

Lab / Assignment 3: Differential Equations

Alex Matheson, Austin Nhung

April 12, 2018

1 Introduction

Differential equations form the backbone of most physics sub-disciplines. Substantial work has been put into analytically evaluating these equations. With the exception of some symbolic languages, computers cannot employ the continuity that analysis requires; methods must be used that convert continuous differential equations to some discrete form. Finite differences uses the formal definition of the derivative to create different equations called 'stencils' that relate a value at a point to values at other discrete points. This lab explored many different stencils that were devised to minimise error, overcome discrete limitations, and counteract phenomena that may artificially arise as a side effect of discretisation.

2 Methods

2.1 Warm-Up

Finite differences allows one to set up an equation relating the value of a variable at a point to the values in the surrounding discrete locations. For instance, a two dimensional Laplace equation of the form $u_{xx} + u_{yy} = 0$ would have the form $u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{i,j} = 0$. By re-writing this equation in terms of $u_{i,j}$, we can have a series of equations that define the value of each point relative to that of its neighbors. If boundary values are supplied, this yields $n \times m$ equations for a region divided into $n + 2$ x-points and $m + 2$ y-points. The **direct method** is used to solve this system of equations exactly, by solving the series of equations using

matrix methods such as Gaussian methods or L-U decomposition. Direct methods however may not always be desirable. This is because the matrix solving algorithms used may be slow and inefficient. Additionally, some data sets may not behave well with certain matrix methods computationally. As a result of this, iterative methods have been developed to address these shortcomings.

Iterative methods involve "guessing" a solution to the equation, and then continually refining the results until they converge to a solution (note that convergence tests may be required to ensure that the system converges to the correct solution). Given a series of boundary values, the inner values may then all be guessed. Next, finite differences may be used to determine an equation that defines the value of a cell based on its neighbors. Whereas the direct method tried to simultaneously determine all cells, the iterative starts with a guess, then uses the finite difference equation to create a new "generation" of grid based on the previous grid's values. In the earlier example, this equation would be $u_{i,j}^{k+1} = (u_{i+1,j}^k + u_{i-1,j}^k + u_{i,j+1}^k + u_{i,j-1}^k)/4$ where k signifies the current iteration of the system. This method continues until certain criteria are met, though the final result will have some associated error. The method benefits, however, by generally performing faster than direct methods, especially when the guess is close to the actual answer.

There are different ways to implement iterative methods. The Jacobi method is closest to what has been described so far. In the Jacobi method, the grid is solved at some iteration k . Once the iteration is complete, the system is compared to exit conditions, and then the next iteration is generated. The

Gauss-Seidel method considers a different approach. We know that the more iterations take place, the more likely a value at a particular point is to be correct. For the first point in a grid we consider, we use all the neighbor values from the previous iteration. In the Jacobi method, the next point also uses all the neighbor points from the previous iteration. The Gauss-Seidel method realizes that for the second point, we already have one "updated" point that should be closer than the previous iteration, so we use the updated neighbor when available instead of the previous iteration to approach the solution more quickly. In practice, the Gauss-Seigle method uses a checkerboard scheme, where points alternate between using only previous iteration values and using a mixture of previous - and current iteration values. This scheme maximizes the number of updated points used in the calculation, speeding up the algorithm. The Jacobi method is more intuitive to implement, since it completes one iteration at a time. The Jacobi method can also be split apart into different intersecting independent grids for parallelization, whereas the Gauss-Seigle method cannot be parallelized easily.

The aforementioned methods can be used to solve numerous physical systems. Many differential equations have general forms that have been given names over time.

Differential equations may be classified according to the discriminant of the generalized form $au_{xx} + bu_{xy} + cu_{yy} + du_x + eu_y + fu + g = 0$. A zero discriminant is classified as parabolic, whereas positive and negative are hyperbolic and elliptic respectively. Based on this, three sample PDEs may be considered. The equation $u_{tt} = 2u_{xx}$ is hyperbolic with a discriminant of 8, $u_{xx} + u_{yy} = 0$ is elliptic with a discriminant of -4, and $u_t - u_{xx}$ is parabolic with a discriminant of 0.

There are three different first order FD operators that may be used on a function. For the function

$u(x) = x^2$, the forward FD is as follows:

$$\begin{aligned}\partial_x^+ u &= \frac{u(x+h) - u(x)}{h} \\ \partial_x^+ u(3) &= \frac{u(3+0.1) - u(3)}{0.1} \\ \partial_x^+ u(3) &= \frac{9.61 - 9}{0.1} \\ \partial_x^+ u(3) &= 6.10\end{aligned}\tag{1}$$

With a finer grid resolution, the result becomes:

$$\begin{aligned}\partial_x^+ u &= \frac{u(x+h) - u(x)}{h} \\ \partial_x^+ u(3) &= \frac{u(3+0.05) - u(3)}{0.05} \\ \partial_x^+ u(3) &= \frac{9.3025 - 9}{0.05} \\ \partial_x^+ u(3) &= 6.0500\end{aligned}\tag{2}$$

In both cases, the result is near the expected result of 6, but is off by a single factor of h . This matches the error of order $O(h)$ for the forward method.

Another FD method is the central method. The same two examples may be computed using this method.

$$\begin{aligned}\partial_x u &= \frac{u(x+h) - u(x-h)}{2h} \\ \partial_x u(3) &= \frac{u(3+0.1) - u(3-0.1)}{2 * 0.1} \\ \partial_x u(3) &= \frac{9.61 - 8.41}{0.2} \\ \partial_x u(3) &= 6.00\end{aligned}\tag{3}$$

$$\begin{aligned}\partial_x u &= \frac{u(x+h) - u(x-h)}{2h} \\ \partial_x u(3) &= \frac{u(3+0.05) - u(3-0.05)}{2 * 0.05} \\ \partial_x u(3) &= \frac{9.3025 - 8.7025}{0.1} \\ \partial_x u(3) &= 6.000\end{aligned}$$

As expected according to the text, the central method was more accurate. The result matched the analytical solution exactly.

