frac{\partial \varphi}{\partial x_w} + \\ \Gamma_n A_n \frac{\partial \varphi}{\partial y_n} - \Gamma_s A_s \frac{\partial \varphi}{\partial y_s} + S\Delta V = 0.\end{aligned} \tag{3}$$

Where the subscripts n, s, e, and w correspond to the cardinal directions of the 2d grid denoting north, south, east and west when viewing the plane. For a uniform grid and non-variable thermal conductance, Γ and A are constant, and do not need the subscripts. The discretization of this equation using a central difference scheme between successive nodes yields the flux equations;

$$\Gamma A_w \frac{\partial \phi}{\partial x_w} = \Gamma A_w \frac{\phi_P - \phi_W}{\delta x_{WP}}, \tag{4a}$$

$$\Gamma A_e \frac{\partial \phi}{\partial x_e} = \Gamma A_e \frac{\phi_E - \phi_P}{\delta x_{PE}}, \tag{4b}$$

$$\Gamma A_s \frac{\partial \phi}{\partial y_s} = \Gamma A_s \frac{\phi_P - \phi_S}{\delta y_{SP}}, \tag{4c}$$

$$\Gamma A_n \frac{\partial \phi}{\partial y_n} = \Gamma A_n \frac{\phi_N - \phi_P}{\delta y_{PN}}. \tag{4d}$$

* declan.bracken@mail.utoronto.ca

Where ϕ_P represents the state for the central node in the domain, commonly referred to as node P, and δy, δx represent the spacing between successive nodes in the y and x directions respectively. Rewriting equation (3) using the discretized equations computed with central difference and grouping like terms by ϕ yields;

$$\begin{aligned}\left(\frac{\Gamma_w A_w}{\delta x_{WP}} + \frac{\Gamma_e A_e}{\delta x_{PE}} + \frac{\Gamma_s A_s}{\delta y_{SP}} + \frac{\Gamma_n A_n}{\delta y_{PN}} - S_p\right)\phi_p = \\ \left(\frac{\Gamma_w A_w}{\delta x_{WP}}\right)\phi_w + \left(\frac{\Gamma_e A_e}{\delta x_{PE}}\right)\phi_e + \left(\frac{\Gamma_s A_s}{\delta y_{SP}}\right)\phi_s + \\ \left(\frac{\Gamma_n A_n}{\delta y_{PN}}\right)\phi_n + S_u,\end{aligned} \tag{5}$$

where S_p and S_u are the source terms separated into components due to their dependence on temperature (S_P is state dependent while S_u is not). The coefficients of each ϕ term are constant for the diffusion problem, hence equation (5) can be rewritten as

$$a_p \phi_p = a_w \phi_w + a_e \phi_e + a_s \phi_s + a_n \phi_n + S_u, \tag{6}$$

with defining coefficients:

a_W	a_E	a_S	a_N	a_P
$\frac{\Gamma_w A_w}{\delta x_{WP}}$	$\frac{\Gamma_e A_e}{\delta x_{PE}}$	$\frac{\Gamma_s A_s}{\delta y_{SP}}$	$\frac{\Gamma_n A_n}{\delta y_{PN}}$	$a_w + a_E + a_S + a_N - S_p$

Table I: Discretization coefficients and their expressions

These coefficients will be used to populate a aquare dependency matrix A of size $n_x \times n_y$ by $n_x \times n_y$ where n_x, n_y are the number of x and y dimensional node points in the 2d grid. The natural shape of the dependency matrix is tri-diagonal, with 2 additional non-zero diagonals beginning at $[0, n_x]$ and $[n_y, 0]$ due to the 2d nature of the problem. This dependency matrix can then be used to solve for the state variable ϕ using the solution array S_u comprising the generative source terms due to the boundary conditions according to:

$$\{A\}\{\phi\} = \{S_u\}. \tag{7}$$

B. Boundary Derivations

There are 3 kinds of boundaries used in this simulation problem; an insulating boundary, fixed temperature, and mixed or conductive boundary. The first 2 boundary derivations are straightforward, while the third requires greater analysis.

For an insulated boundary the flux across the domain is 0 ($\frac{\partial \phi}{\partial y}, \frac{\partial \phi}{\partial x} = 0$), which means that one can simply set the coefficient for that side of the central node to 0.

