Unstructured Mesh Methods

Mohammad Zandsalimy

December 20, 2021

1 Problem

The 2D convection equation (presented as equation 1) is selected for numerical solution on a square shaped domain containing an unstructured mesh with triangular elements. In this equation, u and v are the velocity vector components which are known and T is the only unknown variable.

$$\frac{\partial T}{\partial t} + u\frac{\partial T}{\partial x} + v\frac{\partial T}{\partial y} = 0 \tag{1}$$

The solution domain is a 6×6 square with $T = 0$ on all boundaries. The initial conditions for the solution and velocity domain are presented as equations 2 and 3, respectively.

$$T(x, y) = \exp\left[-5\left(x^2 + (y-1)^2\right)\right] \tag{2}$$

$$\begin{cases} u(x, y) = \pi y \\ \\ v(x, y) = -\pi x \end{cases} \tag{3}$$

This velocity field represents a clockwise rigid body rotation with a period of 2. As a result, at any time the exact solution to the flow is known. The contours of initial conditions for the solution is presented in figure 1. The 3D version of the same contour plot is presented in figure 2. As seen here, the solution has a maximum value of 1.0 at $(x, y) = (0, 1)$. As a result of the lack of viscous dissipation terms, the solution should move in a rigid body motion without any losses. However, in the numerical solution we will witness dissipation in time which is due to discretization errors. Five numerical grids for solution is provided with the (self-) addition of a test mesh containing only two triangle over the domain for testing purposes. These numerical grids include 2, 32, 136, 494, 2068, and 8168 cells and are called *test*, *very coarse*, *coarse*, *medium*, *fine*, and *very fine*, respectively. For the sake of simplicity, let's refer to each mesh with a numeric code. 0 for *test* and 1 through 5 for *very coarse* through *very fine* sounds good. The medium mesh (mesh 3) is presented in figure 3 as an example.

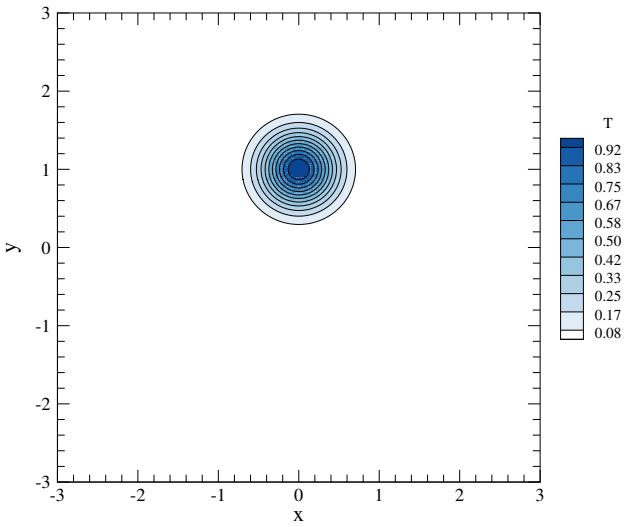

Figure 1: The contours of initial condition for T.

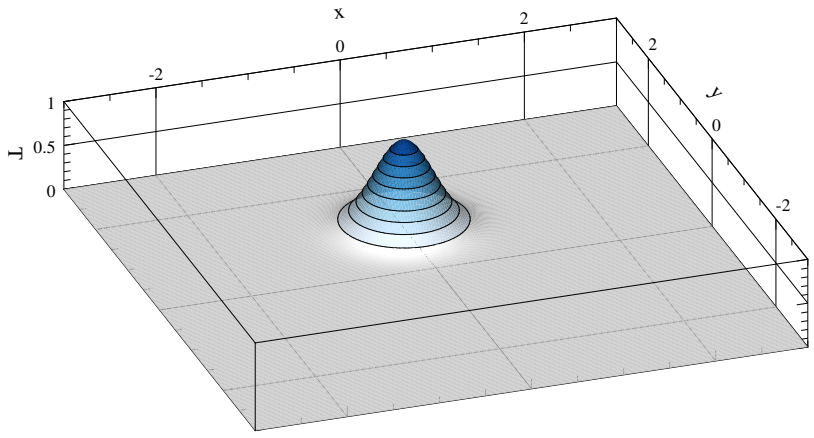

Figure 2: The 3D contours of initial condition for T.

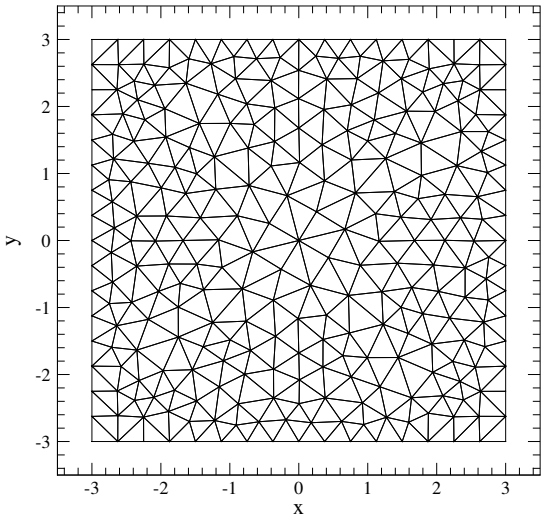

Figure 3: Medium mesh with 136 cells.

