SEMI-IMPLICIT TIME INTEGRATION FOR PARTIAL DIFFERENTIAL EQUATIONS AND THE METHOD OF REGULARIZED STOKESLETS

AN ABSTRACT
SUBMITTED ON THE FIFTH DAY OF MAY, 2023
TO THE DEPARTMENT OF MATHEMATICS
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
OF THE SCHOOL OF SCIENCE AND ENGINEERING
OF TULANE UNIVERSITY
FOR THE DEGREE OF
MASTER OF SCIENCE
BY

BENJAMIN PHILIP STAGER

APPROVED:

TOMMASO BUVOLI, Ph.D.
CHAIRMAN

KYLE KUN ZHAO, Ph.D.

LISA FAUCI, Ph.D.

Abstract

Many processes found in science and engineering are governed by dynamical systems, described by ordinary and partial differential equations. These systems are often complex and do not admit closed form solutions. Therefore numerical methods called time integrators are required for finding solutions. In this thesis we study the efficiency of various integrators for solving three partial differential equations: the heat equation, Viscous Burgers' equation, and Stokes equations (specifically the time integration of velocity field produced by method of regularized Stokeslets). A time integrator's efficiency is quantified by analyzing the amount of computational time required to approximate the solution at a given accuracy. This thesis has three primary components. First, we will discuss implicit and explicit integrators for solving linear PDEs. Second, we will present implicit-explicit and exponential integrators for solving semi-linear PDEs. Lastly, we propose to use a semi-linear time integrator for solving initial value problems arising from the Stokes equations. We find that a semi-implicit integrator can take larger timesteps when modeling stiff springs.

Acknowledgements

Affording myself merely a page for acknowledgements will never do my level of appreciation justice, but I will attempt to summarize to the best of my ability. I owe the utmost gradtitude to the Tulane University Department of Mathematics. I would like to thank those who were not involved in my thesis, but whose classes left me with a desire to continue my mathematical education namely - Marie Dahleh, Tewodros Amdeberhan, and James 'Mac' Hyman. Their teachings inspired me to continue my study of higher level mathematics - in both research and academic spaces. I would like also to thank Lisa Fauci for her brilliant lectures on graduate level applied mathematics - she is truly one of the world's greatest mathematicians. To Kyle Zhao - words will never be able to describe the effect your leadership had on my education. I would not be in the place I am without you. And lastly, but most certainly not least, I would like to thank Tommaso Buvoli. The significant effect of his mentorship throughout the journey ff this thesis can not be overstated. Tommaso has helped usher me into the world of numerical methods, and he is truly one of the brightest minds in the field. I consider him not only one of the most valuable presences in my academic life - but also a friend who I can turn to when I'm struggling with my studies. I look forward to continuing my exploration into this field post-graduation. I also would thank to thank my friends and family for their unwavering support throughout this journey. MORE and less lofty

Dedication

To the unbounded and profound subject of mathematics.

Contents

List of Figures

List of Tables

List of Code

Chapter 1

Introduction

In this paper, we will explore the efficacy and precision of **time integration** methods for a number of different regimes. Time integration is a subclass of numerical methods, implemented to find robust approximations of ordinary and differential equations. We will consider both ODEs (ordinary differential equations) and PDEs (partial differential equations). Building the foundation from the ground up, this paper will serve both research and background purposes. We begin by presenting the fundamental notions of time integration, then will move to a new way to use time integration a famous PDE problem.

The author will accompany his written findings with MATLAB code - in order to help the reader better understand how we can use many of these methods in tangible contexts. All codes are open to the reader to use for their benefit.

1.1 Necessity of time integration

We can use time integration for many dynamical systems and classes of equations that do not have analytical solutions. For an ordinary differential equation $y' = f(t, y)$ with solution $y : \mathbb{R} \to \mathbb{R}$, we could apply a time integrator across discrete points to find an exact solution. Shown below is an example of an application for a time integrator for a linear ordinary differential equation:

We implement the time integrator to approximate the solution at discrete

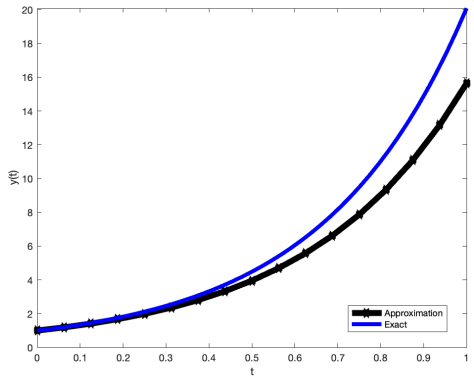

Figure 1.1: Comparison of time integrator and exact solution

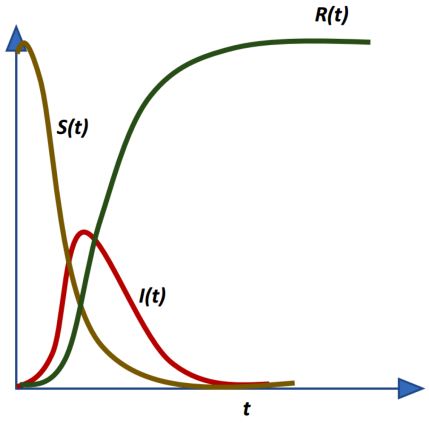

Figure 1.2: A generalized diagram for time integrated SIR model

points across a temporal interval. We can extend this application to much more complicated problems, such as a system of ordinary differential equations, that often do not have closed form solutions. By discretizing our time interval very finely, we can closely match an exact solution. Shown below is a generalized diagram for an SIR disease model, where we can find the value of each equation in the system at each time step.

The goal of these examples is to show that time integration is used to approximate solutions when exact closed form examples do not exist. Throughout the paper we will analyze how efficient time integrators can be in approximating these solutions.

Chapter 2

Overview of time integration

2.1 forward Euler

We will begin by introducing a simple explicit time integrator. Suppose that we have an ordinary differential equation of the form $y' = f(t, y)$ with initial condition $y(t_0) = y_0$. We can write the difference between two discrete values of the function $y(t_{n+1})$ and $y(t_n)$ as:

$$y_{n+1} - y_n = \int_{t_n}^{t_{n+1}} f(t, y)\ dt \tag{2.1.1}$$

$$y_{n+1} = y_n + \int_{t_n}^{t_{n+1}} f(t, y)\ dt \tag{2.1.2}$$

$$\implies y_{n+1} = y_n + hf(t_n, y_n). \tag{2.1.3}$$

The term h is a discretized time step given for a time interval $t \in [t_i, t_f]$ and time steps N_t as:

$$h = \frac{t_f - t_i}{N_t}. \tag{2.1.4}$$

The time integrator is:

$$y_{n+1} = y_n + hf(t_n, y_n). \tag{2.1.5}$$

The above equation is called the **forward Euler** time integrator. As $h \to 0$, the approximation approaches an exact solution.

Suppose we seek to time integrate the ODE $y' = 3y \ , \ y(0) = 1$, using $N_t = [4, 16, 32]$. Note below the diagram of numerical solutions:

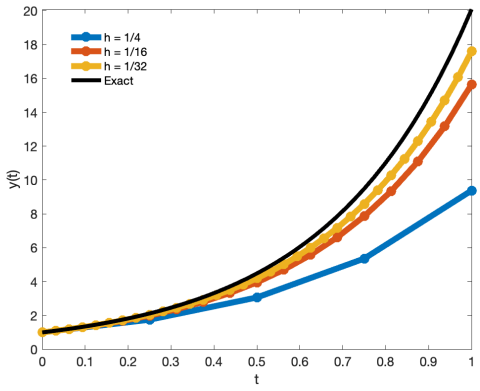

Figure 2.1: A time integrated solution for various h values

Notice that as $h \to 0$, the solution becomes closer to the true solution $y = e^{3t}$. While choosing h to be close to machine precision may seem good in practice, it can often slow down the method. We therefore seek to understand how we can choose h to minimize error while still getting a robust solution. [6]

2.2 Stability region in C

A time integrator's **stability region** governs the regime in which the method produces bounded outputs. One must be careful to choose the time step h small enough so that the method does not become unstable. Stability regions often play a key part in choice of numerical method for problem solving. To understand forward Euler's stability region, apply the scheme to the **Dalquist problem**, such that:

$$y' = \lambda y \ , \ y(0) = y_0 \ \ , \ \ \lambda \in C$$

$$y_{n+1} = y_n + h\lambda y_n \implies y_{n+1} = (1 + h\lambda)y_n.$$

For the sole case of this problem, y at the nth step can be condensed into a product:

$$y_{n+1} = (1 + h\lambda)^n \ y_0 \qquad \Longrightarrow \qquad y_{n+1} = (1 + z)^n y_0 \ , \ z = h\lambda \tag{2.2.1}$$

The term $1+h\lambda$ in (2.1.6) is defined as the amplification factor[6]. We can loosely define stability to satisfy the following requirements:

$$(1+h\lambda)^n = 0 \quad \iff \quad |1+h\lambda| \leq 1$$

The above expression is the stability region for forward Euler [2]. These two equivalent statements can be used to visualize the stability regions geometrically.