For a fixed temperature boundary, the flux into the central cell is calculated in the same way that it is for interior nodes, but using a known temperature value T_B. For example, if the boundary of the west wall were held at a constant temperature, the flux equation would look like:

$$\Gamma A \frac{\partial \phi}{\partial x_w} = \Gamma \frac{\phi_P - T_B}{\delta x_{BP}}, \tag{8}$$

which leads to generative source terms in the diagonal of the dependency matrix (representing the central nodes during solving) and the source array:

a_W	a_p	S_p	S_u
0	$a_N + a_S + a_E - S_p$	$-\frac{2kA}{\delta x}$	$\frac{2kA}{\delta x} T_B$

For the mixed boundary condition, we can start with the convective heat transfer equation, often representative of the boundary between 2 materials:

$$-\Gamma \frac{d\phi}{dy} = h_f(\phi_{\text{ext}} - \phi), \tag{9}$$

where T_{ext} is the temperature of the external material, and h_f is the transfer coefficient between the materials. This equation can be expressed as the difference in temperature between the external temperature T_{ext} and the temperature at the wall T_{wall};

$$-\Gamma \frac{T_{\text{wall}} - T_p}{\delta x_{BP}} = h_f(T_{\text{wall}} - T_{\text{ext}}), \tag{10}$$

$$T_{\text{wall}} = \frac{T_p + C T_{\text{ext}}}{1 + C}, \quad C = \frac{h_f \delta x_{BP}}{\Gamma}. \tag{11}$$

Having the temperature of the wall expressed as a function of known variables T_{ext} and T_p allows us to insert it into equation (**??**) replacing T_B. Simplifying the expression which follows will eventually lead to the derivation of the new source coefficient term:

$$S_u = \frac{T_{\text{ext}} A_{wall}}{\frac{\delta x_{BP}}{\Gamma} + \frac{1}{h_f}}, \tag{12}$$

$$S_p = 0, \tag{13}$$

$$a_{\text{bound}} = \frac{A_{wall}}{\frac{\delta x_{BP}}{\Gamma} + \frac{1}{h_f}}. \tag{14}$$

II. RESULTS AND DISCUSSION

A. Equidistant Grid Spacing

The simulation is setup with the following conditions: A mixed north boundary with ambient temperature $T_{ext} = 300$, fixed temperature east and west walls set to $T_{east} = 100$ and $T_{west} = 10$, and an insulated south wall. The north boundary implements a transfer coefficient of $h_f = 10$ and thermal conduction coefficient of $\Gamma = 20$. The grid size is set to $L_x, L_y = 1$ for 3 different equidistant resolutions of $n_x, n_y = 80, 160, 320$.

The system of linear equations described by equation (7) is solved using a stabilized bi-conjugate gradient iterative solver, with incomplete L-U factorization preconditioning.

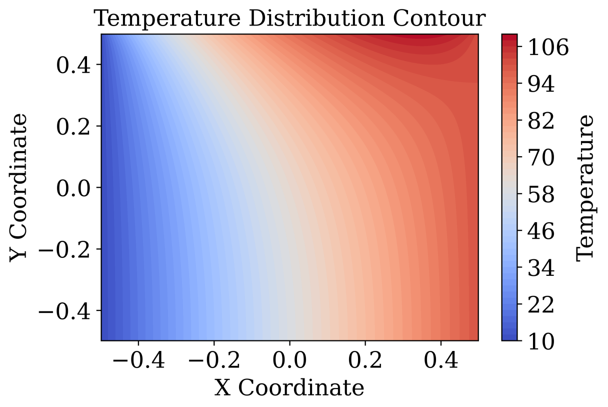

Figure 1: Temperature distribution for a grid with 320×320 nodes.

