Computer Project 3

AA 543 - Computational Fluid Dyanmics
University of Washington

Mishaal Aleem
March 17, 2015

Contents

Introduction

Over the course of the Winter 2015 quarter, a modular MATLAB language computational fluid dynamics (CFD) program was created. This program allows users to first generate a 1-dimensional grid, then creates a function over the grid with specified boundary conditions, and finally solves either the linear advection equation or Burgers' equation with specified parameters.

3.1 Grid Generator

The fist mode in the program produces a 1-dimensional grid. The grid can either be uniformly spaced or clustered at a single location based upon user selection.

Uniform

For the uniform grid, the required inputs are the range starting value (x_{min}), range ending value (x_{max}), and the total number of grid points (i_{max}). The grid location points are then calculated using Eq. 1.

$$x_1 = x_{min}, x_2 = x_1 + dx, x_3 = x_2 + dx, ...x_{max} = x_{max-1} + dx \tag{1}$$

$$\text{where } dx = \frac{x_{max} - x_{min}}{i_{max} - 1}$$

Each of the grid points is corresponded to an integer grid index i ranging from 1 to i_{max}. At this point, the program outputs the vectors of grid indices (i), the vector of grid location points (x), and the vector of grid spacing (Δx) where each Δx value is defined by Eq. 2. For the uniform grid, each value of Δx is, of course, the same by definition.

$$\Delta x_i = x_{i+1} - x_i \tag{2}$$

Clustered

For the clustered grid, the required inputs again include x_{min}, x_{max}, and i_{max}, as well as the clustering location (x_c) and the degree of clustering (ρ). The degree of clustering is defined by Eq. 3 as the ratio of the maximum to minimum grid spacing.

$$\rho = \frac{(\Delta x)_{max}}{(\Delta x)_{min}} \tag{3}$$

Once the inputs are received, the program calculates the clustered grid point locations by solving a system equations. The equations are developed based on the understanding that because the grid is clustered, Δx will be parabolic and therefore the x vector, its integral, will be a cubic of the form $ai^3 + bi^2 + ci + d$.

First, the clustering index (i_c) is rewritten in terms of the polynomial constants using Eq. 4 by knowing that the derivative of Δx will be equal to 0 at the clustering point and therefore the second derivative of the x cubic will be 0.

$$x'' = 0 = 6ai_c^3 + 2b \rightarrow i_c = -\frac{b}{3a} \tag{4}$$

Next, it is noted that the maximum value of Δx will correspond to location x_{min} if the clustering location is right of center or x_{max} if the clustering location is left of center because Δx is a parabola. A simple "if/else" statement is used to select the appropriate boundary to correspond to $(\Delta x)_{max}$. Additionally, the value $(\Delta x)_{max}$ is rewritten using the degree of clustering definition, $(\Delta x)_{max} = \rho(\Delta x)_{min}$.

Then, the following five principles are using to develop Eqs. 5, 6, 7, 8, and 9 which are solved simultaneously for a, b, c, d, and x_{min}:

- The value of x at the first index (1) is equal to x_{min}.
- The value of x at the last index (i_{max}) is equal to x_{max}
- At the cluster index, Δx will equal $(\Delta x)_{min}$
- At the cluster index, x will equal the cluster location x_c
- At either $i = 1$, or $i = i_{max}$, depending on the aforementioned if/else statement, the value of Δx will equal $(\Delta x)_{max}$ which means it will equal $\rho(\Delta x)_{min}$

$$x_{min} = a + b + c + d \tag{5}$$

$$x_{max} = ai_{max}^3 + bi_{max}^2 + ci_{max} + d \tag{6}$$

$$(\Delta x)_{min} = 3ai_c^2 + 2bi_c + c \tag{7}$$

$$x_c = ai_c^3 + bi_c^2 + ci_c + d \tag{8}$$

$$\rho(\Delta x)_{min} = 3ai_{1/max} + 2bi_{1/max} + c \tag{9}$$

Equations 5-9 are solve for a, b, c, d, and x_{min}. With these equations solved, the cubic $x = ai^3 + bi^2 + ci + d$ is defined. The grid point location vector x is generated by solving the cubic for integer values of $i = 1$ to $i = i_{max}$. The Δx vector is again generated per Eq. 2.