Name	Form	Order
Poisson's equation	$\nabla^2 u = f(x, y, z)$	2
Laplace's equation	$\nabla^2 u = 0$	2
3-Dimensional Wave equation	$u_{tt} = c^2 \nabla^2 u$	2
1-Dimensional Heat equation	$u_t - \alpha u_{xx} = 0$	2
1-Dimensional Wave equation	$u_{tt} = c^2 u_{xx}$	2
Ginsburg-Landau equation	$A_t = A_{xx} + \sigma A - A ^2 A$	2
Korteweg-de Vries equation	$u_t + u_{xxx} - 6uu_x = 0$	3

Table 1: Different types of differential equations, their general forms, and order.

References

- [1] Ouyed and Dobler, PHYS 581 course notes, Department of Physics and Astrophysics, University of Calgary (2016).
- [2] W. Press et al., *Numerical Recipes* (Cambridge University Press, 2010) 2nd. Ed.
- [3] T. O'Brian, "Properties of Pulsars", University of Manchester Jordell Bank Observatory, accessed at <http://www.jb.man.ac.uk/distance/frontiers/pulsars/section1.html>.
- [4] D. Hathaway, "The Sunspot Cycle", NASA, accessed at <https://solarscience.msfc.nasa.gov/SunspotCycle.shtml>

There are many different finite difference schemes that may be employed. Table ?? shows many of the different schemes covered in the course.

Of the schemes covered, they could be divided into two types: explicit and implicit. Explicit schemes may be solved cell-by-cell. That is, a cell at time $l + 1$ could be found using a single equation, since it was dependent only on cells at time l . Implicit schemes on the other hand, require that the values in all cells be solved simultaneously, since a variable at time $l + 1$ will be dependent on the values in other cells at time $l + 1$ in such schemes. Implicit schemes therefore require the use of linear algebra to solve equations simultaneously, usually using some matrix-solving method such as LU decomposition. Implicit schemes have the benefit of always

being stable, meaning one is free to chose time and space resolutions as one wishes. The downside of the implicit method is that simultaneous solving is more computationally expensive. One needs to weigh the trade off between run-time and resolutions when deciding on a method to use.

In the previous paragraph, stability of a discrete approximation was touched upon. A simulation becomes unstable if its values begin to explode contrary to physical conservation laws. Given enough time, the values of the variable of interest will explode to infinity. This instability is the result of selecting certain grid spacings that prevent proper transfer of information. To determine when a scheme becomes unstable, Von Neumann stability analysis may be applied. Von Neumann stability analysis takes advantage of the Fourier representation of a function. Our value u is some linear sum of waves. At some future time u^{l+1} our system should be the same set of waves, but evolved. This may be represented as a constituent wave being multiplied by some factor A . If $A > 1$, then after some period of time, that constituent wave will dominate the variable, and explode to infinity. The Van Neumann method therefore determines the domain of such amplitudes, and determines under what conditions $A > 1$. By plugging in the Fourier representation of a variable into our finite difference stencil, we may re-arrange to determine the value of A with respect to the Courant number.

Name	FD Scheme	Order	CFL
Forward Euler	$u_k^{l+1} = u_k^l + r(u_{k+1}^l - u_k^l)$	1	$r \leq 1$
Upwind	$u_k^{l+1} = u_k^l - r (u_k^l - u_{k-1}^l)$	1	$r \leq 1$
Leap Frog	$u_k^{l+1} = u_k^{l-1} + r(u_{k+1}^l - u_{k-1}^l)$	2	$r \leq 1$
Lax-Wendoff	$u_k^{l+1} = u_k^l - \frac{r}{2}(u_{k+1}^l - u_{k-1}^l) + \frac{r^2}{2}(u_{k+1}^l - 2u_k^l + u_{k-1}^l)$	2	$r \leq 1$
Backward Euler	$u_k^{l+1} = (1+r)^{-1}(u_k^l + ru_{k+1}^l + 1)$	1	None
Crank-Nicholson	$u_k^{l+1} = u_k^l + \frac{r}{2\Delta x}(u_{k+1}^{l+1} - 2u_k^{l+1} + u_{k-1}^{l+1} + u_{k+1}^l - 2u_k^l + u_{k-1}^l)$	2	None

Table 2: Different FD schemes, their respective orders and stabilities. The variable r has been used to represent the Courant number $c\Delta t/\Delta x$. The Last two schemes are implicit, while the rest are explicit.

2.2 Time Independent PDEs

2.2.1 Relaxation Methods

The Laplacian equation was considered as an example of a time independent PDE. The equation could be converted into a stencil as shown below:

$$\begin{aligned} & u_{xx} + u_{yy} = 0 \\ & \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{(\Delta x)^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{(\Delta y)^2} = 0 \end{aligned} \quad (4)$$

If the grid spacing in each dimension is equal, the denominators may vanish yielding:

$$u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{i,j} = 0 \quad (5)$$

These equations may be solved using matrix methods, given some boundary conditions, or may be solved iteratively. Iterative solving uses the notion that the limit of $\partial u / \partial t$ should be zero for a time independent system. Therefore, setting non-boundary values to some initial guess should yield a correct solution if the zero in the Laplace stencil is replaced with $\partial u / \partial t$ and allowed to evolve for sufficient time.