2 Finite Volume Method

The finite volume methods on unstructured grid have been thoroughly studied and explained in literature. Hence, we are not going to explain every bit of the method in use. However, I will discuss the important aspects (in my opinion) of the method. Most of the discussions in this section are extracted from the course notes [1]. The outline of a finite volume unstructured mesh scheme (as presented in the notes), can be summarized as follows:

a. Reconstruction of the solution to a certain order of accuracy. In this project, we are using a linear reconstruction which gives second order accuracy.

b. Calculate the fluxes at any required point. We will calculate the flux at midpoint of each edge. This flux can be added to and subtracted from a *receptor* and *donor* cell, respectively. Be sure to use the solution value at the control point for flux calculation. Not amazingly, the average solution inside a cell will not give good results. The summation of flux across each face of a single cell is in fact the residual of the solution in that cell which can be used to progress in time.

c. While conducting flux calculations, make sure you are thinking about the maximum time step allowable for the numerical solution. Each cell in an unstructured grid can have a different size than others. This will restrict

us from advancing the solution too quickly or the scheme will blow right up. Each cell will have a maximum allowable time step which can be calculated using the fluxes. The global time step is the minimum of these local time steps.

d. Advance the solution in time using your explicit time integration method of choice. We will use Runge-Kutta 3 stage in the present project just to be sure.

2.1 Solution Reconstruction

The goal is to approximate the solution at any given point inside the domain with a polynomial of degree k. This approximating polynomial will match the exact function to order h^{k+1}, in which h is the representative cell size. In the present project, we will use a linear approximation for a reconstruction of order 2. There are two sets of different conditions that should be satisfied in a successful reconstruction scheme. First, the approximating polynomial should exactly reproduce (it better do, because it was designed for that reason) the average value of the function inside the cell in question. Second, the polynomial should reproduce the average values of the solution inside the neighboring cells. For a first order reconstruction, as explained in the notes, the Taylor series expansion can be used for cell i as follows.

$$\phi_i^R(x,y) = \phi_i + \left.\frac{\partial\phi}{\partial x}\right|_i (x - x_i) + \left.\frac{\partial\phi}{\partial y}\right|_i (y - y_i) \tag{4}$$

In this equation, $\phi_i^R(x,y)$ is the reconstructed solution inside cell i. ϕ_i, $\left.\frac{\partial\phi}{\partial x}\right|_i$, and $\left.\frac{\partial\phi}{\partial y}\right|_i$ are constant coefficients specific to cell i and are calculated during the reconstruction process.

The finite volume method in use is cell centered and the control point is chosen to be the centroid of each cell. This can help us during the reconstruction process by zeroing out some terms. As easy and accurate as unstructured mesh reconstruction seems, it can be (will be) inaccurate depending on the reconstruction stencil. From the number of neighbors point of view, there are three types of cells in our mesh.

- Cells with only one neighboring cell
- Cells with two neighboring cell
- Cells with three neighboring cell

Our equation has three unknowns and for an exact solution we will require exactly three equations. This only happens in the case of cells with two neighbors which are pretty rare compared to cells with three neighbors (at least in our case).

However, there is nothing to be worried about. We can use singular value decomposition to get an approximate (or a least squared) solution to the over-determined or under-determined system of equations. For this purpose, I have utilized Armadillo, a high quality linear algebra library for the C++ language. It is actually very straight forward and easy to use. Just be sure to have the latest copy of the software installed on your machine and use *openblas* and *lapack* libraries with *-DARMA_DONT_USE_WRAPPER* flag to compile. An example compile command is presented as follows.

```
$ g++ -O3 main.cpp -DARMA_DONT_USE_WRAPPER -lopenblas -llapack
```

2.2 Flux Integration

First order upwind fluxes are used for the purpose of flux integration in the present project. Before anything, we have to determine the flow direction at a control point on the edge. This task is easy enough with a simple dot product of the flow velocity at the control point and the face unit normal which is always pointing to the right of the face. We should mention that the control points of each edge (face) are the center points. If the flow direction is pointing to right, we should use the reconstructed solution value at the edge control point from the left cell and vice versa. Not surprisingly, this is the average flux at control point of the edge. Flux integration over the edge is conducted by multiplying the average flux and the edge length. Finally, this value if subtracted from the donor cell and added to the receptor cell. Note that, this is the total flux inside the cell and should be divided by cell area to get the average residual for time integration.

Further, we have to think of a way to apply the boundary conditions in flux integration. In our solution domain, one might think that the outer boundary is big enough that no flux integration is required over there. However, a closer inspection of this issue revealed that in fact treating the boundaries as solid walls is not a particularly good choice in this case. The solution would slowly start to grow near the boundaries and I was getting non-physical results. A better choice for the boundaries in my opinion is inflow and outflow boundary conditions. Just treat the boundary edges as normal and calculate fluxes over them. This way we are not restricting the fluid flow to inside of the domain.

2.3 Maximum Stable Time Step

The only consideration for maximum solution time step is that the information cannot move more than one cell size at any given iteration. Another way of looking at this is the net incoming flux into a cell cannot be allowed to overfill the cell in one iteration. As a result, just add all the incoming flux up for a certain cell and find the maximum allowable time step for that cell using equation 5.

$$\Delta t_{i,\max} = \frac{A_i}{\oint_i |\text{Incoming Flux}|\, ds} \tag{5}$$

Now, the only thing left is to find the minimum of these allowable local time steps, to use as the global time step. A CFL conditions is also employed to make matters more controllable. Finally, the time step used in our scheme will be equal to CFL multiplied by the global minimuim of maximum allowable time steps in all control volumes.

