1 Introduction

1.1 Computational Difficulty of Solving Nonlinear PDEs

In most introductory Partial Differential Equation (PDE) courses, diffusion is often the first topic covered. This is a result of the key defining property of a PDE: that there is more than one independent variable $x, y, z, ...$ and a dependent variable that is an unknown function of these variables $u(x, y, ...)$. Thus, some of the simplest PDEs are those of low order and are considered fundamental in physical theory. Three of these include

$$\begin{aligned} u_x + u_y = 0 \quad & \text{Transport} \\ \nabla^2 u = f \quad & \text{Poisson (or LaPlace for f = 0)} \\ \nabla^2 u = u_t \quad & \text{Heat/Diffusion Equation} \end{aligned}$$

However, all three of these equations are similar in that they are linear; all of its terms involving u and any of its derivatives can be expressed as a linear combination in which the coefficients of the u-terms are independent of u. This property allows linear PDEs to be classified and solved using predetermined schema (i.e. separation of variables, superposition, Fourier series/Transform, and Laplace Transform). On the other hand, no general techniques exist that can solve a substantial class of nonlinear PDEs at a time; each PDE is treated as a separate problem. This distinction between linear and nonlinear PDEs is exceptionally important in numerical analysis. Furthermore, exact equations are required for testing and verification in a computational setting, as computer scientists and mathematicians must constantly test their models to ensure their relevance and accuracy. This obviously becomes an issue when evaluating nonlinear PDEs. Generally, nonlinear PDEs don't yield analytic solution approaches. Unfortunately, most real-world applications and leading edge work in numerical analysis invovles nonlinear PDEs, and mathematicians spend a great deal of time obtaining approximations these problems/equations. Throughout this paper, I'll focus on one equation, **the Burger Equation**, and how it's exceptionally important in confronting one of the main problems linked to nonlinear PDEs: the appearence of shocks. I'll talk about a familiar application (including its initial and boundary conditions) then discretize the problem so it can be solved using the finite difference algorithm.

1.2 Applications to Traffic

It's pre-quarantine and you're in your car at 6PM, sitting in the O'Neil tunnel. As expected, Boston rush hour traffic is terrible, and matters are made worse when two lanes are closed, forcing three lanes' worth of cars into only one. Instead of modelling cars individually, the number of cars in the tunnel can be estimated using the density $\rho(x, t)$ of cars which are in the interval (x_1, x_2) at time t using the equation:

$$\textit{number of cars between } x_1 \textit{ and } x_2 = \int_{x_1}^{x_2} \rho(x, t)dx$$

Of course, each car is also travelling (albeit very slowly) at some velocity, v, of the cars at position x and time t. Thus, the number of cars which enter and leave through point x at time t is $\rho(x, t)v(x, t)$. As the night progresses, the traffic will eventually loosen its grip on the tunnel. Since I want to model this change, I'll derive an equation for the evolution of the car density:

$$\tfrac{d}{dt} \int_{x_1}^{x_2} \rho(x_1, t)dx = \rho(x_1, t)v(x_1, t) - \rho(x_2, t)v(x_2, t)$$

Integrating this yields:

$$\begin{aligned} \int_{t_1}^{t_2} \int_{x_1}^{x_2} \partial_t \rho(x, t)dxdt &= \int_{t_1}^{t_2} (\rho(x_1, t)v(x_1, t) - \rho(x_2, t)v(x_2, t))dxdt \\ &= -\int_{t_1}^{t_2} \int_{x_1}^{x_2} \partial_x(\rho(x, t)v(x, t))dxdt \end{aligned}$$

Since $x_1, x_2 \in \mathbb{R}, t_1, t_2 > 0$ are arbitrary, it follows that

$$\rho_t + (\rho v)_x = 0 \qquad\qquad x \in \mathbb{R}, t > 0$$

Now, I need an equation for the velocity, v. If I assume that v only depends on ρ and that the maximum velocity is v_{max}, I can represent the current velocity as a ratio of the current velocity's value and the maximum possible velocity:

$$v(\rho) = v_{max}(1 - \tfrac{\rho}{\rho_{max}}) \qquad \text{for } 0 \leq \rho \leq \rho_{max}$$

Which transforms the original equation into

$$\rho_t + (v_{max}\rho(1 - \tfrac{\rho}{\rho_{max}}))_x = 0, \qquad x \in \mathbb{R}, t > 0$$

In order to convert this into a useful form for numerical integration, I'll definte two variables, L and T, to represent the periods for displacement and time respectively. I'll also define a restriction: the average displacement over this time period should be equal to the maxuimum velocity. Now, I have:

$$x' = \tfrac{x}{L} \qquad t' = \tfrac{t}{L} \qquad u' = 1 - \tfrac{2\rho}{\rho_{max}}$$