The iterative method was tested using a set of boundary conditions $u(x, 0) = u(x, 10.0) = u(0.0, y) = 0$ and $u(20.0, y) = 100.0$. The system was tested beginning with a resolution $\Delta x = \Delta y = 5.0$. A code was written to evaluate and display this. Figure 1 shows the results of this system. The result was extremely low resolution, and was thus liable to be inaccurate compared to the analytical solution. To improve the accuracy, the resolution of the system

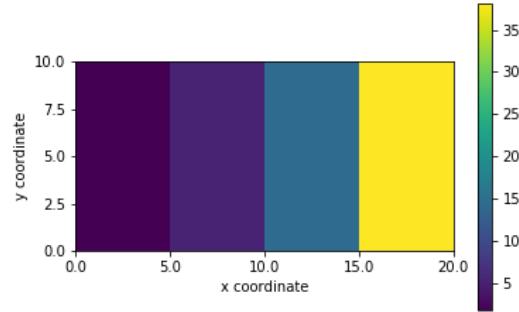


Figure 1: Solution to the Laplace equation for a set of boundary conditions. The wall at $x = 20.0$ is set to $u = 100.0$ while the other edges are set to $u = 0$. The image is low-resolution, measuring 2×4 cells.

was successively doubled four times. This is shown in figure 2.

The Laplacian equation is present in many physical systems. An example of the voltage inside a capacitor was considered. Boundary conditions along the plates were considered:

$$\begin{aligned} u(x, 0) &= 100 \sin\left(\frac{2\pi x}{L}\right) \\ u(x, 1) &= -100 \sin\left(\frac{2\pi x}{L}\right) \\ u(0, y) &= u(L, y) = 0 \end{aligned} \quad (6)$$

Where the plates were of width L and placed a dis-

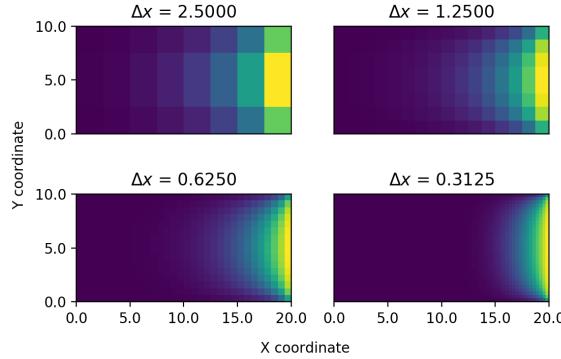


Figure 2: Solution to the same Laplace system as figure 1. The system has been evaluated at a higher resolution for each successive panel.

tance of L apart. The problem was approached using two variations of the relaxation method. The first was the Jacobi method, which was used in the previous problem. The second was the Gauss-Seidel method. The Gauss-Seidel method should be more efficient, as it uses an updated cell value during its sweeps if there is an updated value for one of the stencil cells. The results of the two methods are shown in figure 3. To test which method converged to a solution faster, a convergence criterion was used. Once no cell on the grid was more than 10^{-8} in difference from the previous iteration, the result was considered correct. The Jacobi method converged after 5390 iterations, while the Gauss-Seidel method converged after 3809 iterations.

2.2.2 The Schrödinger Equation

This section considered a common model for the Schrödinger equation that was a variant of the Laplace equation:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} - \alpha u = 0 \quad (7)$$

The stencil for this differential equation was largely similar to that of the Laplace equation, with a single added term. The stencil is as follows:

$$u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{i,j} - \alpha u_{i,j} = 0 \quad (8)$$

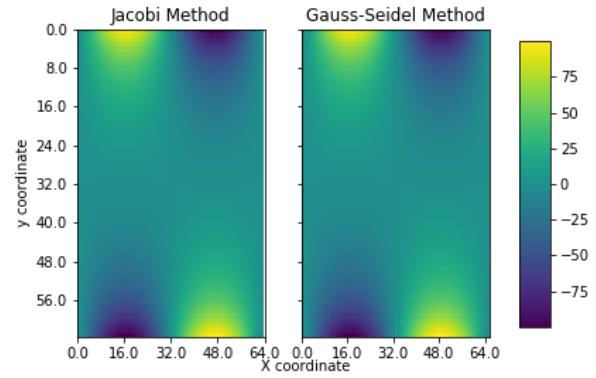


Figure 3: Final solutions obtained for a capacitor voltage problem using the Jacobi and Gauss-Seidel methods. Both methods yielded the same result. Note that axes are not to scale.

As previously mentioned, this system may be solved as a series of linear equations using matrix methods. To demonstrate this solution, the problem was considered for a square of length 1, resolution 50×50 , with u at all boundaries set to 1. A python script was written to evaluate the system of equations, by flattening the 2D u array into a one dimensional array. Next, the stencil was also flattened into a $50^2 \times 50^2$ 2D matrix. These terms could then be used to solve an $\mathbf{Ax} = \mathbf{b}$ system. The script was made more quickly using a variant of LU decomposition for banded matrices that sped the execution time to approximately 1.3 seconds. The result is shown in panel 1 of figure 4.

Another method used to evaluate such systems was a weighted Jacobi method. In essence, the method weights how heavily to consider the stencil terms rising from the time derivative versus how heavily to consider the stencil terms from the spacial derivatives. A series of 5 weighting coefficients were examined for the same problem as above with $\alpha = 1$. The results of this method are shown alongside the direct method in figure 4. The method was also tested for a variant equation with $\alpha = 1000$. The com

Method	Time to compute $\alpha = 1$	Iterations to compute $\alpha = 1$	Time to compute $\alpha = 1000$	Iterations to compute $\alpha = 1000$
Direct	0.28s	1	0.24s	1
Weighted $w = 0.1$	327.8s	30000	11.8s	1039
Weighted $w = 0.25$	264.8s	22529	4.81s	446
Weighted $w = 0.5$	144.9s	11956	2.46s	232
Weighted $w = 0.75$	95.1s	8239	1.68s	158
Weighted $w = 0.9$	75.9s	6967	1.44s	132

Table 3: Computation Speed of both the direct and weighted Jacobi methods. The weighted Jacobi method would had an iteration exit condition of 30000 loops.