A key point to note is the inherent error when generating a clustered grid. There will always be an error when describing a non-uniform grid because the grid is truly defined by a continuous function but is being modeled by finite, discretized points. For a better grid, increasing the total number of points (i_{max}) decreases the error. As an example, a plot of error percentage in the degree of clustering as a function of grid points is presented in Fig. 1. A strong trend of decreasing error with increasing grid points is visible. For this particular grid, a grid from 0 to 10 clustered at 5 with degree of clustering of 3, the error is is 1.34% with 100 grid points but is decreased to just 0.13% with 1000 grid points.

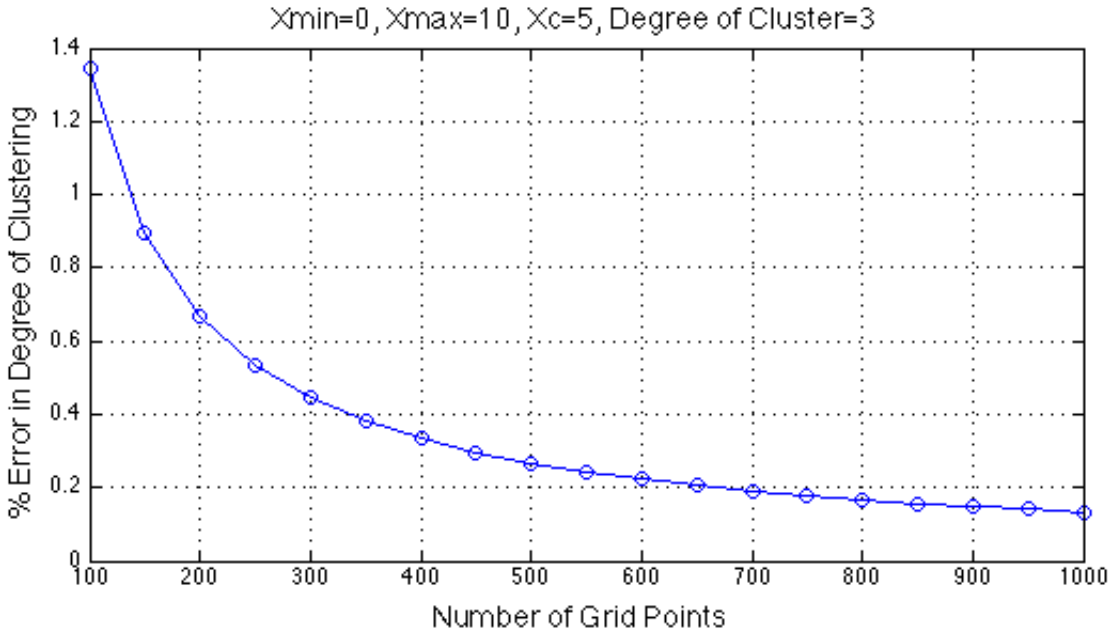


Figure 1: Error in Clustered Grid

3.2 Initial Function Generator

The second mode in the program generates a function $u(x)$ with specified boundary conditions. The user can select either a square wave, a Gaussian function, or a sine wave.

If the user selects a square wave, they simply enter the width of the wave (x_{width}) and the location of the wave (x_{peak}) and it is assumed the wave is centered about the peak at a value of 1

and the value is 0 everywhere else. If the user selects a Gaussian function, the same inputs are required as for square wave and then the function is simply defined by Eq. 10. If a user selects a sine wave, the simply specify if $u(x) = \sin(x)$ or $u(x) = -\sin(x)$.

$$u(x) = 1 + e^{-10/x_{width} * ((x - x_{peak}) * (x_{max} + x_{min})/(x_{max} - x_{min}))^2} \tag{10}$$

From these definitions and the grid location vector x from the grid generation mode, an initial function vector u is generated. Next, the user specifies the boundary conditions. These boundary conditions are either user-specified Dirichlet, program-specified Von Neumann, or periodic. Note that the user has the option to set one boundary as Von Neumann and the other as Dirichlet, if desired.