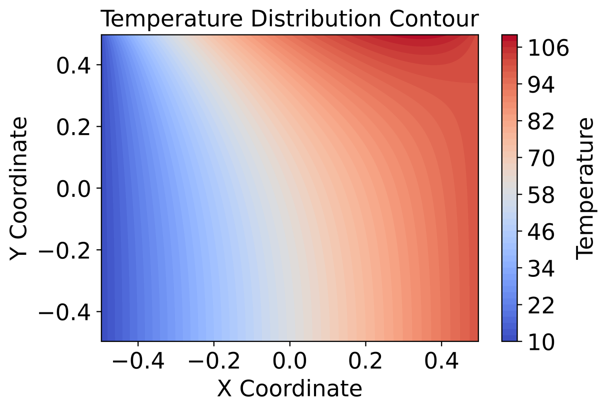

Figure 2: Temperature distribution for a grid with 160×160 nodes.

Analyzing figures 1 through 3, there's very little noticeable change in the contour graph, even with 60 contour separations. If the images are expanded, there is a small noticeable difference between the 80×80 grid and the 160×160 grid, and an even smaller difference between the 160 and 320. The L2 norm of the residual for the iterative BiCGSTAB solver is calculated at every iteration, with a solver tolerance set at 10^{-5} for each of the resolutions. The L2 norm of the residual reached by each solver (to 3 significant figures) is 0.702, 0.468, and 0.226 for the fine, medium, and course meshes respectively. The convergence plots are available below.

Finally, the order of convergence for the solver using an equidistant mesh is calculated according to equation 15.

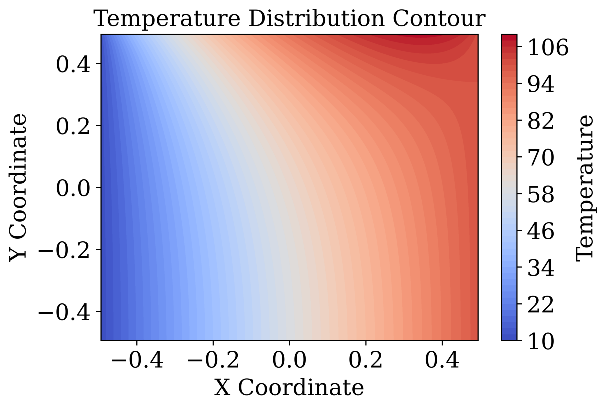

Figure 3: Temperature distribution for a grid with 80×80 nodes.

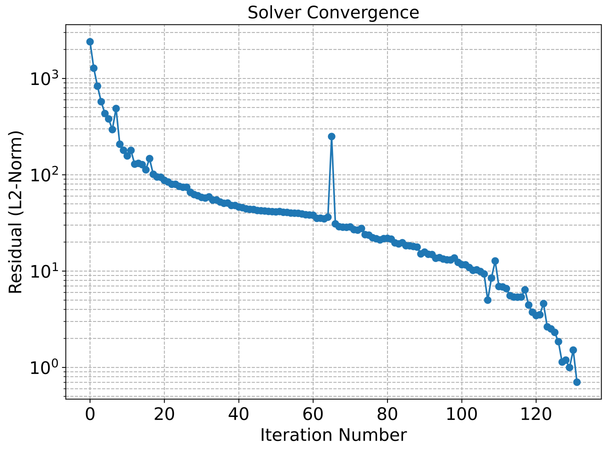

Figure 4: Convergence of residual's L2 norm for a grid with 320×320 nodes.

$$0 \approx \frac{\log(|e_c|/|e_f|)}{\log(h_c/h_f)}. \tag{15}$$

Where e_c and e_f are the error norms for a course and a medium mesh (with respect to a fine mesh) and h_c and h_f are the spacing of the course and medium mesh respectively. The error norms can be calculated according to equation (16);

$$|e| = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\phi_f - \phi_c)^2}, \tag{16}$$

where ϕ_f is the temperature distribution for the finest grid (estimating the analytical solution) at 320×320, and ϕ_c is either the course or the medium temperature distribution. Since the meshes don't have the same sampling points, an interpolation and extrapolation step is performed so that the course and medium meshes are sampled at the same locations as the fine mesh.