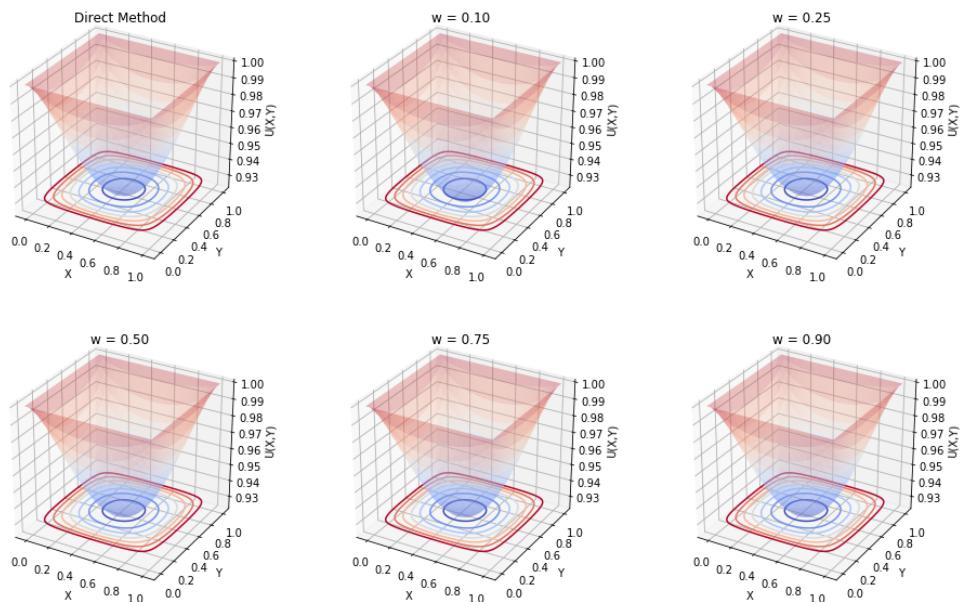


Figure 4: Solution to a model Schrödinger equation with $\alpha = 1$. All methods produced approximately the same result, with the only difference being in computation time.

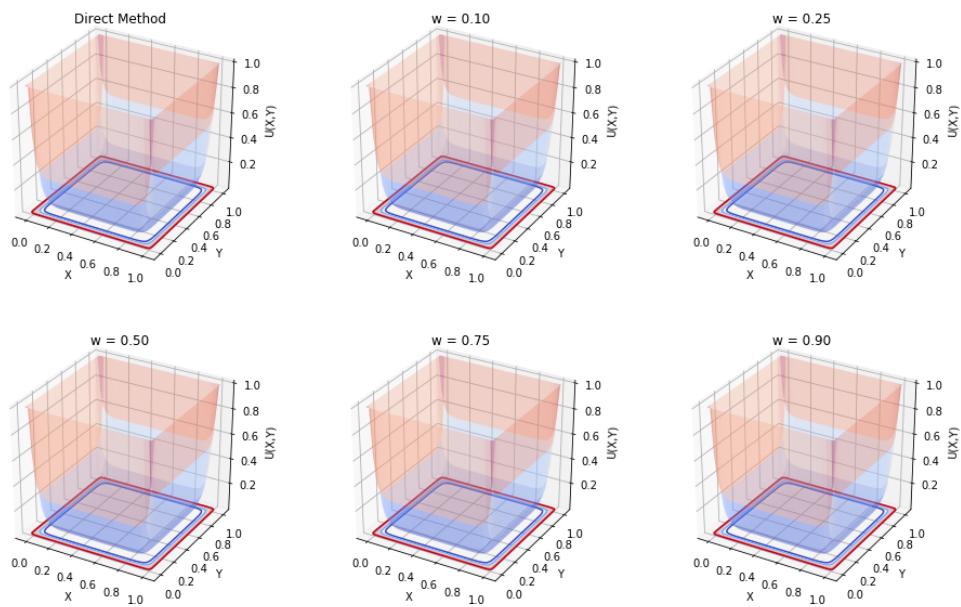


Figure 5: Solution to a model Schrödinger equation with $\alpha = 1000$. All methods produced proximately the same result, with the only difference being in computation time.

2.3 Hydrodynamics

2.3.1 Code Introduction

An example of rigorous code that simulates hydrodynamics in 2 dimensions was provided for the purposes of this lab. The code simulates the pressure, flow velocity, and density of a fluid or gas using the Navies-Stokes equations and related advection, diffusion, and dispersion terms such as those described in this lab. The code takes advantage of a staggered mesh. A staggered mesh is where different quantities are not evaluated at the same points, and instead are stored in separate meshes. In the case of this code, the scalar quantities (density, pressure) were located co-locally and the vector quantities (x and y flow velocity) were located at half steps to either side of the scalars. This functioned as the scalars being located in the middle of some unit cell, and the vectors (used as a flux quantity) located at the faces of the unit cube. Staggered grids are desirable, as they eliminate odd-even decoupling that can occur in co-local schemes.

As the code provided was written in Fortran, it required compilation. The various dependencies in the source code were managed using a makefile. This makefile was written to compile the code, create a new sub-directory containing set-up documents, and to clean directories of old compiled versions.

The code provided contained a default implementation of four initial high pressure points that were then allowed to diffuse. The code was compiled and allowed to run. Figure 6 shows the resulting behaviour. The pressurized points diffuse as expected. When encountering the obstacle, the pressure waves appear to reflect. No reflections occurred at the edges, indicating that matter in the simulation zone was allowed to leave the active zone and flow out. Figure 7 shows the flow velocity behaviour for the same simulation.