Which, again, allows us to redefine the previous equation to:

$$\partial_t \rho = \tfrac{1}{T}\partial_{t'}(\tfrac{\rho_{max}}{2}(1-u)) = -\tfrac{\rho_{max}}{2T}\partial_{t'}u$$

and

$$\partial_x(v_{max}\rho(1 - \tfrac{\rho}{\rho_{max}})) = \tfrac{1}{L}\partial_{x'}(v_{max}\tfrac{\rho_{max}}{2}(1-u)\tfrac{1}{2}(1+u)) = -\tfrac{\rho_{max}}{2T}\partial_{x'}(\tfrac{u^2}{2})$$

Now, I can rewrite the original equations to get:

$$u_t + (\tfrac{u^2}{2})_x = 0, \qquad x \in \mathbb{R}, t > 0 \quad \text{and} \quad u(x,0) = u_0(x), \qquad x \in \mathbb{R}$$

More commonly, these equations are denoted:

$$\tfrac{\partial u}{\partial t} + u\tfrac{\partial u}{\partial x} = 0$$

$$u(x,0) = u_0(x)$$

Of couse, the initial condition represents a "shock" in the system and u represents the wave speed. This equation is conservative; no cars can leave the lane and no cars can enter. It actually has a special name; the *Inviscid Burger's Equation*. However, if a diffusive term is added, the equation becomes the 1D convection-diffusion equation. In its general form:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = v\frac{\partial^2 u}{\partial x^2}$$

2 Analysis of Burger's Equation

2.1 A Centered Finite Difference Schema for Burger's Equation

Finite difference methods rely on the replacement of derivatives in an ODE or a PDE with approximations. For this problem, I'll apply a forward difference at time t_n and a second order central difference for the space derivative at position x_j. To help illustrate this, I'll provide the stencil (geometric arrangement of a nodal group that relates to a discretized point of interest by using a numerical approximation schema) for this 1D problem:

The domain is discretized by introducing a uniformily partitioned mesh. The points in the mesh can be labelled $t_n = n\Delta t, n = 0, 1, ..., N_t$, where $\Delta t = T/N_t$ is the constant length of the time steps. Similarly, I'll define $x_n = n\Delta x, n = 0, 1, ..., N_x$ and $\Delta x = X/N_x$ is the constant length of displacement steps. I'll also

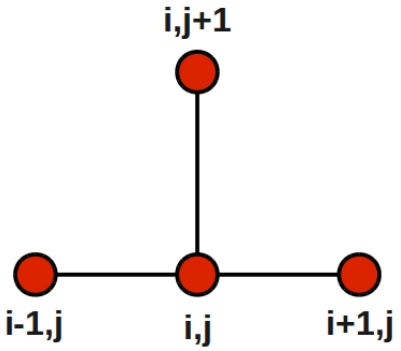

Figure 1: The stencil that will be used to solve the 1D Burger's Equation (Forward Euler)

define a mesh function, u^n for $n = 0, 1, ..., N_t$, which approximates the exact solution at the mesh points. The mesh function can be computed from algebraic equations derived from the differential equation problem. The PDE should be satisfied at each mesh point:

$$\frac{\partial u(x_n,t_n)}{\partial t_n} + u(x_n, t_n)\frac{\partial u(x_n,t_n)}{\partial x_n} = v\frac{\partial^2 u}{\partial x^2}$$

Now, I'll replace the derivatives with the finite difference approximation:

$$\frac{u_i^{n+1}-u_i^n}{\Delta t} + u_i^n\frac{u_i^n-u_{i-1}^n}{\Delta x} = v\frac{u_{i+1}^n-2u_i^n+u_{i-1}^n}{\Delta x^2}$$

Multiplying both sides by Δt then subtracting the second term from both sides gives me

$$u_i^{n+1} = u_i^n - u_i^n\frac{\Delta t}{\Delta x}(u_i^n - u_{i-1}^n) + v\frac{\Delta t}{\Delta x^2}(u_{i+1}^n - 2u_i^n + u_{i-1}^n)$$

2.1.1 Stability of Explicit Method

Using this explicit method, I can define a new constant, r, to measure divergence:

$$r = \frac{\Delta t}{\Delta x^2}$$

With the defined recurrence relation, I can iterate knowing the values at time n to get the values at time $n + 1$. Furthermore, I can replace u_0^n and u_j^n with the boundary conditions. This explicit method is also known to be numerically stable (convergent) when $r \leq 1/2$. In other words, when the numerical errors are proportional to the time step and the square of the time step. This can be expressed as:

$$\Delta u = \boldsymbol{O}(\Delta t) + \boldsymbol{O}(\Delta x^2)$$

2.2 Analytical Solution

As previously mentioned, comparing the results of an approximation to the analytical solution is necessary for computational scientists, but deriving analytical solutions to many nonlinear PDEs isn't always possible. In this case, however, an analytical solution can be derived using the **Cole-Hopf transformation**, a transformation that turns the our strongly nonlinear equation into a heat equation problem. It's defined by:

$$u(x,t) = -2v\frac{\phi_x}{\phi} \qquad (*)$$

Which means that

$$u_t = \frac{2v(\phi_x\phi_t - \phi\phi_{xt})}{\phi^2}$$

by diffrentiation. Similarly, I can derive

$$uu_x = \frac{4v^2\phi_x(\phi\phi_x x - \phi_x^2)}{\phi^3}$$

and

$$vu_{xx} = \frac{2v^2(2\phi_x^3 - 3\phi\phi_{xx}\phi_x + \phi^2\phi_{xxx})}{\phi^3}$$