2.4 Time Advance Scheme

Runge-Kutta 3 stage is selected as the explicit time advance scheme which is 3rd order accurate in time (on paper at least). This scheme is presented as equation 6. λT^n in this equation correlates to the residual of solution at each stage. The amplification ratio of Runge-Kutta 3 time advance scheme is presented in equation 7. This stage is straight forward with the only consideration being that we have to save two copies of the solution to prevent mixing up $T^{(1)}$ and $T^{(2)}$ with $T^{(n)}$ or $T^{(n+1)}$.

$$\begin{cases} T^{(1)} = T^n + \dfrac{\Delta t}{3}\left(\lambda T^n\right) \\ T^{(2)} = T^n + \dfrac{\Delta t}{2}\left(\lambda T^{(1)}\right) \\ T^{n+1} = T^n + \Delta t\left(\lambda T^{(2)}\right) \end{cases} \tag{6}$$

$$\sigma = 1 + \lambda\Delta t + \frac{(\lambda\Delta t)^2}{2} + \frac{(\lambda\Delta t)^3}{6} \tag{7}$$

2.5 Numerical Integration

Finite volume methods traditionally require several different area integration in different parts of the solver (e.g. finding the control volume averages and flux averages from the exact data). An effective method for numerical area integration is Gaussian quadrature which has several different implementations. I will use a 120 point quadrature presented by [2] which can exactly integrate a polynomial of up to order 25. The quadrature point coordinates and weights can be found in the *.tex* file of the article (https://arxiv.org/abs/math/0501496). These points are stored in the file *quad_coords.dat* in the same directory as the our C++ program. Furthermore, the program is very flexible on the quadrature method in use. This means that providing any quadrature file (following the same format obviously) containing any number of quadrature points inside the reference triangle will work just fine without any alterations of the code.

The 120 quadrature points are shown in figure 4. These points are presented for a reference right angle triangle and should be linearly mapped to an arbitrary triangle inside the mesh for later use. To find the mapping, first, we need to map three corners of the reference triangle $(0,0)$, $(1,0)$, and $(0,1)$ to an arbitrary triangle with corners (x_1, y_1), (x_2, y_2), and (x_3, y_3). Suppose a linear mapping is presented as equation 8 which maps a point (x, y) to the point (x', y'). Substituting the three corners of the reference and destination triangles

gives six equations which can be solved to get the six unknown coefficients in equation 8. Doing so, the final linear mapping is presented by equation 9. As a result of this mapping, the quadrature points and triangle in figure 4 is mapped to figure 5.

$$\begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} \begin{bmatrix} 1 \\ x \\ y \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix} \tag{8}$$

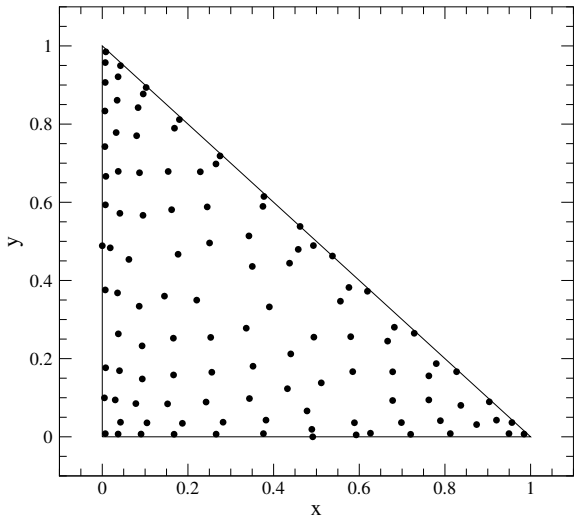

Figure 4: 120 point quadrature on the reference triangle.

$$\begin{bmatrix} x_1 & x_2 - x_1 & x_3 - x_1 \\ y_1 & y_2 - y_1 & y_3 - y_1 \end{bmatrix} \begin{bmatrix} 1 \\ x \\ y \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix} \tag{9}$$

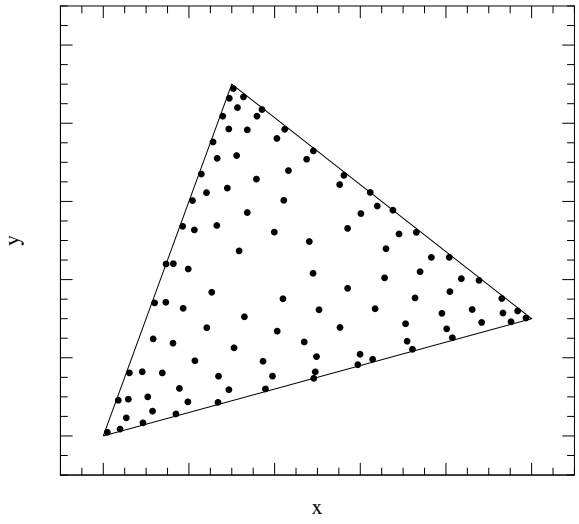

Figure 5: 120 point quadrature on the destination triangle.

2.6 Data Structure

Edge based data structure is utilized in the present project as it is the most convenient and easiest to implement. That being said, I will have three separate data structures to store the data for vertices, edges, and cells. Vertex type data structure includes double precision numbers for x and y coordinates of each vertex, value of T and the initial solution at that point, and velocity components in horizontal and vertical directions. The ordering of vertex data shows the index for each vertex. This means that for example vertex number 10 (9 in C++ programming language) is the tenth vertex in the vertex data structure.

Edge type data structure contains more information including, the coordinates of the center location, length of the edge, unit normal vector components, velocity components at center point, and a Boolean to indicate whether the edge lies on the boundary or not. Most important information in the edge type data structure, however, are the starting and destination vertex indices and left and right adjacent cells. In this way the starting vertex of any edge can be accessed with *vertex[edge.starting_vertex]* command and the left cell of the edge can be accessed with *cell[edge.left_cell]* command. The same applies for other information of the edge as well.