2.3.2 CFL Condition

The code as distributed did not have a proper CFL condition implemented. This prevented the code from running at an optimal speed. In order to compute the CFL at any given step, the speed of sound everywhere must be known. The speed of sound may

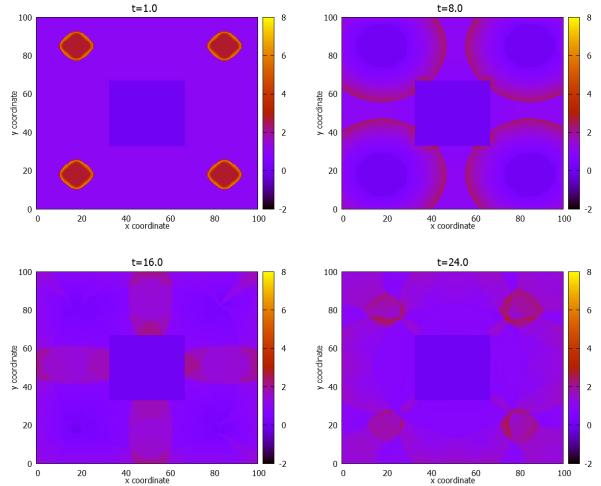


Figure 6: Initial configuration of the hydrodynamics simulation. Initial conditions specify four high pressure zones. No additional forcing was provided, and the subsequent frames allow the pressure zones to disperse. A square obstacle has been placed at the centre of the active zone, and can be seen as a solid zone of colour when pressure disturbances approach.

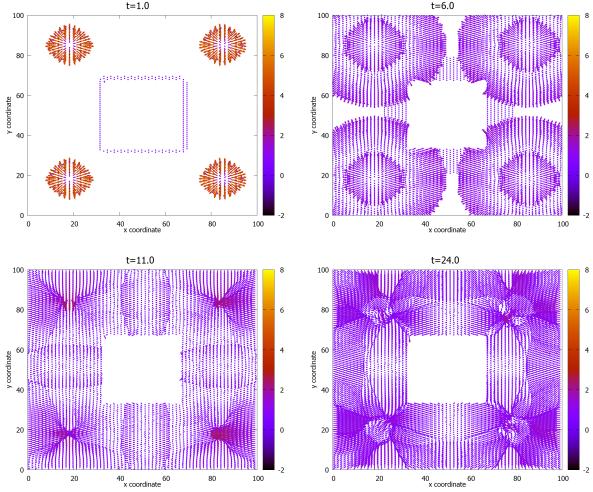


Figure 7: Initial configuration of the hydrodynamics simulation in terms of velocities. Vectors point in the direction of velocity and are both scaled and color coded by magnitude. A square obstacle has been placed at the centre of the active zone, which is shown as white in this figure.

be defined by the equation:

$$c_s = \sqrt{\left(\frac{\partial p}{\partial \rho}\right)_s} \quad (9)$$

A further equation used by the simulation used an adiabatic equation to define the pressure relative to the density:

$$p = \rho^\gamma \quad (10)$$

These two equations were combined to determine the speed of sound, with the adiabatic index $\gamma = 5/3$:

$$c_s = \sqrt{\left(\frac{\partial p}{\partial \rho}\right)_s} = \sqrt{\gamma \rho^{\gamma-1}} \quad (11)$$

The code manipulates this result to take both pressure and density into account, by re-substituting pressure:

$$\sqrt{\gamma \rho^{\gamma-1}} = \sqrt{\frac{\gamma \rho^\gamma}{\rho}} = \sqrt{\frac{\gamma p}{\rho}} \quad (12)$$

To implement this CFL condition, the general 2D CFL condition was used:

$$C = \frac{u_x \Delta t}{Deltax} + \frac{u_y \Delta t}{\Delta y} \quad (13)$$

Where u is the speed of the medium in that direction, which will usually be c_s . This could be re-arranged using $C = 0.5$ to find Δt at each step. Important to note is that u is not the same at all points being simulated. To determine the u value to use, the maximum u occurring in the grid must be used. This maximum value is either the speed of sound, or the highest flow speed in the medium, whichever is higher.

The CFL file was modified to determine both the maximum speed and the CFL condition using the speed of sound equation defined above. Before running the CFL condition, the default simulation running to $t = 10.0$ took 135.7s with an improper $dt = 10^{-3}$. After implementing dynamic CFL, the same simulation took 14.1s. The speed difference is due to the CFL condition choosing the highest dt possible without causing instability. This allows the system to reach its time-goal much more quickly in runtime.

2.3.3 Injection Inlet

Once the simulation was optimized, physical situations could be modeled. One such application is a wind-tunnel, where a stream of high velocity fluid is injected from the left. The setobjects and studentsetup files were modified to add a jet in the x direction, entering from the left. Figures 8 and 9 show the behaviour of both the pressure and the velocity for this setup.

Further experimentation was performed by varying the density and the velocity of the jet. Increasing the jet density resulted in a very sudden high-density front moving across the active zone: much more pronounced than in figure 8. The shape of the pressure wave also changed, with the front of the pressure-wave being much thicker and flatter. Where the default jet had been almost parabolic in shape, the increased density jet looked more like a rounded rectangle. Increased density also caused a change in the velocity profile, with the former tapering beam of

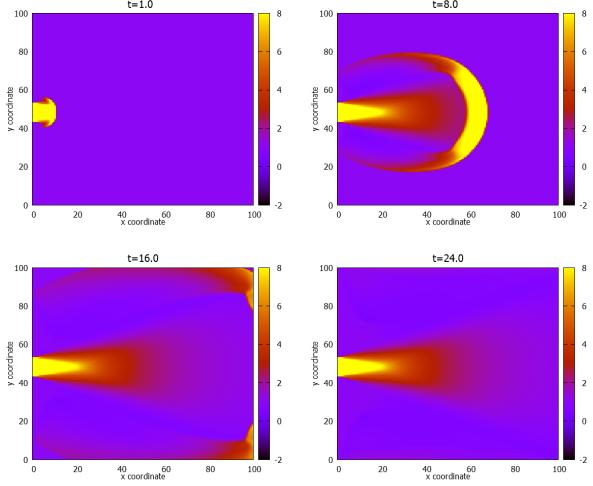


Figure 8: Pressure simulation of a jet of fluid entering from the right. The fluid is allowed to flow unimpeded to the right side boundary.

high velocity becoming a broad cone of high velocity, trapped behind the pressure front. Increasing the x velocity also changed the behaviour of the system. The pressure front was narrow, and didn't spread out very much. The pressure also tapered off rather quickly. The velocity was again a very strong, narrow beam. Where in the default jet, velocity spread out in a circular shape around the beam, the high velocity variant had velocity turbulations extending out in a planar manner parallel to the jet.