Substituting this into the original form gives me

$$\begin{aligned} \frac{2v(-\phi\phi_{xt} + \phi_x(\phi_t - v\phi_{xx}) + v\phi\phi_{xxx})}{\phi^2} &= 0 \\ \iff -\phi\phi_{xt} + \phi_x(\phi_t - v\phi_{xx}) + v\phi\phi_{xxx} &= 0 \\ \iff \phi_x(\phi_t - v\phi_{xx}) &= \phi(\phi_{xt} - v\phi_{xxx})_x \\ &= \phi(\phi_t - v\phi_{xxx})_x \end{aligned}$$

On the LHS, we can observe that ϕ solves the heat equation, $\phi_t - v\phi_{xx} = 0 \quad \forall x \in \mathbb{R}$. Thus, $u(x,t)$ given by the transformation defined by (*) solves Burger's equation. Going back to (*), we can use logarithmic properties to write

$$u(x,t) = -2v(log\phi)_x$$

Solving for ϕ by dividing by -2v and integrating by x gives me gives me

$$(log\phi) = -\int \frac{u(x,t)}{2v} dx$$

Finally, I solve for ϕ by raising each side by e:

$$\phi(x,t) = exp(-\int \frac{u(x,t)}{2v} dx)$$

I can now apply the arbitrary initial value condition:

$$\therefore \phi(x,0) = \phi_0(x) = exp(-\int_0^x \frac{u_0(y)}{2v} dy), \forall x \in \mathbb{R}, t > 0$$

To solve the heat equation, I can take the Fourier Transform of x for both the heat equation and the initial condition, $\phi_0(x)$. I now have the set of equations:

$$\begin{aligned} \widehat{\phi}_t = \xi^2 v\hat{\phi} \qquad & \xi \in \mathbb{R}, t > 0, v > 0 \\ \widehat{\phi}(\xi, 0) = \hat{\phi}_0(\xi) \qquad & \xi \in \mathbb{R} \end{aligned}$$

I can write $\widehat{\phi}(\xi,t)$ and its solution as:

$$\widehat{\phi}(\xi,t) = \int_{\mathbb{R}} \phi(x,t) exp(i\xi x) dx$$

$$\widehat{\phi}(\xi,t) = \widehat{\phi_0}(\xi) exp(\xi^2 vt)$$

To recover the analytic expression of $\phi(x,t)$ I have to use the inverse Fourier Transform $\mathcal{F}^{-1}$:

$$\begin{aligned} \phi(x,t) &= \mathcal{F}^{-1}(\widehat{\phi}(\xi,t)) \\ &= \mathcal{F}^{-1}(\widehat{\phi}_0(\xi) exp(\xi^2 vt)) \\ &= \phi_0(x) * \mathcal{F}^{-1} exp(\xi^2 vt) \end{aligned}$$

where $*$ denotes the convolution product. Now, I solve for the exoression within the transform:

$$\mathcal{F}^{-1}(exp(\xi^2 vt)) = \frac{exp(\frac{-x^2}{4vt})}{2\sqrt{\pi vt}}$$

Thus, the initial value problem has the analytic solution of

$$\phi(x,t) = \frac{1}{2\sqrt{\pi vt}} \int_{\mathbb{R}} \phi_0(v) exp(\frac{-(x-\xi)^2}{4vt}) d\xi$$

Thus, I obtain the analytic solution for Burger's equation (using the transform defined by (*)):

$$u(x,t) = \frac{\int_{\mathbb{R}} \frac{x-\xi}{t} \phi_0(\xi) exp(\frac{-(x-\xi)^2 d\xi}{4vt})}{\int_{\mathbb{R}} \phi_0(\xi) * exp(\frac{-(x-\xi)^2 d\xi}{4vt})}$$

I can use MATLAB's computer algebra system to compute the integrals and get a more useful expression of the analytic solution:

$$u(x,t) = -2v \frac{\partial \phi / \partial x}{\phi}$$

$$\phi(x,v) = exp(\frac{-(x-4t)^2}{4v(t+1)}) + exp(\frac{(-x-4t-2*\pi)^2}{4v(t+1)})$$

$$\frac{\partial \phi}{\partial x} = -\frac{x-4t}{2v(t+1)} exp(\frac{-(x-4t)^2}{4v(t+1)}) - \frac{x-4t-2\pi}{2v(t+1)} exp(\frac{-(x-4t-2\pi)^2}{4v(t+1)})$$

3 Implementation of the Algorithm and Discussion of Results

3.1 Pseudocode: the Finite Difference Algorithm Applied on the 1D Burgers' Equation

In section 2, I outlined the discretization of this problem. I'll now go over the exact constants, initial and boundary conditions, and constants that the algorithm takes in to solve.