Cell type data structures have the most information stored out of the three data types utilized. Double precision data include two variables to store the average solution value at the control point at each time step (T and T_old), the exact average value in each cell (T_exact), 120 quadrature point coordinates each including x and y locations as well as the integration weight w. Area of

the cell, maximum allowable time step, the location of the centroid (control point), three moments of neighboring cells, and three constant coefficients of solution reconstruction (ϕ, $\frac{\partial \phi}{\partial x}$, and $\frac{\partial \phi}{\partial y}$) are other information stored in each cell. Finally once again, the most important data stored in the cell type data structure are six integer values for the vertex index of each of the three corners and three neighboring cells. Vertices are stored in a counter clock wise fashion. We should also mention the cases where a cell has less than three neighbors. In such cases, a negative value is stored in the corresponding variable. As an example let's access the x location of the third vertex of the second neighbor of a cell i with *vertex[cell[cell[i].second_neighbor].third_vertex].x_location*. As we can see, it is possible to access data so far away that they start becoming irrelevant which means more options, and options are good.

3 Validation Plan

An important first step in developing any successful CFD solver is to write a validation plan. That is, we need to be able to identify the key parts of the program and write test codes for them. Such tests can be conducted for each function separately or we can have global tests. Of course, using both local and global tests will confirm the correctness of the program readily with an easier debugging process.

3.1 Numerical Grid

Mesh files come in many different extensions and formats. As explained earlier, a good data structure to utilize in finite volume analysis is edge based data. A popular edge based numerical mesh extension is *.mesh* files. The first line in such files includes four integers showing the number of cells, edges, boundary edges, and vertices, respectively. Then, using the following data for vertices and edges we can easily setup our data matrices for the numerical solution. A test should be conducted to check the legitimacy of the loaded mesh. We should focus on the numbering and prevent any unintended negative values. For this test, just read the mesh and write out the data with an arbitrary solution to check if the third party post processing software can load the output file.

3.2 Cell Area Integration

There are coordinates of 120 quadrature points and their corresponding weights stored in each cell. Quadrature can be utilized for numerical integration over area. On the other hand, the exact area of an arbitrary triangle with corners (x_1, y_1), (x_2, y_2), and (x_3, y_3) can be calculated by a simple cross product of any two edges. As a result, the exact area of this triangle can be calculated as equation 10. The validation process for quadrature in cell area integration includes calculating the exact area of each cell using equation 10 and comparing

it to the numerical area integration using quadrature. Further, the summation of all cell areas should be equal (very close) to the domain area which is 6×6. This is the first test for quadrature.

$$A = \frac{1}{2}\det\begin{vmatrix} 1 & 1 & 1 \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{vmatrix} = \frac{1}{2}\text{abs}((x_2y_3 - x_3y_2) - (x_1y_3 - x_3y_1) + (x_1y_2 - x_2y_1)) \tag{10}$$

3.3 Initial Data Integration

A few tests can be conducted on the initializing function. While initializing, we mainly focus on finding the average values of the starting exact data in each cell. The integral of initial data in each cell can be calculated numerically using the quadrature points. This integral of solution values over the cell can be divided by cell area to get the average solution value in the cell. Not surprisingly, the summation of quadrature integral of T in all cells should be equal to the exact integral of the initial data which is definite as presented in equation 11. This is another test for quadrature.

$$\iint_{\text{Domain}} T(x, y)dxdy = 0.628318530638 \tag{11}$$

3.4 Singular Value Decomposition

We have utilized singular value decomposition from Armadillo, to solve the linear system from reconstruction and will write a test function for this method. Of course, any other applicable method can be used. But be sure to test the method beforehand. Suppose the linear system of equation 12. SVD decomposes a real or complex $m \times n$ matrix $\boldsymbol{M}$ into multiplication of equation 13 in which $\boldsymbol{U}$ is $m \times m$ real or complex unitary, $\boldsymbol{\Sigma}$ is $m \times n$ rectangular diagonal with non-negative real numbers on the diagonal, and $\boldsymbol{V}$ is an $n \times n$ real or complex unitary matrix. The solution vector can be found using equation 14

$$\boldsymbol{M}\vec{a} = \vec{b} \tag{12}$$

$$\boldsymbol{M} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T \tag{13}$$

$$\vec{b} = \boldsymbol{M}^\dagger\vec{a} = \boldsymbol{V}\boldsymbol{\Sigma}^\dagger\boldsymbol{U}^T\vec{a} \tag{14}$$

After decomposing $\boldsymbol{M}$ into three matrices $\boldsymbol{U}$, $\boldsymbol{\Sigma}$, and $\boldsymbol{V}$, we should make sure this process is correct so that we can calculate the solution vector $\vec{b}$ using equation 14, confidently. For this purpose, just do the multiplication $\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$ and we should get exactly $\boldsymbol{M}$. Further, $\boldsymbol{U}\boldsymbol{U}^T$ and $\boldsymbol{V}\boldsymbol{V}^T$ should both be identity matrices. We can use the data while reconstructing the solution or just use any

arbitrary matrix $\boldsymbol{M}$. The advantage of the former is being closely related to the solution that we are looking for.