Once the jet was established, different solid shapes could be placed in the path of the jet in order to examine aerodynamic properties. First, built-in subroutines were used to place a vertical wall in the path of the jet. The pressure and velocity simulations are shown in figures 10 and 11 respectively. The fluid does not show any reflection, instead the pressure simulation shows a high pressure buildup in front of the wall. This is likely due to the boundary conditions imposed by the wall. Once particles come in contact with the wall, their velocity is set to 0. This causes particles to build up in front of the wall, and not be moved around it. Looking at the velocity figure seems to confirm this, with the jet largely halted

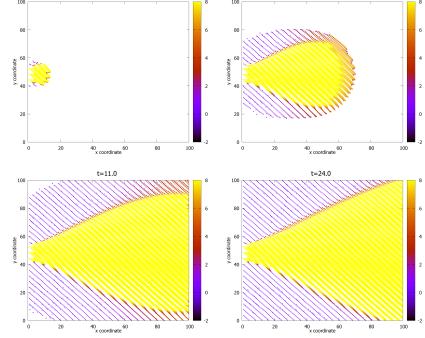


Figure 9: Velocity simulation of a jet of high speed fluid entering from the left. The fluid is left unimpeded to flow to the right boundary. The vectors have been scaled and color coded according to magnitude.

by the wall, with some spill off around the edges of the jet that roughly correspond to the edge of the wall. This is different to the edges of the active zone, where there are not solid walls. This allows particles to flow out of the active zone. One can imagine that if the active zone was bounded by a box with similar velocity properties to the wall, there would be similar pressure build-ups around the active zone.

The next obstacle considered was a wall oriented parallel to the jet's path. Figures 12 and 13 show the density and velocity simulations. In this case, density once again built up in front of the leading edge. Since the wall was now smaller than the width of the jet, the jet was able to pass around the obstacle around the sides. The wall left a pressure wake behind it, which collapsed in on itself to form a slightly more turbulent pressure front once again, with some pressure oscillations between the front and the wall. The velocity profile shows the jet being split into two beams, diverging away from the wall at approximately 45° each. This makes sense, as any particles moving back towards the wall would lose their velocity. This reinforces the velocities away from the wall, causing the stream to diverge away from the wall.

Lastly, an isosceles triangle was placed in the path of the jet, as documented in figures 14 and 15. As in

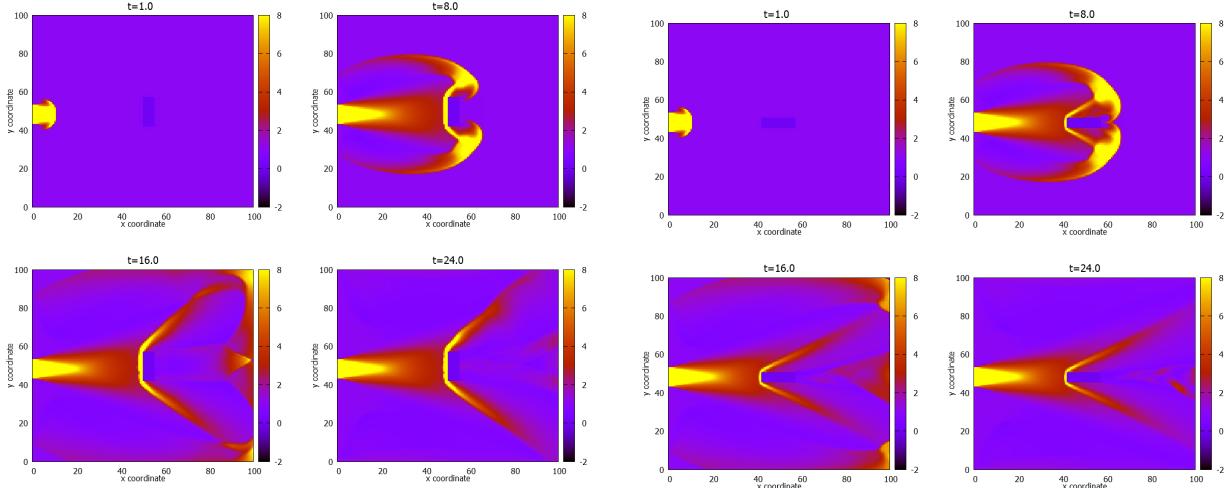


Figure 10: Pressure evolution of a jet pointed at a vertical wall.

Figure 12: Pressure simulation of a horizontal placed in the path of a fluid jet.