3.1.1 Inputs

The first constants I'll define are the time and space steps. Most relevantly, Burger's Equation models a dissapative system: problems that are defined by an initial "shock" or impulse (heat from some source, an introduction of soy sauce to water, etc.) that eventually naturally evolves into a steady state. Thus, the time and space steps should be small relative to their respective intervals in order to test the system properly. Since the initial condition is a sawtooth wave, the boundary conditions should be periodic: $u^n_{i=0} = u^n_{i=max}$. Furthermore, there should be some "viscosity" value, v, that represents how fast the system changes with one time or space step (also known as the diffusion term).

3.1.2 Representation of Iterations Over Time and Space

Since this equation is second order defined by partials with respect to two distinct variables, an appropriate numerical scheme for the iterations over the terms involves two variables as well. Per the previous section, I'll use i to represent indices on the mesh in x and n to represent indices on the mesh in t. Now, I can write out the pseudocode for my algorithm and explain it. I'll use a python3- flavored pseudocode to describe my algorithm due to its similarity to MATLAB:

```
//initialize constants:
  nt; tmax; dt; nx; xmax; dx = xmax/(nx - 1); v;

// represent points within the interval of interest at each x step
for i between 1 and nx:
  x(i) = i * dx
  ipos(i) = i + 1
  ineg(i) = i - 1
ipos(end) = 0
ineg(1) = nx - 1;

// represent the initial conditions
for i between 1 and nx:
  phi = exp(-x(i)^2/(4*v)) + exp(-(x(i)-2*pi)^2/(4*v))
  dphi = -(0.5/vis)*exp(-(x^2)/(4*vis))-(0.5*(x-2*pi)/v)*exp(-(x-2*pi)^2/(4*v))
  u(i, 0) = -2*v*(dphi/phi)

// Now, carry out the numerical computation by stepping over time and
// using an inner loop to iterate over space.
for n between 2 and nt - 1
  for i between 1 and nx - 2
  u(i,n+1) = u(i,n)-u(i,n)*(dt/dx)*(u(i,n)...
             -u(ineg(i),n))+v*(dt/dx^2)*(u(ipos(i),n)-2*u(i,n)+u(ineg(i),n))

// Now, use compute the exact solution at the same points and plot to show the difference
for i between 1 and nx:
  plot(u, x)
  plot(u_analytic, x)
```

3.2 Tests and Figures

3.2.1 Running the Algorithm with a Δx of 0.04189, Δt of 0.00333, r of 1.8977

The numerical solution was computed using a time-march with 150 iterations. The domain for displacement is $[0, 2\pi]$ radians and the domain for the time is $[0, 0.5]$ seconds. In order to promote diffusion towards a steady-state, the viscosity is 0.1. Since there were 150 iterations performed, a single x step was $2\pi/150 \approx 0.04189$ radians and a single t step was $0.5/150 \approx 0.00333$.

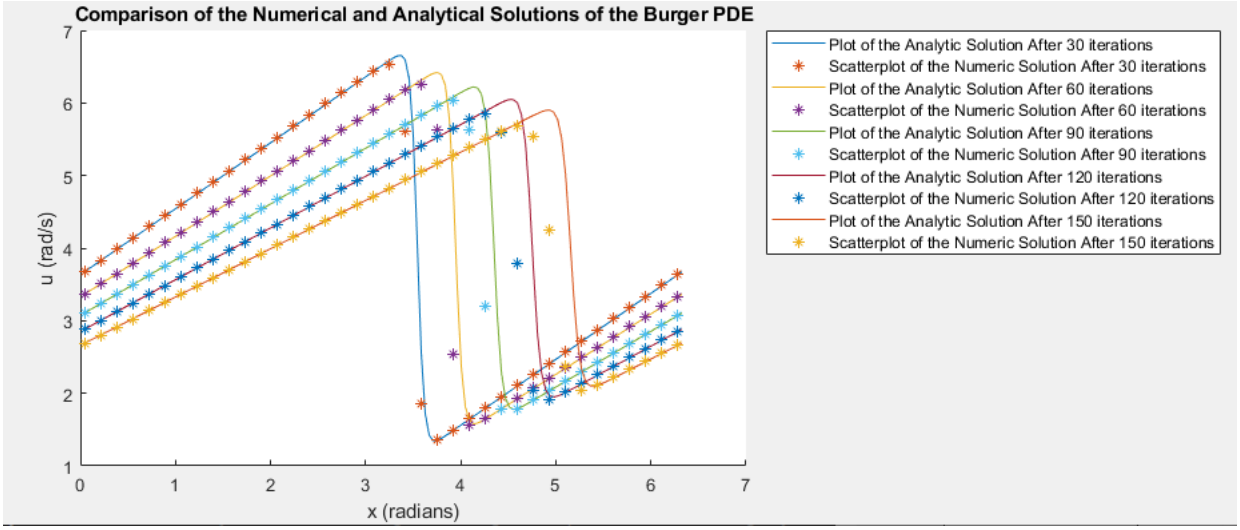

Figure 2: Multiple Plots of $u_{analytic}(x, t)$ and $u_{numerical}(x, t)$ at various iterations (time steps)