3.5 Reconstruction

The reconstruction process will produce a polynomial of a certain degree k (1st order in our case) which has an accuracy of order $k+1$. However, as explained earlier, the reconstruction does not always produce the exact answer for the linear system of equations. This is due to the system being over/under-determined. In any case, we should be able to integrate the reconstructed solution to get a close approximation to the average value in the cell and its neighbors. As a result, use the reconstruction parameters in cell i and find the average solution value. This value should be close the average T from the numerical solution. Using the same reconstruction parameters of cell i, find the average solution value in the neighboring cells. These average values should also be close to the average T in the numerical solution of the neighbors of cell i.

I have found that for cells with only two neighbors, the reconstruction produces very accurate average values. Amazingly, this was true for cells with only one neighbor inside the domain (there are only four of these in the domain in each corner). However, for cells with three neighbors inside the domain (the over-determined case) we don't get very good results. Expect errors ranging from order of 10^{-6} to 10^{-1}. Further, I tried removing one of the neighboring cells in the reconstruction stencil from the over-determined case (at random) and adding a neighbor of the single neighbor in the under-determined case. This was all done to make the reconstruction stencil have as many degrees of freedom as equations (a 3 by 3 matrix). I didn't get good results out of this method.

3.6 Flux Integration

The velocity domain is a solid body rotation with an angular velocity of π. As a result, at any angle of rotation θ (any time) we have the exact solution using a simple rotation of the axis with equation 15. Now, we can find the exact flux in each cell. Using our quadrature integration, the integral of flux over each cell can be found and compared to the numerical flux summation over three edges of the cell. The exact flux is presented in equation 16. $T(x,y)$, $u(x,y)$, and $v(x,y)$ can be found in equations 2 and 3.

$$\begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix} \tag{15}$$

$$\text{Flux} = -u\frac{\partial T}{\partial x} - v\frac{\partial T}{\partial y} = -10xT(x,y)u(x,y) - 10(y-1)T(x,y)v(x,y) \tag{16}$$

3.7 Time Integration

Testing the correctness of time integration scheme (Runge-Kutta 3 in our case) is straight forward. Just solve a simple ODE with the scheme in use and check the results. This way, the accuracy of the system in time can be confirmed independent of the space discretization scheme. The one dimensional wave equation with periodic boundary setting is a good choice for the ODE which has an exact solution as well.

4 results

The proposed C++ program requires mesh files as well as the quadrature data to be in the same directory as the code. It will produce two output files called *data1d.dat* and *data2d.dat* which contain the 1D data of the solution value at $(r, \theta) = (1, \pi t)$ versus θ and the 2D data of T as well as the velocity components, respectively. As mentioned earlier compiling the program requires the latest version of Armadillo. Use *openblas* and *lapack* libraries with *-DARMA_DONT_USE_WRAPPER* flag to compile. An example compile command is presented as follows.

```
$ g++ -O3 main.cpp -DARMA_DONT_USE_WRAPPER -lopenblas -llapack
```

There are only four parameters that control the solution which are nested in preprocessor macros section, *TEST*, *MESH*, *CFL*, and *THETA*.

- *TEST* controls test functions. Choose this to be *true* or *false* to turn the testing on or off. At the current state of the program turning testing off is the logical choice for solution time.
- *MESH* is an integer in $\{0, 1, 2, 3, 4, 5\}$ corresponding to {test, verycoarse, coarse, medium, fine, veryfine} meshes. Choose whichever mesh you want to solve the problem on.
- *CFL* is pretty self-explanatory. I use a CFL of 1 in most of the simulations.
- *THETA* is the final angle of rotation.

Running the program for the mesh 5 at a CFL number of 1.0 to 90 degrees gives the contours of solution as in figure 6. As seen, only the non-zero part of the solution is shown which lies on the x-axis now. Further, the contour lines are concentric but not circles which is due to the unstructured mesh and reconstruction. As mentioned before, the exact solution is available at any angle of rotation which enables us to find numerical solution error at any stage, confidently. We can utilize this to find the order of accuracy of the numerical method. Suppose that a numerical solution on a specific mesh with the representative mesh size of h_1 is the summation of the exact solutions and an error of order k, as in equation 17. The representative mesh size is the square root of domain area divided by the number of cell as presented by [3]. The same

rule applies for a mesh with smaller representative grid size of h_2 as in equation 18. Let's divide the two equations and rewrite (equation 19). As a result, just divide two consecutive L_2 differences with the exact solution and solve for k in equation 19 to get the order of accuracy.

$$||\hat{u}_{h_1}|| = ||u_e|| + C(h_1)^k \tag{17}$$

$$||\hat{u}_{h_2}|| = ||u_e|| + C\,(h_2)^k \tag{18}$$

$$\frac{||\hat{u}_{h_1}|| - ||u_e||}{||\hat{u}_{h_2}|| - ||u_e||} = \left(\frac{h_1}{h_2}\right)^k \quad \rightarrow \quad k \log\left(\frac{h_1}{h_2}\right) = \log\left(\frac{||\hat{u}_{h_1}|| - ||u_e||}{||\hat{u}_{h_2}|| - ||u_e||}\right) \tag{19}$$

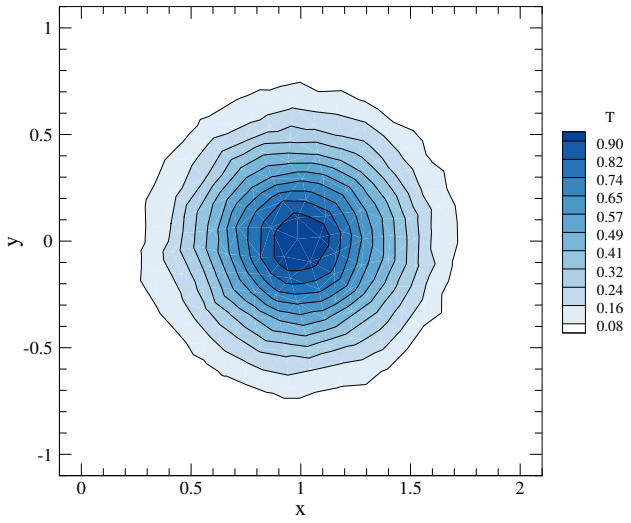

Figure 6: Solution at $\theta = \dfrac{\pi}{2}$ on mesh 5.