It is easy to see that the region of stability is a circle in $\mathbb{C}$, centered about $(-1, 0)$ with $r = 1$. We will notice that the method is not stable for $|\lambda|$ that is large. Note the region of stability in the complex plane for forward Euler below:

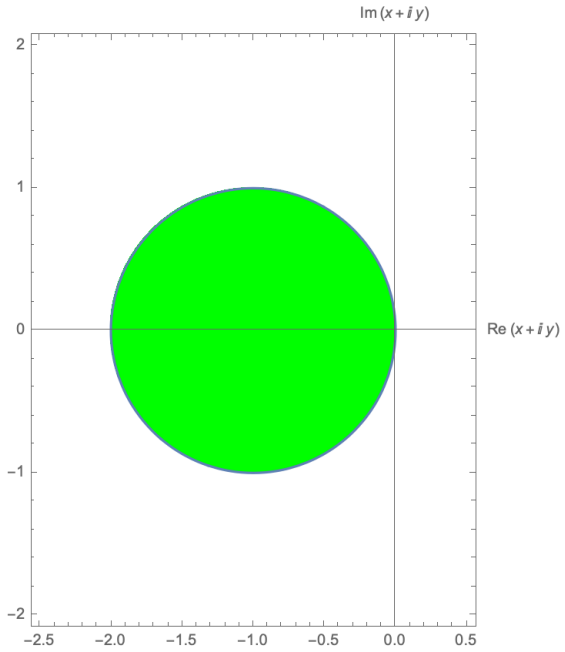

Figure 2.2: Stability region in $\mathbb{C}$ for the forward Euler time integrator

2.3 Error convergence

A method's convergence refers to the evolution of error as a function of the time step h. Error can be defined by the expression:

$$e(h) = ||\mathbf{y}^* - \mathbf{y}(h)||. \tag{2.3.1}$$

$||\cdot||$ is any valid norm. The term $\mathbf{y}(h)$ represents the approximate solution found using stepsize h. $\mathbf{y}^*$ is the exact solution denotes an exact or temporally 'fine' solution.

All time integrators have a corresponding order of accuracy. Suppose that for a solution defined using h if:

$$h \to \frac{h}{2} \implies e(h) \to \frac{e(h)}{2^p} \tag{2.3.2}$$

then we say a method has order $\mathcal{O}(h^p)$. Suppose we measure error for several h values such that $h = 10^{-n}$, $n = 1, \ldots, 5$. We measure each error using (2.1.5), where $||\cdot||$ is the 2-norm. Note the error convergence of forward Euler below:

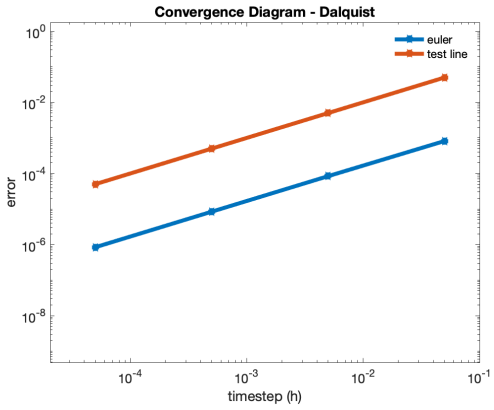

Figure 2.3: Error convergence of the forward Euler time integrator, on log-log scale

Note that forward Euler order of accuracy 1, or $\mathcal{O}$ [2]. In other words, $h \to \frac{h}{2}$, means that $e(h) \to e(h)/2$. Forward Euler while computationally efficient per time step, is often slow to compute high accuracy solutions because of its linear error convergence.

2.4 MATLAB implementation

Shown below is a MATLAB implementation of Forward Euler:

Listing 2.1: **Forward Euler**

```
function [ys,cpu_time] = euler(f,tspan,y0,N)
```

```
ys = zeros(length(y0),N+1);
ys(:,1) = y0;
y = y0;
dt = diff(tspan)/N;
t = tspan(1);
tic
for i = 1:N
    y = y + dt*f(t,y);
    ys(:,i+1) = y;
    t = t+dt;
end
cpu_time = toc;
```

2.5 Implicit integrators

Implicit integrators are amongst another class of time integrators. We seek to implement implicit integrators as a way to alleviate problems that explicit integrators have caused by numerical stiffness

Recall that the we can calcualte the change from $y_n \to y_{n+1}$, for an ODE $y' = f(t, y)$ as

$$y_{n+1} = y_n + \int_{t_n}^{t_{n+1}} f(t, y)dt. \tag{2.5.1}$$

If we take the integral using a right Reimann sum, we get:

$$\begin{aligned} y_{n+1} &= y_n + (t_{n+1} - t_n)f(t_{n+1}, y_{n+1}) \\ \implies y_{n+1} &= y_n + hf(t_{n+1}, y_{n+1}). \end{aligned} \tag{2.5.2}$$

Equation (2.2.2) is known as the **Backward Euler** method [2][6], different only in the endpoints of the rectangle approximation that we impose. Note that Backward Euler also has $\mathcal{O}$ accuracy. For the Dalquist problem the scheme becomes:

$$y_{n+1} = y_n + h\lambda y_{n+1} \implies y_{n+1} = (1 - h\lambda)^{-1} y_n. \tag{2.5.3}$$

Note here that the amplification factor is now $\frac{1}{1-h\lambda}$, which will change our stability

region.

2.6 Region of stability

Backward Euler differs from its explicit counterpart solely in region of stability. The method applied to the Dalquist problem is:

$$y_{n+1} = y_0\left(\frac{1}{1-h\lambda}\right)^n \tag{2.6.1}$$

where

$$|\frac{1}{1-h\lambda}|\leq 1 \tag{2.6.2}$$

is the amplification factor [6]. Expression (2.2.5) is the stability expression for Backward Euler [2]. The domain of stability is $\mathcal{S} = \{z \in \mathbb{C} : amp(Z) \leq 1\}$ [6]. We will be able to see from this notion that the stability region is much larger. The stability region for Backward Euler is shown below:

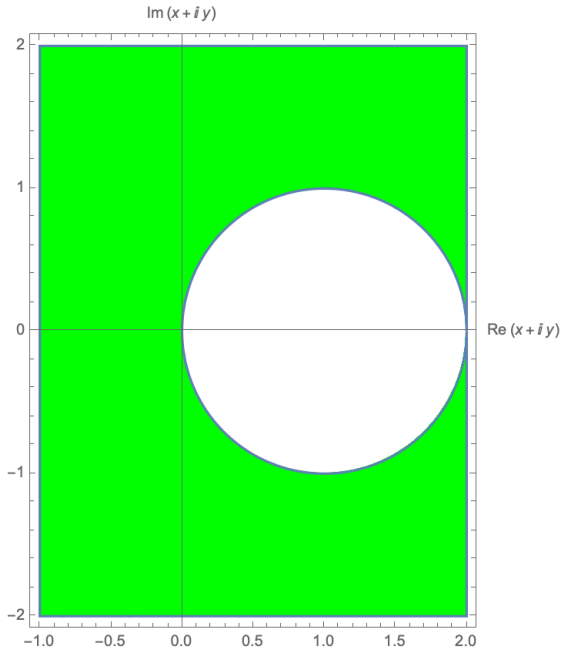

Figure 2.4: Stability region for Backward Euler in C

Note the unstable region is a circle centered at $(1, 0)$ with radius 1. Additionally if $\text{Re}(z) < 0$, the method is unconditionally stable, such that there is no constraint

Im(z). This notion is what makes implicit methods so important. By having such a large stability region, we can let the stepsize be much larger, and in turn can take less steps N_t and use less computational time.

2.6.1 MATLAB implementation

Shown below is a MATLAB implementation of Backward Euler:

Listing 2.2: **Backward Euler**

```
function [ys,cpu_time] = backwardsEulerLin(A,tspan,y0,N)
ys = zeros(length(y0),N+1);
y = y0;
ys(:,1) = y0;
dt = diff(tspan)/N;

if(issparse(A))
    I = speye(length(y0));
else
    I = eye(length(y0));
end

tic
for i = 1:N
    y = (I-dt*A)\y;
    ys(:,i+1) = y;
end
cpu_time = toc;

end
```

2.7 Runge-Kutta methods

Runge-Kutta methods are a class of time integrators that are implemented using **stage values**, or multiple intermediaries to compute the difference between y_{n+1} and y_n [2]. An *explicit* Runge-Kutta integrator is schemed using the general

expression:

$$y_{n+1} = y_n + h\sum_{i=1}^{s} b_i k_i \tag{2.7.1}$$

where

$$k_1 = f(t_n, y_n) \tag{2.7.2}$$

$$k_2 = f(t_n + c_2, y_n + h(a_{21}k_1)) \tag{2.7.3}$$

$$k_3 = f(t_n + c_3, y_n + h(a_{31}k_1 + a_{32}k_2)). \tag{2.7.4}$$

And the k_i can be abstracted as:

$$k_i = f\left(t_n + c_1, y_n + h\sum_{j=1}^{i-1} a_{ij}k_j\right) \tag{2.7.5}$$

.Note that $s \in \mathbb{N}$, and we say the method has s stages. For implicit Runge Kutta methods, the k_i at the ith step is defined as:

$$k_i = f\left(t_n + c_i, y_n + h\sum_{j=1}^{s} a_{ij}k_i\right) \tag{2.7.6}$$

For an explicit method, entries $a_{ij} = 0 \forall j \geq i$, otherwise the method would become implicit. The coefficients c_i, b_i, and a_{ij}, can be represented using a *Butcher Tableu* [13]:

$$\begin{array}{c|cccc} c_1 & a_{11} & a_{12} & \dots & a_{1s} \\ c_2 & a_{21} & a_{22} & \dots & a_{2s} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & a_{s2} & \dots & a_{ss} \\ \hline & b_1 & b_2 & \dots & b_s \end{array}$$

$$Y_i = y_n + h\sum_{j=1}^{s} a_{ij} f(Y_i) \tag{2.7.7}$$

$$y_{n+1} = y_n + h\sum j = 1^s b_i f(Y_i) \tag{2.7.8}$$

Implicit methods are taken across s stages[13].