Dirichlet Boundary Conditions

For Dirichlet boundary conditions, the user can simply select values for both the first and last element of the u vector, i.e. $u(1) = u(x_{min})$ and $u(end) = u(x_{max})$.

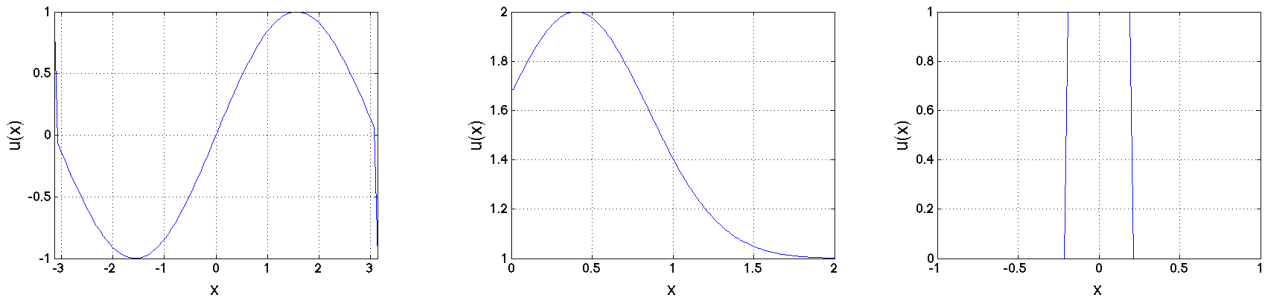
Von Neumann Boundary Conditions

For the Von Neumann boundary conditions, the gradient of the function is defined as 0 at the boundaries. This is achieved by simply setting the first and last elements of the u vector such that $u(1) = u(2)$ and $u(end) = u(end - 1)$, making the derivative at the boundaries equal to 0.

Periodic Boundary Conditions

For periodic boundary conditions, the last element of the u vector is set as equal to the first element, i.e. $u(end) = u(1)$. Additionally, a ghost-point is added for periodicity so that not only do the values at the boundary match, but so do the gradients. This is implemented by extending the u vector by element, i.e. $u(end + 1) = u(2)$.

An example of each of the functions, each with a different boundary condition type, is presented in Fig. 2.



Sine Wave with Dirichlet Boundary Conditions

Gaussian Function with Von Neumann Boundary Conditions

Square Wave with Periodic Boundary Conditions

Figure 2: Initial Functions Generated with various Boundary Conditions

After the initial function has been generated, the user can choose to solve either the linear advection equation or Burgers' equation. Each of these 2 two solver modes are discussed in the two following sections.

3.3 Linear Advection Equation Solver

The Linear Advection Equation Solver mode solves the time-dependent linear advection equation, given by Eq. 11, using the explicit Lax-Wendroff method.

$$\frac{\partial u}{\partial t} + a\frac{\partial u}{x} = 0 \tag{11}$$

For this mode, the user must input the value of the CFL number, defined by Eq. 12, and the final time (or choose to stop at $t = i_{max}\frac{\Delta t}{4}$).

$$\text{CFL} = \frac{a\Delta t}{\Delta x} \tag{12}$$

For the purpose of this program, the wave speed (a) is assumed to be 1 and thus the time step is simply $\Delta t = \text{CFL}\Delta x$. For a clustered grid, the minimum value of the Δx vector is used as that creates the most conservative restriction on Δt.