Representative mesh size of the numerical grids is presented in table 1. We run the program at CFL number of 1 on all the meshes to $t = 0.5$ and present the L_2 error between the numerical and the exact solutions in table 2. Table 3 presents the exact same test only with CFL=0.5. As seen in these tables, the spatial order of accuracy in the finest mesh is about 1.75 which is close enough to 2. There is something going on in mesh 3 which makes the solution error to go up. I think this issue is related to the reconstruction process. To study the order of accuracy in time we can just do one iteration with several fixed time steps (preferably decreasing with a ratio of 2) on a single mesh and find the solution change in the form of L_2 error. This test is carried out on the very fine and fine meshes with different time steps and the results are presented in

tables 4 and 5. We can see the third order accurate results in time for larger time steps. However, for smaller Δt, the order of accuracy decreases.

Table 1: Representative mesh size of the numerical grids.

Mesh	h
1	1.06066
2	0.514496
3	0.269953
4	0.13194
5	0.0663886

Table 2: Behavior of solution error with mesh refinement at CFL=1

Mesh	L_2 error	Ratio	Order of Accuracy
1	0.0837796		
2	0.040204	2.084	1.015
3	0.0265136	1.516	0.646
4	0.00946992	2.800	1.438
5	0.00283304	3.343	1.757

Table 3: Behavior of solution error with mesh refinement at CFL=0.5

Mesh	L_2 error	Ratio	Order of Accuracy
1	0.0833729		
2	0.0402017	2.074	1.008
3	0.0265135	1.516	0.645
4	0.00946991	2.800	1.438
5	0.00283304	3.343	1.757

Table 4: Behavior of solution change in one iteration with time step refinement for mesh 4

Δt	L_2 error	Ratio	Order of Accuracy
0.4	10.9251		
0.2	1.32078	8.272	3.048
0.1	0.165259	7.992	2.999
0.05	0.0361131	4.576	2.194
0.025	0.0187456	1.926	0.946

Table 5: Behavior of solution change in one iteration with time step refinement for mesh 5

Δt	L_2 error	Ratio	Order of Accuracy
0.2	2.03208		
0.1	0.242431	8.382	3.067
0.05	0.0387887	6.250	2.644
0.025	0.0149867	2.588	1.372
0.0125	0.00760248	1.971	0.979

Another important parameter to study in the present problem can be the solution at $r = 1$ versus θ. For this purpose, the solution at $r = 1$ on all meshes is extracted and plotted vs. θ (up to $\theta = \frac{\pi}{2}$) in figure 7 with a constant time step of 0.000125. The exact solution is indicated with a red dashed line in this figure. As seen, the solution on all meshes decay as time passes by, with the most and least decay related to the very coarse and very fine meshes, respectively. The reduction in solution value is related to discretization error which decreases with mesh refinement.

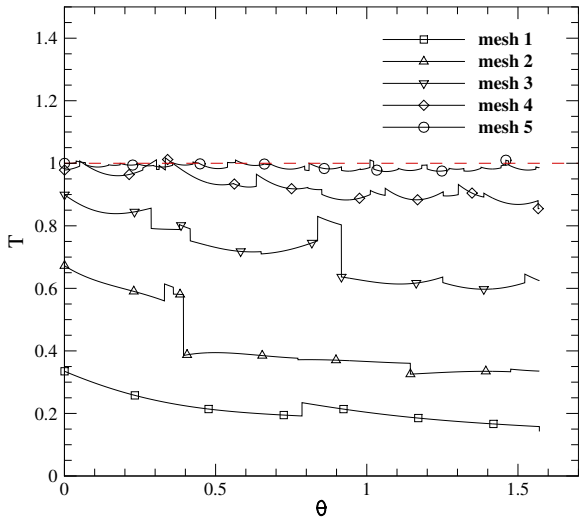

Figure 7: Solution at $r = 1$ vs. θ at a constant time step of 0.000125.

The same test is conducted, this time on a single mesh at different time steps. The results are presented in figures 8 and 10 for meshes 1 and 3, respectively. As seen in these figures, refining Δt only shifts the solution decay process to

left (earlier decay). Other than that, some oscillations form in smaller time steps along the jumps in the solution. What are these solution jumps? To find out, I tracked the $(r, \theta) = (1, \pi t)$ point to check the position of reconstruction in relation to mesh cells. Turns out, the jumps in solution that are seen in figures 8 and 10 happen when the reconstruction point moves across cell edges. The path that $r = 1$ takes is depicted with a red solid line in comparison to the grid in figures 9 and 11. Small red circles show the points where the path crosses cell faces, which I call intersection points. The angle of intersection for each mesh is depicted with dashed black lines in corresponding figures 8 and 10. As seen here, clearly, crossing cell faces causes the solution at $r = 1$ to have a discontinuity in time. I suspect the reconstruction process causing this problem. As to why sometimes the solution increases while crossing the cell face, and sometimes decreases, once again, I suspect the reconstruction stencil.