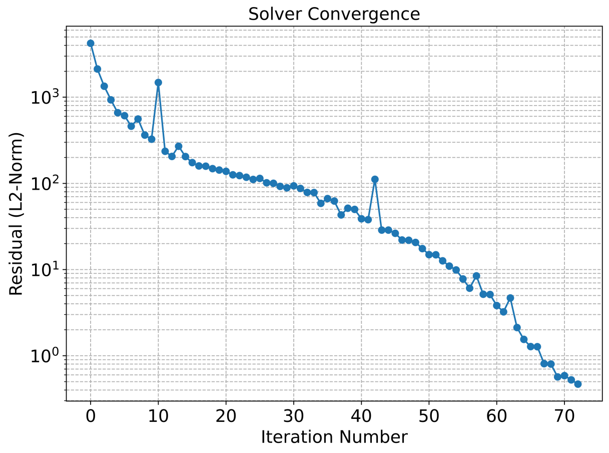

Figure 5: Convergence of residual's L2 norm for a grid with 160×160 nodes.

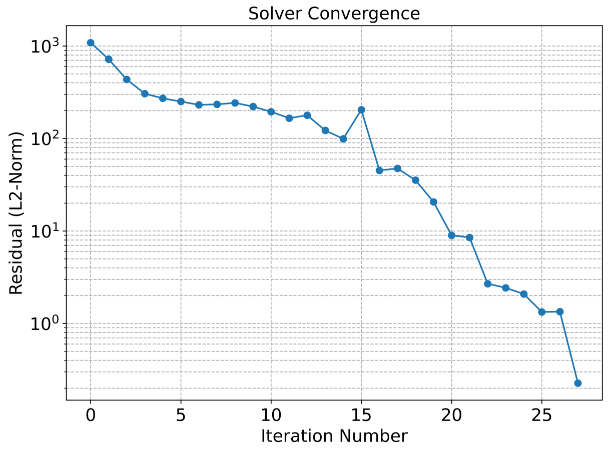

Figure 6: Convergence of residual's L2 norm for a grid with 80×80 nodes.

Table II: Errors and Order of Convergence for Different Mesh Sizes

Mesh Size	**Error**
320x320 (fine)	Analytical Solution
160x160 (medium)	46.7505
80x80 (coarse)	46.7573
Order of Convergence: 0.00021012	

B. Inflated Grid

As seen in figures 1 through 3, a larger proportion of the temperature gradients is concentrated near the edges of the domain due to the boundary conditions. In order to resolve the simulation better along the boundaries, we may implement a non-equidistant mesh utilizing an x-y symmetric inflation factor r. The x spacing along the edges of the domain are initialized at;

$$\Delta x_0 = \frac{(1-r)L_x}{(1-rn_x/2)2}, \tag{17}$$

where $L_x = 1$ is the width of the domain and n_x is the resolution for the x-dimension. The exact same equation is applied for the y dimension, making the grid symmetric. Then, past $i = 0$, each successive spacing between nodes is calculated for;

$$\Delta x_{i+1} = \begin{cases} r\Delta x_i & \text{if } x \leq L_x/2, \\ \frac{\Delta x_i}{r} & \text{if } x > L_x/2. \end{cases} \quad (18)$$

Applying an inflation factor of $r = 1.05$ to the course grid yields a noticeably expanded mesh distribution, with a minimum node area of $1.713 * 10^{-5}$ and a maximum node area of $8.491 * 10^{-4}$. This finite element distribution is only an example used to demonstrate the grid visually, as with an inflation factor this large the nodes tend to be overly concentrated to the outside of the domain. For better analysis, an inflation factor of $r = 1.01$ is set, and the contour plot for the resulting medium grid is shown.

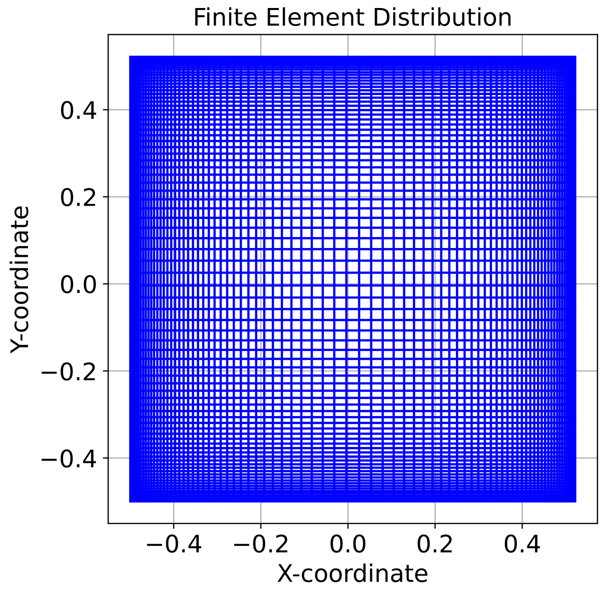

Figure 7: Mesh distribution for an 80 × 80 grid with an inflation factor of $r = 1.05$.