2.7.1 Heun's method

Heun's Method is a two stage method explicit Runge-Kutta method. The corresponding Butcher tableau is shown below.

$$\begin{array}{c|cc} 0 & 0 & 0 \\ 1 & 1 & 0 \\ \hline & 1/2 & 1/2 \end{array}$$

As previously done, we will analyze stability using the Dalquist equation. With $s = 2$, the scheme is:

$$y_{n+1} = y_n + h(b_1 k_1 + b_2 k_2) \tag{2.7.9}$$

where

$$k_1 = f(t_n, y_n) = \lambda y_n \tag{2.7.10}$$

$$k_2 = f(t_n + c_2, y_n + h a_{21} k_1) = \lambda(y_n + h\lambda). \tag{2.7.11}$$

Putting k_1, k_2 together as a linear combination of b_1, b_2, we have the final scheme as:

$$y_{n+1} = y_n + h\left(\frac{\lambda}{2} y_n + \frac{\lambda}{2}(y_n + h\lambda)\right) = y_n\left(1 + h\lambda + \frac{h^2\lambda^2}{2}\right). \tag{2.7.12}$$

Although the method is two stages, we can write the Dalquist as a combination of $s = 1, 2$ stages. Heun's method applied to the Dalquist equation is:

$$y_{n+1} = \left(1 + h\lambda + \frac{h^2\lambda^2}{2}\right)^n y_0 \tag{2.7.13}$$

The stability condition is subsequently:

$$|1 + h\lambda + \frac{h^2\lambda^2}{2}| \leq 1 \tag{2.7.14}$$

$$\implies |1 + z + \frac{z^2}{2}| \leq 1 \tag{2.7.15}$$

2.7.2 Region of stability

The stability condition (2.3.13) [13] is not as simple as forward Euler to analytically understand. Using *Mathematica* software, we can plot the region in the complex plane. Shown below is the region of stability for Heun's method:

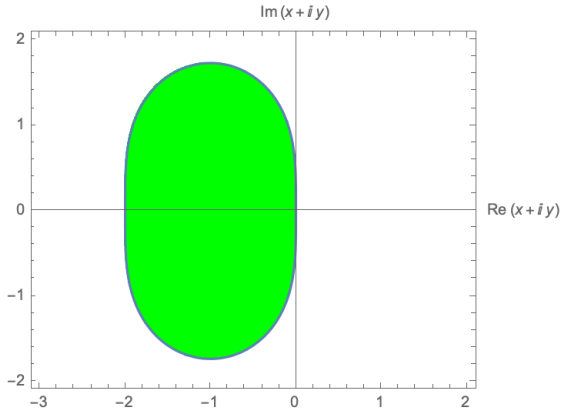

Figure 2.5: Stability region for Heun's method

The stability region does not provide much more than Forward Euler, but we will see that Heun is valuable because of its accuracy.

2.7.3 Second order error convergence

Heun, with $s = 2$ stages, has order of accuracy 2, or $\mathcal{O}(h^2)$. This means:

$$h \to \frac{h}{2} \quad \implies \quad e(h) \to e(h)/4. \tag{2.7.16}$$

This is valuable, as we now have nonlinear accuracy, gaining quadratically decreasing error progagation per timestep. Note the convergence diagram for Heun, compared to Forward Euler, below:

Heun's convergence, $\mathcal{O}(h^2)$, is often helpful as we do not need to take as many time steps to reach a desired accuracy.

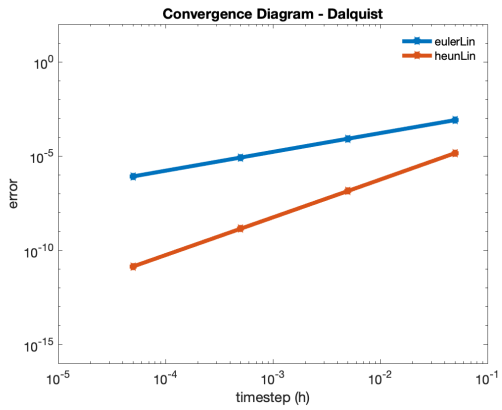

Figure 2.6: Error convergence for Heun's Method

2.7.4 MATLAB implementation

Shown below is a MATLAB implementation of Heun's method:

Listing 2.3: **Heun's method**

```
function [ys,cpu_time] = heun(f,tspan,y0,N)

ys = zeros(length(y0),N+1);
ys(:,1) = y0;
y = y0;
dt = diff(tspan)/N;
t = tspan(1);

tic
for i = 1:N
    k1 = f(t,y);
    k2 = f(t,y+dt*k1);
    y = y +.5*dt*(k1+k2);
    ys(:,i+1) = y;
    t = t+ dt;
end
cpu_time = toc;
```

2.7.5 Implicit midpoint method

The implcit midpoint method is an implcit one-stage Runge Kutta method. Its Butcher Tableau is:

$$\begin{array}{c|c} 1/2 & 1/2 \\ \hline & 1 \end{array}$$

The method can be written as

$$y_{n+1} = y_n + h(b_1 k_1) \implies y_{n+1} = y_n + \frac{h}{2} k_1 \tag{2.7.17}$$

$$k_1 = f(t_n + c_1 h, y_n + h a_{11} k_1) \implies k_1 = \lambda(y_n + \frac{h}{2} k_1). \tag{2.7.18}$$

Now, since k_1 is defined implicitly, we must compute a solve at the $i = 1$ step for k_1, such that for the Dalquist equation:

$$k_1 - \lambda \frac{h}{2} k_1 = \lambda y_n \implies k_1 = \frac{\lambda y_n}{1 - \frac{h\lambda}{2}}. \tag{2.7.19}$$

We can now compute the y_{n+1} step by substituting k_1, such that:

$$y_{n+1} = y_n + \frac{h}{2}\left(\frac{\lambda y_n}{1 - \frac{h\lambda}{2}}\right) \implies y_{n+1} = \left(1 + \frac{h\lambda}{2(1 - \frac{h\lambda}{2})}\right) y_n. \tag{2.7.20}$$

Note that from here we are able to visualize the condition for stability, such that:

$$\left(1 + \frac{h\lambda}{2(1 - \frac{h\lambda}{2})}\right)^n \to 0 \implies 1 + \frac{h\lambda}{2(1 - \frac{h\lambda}{2})} \leq 1. \tag{2.7.21}$$

The stability region, as is the case with many implicit methods, is much larger than that of an explicit method - alleviating the difficult of stiffer problems a. Note the stability region of implicit midpoint method below [2]:

Similarly to Backward Euler, we note the method as *A-stable*, i.e. $\forall\ \lambda < 0$, the method will converge to a solution regardless of step size h. Implicit midpoint has order of accuracy 2, or $\mathcal{O}(h^2)$.

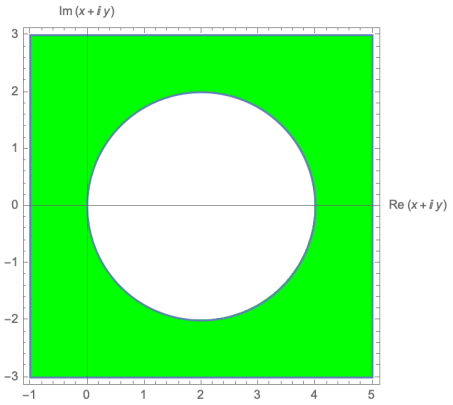

Figure 2.7: Stability region of implicit midpoint method

2.7.6 MATLAB implementation

Shown below is a MATLAB implementation of implicit midpoint:

Listing 2.4: **Implicit midpoint method**

```
function [ys,cpu_time] = impMidpointLin(A,tspan,y0,N)

ys = zeros(length(y0),N+1);
y = y0;
ys(:,1) = y0;
dt = diff(tspan)/N;

if(issparse(A))
    I = speye(length(y0));
else
    I = eye(length(y0));
end

tic
for i = 1:N
    Y1 = (I - (dt/2) * A) \ y;
    y  = y + dt * A * Y1;
    ys(:,i+1) = y;
end
cpu_time = toc;

end
```

2.8 Stability for systems and stiffness

We have discussed stability for the Dalquist equation, here we will extend it to systems. For the Dalquist equation, stability is governed by:

$$y' = \lambda y \quad \Longrightarrow \quad \text{Stability criteria} : h\lambda \tag{2.8.1}$$

whereas for a system of linear ODEs, it is:

$$\mathbf{y}' = A\mathbf{y} \quad \Longrightarrow \quad \text{Stability criteria} : \rho(hA) \tag{2.8.2}$$

where $\rho(hA)$ is the spectral radius of the matrix A. When using explicit integrators for stiff problems, the stepsize h must be chosen small enough that $h\lambda$ fits within the stability region. The benefit of implicit integrators is that if $\mathrm{Re}(\lambda) < 0$, then there is no restriction on stability of the method. We will see in the following section how this affects each method.