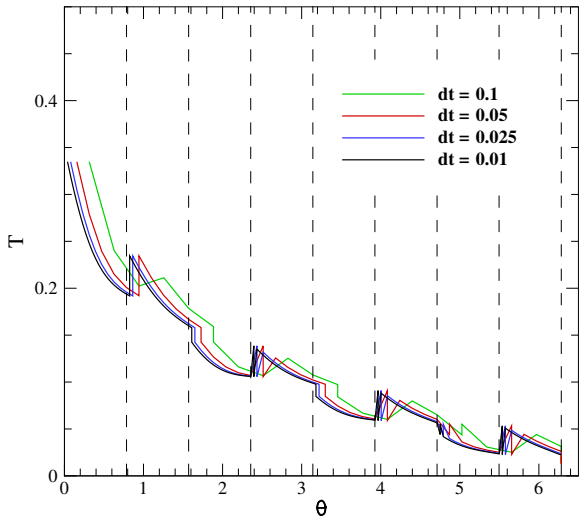

Figure 8: Solution at $r = 1$ vs. θ at different time steps for mesh 1.

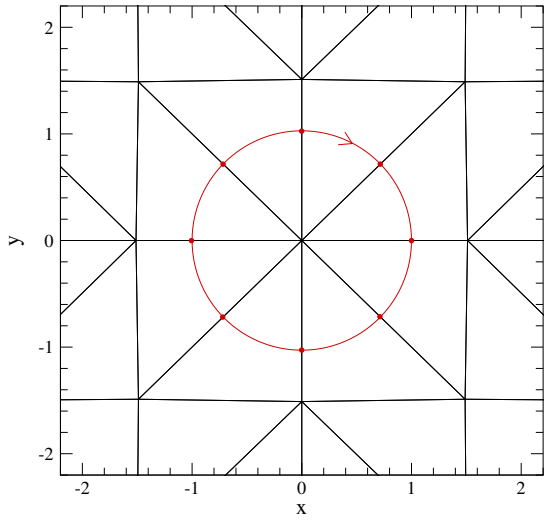

Figure 9: Close up of the movement path of $r = 1$ over mesh 1.

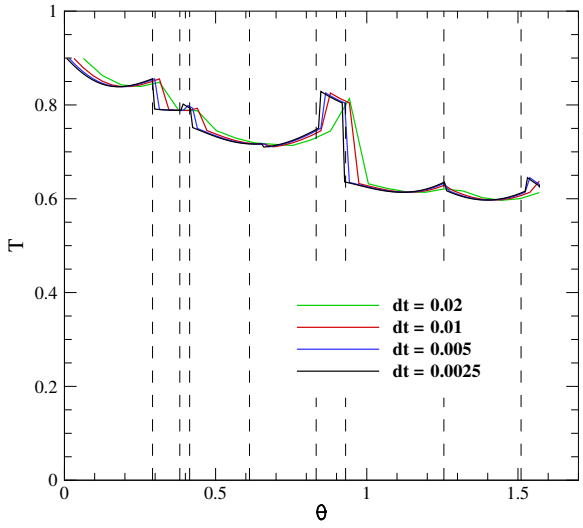

Figure 10: Solution at $r = 1$ vs. θ at different time steps for mesh 3.

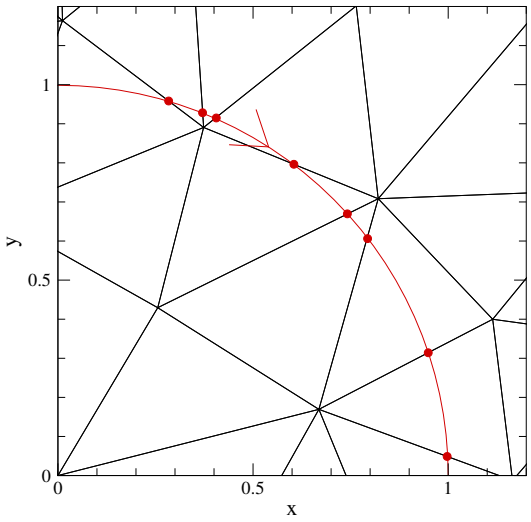

Figure 11: Close up of the movement path of $r = 1$ over mesh 3.

Now, let's modify one of the meshes and see if we can get a more smooth result for the solution at $r = 1$. For this purpose, I changed the very coarse mesh to get a new mesh as presented in figure 12. Comparing this new mesh with the original version in figure 9 reveals the differences. The solution over the new mesh is conducted with a constant time step of 0.00125 and the result is presented in figure 13. As seen here, crossing the first and second edges in this setting gives a much smoother solution. I think the length of each edge might have an effect on the solution. Let's modify the mesh in a way that all the edges in the path of $r = 1$ curve have the same length with a 45 degree angle increment. This second modification of the very coarse mesh is presented in figure 14. The numerical solution is conducted on this mesh for a constant time step of 0.000125 and the results are presented in figure 15. As seen here, we have a much smoother transition in the solution value at $r = 1$ compared to the original very coarse mesh. Figure 16 shows the solution at $r = 1$ on the original mesh 1 and its two modifications. As seen here, the second modification of mesh 1 depicts the least amount of decay in solution. As a result, edges with equal lengths are good for our solution. Another possibility is the point where the path cuts the edge. Maybe, going through the center point of each edge introduces the least amount of decay in the solution.