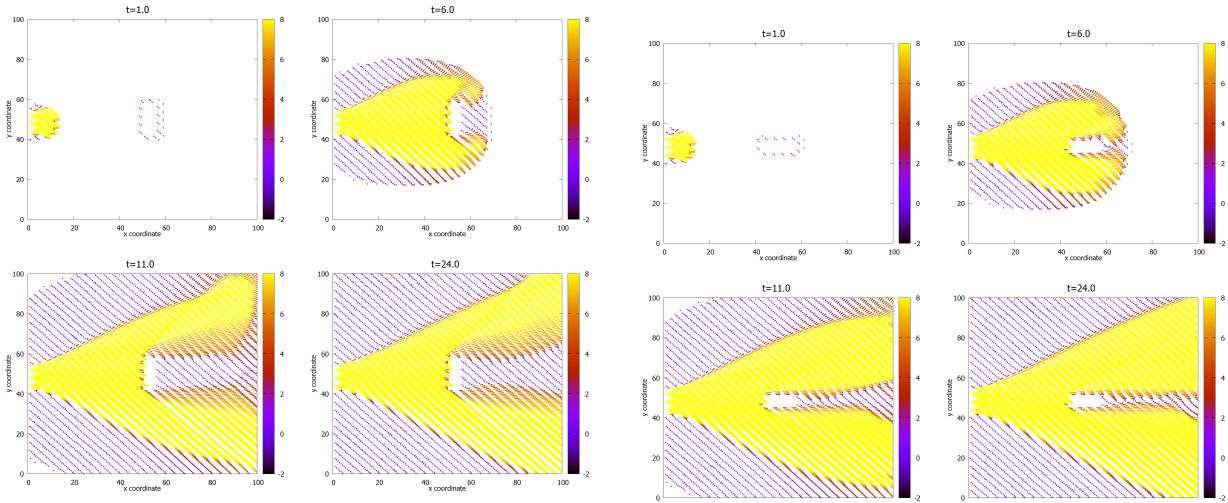


Figure 11: Velocity evolution of a jet pointed at a vertical wall.

Figure 13

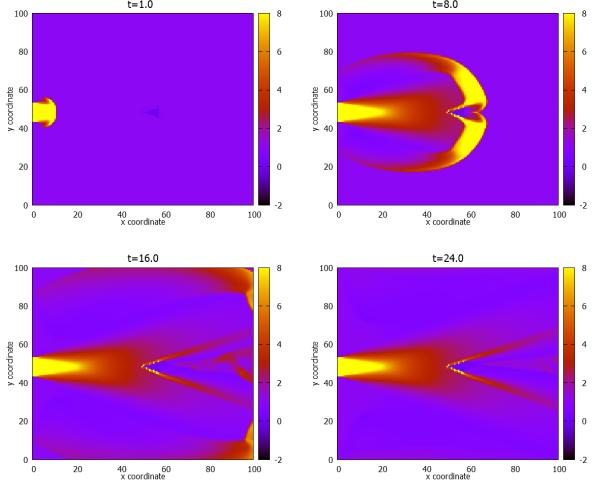


Figure 14: Pressure simulation for a triangular obstacle in the path of the injection jet.

the previous example, there is a pressure buildup immediately in front of the object, however it is greatly reduced compared to previous examples. This is sensible due to the more aerodynamic shape. As density builds up, it is easy for it to slide at an angle compared to the flat surfaces examined so far. The velocity profile again shows the stream being diverted by the wall, however the smaller profile of this object allows dispersion to merge the two streams somewhat together again.

2.3.4 Airplane Aerodynamics

The code built in this lab can function as a test for aerodynamic properties. Using the shapes constructed in the previous section, a crude airplane silhouette was constructed. It no longer made sense to consider a narrow jet. Instead, the system was modified to be a unidirectional flow throughout the active zone. For the following exercises, the flow velocity was varied to simulate an airplane moving at the same speed through the air. The density of the fluid considered was set to enter at a value of 5. In the previous CFL section, it was shown that the speed of sound in a medium depends on the density of the

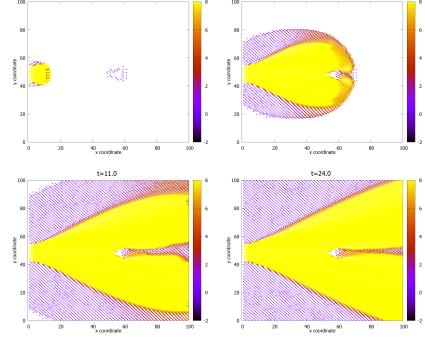


Figure 15: Velocity simulation for a triangular obstacle in the path of an injection jet.

material. For this density, the speed of sound was as follows:

$$\sqrt{\gamma \rho^{\gamma-1}} = \sqrt{\frac{5}{3} \times 5^{\frac{2}{3}}} \quad (14)$$

An area of particular interest in aviation is the behaviour of vehicles relative to the speed of sound. The Mach number is a way to characterize speed relative to a medium. The Mach number M is defined as how fast something is traveling relative to the speed of sound of the surrounding medium. For this exercise, a model airplane was simulated traveling at four different Mach numbers: $M = 0.2$, $M = 1.1$, $M = 3.0$, and $M = 7.5$.

Looking at figure 16, the general behaviour is similar between plots. In front of the plane wings is a region of high pressure, known as a bow-shock. Trailing behind is a region of generally low pressure compared to the undisturbed medium. There is significant turbulence in the wake of the plane. As the speed of the plane increases, the pressure in front of the wings grows significantly. In addition, the bow-shock grows, so that the region of high pressure thickens in front of the wings.