2.9 Method comparison and precision

Up to this point, we have discussed for different time integrators. They are compared below: We can see that each method has its advantages and disadvantages.

Method	Scheme type	Convergence	Stages	Stability
Foward Euler	Explicit RK	$\mathcal{O}(h)$	1	Restrictive
Backward Euler	Implicit RK	$\mathcal{O}(h)$	1	L-stable
Heun	Explicit RK	$\mathcal{O}(h^2)$	2	Partially restrictive
Implicit midpoint	Implicit RK	$\mathcal{O}(h^2)$	1	A-stable

Table 2.1: Comparison of explicit and implicit time integrators

To understand the benefitx of each method, we will briefly study the Dalquist problem for different values of λ. A method's **precision diagram** will shows $e(h)$ against computational time.

2.9.1 Precision of time integrators for variant λ

We defined the *stiffness* of a linear first-order non-autonomous ODE by its value of $|\lambda|$. As $|\lambda| \to \infty$, the solution curve $y(t)$ becomes 'steeper', or more oscillatory, and requires a smaller h - for explicit methods. Recall that the expression for error at the final time step for the Dalquist problem is:

$$e_f = ||y_{N_t} - e^{t_f} y_0|| \tag{2.9.1}$$

Where y_{N_t} is the time integrated approximation at final t_f using N_t steps. First consider the case in which $\lambda = -1$, giving a particular solution of $y(t) = e^t$. We will apply each method along the temporal interval $t \in [0, 5]$, and measure the error at $t_f = 5$ where $N_t \in \{10^3, 10^4, 10^5, 10^6, 10^7\}$.

Shown below is the combined convergence diagram of each method:

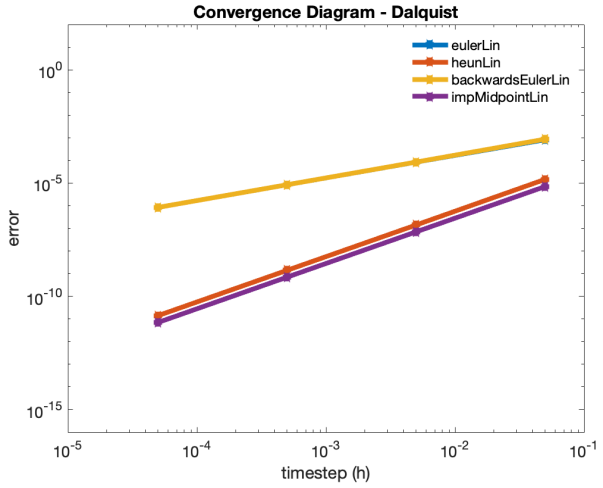

Figure 2.8: Temporal convergence diagram for λ = -1

Note that the Heun and implicit midpoint have $\mathcal{O}(h^2)$ accuracy. Forward and Backward Euler are overlapping. More importantly, shown below is the precision diagram for all for methods:

We can see that for an unstiff problem, error is significantly lower for the Runge-Kutta integrators. This is important, as we do not need to take additional

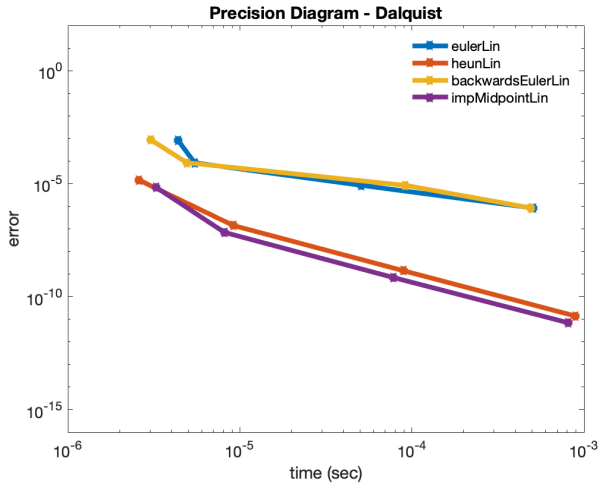

Figure 2.9: Temporal precision diagram for λ = -1

computational time, or time steps to reach lower error. Furthermore, as time continues, the error is of quadratic decrease on a log-log scale. We conclude that Runge-Kutta integrators for this simulation are the best choice.

Now consider a case where the problem is of stiff decay. We will set $\lambda = 3$. The particular solution for $y(t_0 = 1)$ is $y(t) = e^{3t}$. The previously mentioned parameters remain the same. Note the precision diagram below:

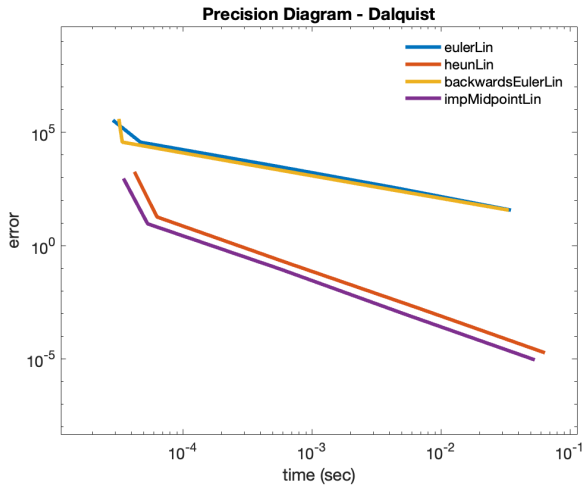

Figure 2.10: Temporal precision diagram for $\lambda = 3$

Again, both 2 stage integrators are a better choice. Additionally, the approximations do not even converge appropriately for the 1 stage integrators, retaining $e_f > 1$ even at the finest time step. A solution y_{N_t} that has a difference of more than 1 is a poor approximation. One may be tempted to ascertain that second-

order integrators are the best choice, regardless of the problem. This is entirely untrue, especially when moving to partial differential equations, where we are no longer returning a scalar value for each y_{n+1}. This will be explored further in the next chapter of the paper.

Chapter 3

Time integration for linear and nonlinear PDEs

3.1 The heat equation

The heat equation is a partial differential equation that describes the diffusion of heat across an n-dimensional space [7]. For a function $u(\mathbf{x}, t)$ the function maps $u : \mathbb{R}^{n+1} \to \mathbb{R}$, where $\mathbf{x} \in \mathbb{R}^n$, $t \in \mathbb{R}$. The general form of the heat equation n spatial dimensions is:

$$\frac{\partial u}{\partial t} = k\Delta u \tag{3.1.1}$$

Where Δ is the Laplacian operator on $u(\mathbf{x}, t)$, returning a scalar value $\Delta u = \frac{\partial^2 u}{\partial x_1^2} + ... + \frac{\partial^2 u}{\partial x_n^2}$ [7]. The diffusion constant k dictates the rate of temporal heat diffusion. For the sake of simplicity, we consider the heat equation in 1-dimensional space, such that $\mathbf{x} \in \mathbb{R}$. While this paper relates to efficiency of integrators for the heat equation, we will briefly present the analytical solution to the heat equation:

3.1.1 Analytical solution of the 1-D heat equation

Consider the 1-dimensional heat equation:

$$\frac{\partial u}{\partial t} = k\frac{\partial^2 u}{\partial x^2} \tag{3.1.2}$$

For simplicity, we set $k = 1$ and omit all further instances of it. We also must define specified boundary and initial conditions. We set $x \in [-1, 1]$, and $t \in \mathbb{R}^+$, but will only be numerically integrated on the interval $[0, .1]$. We define the BCs and IC as:

$$\text{BC} : u(0,t) = u(1,t) = 0 \tag{3.1.3}$$

$$\text{IC} : u(x,0) = \frac{8\pi}{3}\sin(\pi x) + \frac{4\pi}{6}\sin(2\pi x) - \frac{5\pi}{6}\sin(10\pi x) \tag{3.1.4}$$

We choose a multi-frequency initial condition to add some early complexity to our problem. This will be explained later. The fundamental solution to the heat equation on $x \in [-1, 1]$, $t \in [0, \text{inf})$ is widely known [7]:

$$u(x,t) = \sum_{n=1}^{\infty} b_n \sin(n\pi x)\ e^{-n^2\pi^2 t} \tag{3.1.5}$$

For sake of numerics, we only choose a finite amount of n, as one can not numerically analyze the solution at $n \to \infty$. We choose a finite amount of $n \in A = [1, 2, 10]$ for our numerical simulation, such that the particular solution is:

$$u(x,t) = \sum_{n \in A} u_n(x,t), \tag{3.1.6}$$

$$u_1(x,t) = \frac{8\pi}{3}\sin(\pi x)e^{-\pi^2 t}, \tag{3.1.7}$$

$$u_2(x,t) = \frac{4\pi}{6}\sin(2\pi x)e^{-(2\pi)^2 t}, \tag{3.1.8}$$

$$u_{10}(x,t) = -\frac{5\pi}{6}\sin(10\pi x)e^{-(10\pi)^2 t}. \tag{3.1.9}$$

We choose a high frequency solution so that we can test the error properties of our integrators on a solution with both slowly and rapidly evolving solution components. Similarly to the exact solution of the Dalquist problem, we will use this as our reference solution for this chapter of time integration.