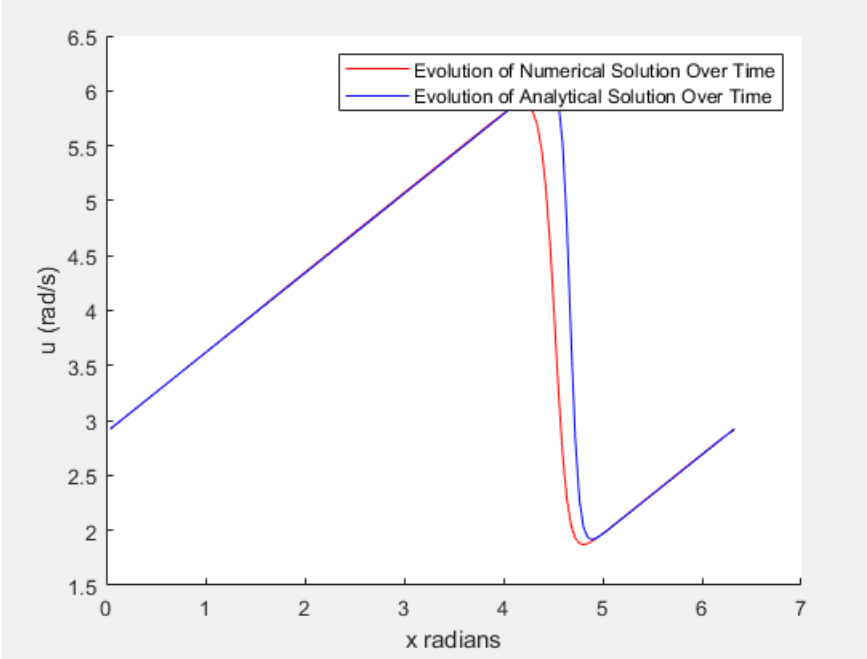

Figure 3: A frame from the animated dissipative system solutions

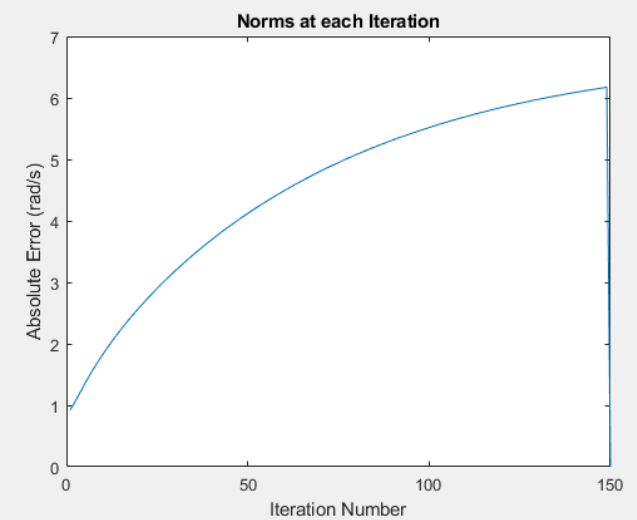

Figure 4: A plot of the norms (error) at each iteration

3.2.2

3.2.3 Running the Algorithm with a Δx of 0.06283, Δt of 0.00100, r of 0.25333

The numerical solution was computed using a time-march with 500 steps in the t-space and 100 steps in the x-space. The domains for displacement and time remain unchanged. Since there were 500 iterations performed in the x-space, a single x step was $2\pi/100 \approx 0.06283$ radians and a single t step was $0.5/500 \approx 0.00100$. This results in a t of $0.00100/(0.06283)^2 = 0.25333$.