Once the Δt value is calculated, the Lax-Wendroff algorithm, given by Eq. 13, is implemented continuously, incrementing time with a value Δt at each step until the final time is reached. At each time step, the function is plotted and output to the screen allowing the user to easily visualize the propagation of the wave.

$$u_j^{n+1} = u_j^n - \frac{\text{CFL}}{2} + \frac{\text{CFL}^2}{2}(u_{j+1}^n - 2u_j^n + u_{j-1}^n) \tag{13}$$

For stability, the CFL number must be greater than or equal to -1 and less than or equal to 1, as given by the Von Neumann stability criteria for the Lax-Wendroff algorithm. Additionally, because the Lax-Wendroff algorithm is being used, there is dispersion in the solution that varies with CFL number. Specifically, smaller magnitudes of CFL number cause greater dispersion.

An example solution from the Linear Advection Equation Solver is presented in Fig. 3. The plot shows the wave at various times in a single plot. This plot was generated by propagating the wave until the 5 specified time values and plotting the output from each run together in a single plot.

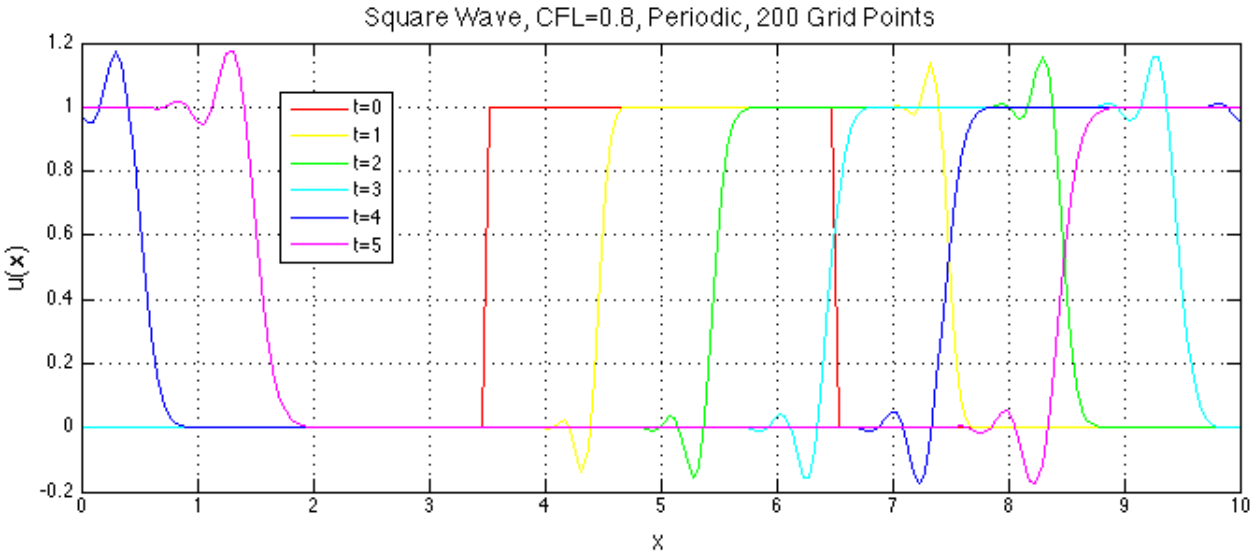


Figure 3: Example of Linear Advection Solution

Note that for the Linear Advection Equation Solver, the wave continues to propagate across the screen if the function is periodic, whereas it only propagates once across (and does not return, similar to the manner showed in Fig. 3) if the boundary is non-periodic. This is due to the definition of periodicity.

3.4 Burgers' Equation Solver

The alternate solver mode that the user may select is the Burgers' Equation Solver. This mode solves the viscous Burgers' equation which is presented in conservative form in Eq. 14.

$$\frac{\partial u}{\partial t} + \frac{\partial \frac{1}{2}u^2}{\partial x} = \nu \frac{\partial^2 u}{\partial x^2} \tag{14}$$

For this mode, the user inputs the viscosity coefficient (ν), the CFL number, and the final time. Again, a wave speed of 1 is assumed and thus the time step is simply $\Delta t = \text{CFL}\Delta x$ and for a clustered grid the minimum value of the Δx is used.