3.1.2 Finite differences: FTCS method

To be able to understand time integration of the heat equation, we must first understand a simple numerical method to solving the heat equation. Consider the first order time derivative approximation of a function $y : \mathbb{R} \to \mathbb{R}$:

$$y' \approx \frac{y_{n+1} - y_n}{\Delta t} \tag{3.1.10}$$

This is known as a **forward approximation** in time, something that we will use throughout the paper [8]. Since space and time are both quantities in the heat equation, we will have to approximate both. We use a first order forward approximation for $\frac{\partial u}{\partial t}$ and a second order centered in space approximation for $\frac{\partial^2 u}{\partial x^2}$ such that:

$$u_t \approx \frac{u_j^{i+1} - u_j^i}{\Delta t} \tag{3.1.11}$$

$$u_{xx} \approx \frac{u_{j+1}^i - 2u_j^i + u_{j-1}^i}{(\Delta x)^2} \tag{3.1.12}$$

Substituting them in $u_t = u_{xx}$, we have:

$$\frac{u_{j+1}^i - u_j^i}{\Delta t} = \frac{u_j^{i+1} - 2u_{j+1}^i + u_j^{i-1}}{(\Delta x)^2} \tag{3.1.13}$$

$$\implies u_{j+1}^i = u_j^i + \alpha(u_j^{i+1} - 2u_{j+1}^i + u_j^{i-1}) \tag{3.1.14}$$

Where $\alpha = \Delta t/\Delta x^2$. Note that $u^i \to u(x_i)$ and $u_j \to u(t_j)$. Therefore each u_j is a column vector of heat values at various x_i, for each t_j [8][7]. Note that there is a restriction on stability for the finite difference method, such that:

$$\frac{\Delta t}{\Delta x^2} \leq \frac{1}{2}$$

This condition must be met to produce a stable solution. Going forward, we will not use finite differences, however it is important to learn as many of its notions will be used.

3.1.3 MATLAB Implementation

Listing 3.1: **Finite differences FTCS method**

```
Nx = 200;
Nt = 20000;
xspan = [-1,1];
tspan = [0,.1];

dx = diff(xspan)/(Nx-1);
dt = diff(tspan)/(Nt-1);

xs = linspace(xspan(1),xspan(end),Nx);
ts = linspace(xspan(1),xspan(end),Nt);
alpha = dt/dx^2;

u = zeros(Nx,Nt);

u(:,1) = sin(pi*xs);
u(1,:) = 0;
u(Nx,:) = 0;

for j = 1:Nt-1
    for i = 2:Nx-1
        u(i,j+1) = alpha*(u(i+1,j)-2*u(i,j)+ u(i-1,j))+
           u(i,j);
    end
end
```

Note the numerical approximation using finite differences below:

We can see the diffusive nature of the surface plot. Per the maximum principle for PDEs, the maximum value of the function occurs on the boundary of the domain [7]. Finite difference produces an extremely precise solution- although we can improve our approach to produce a more robust approximation.

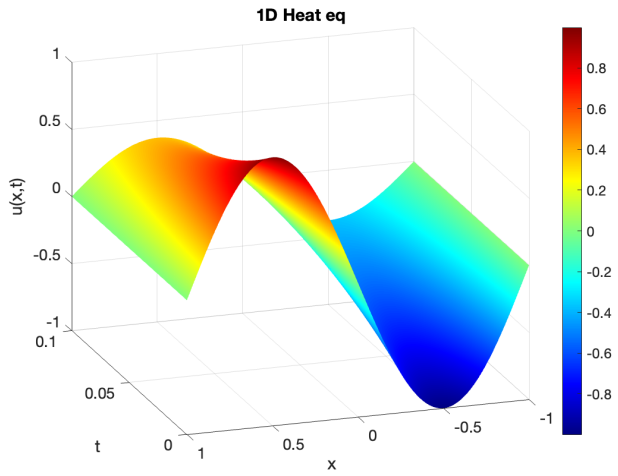

Figure 3.1: Numerical solution of the 1-D heat equation using finite differences

3.1.4 Implementing time integration: Method of lines

Recall that finite differences uses temporal and spatial **discretization**, approximating the solution at x_i, t_j such that:

$$x_i = (i-1)\Delta x \ , \ i = 1, \ldots, N_x \tag{3.1.15}$$

$$t_j = (j-1)\Delta t \ , \ j = 1, \ldots, N_t \tag{3.1.16}$$

As N_t , $N_x \to \infty$, we converge to a spatially and temporally fine solution. However, this is computationally impossible - making our solution an approximation regardless of N_x and N_t.

We can create a more flexible approximation by refining exactly in time, implementing the **method of lines** [14]. This approach differentiates exactly in time, while leaving a spatial discretization.

We denote D_{xx} as the centered in space finite difference matrix, $\mathbf{u}$ and its corresponding derivative is now a vector of $\mathbf{u} = (u_1(t), ..., u_N x(t))$ - creating a system of linear first order ODEs. The approximation can be written in augmented

matrix format as:

$$\begin{bmatrix} u_2' \\ u_3' \\ \vdots \\ u_{Nx+1}' \end{bmatrix} = \frac{1}{(\Delta x)^2} \begin{bmatrix} -2 & 1 & \dots & 0 \\ 1 & -2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 1 & -2 \end{bmatrix} \begin{bmatrix} u_2 \\ u_3 \\ \vdots \\ u_{Nx+1} \end{bmatrix} \tag{3.1.17}$$

$$\frac{d\mathbf{u}}{dt} = \frac{u_j^{i+1} - 2u_j^i + u_j^{i-1}}{\Delta x^2}\mathbf{u} \tag{3.1.18}$$

$$\implies \frac{d\mathbf{u}}{dt} = D_{xx}\mathbf{u} \tag{3.1.19}$$

Note that we slightly switch the discretization to account for homogeneous boundary conditions. If we have N_x spatial points, we set $u(x_1, t)$ and $u(x_{Nx+2}, t) = 0$, and therefore only define our system on interior grid points, such that $u_2', \dots, u_{N_x+1} \neq 0$. Note that the discretized heat equation (3.1.19) has *exact* temporal solution:

$$\mathbf{u}(t) = e^{D_{xx}t}\mathbf{u}_0 \tag{3.1.20}$$

Again, $\frac{d\mathbf{u}}{dt}$ is a vector of solution values at each spatial grid point. If we have $N_x + 2$ spatial points, we have N_x nonzero rows of the finite difference matrix. Now that we have a system of form $\mathbf{u}' = A\mathbf{u}$, a linear system, we can apply any time integrator to solve the system precisely in time. At each time step, $\mathbf{u}_n \in \mathbb{R}^{N_x}$ will produce a vector of solution values at each spatial grid point[7][8][14].

3.1.5 Temporal error for the heat equation

Now that we have a system of ODE's of form $\mathbf{u}' = \mathbf{f}(\mathbf{u}) = A\mathbf{u}$, we can apply any of the four time integrators described in Chapter 2. For example, below is an application of forward Euler:

$$\mathbf{u}_{n+1} = \mathbf{u}_n + hD_{xx}\mathbf{u}_n \tag{3.1.21}$$

The term $\mathbf{u}_n$ represents the solution at each spatial gridpoint $x_n, n = 1, \ldots, N_x$ at time $t_n = (n-1)\Delta t$. As usual, h is the time step. Since $\mathbf{u}' = A\mathbf{u}$, the exact solution is similar to the Dalquist problem such that $\mathbf{u}(t) = e^{D_{xx}t}\mathbf{u}_0$, where $\mathbf{u}_0 = u(x, 0)$ the initial condition. The error is subsequently:

$$e_{t_f} = ||\mathbf{u}_{N_{t+1}} - e^{D_{xx}t_f}\mathbf{u}_0|| \tag{3.1.22}$$

Where $\mathbf{u}_{N_{t+1}}$ are solution values at time $t_{N_{t+1}}$ at each spatial grid point. Since we are dealing with vectors in $\mathbb{R}^{N_x}$ we implement at 2-norm, since we are performing vector differences. The remaining time integrator schemes for Backward Euler, Heun, and implicit midpoint respectively become:

$$\mathbf{u}_{n+1} = (I - hD_{xx})^{-1}\mathbf{u}_n \tag{3.1.23}$$

$$\mathbf{u}_{n+1} = \mathbf{u}_n + hD_{xx}\mathbf{u}_n + \frac{h^2 D_{xx}^2}{2}\mathbf{u}_n \tag{3.1.24}$$

$$\mathbf{u}_{n+1} = \mathbf{u}_n + h\left[(I - \frac{h}{2}D_{xx})^{-1}D_{xx}\mathbf{u}_n\right] \tag{3.1.25}$$

Both implicit methods now require a solve of the tridiagonal matrix $(I - hD_{xx}$ for each time step t_n. This can be computationally expensive for large spatial discretizations N_x, and reduces much of the benefit we saw from our implicit integrators on Dalquist [2]. This will be discussed in the next section.