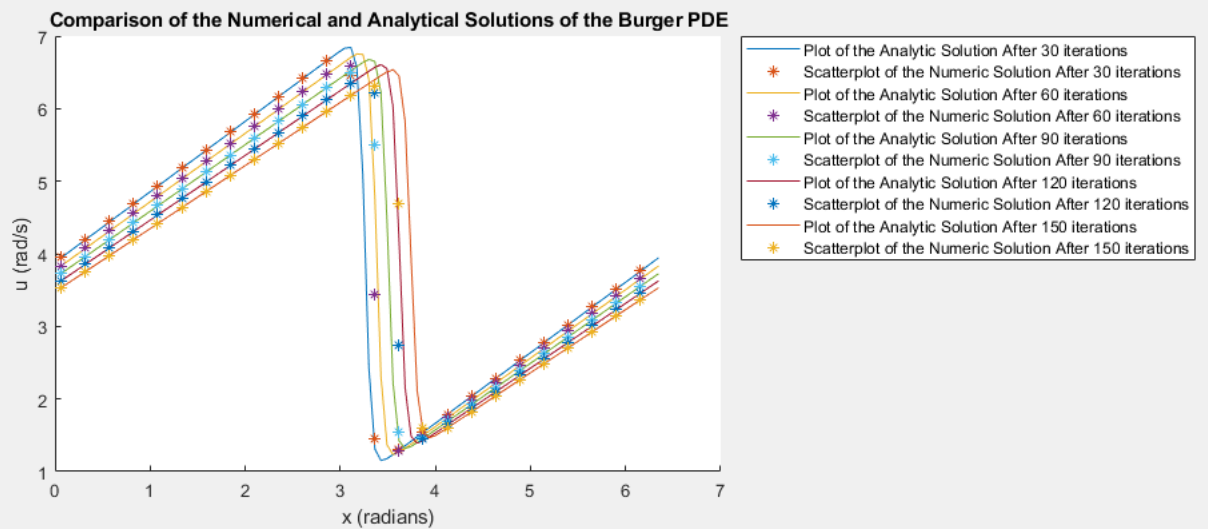

Figure 5: Multiple Plots of $u_{analytic}(x,t)$ and $u_{numerical}(x,t)$ at various iterations (time steps)

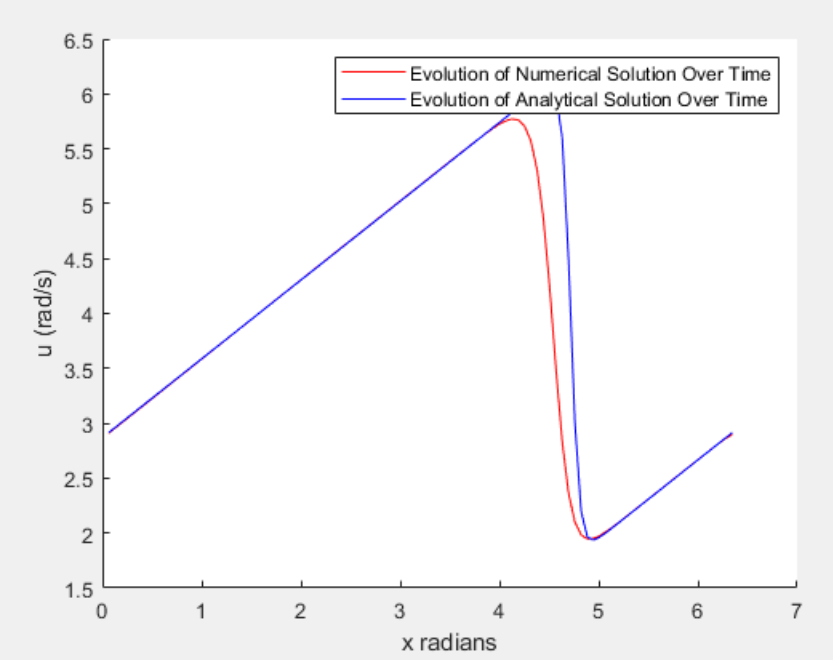

Figure 6: A frame from the animated dissipative system solutions

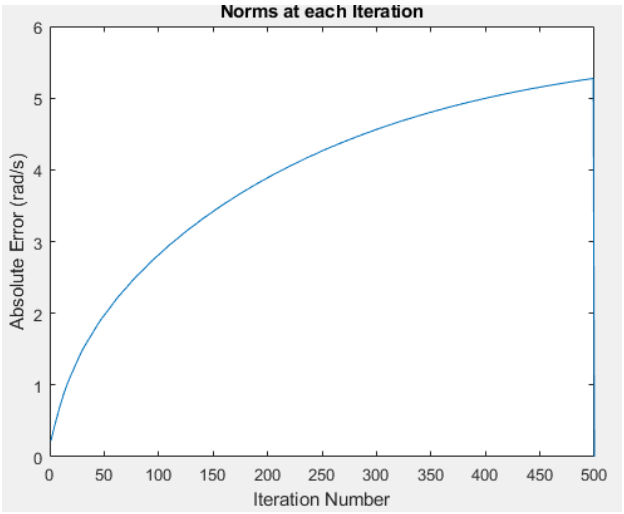

Figure 7: A plot of the norms (error) at each iteration

3.2.4 Running the Algorithm with a Δx of 0.02513, Δt of 0.00200, r of 3.16647

The numerical solution was computed using a time-march with 500 steps in the t-space and 100 steps in the x-space. The domains for displacement and time remain unchanged. Since there were 500 iterations performed in the t-space, a single x step was $2\pi/100 \approx 0.06283$ radians and a single t step was $0.5/500 \approx 0.00100$. This results in a t of $0.00100/(0.06283)^2 = 0.25333$.