Modern aircraft use a swept-wing design, where the wing tapers backward at an angle. Figure 17 shows the difference that is made by using a swept-wing design. It can be seen that the swept-wing design reduces the bow-shock in advance of the wing. While it cannot be seen in this figure, the growth rate of the bow-shock is also greatly reduced as well. Where

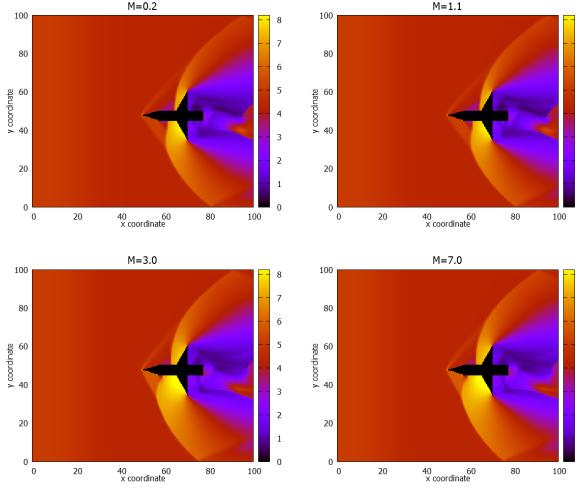


Figure 16: Pressure profile surrounding a crude airplane traveling at high speeds. The text above each panel indicate the Mach number of the airplane's speed in that panel. All panels were snapshots taken at $t = 44$.

the broad-wing design of the original plane showed stronger bow-shocks progressively as the speed of the plane increased, this effect was not nearly as strong in the swept-wing case. These two designs suggest that the swept-wing design is better for strength and stability. The strong pressure in advance of the wing contrasted with the low pressure following causes shear and strain on the wings - reducing this would be crucial in ensuring that the wings are not torn off in high-speed flights.

The success of these simulations was dependent upon staggered grids and the finite volume method. Used together, the model may be seen as a series of volume elements with flow defined on the faces of volume-cubes. If a finite element method had instead been used, angled obstacles may have caused problems. This is because velocities became zero when hitting an object. For finite elements, this would cause a loss of both x and y velocity in the pixel ahead of the obstacle. By using the staggered grid, the y velocity may still be preserved in the case of triangles, allowing the fluid to flow along a triangle edge.

3 Discussion

This lab examined both iterative and direct methods for the solution of differential equations. Of the two methods, direct methods were found to be faster. This however may be misleading, due to the choice of python for solving these systems. In the code written, iterative methods were solved index-by-index, over a series of loops. While understandable, this code may not have taken complete advantage of the tools available both within python and other languages. The code for the direct method, by comparison, used a publicly available library, that took advantage of the banded nature of stencil matrices to better optimise the problem. The relative timing of the two methods presented in this lab should then be taken with a grain of salt. As with many areas of coding, a more thorough examination of optimization methods may be required for a definitive answer of which method is faster, and by what margin.

There are a multitude of recipes available for us-

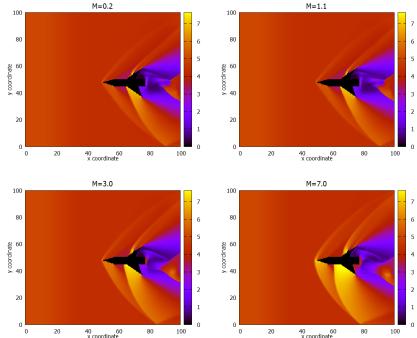


Figure 17: Pressure profile of a crude airplane with a swept wing design. The same numbering convention as figure 16 was used. All panels were snapshots taken at $t = 44$.

ing finite differences to solve differential equations. Many of these recipes were created to address shortcomings that arise from the simple central difference method. For instance, simple methods may have difficulty dealing with shock behaviour. All of these methods have different combinations of accuracy and stability that must be accounted for when choosing which system to use. The choice of method should depend on the problem at hand. For instance, if you only have an initial condition at a single time step, it would likely not be suitable to choose a method that requires two time steps. Likewise, not every problem has sudden shocks and discontinuities; simpler methods that assume continuous functions may suffice. Another factor to consider when choosing methods is how the method physically relates to the system being studied. Methods using advection models should be considered if the system itself is expected to physically experience this. In some cases, the information gained from a physically intuitive system may outweigh the error that accompanies it.

Time dependent methods were shown to be of use to steady state systems as well, taking advantage of the tendency of some recipes to tend towards equilibrium. This lab showed a few examples where spacial systems were examined by allowing the system to evolve from an initial guess. In these cases, the stencils used allowed information from a boundary condition to migrate to all areas under consideration. One particular method applied demonstrated this migratory behaviour. The weighted Jacobi method essentially weighted whether the stencil placed more priority on the neighboring cells, or the cell's own previous state. It was found that preferring the neighboring cells allowed the system to converge faster. This may be interpreted as the boundary information migrating more quickly through the solution.

Hydrodynamics was shown as an important application of finite difference and finite volume methods. The code was able to demonstrate its applicability to the design of aerodynamic surfaces. The examples in this lab showed that symmetry played a large part in how simulations fared. Because of the construction of the system, edges of surfaces were pixelated, and differences in rounding to place points on the pixelated grid sometimes resulted in asymmetrical designs. In

such cases, it was seen that even these slight asymmetries would cause different jet reflections. The most prominent of these issues took place with the simulation of aircraft. Due to some slight asymmetries, the upper wing of the aircraft generated a significantly smallled bow-shock. This effect was amplified as time went on. This asymmetry carried through to the chosen points of the swept-wing aircraft. The top wing in this design experienced significantly less bow-shock, while the lower wing was substantially larger. This made the lower wing almost as poor as the un-swept wing. Errors such as these are fixed by higher resolution methods, and by ensuring that designs are placed in a perfectly symmetrical manner.

4 Conclusion

Finite differences is a starting point to the discrete simulation of differential equations. This lab has shown how a simple discretization of the definition of the derivative may be expanded into a cornucopia of different recipes to model differential equations. Prior to the advent of computers, much of physics was devoted to the understanding and classification of differential equations. The use of finite differencing allows these models and advances to be transferred to high-power computational methods with minimal loss of accuracy. The use of finite differences allows the simulation of change with respect to time, providing a push to the forecasting nature of modern science.

5 Appendix

For access to the source codes used in this project, please visit https://github.com/Tsintsuntsini/PHYS_581 for a list of files and times of most recent update.