3.1.6 Convergence and precision of integrators for heat equation

We proceed by measuring the 2-norm of the error at the final time step at each spatial grid point for the method of lines. We run our integrators at time steps $N_t \in \{10^2, \ldots, 10^6\}$ and choose $N_x = 50$. The effect of N_x on stiffness will be

discussed in the next section. As previously mentioned, the BCs and IC are:

$$\text{BC} : u(0,t) = u(1,t) = 0, \tag{3.1.26}$$

$$\text{IC} : u(x,0) = \frac{8\pi}{3}\sin(\pi x) + \frac{4\pi}{6}\sin(2\pi x) - \frac{5\pi}{6}\sin(10\pi x). \tag{3.1.27}$$

Note below the convergence (error vs. h) and precision (error vs. computational time) diagrams below:

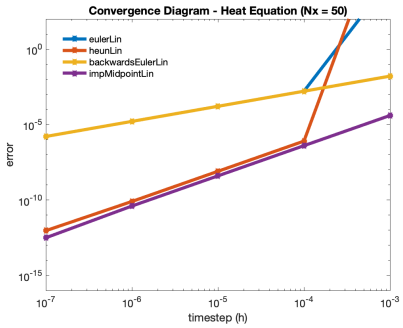

Figure 3.2: Convergence diagram for the 1-D heat equation

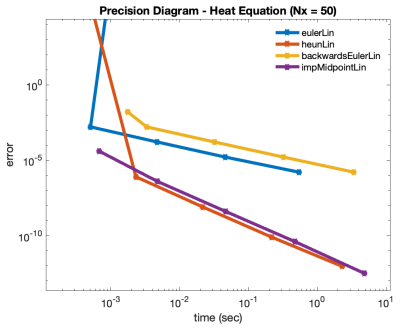

Figure 3.3: Precision diagram for the 1-D heat equation

For the convergence diagram, we notice that both forward Euler and Heun do not converge for $\Delta t > 10^{-4}$. This is due to their finite stability regions, as the eigenvalues of the finite difference matrix fall outside their stability region [8]. The implicit methods succeed because of their A-stable regions.

For the sake of comparison, we will choose a temporal threshold of error $\leq 10^{-1}$ i.e. $||\mathbf{u}_{Nt+1} - e^{D_{xx}t_f}\mathbf{u}_0|| \leq 10^{-1}$. We see that in fact, forward Euler gives us the quickest method of solution, reaching the threshold in $T = .0001$ seconds. Although implcit midpoint has $O(h^2)$ temporal convergence, it takes 10 times the

amount of time to even begin the method, due to each matrix solve that must be computed. Similarly to the Dalquist problem, unstiff systems are best solved by an explicit integrator. We will continue by investigating the effect of spatial grid size on each time integrator.

3.1.7 Spatial error and spectral radius of FD matrix

Recall that the method of lines gives us a linear system of ODEs, such that:

$$\mathbf{u}' = D_{xx}\mathbf{u}$$

With solution $\mathbf{u}(t) = e^{D_{xx}t}\mathbf{u}_0$. While the equation is exact in time, the spatial derivative u_xx was replaced with the second-order approximation:

$$\frac{\partial^2 u}{\partial x^2} \approx \frac{u^i_{j+1} - 2u^i_j + u^i_{j+1}}{(\Delta x)^2} \tag{3.1.28}$$

We can measure spatial error by taking the 2-norm of the difference between $e^{D_{xx}t_f}$, the exact in time solution, and the vector $\{u(x_i,t)\}_{i=1}^{N_x}$, where we evaluate the heat kernel solution at each discretized point of x_i. The error expression is:

$$\text{error}_s = ||\{u(x_i,t)\}_{i=1}^{N_x} - e^{D_{xx}t_f}\mathbf{u}_0|| \tag{3.1.29}$$

If we plot the temporal error and spatial error on the same axis, we anticipate $|\{u(x_i,t)\}_{i=1}^{N_x}| - e^{D_{xx}t_f}\mathbf{u}_0||_2$ to be a horizontal line, as both terms in the norm are not computed using a time integrator. Therefore the colored lines are temporal error diagrams, and the horizontal line is spatial error. We anticipate spatial error to decrease as $N_x \to \infty$. Note the convergence diagrams with spatial error for $N_x = 150$ below:

We first note that the spatial error is indeed a constant line such that $\text{error}_s = 4.7e^{-5}$- the number is rather small because of the highly fine spatial discretization. Examining the convergence diagram, we see that for Euler and Heun once error > error_s, the methods become unstable. Conversely, the implicit integrators are

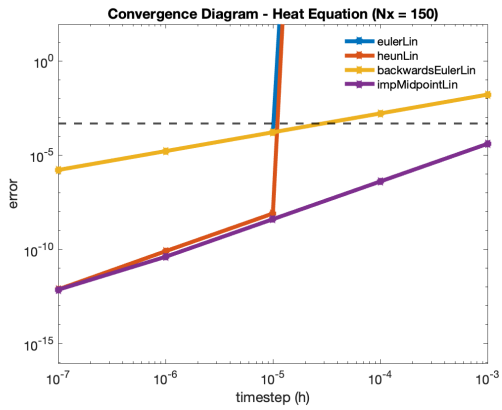

Figure 3.4: Temporal convergence diagram for the heat equation

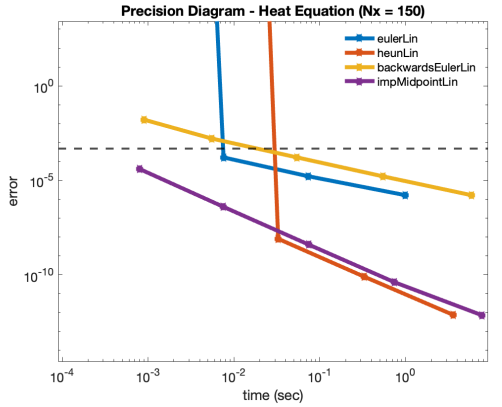

Figure 3.5: Temporal precision diagram for the heat equation

unconditionally convergent, because they are stable for $\mathrm{Re}(z) < 0$.

For the precision diagram, we choose an error threshold such that $\epsilon = \mathrm{error}_s$. The clear viable method choice is implicit midpoint - as it reaches the spatial error threshold at $T = 10^{-3}$ seconds. All the other integrators take significantly more computational time to reach this threshold. This is an example where implicit midpoint is the best choice.

To understand why N_x can cause an explicit method to go unstable, recall the finite difference matrix D_{xx}:

$$D_{xx} = \frac{1}{(\Delta x)^2}\begin{bmatrix} -2 & 1 & & \\ 1 & \ddots & \ddots & \\ & \ddots & \ddots & 1 \\ & & 1 & -2 \end{bmatrix} \tag{3.1.30}$$

If we let $N_x \to \infty$ this implies $\Delta x \to 0$ which implies $1/\Delta x^2 \to \infty$. Since the finite difference approximation is scaled by this term, each entry $\{D_{xx_{ij}}\} \to \infty$. Similarly to the value of λ for the Dalquist problem, D_{xx} has N_x eigenvalues $\{\lambda_i\}_{i=1}^{N_x}$ that solve $\det(D_{xx} - \lambda_i I) = 0$. The FD matrix eigenvalues becomes extremely large as $N_x \to \infty$. This can cause them to fall out of the region of stability of all explicit methods [14]. We will formally assign stiffness to N_x in the next section.

3.1.8 Refining in space and time simultaneously

Recall that the MOL form of the heat equation is:

$$\mathbf{u}' = D_{xx}\mathbf{u} \tag{3.1.31}$$

We implement a second-ordered finite difference pace approximation for the $\frac{\partial^2 u}{\partial x^2}$. The eigenvalues are given by $\det(D_{xx} - \lambda I) = 0$ such that:

$$(D_{xx} - \lambda I) = \frac{1}{\Delta x^2}\begin{bmatrix} -2-\lambda & 1 & 0 & \dots & 0 \\ 1 & -2-\lambda & 1 & \dots & 0 \\ 0 & 1 & -2-\lambda & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & 1 & -2-\lambda \end{bmatrix} \tag{3.1.32}$$

Recall that the stiffness in the Heat equation comes from $\rho(hD_{xx})$. To determine the spectral radius of D_{xx} we will derive the general expression for the eigenvalues of the second derivative. Consider the 2nd ordered centered in space expression for $u(x,t)$ where time is suppressed and n is the amount of spatial grid points, using $Au_i = u\lambda_i$ [4]:

$$\frac{u_{i+1} - 2u_i + u_{i-1}}{dx^2} = u_i\lambda_i \tag{3.1.33}$$

$$\implies u_{i+1} = (2 + dx^2\lambda)u_i - u_{i-1} \tag{3.1.34}$$

Let $2\alpha = (2 + h^2\lambda)$

$$\implies u_{k+1} = 2\alpha u_k - u_{k-1} \tag{3.1.35}$$

Letting a_i be a Chebychev polynomial:

$$\alpha_i = \cos\left(\frac{i\pi}{n+1}\right) \tag{3.1.36}$$

$$\implies 2\cos\left(\frac{i\pi}{n+1}\right) = dx^2\lambda_i + 2 \tag{3.1.37}$$

$$\implies \lambda_k = \frac{-4}{dx^2}\sin^2\left(\frac{i\pi}{n+1}\right) \implies \max\{\lambda_i\}_{i=1}^{n} = \lambda_{\max} = \frac{4}{dx^2} \tag{3.1.38}$$