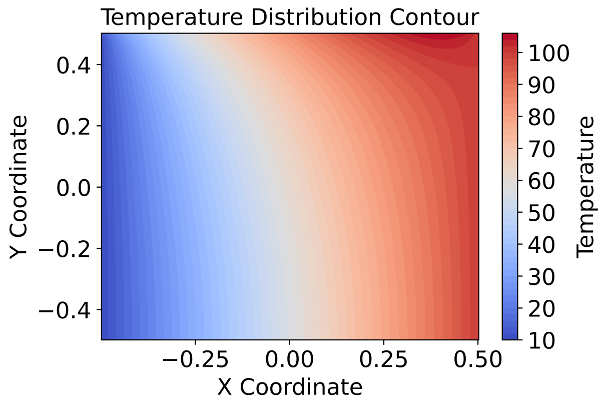

Figure 8: Contour plot showing the temperature distribution for the medium, 160 × 160 grid using an inflation factor of $r = 1.01$.

The L2norm of the residuals and the order of convergence are recalculated for the course and medium meshes using the $r = 1.01$ inflation factor. Note that the domain now reaches slightly past the $x = 5.0$ mark, which is a product of the inflation of the mesh spacing and it's cumulative sum, and is not an error in the code.

Table III: Errors and Order of Convergence for Different Mesh Sizes with Inflation

Mesh Size	**Error**
320x320 (fine)	Analytical Solution
160x160 (medium)	50.2059
80x80 (coarse)	50.9714
Order of Convergence: 0.02216	

III. CONCLUSIONS

The Finite Volume Method has become a staple in Computational Fluid Dynamics (CFD) due to its inherent properties and advantages. One of its primary strengths is its grounding in the fundamental principle of conservation. By dividing the domain into numerous control volumes, FVM ensures the conservation of quantities such as mass, momentum, and energy as they enter and exit these volumes. This localized conservation approach aligns closely with the physical principles governing fluid flows, making FVM a natural choice for CFD.

Moreover, FVM is highly versatile, handling irregular and complex geometries with relative ease. This is because the control volumes can be of any shape, allowing for great flexibility in grid generation around intricate geometries commonly found in real-world engineering problems. FVM's mesh shape insensitivity allows it to be applied to a wide range of problems.

Another advantage is the method's robustness. FVM is known to be less sensitive to grid quality, making it more stable and reliable in practice, especially when faced with grids that might not be perfectly uniform or may contain some skewed elements.

This implementation of the finite volume method has shown great promise in constructing solutions to the 2D diffusion problems using a variety of grid sizes and element distributions. While results are promising, there are still several limitations to the model created. One limitation is the coding architecture, which has not been fully modularized using python classes. Implementing the code as python classes will allow for easier debugging in future iterations, and makes different elements of the code easier to use in isolation, like calculating boundary coefficients. While all of the code except for the initialization variables resides in python functions, it would be optimal to transform the architecture into classes.

Another obvious improvement is the inclusion of the full Navier-Stokes equation, and time dependency. This CFD architecture has been designed with solely a diffusion problem in mind, but future iterations will require time and velocity dependant terms which the scripts will need to be adjusted for.

Another improvement would be improving the efficiency of the dependency matrix formation code. Right now the dependency matrix $\{A\}$ is constructed as a 2D Numpy array of dictionaries, where each dictionary holds neighbor and boundary information for that cell. Population of the dependency matrix requires looping through all grid points, and then all neighbors/boundaries, until each coefficient is individually calculated and placed in the appropriate location of the dependency matrix. While I believe that looping is necessary for dependency population, especially for asymmetric meshes and boundary conditions, I think there could be a faster way to create the matrix which requires exploring.

[1] H. K. Versteeg and W. Malalasekera, *An Introduction to Computational Fluid Dynamics: THE FINITE VOLUME METHOD*, 2nd ed. Pearson Education Limited, 2007.