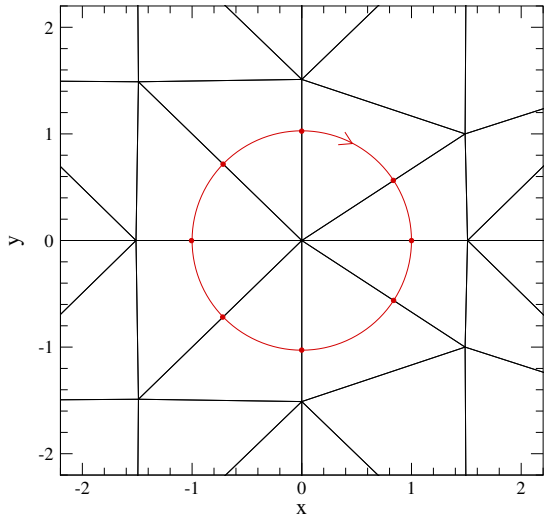

Figure 12: Close up of the movement path of $r = 1$ over the first modification of mesh 1.

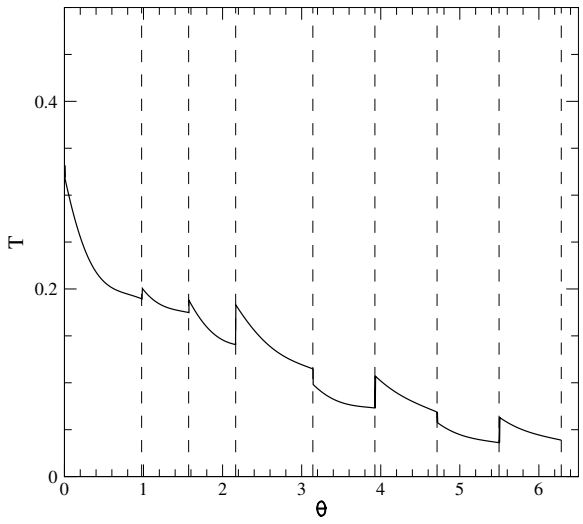

Figure 13: Solution at $r = 1$ vs. θ with $\Delta t = 0.00125$ for the first modification of mesh 1.

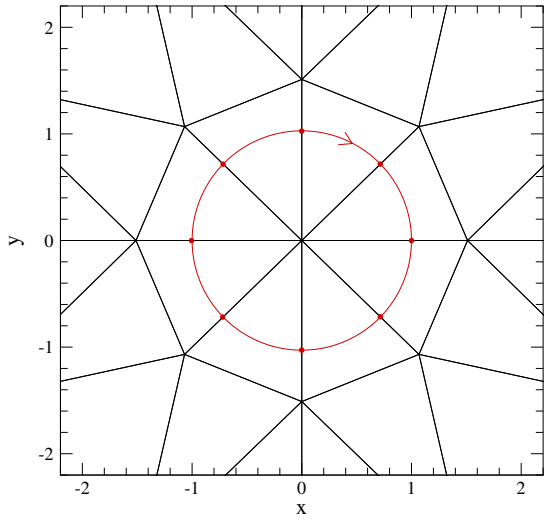

Figure 14: Close up of the movement path of $r = 1$ over the second modification of mesh 1.

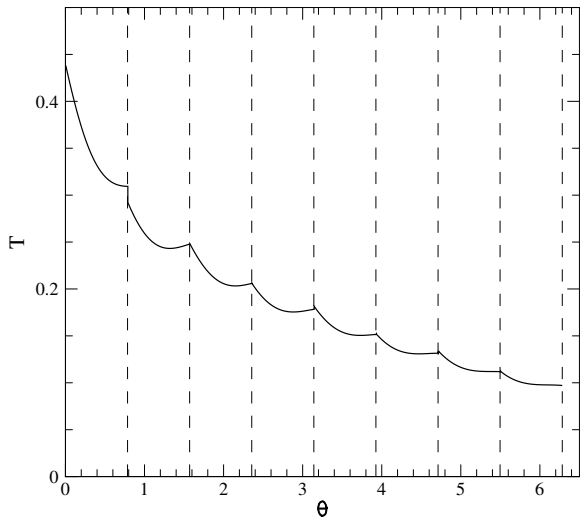

Figure 15: Solution at $r = 1$ vs. θ with $\Delta t = 0.00125$ for the second modification of mesh 1.

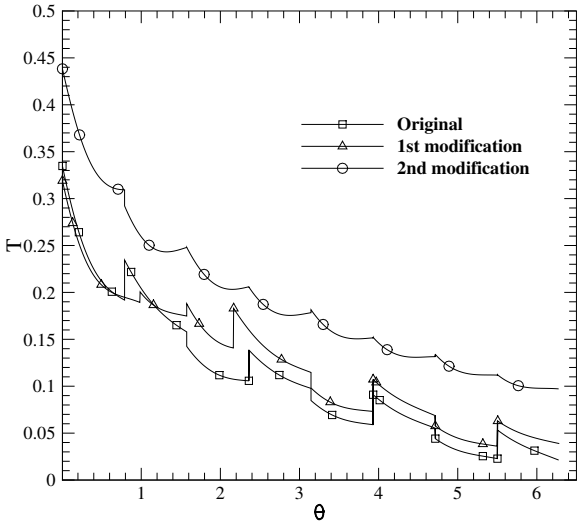

Figure 16: Solution at $r = 1$ vs. θ with $\Delta t = 0.00125$ on the original mesh 1 and its two modifications.

References

[1] Carl Ollivier-Gooch. Lecture notes in mech 511, February 2020.

[2] Mark A. Taylor, Beth A. Wingate, and Len P. Bos. Several new quadrature formulas for polynomial integration in the triangle, 2005.

[3] Procedure for Estimation and Reporting of Uncertainty Due to Discretization in CFD Applications. *Journal of Fluids Engineering*, 130(7), 07 2008.