For a finite difference matrix of size $N_x \times N_x$, we have the set $\Lambda = \{\lambda_i\}_{i=1}^{N_x}$. Note that for the heat equation each $\lambda_i \leq 0$ which implies that any implicit integrator will converge, as the eigenvalues lie entirely on the $\mathrm{Re}(z) < 0$ axis, in the negative half plane. Shown below is this phenomenon:

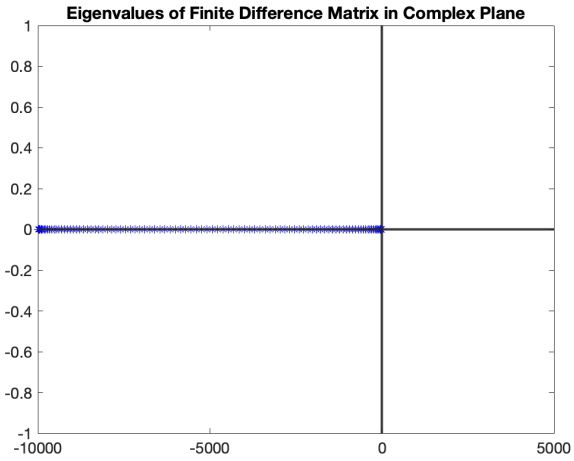

Figure 3.6: Eigenvalues in C for D_{xx} for given parameters

Notice that the for the stability region of forward Euler in $\mathbb{C}$, the eigenvalues λ, and specifically the spectral radius $\rho(hD_{xx}) = \lambda_{max}$ fall out of the inequality $|1 + h\lambda| \leq 1$, causing the method to go unstable. Implicit integrators do not place a restriction on stepsize for this specific problem.

We can now apply a formal definition of stability for the heat equation. Using

the stability condition of forward Euler, and plugging in the value of $\lambda_{\max}$:

$$|1 + h\lambda_{\max}| \leq 1 \implies |1 + \Delta t \frac{4}{\Delta x^2}| \leq 1 \tag{3.1.39}$$

$$\implies h \leq \frac{dx^2}{2} \tag{3.1.40}$$

Letting $x \in [-1, 1]$ and N_x spatial grid points, we can write h as function of N_x, such that:

$$\Delta t \leq \frac{2}{(N_x + 1)^2} \tag{3.1.41}$$

Since we use $N_x + 1$ non homogeneous grid values [4]. Note that Heun has the same stability region for $\mathrm{Re}(z)$.

Recall that for both backward Euler and implicit midpoint there exists no stability restriction. However, we still must select h carefully in order to balance spatial and temporal error.

For a method of $\mathcal{O}(h^p)$ temporal convergence, and $\mathcal{O}(h^q)$ spatial convergence, the error can be written as a function of temporal and spatial grid points:

$$\text{Temporal error} = \mathcal{O}\left(\frac{N_{t_0}}{N_t}\right)^p \tag{3.1.42}$$

$$\text{Spatial error} = \mathcal{O}\left(\frac{N_{x_0}}{N_x}\right)^q \tag{3.1.43}$$

Where N_{t_0} and N_{x_0} represent the initial amount of temporal and spatial steps taken. We can fine tune the amount of temporal steps taken by N_x through choices of N_{t_0}, N_{x_0}, by equating the two expressions and solving for N_t:

$$\left(\frac{N_{t_0}}{N_t}\right)^p = \left(\frac{N_{x_0}}{N_x}\right)^q \implies \ln\left(\frac{N_{t_0}}{N_t}\right) = \frac{q}{p}\ln\left(\frac{N_{x_0}}{N_x}\right) \tag{3.1.44}$$

$$\implies N_t = \frac{N_{t_0}}{e^{\frac{q}{p}\alpha}}\ ,\ \alpha = \ln\frac{N_{x_0}}{N_x} \tag{3.1.45}$$

Now we have $N_t = f(N_x)$, allowing us to refine simultaneously in space and time. It is obvious that for the Heat equation, $q = 2$ regardless of method, as we take

second ordered difference in space. To recap, the equations that prescribe N_t as a function of spatial grid size N_x, we have:

$$\text{Euler} := \frac{2}{(N_x+1)^2} \ , \ \text{Heun} = \frac{2}{(N_x+1)^2} \tag{3.1.46}$$

$$\text{Backward Euler} = \frac{414}{(\frac{50}{N_x})^2} \ , \ \text{Implicit Midpoint} = \frac{18}{(\frac{50}{N_x})} \tag{3.1.47}$$

We can find the target N_{t_0} and N_{x_0} numerically, then proceed by applying those to the expression that we derived in (79). Shown below is the algorithm and subsequent convergence. precision diagram for refining simultaneously in space and time for the 1-D heat equation. The parameters in the code are the parameters used for generating the diagrams:

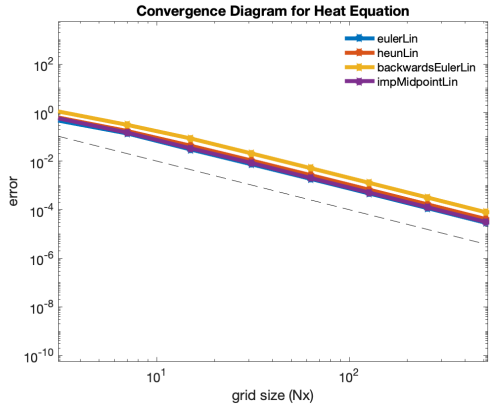

Figure 3.7: Spatial convergence diagram $\mathcal{O}(\Delta x^2)$ for spatial and temporal refinement, heat equation

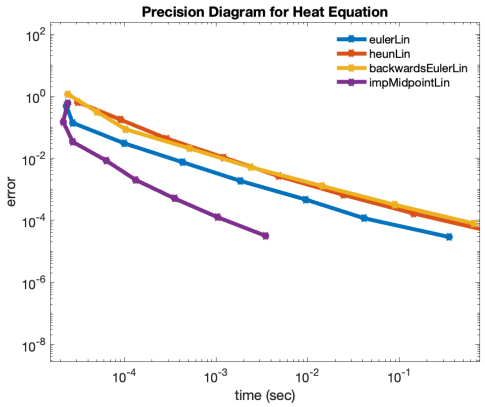

Figure 3.8: Precision diagram for spatial and temporal refinement, heat equation

The convergence diagram is now showing $||\{u(x_i,t)\}_{i=1}^{N_x} - e^{D_{xx}t_f}\mathbf{u}_0||$, the spatial error at each step. Since we used centered second order differences for D_{xx}, each line has $\mathcal{O}(N_x^{-2})$ convergence. The dotted line shows second order spatial conergence.

The precision diagram gives us a conclusive notion about solving the Heat equation using simultaneous refinement in time and space. We conclude that implicit midpoint is the most robust and effecient method, reaching the lowest error threshold in the smallest amount of computational time. This is due to both its order $\mathcal{O}(h^2)$ temporal convergence, and its stability region, leading to less N_t steps taken and therefore much less computational time. We now continue our study of time integration with a non-linear partial differential equation [14][2].

3.2 Viscous Burgers' equation in 1-dimension

We now seek an understanding of time integrators for a quasi-linear partial differential equation. The fundamental behavior and efficiency of each integrator will change as we introduce nonlinearity to our problem.

Consider the **convective-diffusive** wave equation, also known as the **viscous Burgers** equation. Similarly to the heat equation, the solution will only governed by $u(\mathbf{x},t)$, such that $\mathbf{x} \in \mathbb{R}$ (the one dimensional case) [16].

$$u_t = \epsilon u_{xx} - u u_x \tag{3.2.1}$$

In the inviscid case ($\epsilon = 0$) the equation defaults to the inviscid Burgers equation, written as:

$$u_t = -u u_x \tag{3.2.2}$$

The viscous Burgers' equation governs a traveling wave in one dimension, that is dampened by diffusion. The term $\epsilon\frac{\partial^2 u}{\partial x^2}$ is the contributing factor that leads to diffusion, while $\frac{\partial u}{\partial x}$ is the convective behavior [16]. We are less concerned with an

analytical method for Burger's equation, but will briefly discuss it.