The MacCormack algorithm is used to solve the viscous Burgers' equation with the source term given by Eq. 15, the predictor given by Eq. 16, and the corrector given by Eq. 17. Note that $\bar{S}$ means that the source term is evaluated with the predictor $\bar{u}$.

$$S = \nu \left(\frac{u_{j+1} - u_j}{(x_{j+1} - x_j)(x_j - x_{j-1})} - \frac{u_j - u_{j-1}}{(x_j - x_{j-1})^2} \right) \tag{15}$$

$$\bar{u}_j = u_j^n - \frac{\Delta t}{x_j - x_{j-1}} \left(\frac{1}{2}u_j^2 - \frac{1}{2}u_{j-1}^2 \right) + \Delta t S_j^n \tag{16}$$

$$u_j^{n+1} = \frac{1}{2}(u_j^n + \bar{u}_j) - \frac{\Delta t}{2(x_{j+1} - x_j)} \left(\frac{1}{2}\bar{u}_{j+1}^2 - \frac{1}{2}\bar{u}_j^2 \right) + \frac{\Delta t}{2}\bar{S}_j \tag{17}$$

An example solution from the Burgers' Equation Solver is presented in Fig. 4. The plot is of a negative sine wave based upon a grid clustered around the shock location, $x = 0$, with a degree of clustering equal to 3.

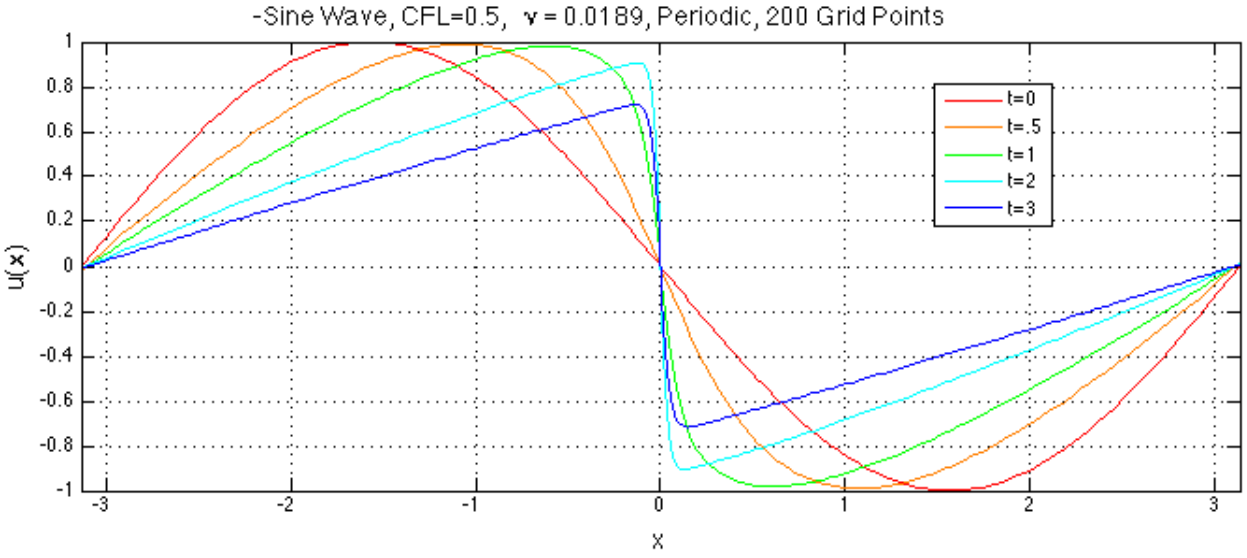


Figure 4: Example of Burgers' Equation Solution

Stability

Because of the nature of the viscous Burgers' equation and the use of an explicit algorithm, stability considerations are of great importance for the Burgers' Equation Solver. Specifically, an appropriate combination of grid points, CFL number, and viscosity coefficient must be made.