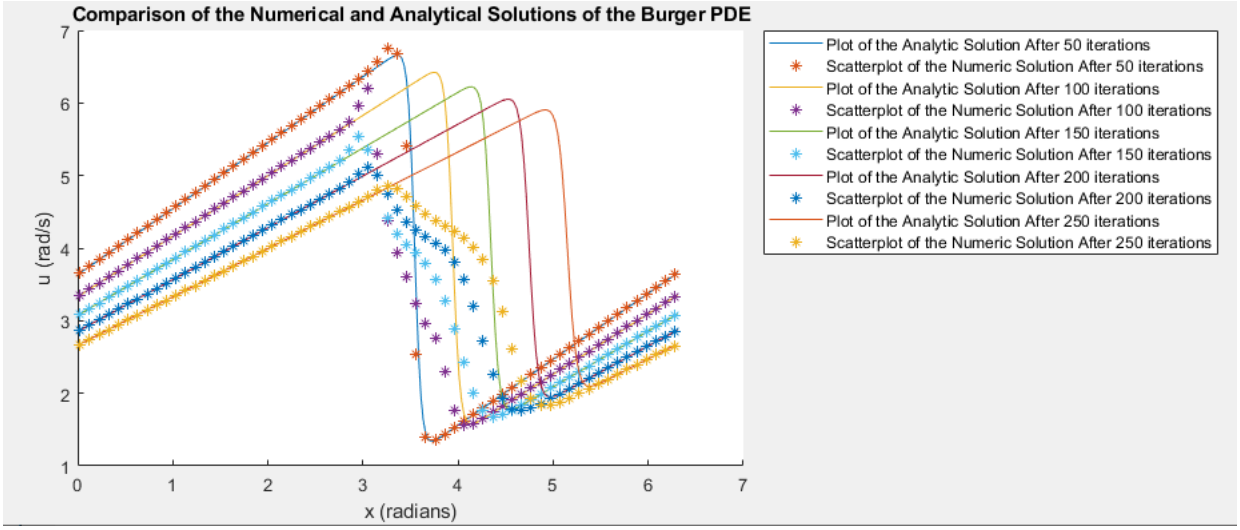

Figure 8: Multiple Plots of $u_{analytic}(x,t)$ and $u_{numerical}(x,t)$ at various iterations (time steps)

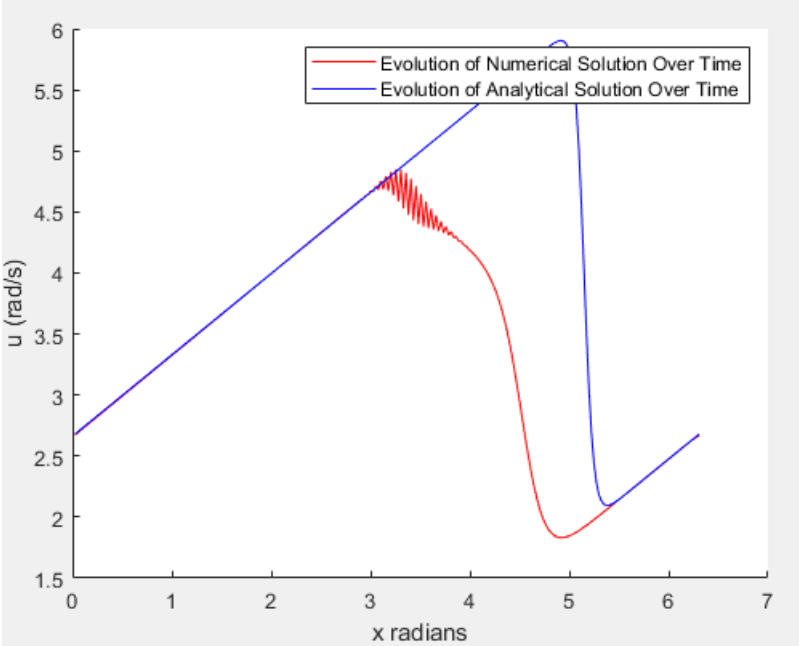

Figure 9: A frame from the animated dissipative system solutions

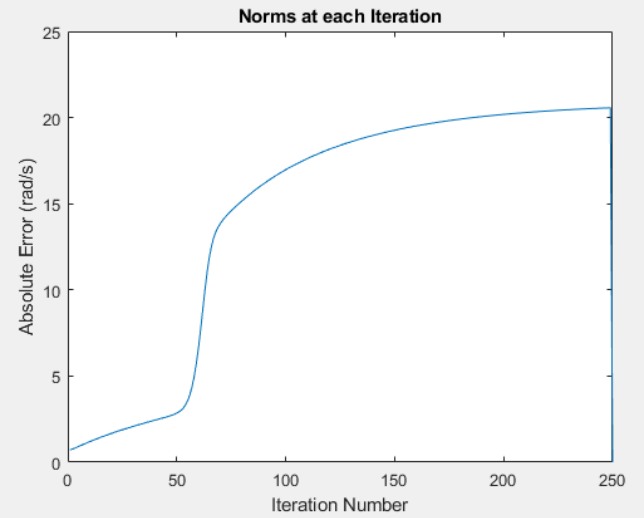

Figure 10: A plot of the norms (error) at each iteration

3.3 Discussion of Error

3.3.1 For 150 Steps in x and t

As seen in Figure 4, each point, on average, was close to the analytical solution. By dividing the norm at an iteration by the length of the vector, the average distance per point can be computed. In my numerical solution, the maximum difference can be found in iteration 150. This difference is $6.173/150 \approx 0.04115 \approx 4.115\%$. More noticably, the error increased logarithmically as time went on.