By running the program for various combinations of the three aforementioned variables, it was found experimentally that the stability criteria for the Burgers' Equation Solver is given by Eq. 18. In the case of a non-uniform grid, the limiting Δx for a clustered grid will simply be the

most conservative case, i.e. minimum value of (Δx). Note also that the CFL number must be given such $0 < \text{CFL} < 1$.

$$\nu \leq \frac{1}{2\text{CFL}}\Delta x \tag{18}$$

Lessons Learned

Various lessons were learned over the course of developing Computer Project 3. These ranged from insights about the solution algorithms to coding optimization. Some of the particularly meaningful lessons learned are discussed following.

Topic Overlap

A very simple yet important lessons learned while developing this project came from the fact that the computer project was developed in pieces which built upon each other. This helped me to understand the basic structure of coding for CFD. First, a grid needs to be generated. From this grid can be generated a function. Finally, the function can be used in a fluids equations and those equations can be solved. In this project, each of the solver modes are dependent on the function generation mode, which is itself dependent on the grid generation mode. However, it was also learned that not all processes build upon the previous; the Linear Advection Equation Solver and the Burgers' Equation Solver are at equal levels of computation, the user just selects either one of the other. This structure of how the modes build upon each other is presented visually in a simple program flowchart in Fig. 5.

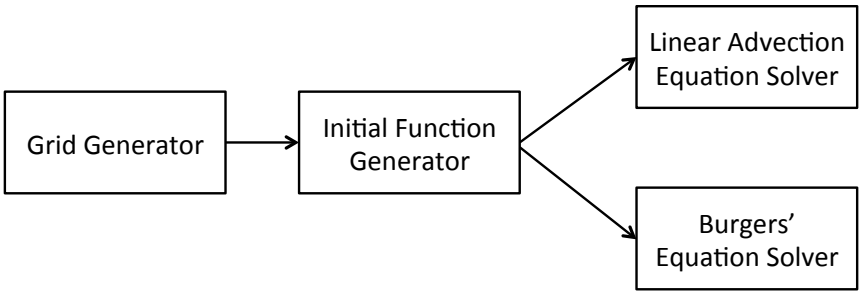


Figure 5: Programs Flowchart

Grid Selection and Solution Quality

One of the most important lessons learned while developing this computer project was implications of the initial grid. The first mode, where the user generates the grid, is a key step because, as aforementioned, each additional mode in the program builds upon the grid.

As described with Fig. 1 previously, by having more grid points, there is less error in a clustered grid. However, this is at the expense of time. More grid points means the grid generator takes more time to define the grid. Additionally, both the Linear Advection Equation Solver and the Burgers' Equation solver are much slower as the number of grid points is increased.

Another reason grid selection is so important is because it corresponds directly to stability. This is because the CFL number, and in the case of the Burgers' Equation Solver the viscosity coefficient, too, corresponds to the stability criteria. Both of these constants are limited by the grid spacing.

Furthermore, the grid can also vary the quality of the solution. This was particularly noticeable with the Burgers' Equation Solver. If the grid was clustered at the shock location, the shock resolution was improved. This points to an important CFD principle: if it is known where

interesting problem phenomena are going to occur, clustering the grid provides more information at that particular location and thus better simulates the phenomena of interest.

Stability

One of the key concerns with any CFD code is stability. Lack of stability refers to growth of computation errors that cause a large deviation of final answer from the true solution. For both the Lax-Wendroff and the MacCormack algorithm (used in the two solver modes) there are specific stability criteria because the algorithms are explicit.

While the chosen algorithms were sufficient for this program developed for an introductory CFD course, they are not particularly robust. Small changes in input data can cause different behavior and potential instability. For example, changing the CFL number in the Linear Advection Equation Solver from 1 to 1.2 causes instability. Changing the CFL number from 1 to 0.8 make the solution stable, though much more dispersive. Similarly, changing the viscosity coefficient too much from an acceptable value can also cause the Burgers' Equation Solver to fail due to instability. These issues make the range of applicability for this code quite limited.