3.3.2 For 100 Steps in x and 500 t

As seen in Figure 7, each point computed by the numerical algorithm is quite close to the analytical solution. By dividing the norm at an iteration by the length of the vector, the average distance per point can be computed. In my numerical solution, the maximum difference can be found in iteration 500. This difference is $20.189/250 \approx 0.0104 \approx 1.04\%$. Again, the logarithmic error exists within these difference computations as well. However, the difference is much less: from 4.12% to 1.04%. This difference can be credited to the smaller t value: from 1.8977 to 0.2533. As discussed in Section 2.1.1, the Forward Euler method is stable for $r \leq 0.5$. Since the r for this run came out to be 0.2533, the algorithm converged faster and is more accurate per iteration.

3.3.3 For 250 Steps in x and 250 t

As seen in Figure 9, each point computed by the numerical algorithm greatly differs from the analytical solution. By dividing the norm at an iteration by the length of the vector, the average distance per point can be computed. In my numerical solution, the maximum difference can be found in iteration 250. This difference is $20.189/250 \approx 0.0.8076 \approx 8.08\%$. Again, the logarithmic error exists within these difference computations as well. Due to the extremely high r value of 3.1665, this run of the algorithm is increasingly divergent (see Figure 8 and 9 for reference). When compared to the most accurate results of the algorithm (see Section 3.2.3 for these results), this run is approximately 8 times more inaccurate per iteration.

3.3.4 Numerical Dissipation and the Accuracy of Numerical PDE Solvers

While increasing and decreasing the r values proved to greatly affect the results of the numerical algorithm, the algorithm still contained error for a relatively low r value (results found in Section 3.2.3). This error can be credited to a phenomenon known as **numerical dissipation**. Numerical dissipation is a diffuculty with computer simulations of continua, such as fluids, in which the simulated (numerical) solution exhibits a higher diffusivity than the true (analytic) solution. In computations that are aimed to solve differential equations, time and space are discretized into discrete meshes while the differential equations are discretized into finite difference equations. The discrete equation, described in Section 2, is more diffusive than the original differential equation. The explanation is physical in nature: shock waves such as the one modelled in this problem tend to be infinitely thin. When a shock wave is induced on a surface, the wave will spread out in all directions relative to the impulse. The discretized equation, however, only steps along the mesh orthogonally, making it impossible to measure the shockwave's exact direction. After a few iterations in both displacement and time-space, the shock wave will have spread out through the mesh via orthogonal nodes. The numerical effect, of course, is that there's an extra high diffusion rate whose impact on the system becomes more pronounced as the algorithm iterates.

4 Conclusion

Burger's Equation is one of the most important PDEs in the theory of nonlinear conservation laws due to its combining of both nonlinear propogation and diffusive effects. Using a finite - difference method, Burger's equation with an initial value defined by the Cole-Hopf Transformation and periodic boundary conditions can be solved. This finite - difference method is expressed as:

$$u_i^{n+1} = u_i^n - u_i^n \frac{\Delta t}{\Delta x}(u_i^n - u_{i-1}^n) + v\frac{\Delta t}{\Delta x^2}(u_{i+1}^n - 2u_i^n + u_{i-1}^n)$$

and served as the basis of my computations throughout this project. It was derived using the stencil found in Section 2, which is a common stencil for solving diffusion equations due to its second-order method in time and its numerical stability. When compared to the analytic solution, derived in Section 2, the algorithm proved to be quite accurate: within 1.04% within 500 iterations for the most accurate run of the algorithm. When the ratio of the time step to the square of the displacement step is altered, the algorithm's result becomes more or less accurate. This value, also known as the r value, greatly affects the convergence of the Forward Euler algorithm. When the r value is relatively low, the absolute difference between the analytical and numerical solution remains relatively low. As shown in Sections 3.2.X, the absolute error

between thge analytical solution and the numerical solution increases as the r value increases, as expected. The logarithmically increasing error is likely credited to numerical dissipation, which is a side-effect of discretizing space and time as uniformally-spaced orthonormal nodes on a mesh.

5 Works Cited

Marino, A., and M. K. V. Murthy. Nonlinear Variational Problems and Partial Differential Equations. Longman Scientific amp; Technical, 1995.

Ebert, Marcelo R., and Michael Reissig. Methods for Partial Differential Equations: Qualitative Properties of Solutions, Phase Space Analysis, Semilinear Models. Springer International Publishing, 2018.

Polyanin, Andrei Zaitsev, Valentin. (2012). Handbook of Nonlinear Partial Differential Equations, Second Edition.

Lax, P. D. (1954), Weak solution of nonlinear hyperbolic equations and their numerical computation, Comm. Pure Appl. Math. 7.

Kutz, J. Nathan. Data-Driven Modeling amp; Scientific Computation: Methods for Complex Systems amp; Big Data. OUP Oxford, 2013.