The program as a whole could be improved by using implicit algorithms such as the θ-method. The relevant simplicity of the explicit algorithms, however, makes coding them much simpler. Because of their limitations, these explicit algorithms should only be implemented when it is known the conditions of the problem will not approach values that could cause errors in the solution.

Program Performance

Another important lesson learned was about program performance When developing the code for generating a clustered grid, initially a bisection method was used rather than a 5 variable system of equations. However, this bisection method had two performance related issues: an acceptable tolerance value had to be defined in order to stop the bisection method loop and the method was very slow.

For example, generating a particular clustered grid with degree of cluster equal to 4 in the program as it is now took only 5 seconds whereas with the bisection method took over 15 seconds. This time could be decreased by increasing the error tolerance (the aforementioned example had an error tolerance of 10^{-6} on the final degree of cluster when using the bisection method). However it was not specified by project instructions to have a user input of error tolerance. A truly "acceptable" tolerance, however, is highly problem-specific. By using the system of equations method rather than the bisection method, the program became significantly faster and more robust. It also avoided the issue of having to have the user input an acceptable error tolerance.

The lesson this provided connects directly to real CFD applicability. It is ideal to write a computer program that executes the code and solves the problem in as little computing time as possible and is as robust as possible. In CFD applications, increased computing time corresponds directly to increased costs. Also, requiring additional user inputs that may not be necessary decreases ease of use of the code. Using good coding practices considerably increases the quality of the program.

Conclusion

Over the course of the Winter 2015 quarter, Computer Project 3 was developed. This program, written in MATLAB, allows the user to first develop a grid, then initiates a function (with

user-specified boundary conditions) over the grid and finally solves either the linear advection equation or Burgers' equation.

The first mode generates a grid, either uniformly spaced or clustered at a single location. To define the grid, the user must input the range starting point, range ending point, total grid points, and in the case of a clustered grid, the clustering location and degree of clustering. For the clustered grid, the grid is assumed to be defined by a cubic equation (which implies a parabolic derivative indicating minimal spacing at the cluster point). The cubic equation is solved for using a system of five equations. There is inherently an error in the clustered grid due to it attempting to describe a continuous equation with a finite set of points.

The next mode initiates a either a sine, Guass, or square wave function. For the case of the Guass or square wave, the user inputs the width of the wave and the peak of the wave. For a sine wave, the user selects a positive or negative sine wave. Next, the user defines the boundary conditions. If periodic boundary conditions are selected, the program assumes that the value at the last node of the function is equivalent to the value at the first node. Then, to ensure periodicity, the program adds a ghost node at the end of function vector that is equivalent to the value of function at the second node. In other boundary condition cases, the user can select to define the value at the boundary or assume the gradient at the boundary is zero.

Next, the user selects either the Burgers' Equation Solver or the Linear Advection Equation Solver mode. The former solves the viscous Bugers' equation using the MacCormack algorithm with a user specified CFL number, viscosity coefficient, and final time. The method is stable for CFL numbers from 0 to 1 and an appropriately low viscosity. For the Linear Advection Equation Solver, the Lax-Wendroff algorithm is used to solve the time-dependent linear advection equation with a user input CFL number and final time. This method is stable for CFL numbers from -1 to 1, though the method does have more and more dispersion for smaller magnitudes of CFL number.

Creating this program served as an introduction into CFD. Through the development of the program, many important lessons were learned regarding grid selection, solution quality, computing performance, and more. This program was a highly useful introduction to CFD because, despite its relative simplicity, it captures many key details regarding the fundamentals of CFD.

References

[1] Eberhardt, D.S. and Shumlak, U. AA543 Computational Fluid Dynamics I, University of Washington, Seattle, WA, 2013.