3.2.1 The Cole-Hopf transformation

The goal of the *Cole-Hopf Transformation* is to transform the viscous Burgers' equation into a linear PDE that can be analytically solved. The PDE can be first rewritten as:

$$u_t = \epsilon u_{xx} - u u_x \tag{3.2.3}$$

$$\implies u_t = \epsilon u_{xx} - \frac{d}{dx}\left(\frac{u^2}{2}\right) \tag{3.2.4}$$

Or in Hamilton-Jacobi form as

$$U_t + \frac{1}{2}(U_x)^2 = \epsilon U_{xx} \ , \ U_x = u \tag{3.2.5}$$

Introducing the *Cole-Hopf* relation gives:

$$U(x,t) = -2\kappa \ log[\phi(x,t)] \tag{3.2.6}$$

Taking the necessary partial derivatives of $U(x,t)$ and substituting into (3.2.6), we get:

$$\phi_t = \kappa \phi_{xx} \tag{3.2.7}$$

The equation $\phi(x,t)$ is the fundamental solution to the heat equation in 1-dimension. As we previously noted, fundamental solution is the *heat kernel* $H(x,t)$ convolved with the initial condition, such that:

$$\phi(x,t) = \int_{-\infty}^{\infty} H(x-y,t) \ \phi(y,0) \ dy \tag{3.2.8}$$

$$= \int_{-\infty}^{\infty} \frac{1}{\sqrt{4\pi\kappa t}} e^{\frac{-(x-y)^2}{4\kappa t}} \ \phi(y,0) \ dy \tag{3.2.9}$$

We can now assemble the solution to $U(x,t)$ and subsequently $u(x,t)$ the Viscous Burger's Equation, such that:

$$U(x,t) = -2\kappa \log\left[\int_{-\infty}^{\infty} \frac{1}{\sqrt{4\pi\kappa t}} e^{\frac{-(x-y)^2}{4\kappa t}}\ \phi(y,0)\ dy\right] \tag{3.2.10}$$

$$\implies u(x,t) = \frac{\int_{-\infty}^{\infty} \frac{x-y}{t} \frac{1}{\sqrt{4\pi\kappa t}} e^{\frac{-(x-y)^2}{4\kappa t}}\ \phi(y,0)\ dy}{\int_{-\infty}^{\infty} \frac{1}{\sqrt{4\pi\kappa t}} e^{\frac{-(x-y)^2}{4\kappa t}}\ \phi(y,0)\ dy} \tag{3.2.11}$$

Equation (3.2.11) is an analytical solution to the Viscous Burger's equation [9]. We present this for two reasons. The first being to understand the important relationship between the heat kernel and the traveling wave solution. One can also see the difficulty in finding an analytical solution. A majority of nonlinear partial differential equations do not retain a closed form solution, necesitating time integration.

3.2.2 Method of lines for a nonlinear PDE

Reconsider the viscous Burger's equation:

$$u_t = \epsilon u_{xx} - u u_x \tag{3.2.12}$$

The product of uu_x causes nonlinearity. The PDE can be rewritten as:

$$u_t = \epsilon u_{xx} - \frac{d}{dx}\left(\frac{u^2}{2}\right) \tag{3.2.13}$$

As we did with the heat equation, we can discretize our spatial domain, creating an autonomous system of nonlinear ODEs, such that:

$$\mathbf{u}' = \epsilon D_{xx}\mathbf{u} - \frac{d}{dx}(\frac{\mathbf{u}^2}{2}) \tag{3.2.14}$$

Similarly to $\frac{\partial}{\partial x^2}$, we can discretize $\frac{\partial}{\partial x}$, using a first ordered center difference, such that:

$$\frac{\partial u}{\partial x} \approx \frac{u_j^{i+1} - u_j^{i-1}}{2\Delta x} \tag{3.2.15}$$

The subsequent first order finite difference matrix, D_x, is:

$$D_x = \frac{1}{2\Delta x} \begin{bmatrix} 0 & 1 & \dots & 0 \\ -1 & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 1 \\ 0 & 0 & -1 & 0 \end{bmatrix} \tag{3.2.16}$$

Note that the autonomous system in finite difference matrix discretized form, using $\mathbf{u} = [u_2 \dots, u_{N_x+1}]$ interior non-zero grid points, Burger's is written as:

$$\begin{bmatrix} u_2' \\ \vdots \\ u_{N_x}' \\ u_{N_x+1}' \end{bmatrix} = \frac{\epsilon}{(\Delta x)^2} \begin{bmatrix} -2 & 1 & \dots & 0 \\ 1 & -2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 1 & -2 \end{bmatrix} \begin{bmatrix} u_1 \\ \vdots \\ u_{N_t-1} \\ u_{N_t} \end{bmatrix} \dots$$
$$\dots - \frac{1}{2\Delta x} \begin{bmatrix} 0 & 1 & \dots & 0 \\ -1 & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 1 \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} u_1^2/2 \\ \vdots \\ u_{N_t-1}^2/2 \\ u_{N_t}^2/2 \end{bmatrix} \tag{3.2.17}$$

Condensing it using $\mathbf{u}$ as a vector of each time derivative, we have:

$$\mathbf{u}' = \epsilon D_{xx} - D_x \frac{\mathbf{u}^2}{2} \tag{3.2.18}$$

Each integrator then takes the following form, where $\mathbf{u}_{n+1}$ is the height of the wave at time t_{n+1} at each x_i along the spatial disretiztion [16]. The numerical

solution to Burgers' equation using forward Euler is:

$$\mathbf{u} = \mathbf{u}_n + h\left(D_{xx}\mathbf{u}_n - D_x\frac{\mathbf{u}_n^2}{2}\right) \tag{3.2.19}$$

Similarly, Heun yields:

$$\mathbf{k}_1 = \epsilon D_{xx}\mathbf{u}_n - D_x\frac{\mathbf{u}_n^2}{2} \tag{3.2.20}$$

$$\mathbf{k}_2 = \epsilon D_{xx}(\mathbf{u}_n + h\mathbf{k}_1) + D_x\left[\frac{(\mathbf{u}_n + h\mathbf{k}_1)^2}{2}\right] \tag{3.2.21}$$

$$\mathbf{u}_{n+1} = \mathbf{u}_n + h\left(\frac{\mathbf{k}_1}{2} + \frac{\mathbf{k}_2}{2}\right) \tag{3.2.22}$$

Yet we encounter additional complexity when attempting to use an implicit integrator, displayed below:

$$\mathbf{u}_{n+1} = \mathbf{u}_n + hf(\mathbf{u}_{n+1}) \tag{3.2.23}$$

$$\implies \mathbf{u}_{n+1} = \mathbf{u}_n + h\left(\epsilon D_{xx}\mathbf{u}_{n+1} - D_x\frac{\mathbf{u}_{n+1}^2}{2}\right) \implies \mathbf{u}_{n+1} = ? \tag{3.2.24}$$

The term $\mathbf{u}_{n+1}^2/2$ is nonlinear, requiring a *nonlinear solve* at each $\mathbf{u}_{n+1}$ step. This must be done numerically, and requires significantly more computational time. In this work, we will not consider the implementation of implicit integrators, in favor of a different integrator formula that will be discussed in subsection 3.2.3.

Before we analyze the effeciency of integrators for Burger's equation, we will compute its numerical solution. We set $x \in [-1, 1]$, $t \in [0, .1]$ and choose $N_t = 10^4$, $N_x = 63$. The initial position of the wave is set to $u(x, 0) = -\sin(\pi x)$. We will use a forward Euler scheme to compute the solution. The solution of Viscous Burger's by Euler's method is shown below:

The initial conditon is by $u(x, 0) = -\sin(\pi x)$, which leads to a shock as the peaks and troughs move closer. We can see this by plotting various curves in the $(x, u(x, t_i))$ plane, where t_i is fixed:

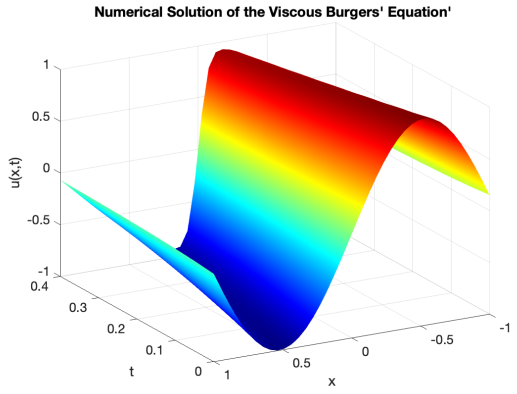

Figure 3.9: Numerical solution to the Viscous Burger's equation in 1-D

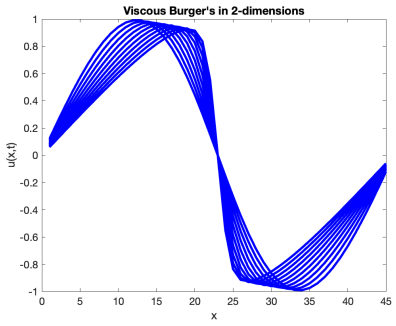

Figure 3.10: Numerical solution to Burger's for various $u(x, t_i)$ in $\mathbb{R}^2$

3.2.3 Implicit-Explicit (IMEX) time integration

Since we are unable to implement fully implicit integrators for this equation, we seek to gain stability through the linear component of viscous Burgers', where $L = \epsilon D_{xx}$. We will introduce other methods of time integration that handle linear and nonlinear components separately.

Recall that the difference of $y_{n+1} - y_n$ can be written as an integral formula, such that:

$$y_{n+1} - y_n = \int_{t_n}^{t_{n+1}} f(t, y)\, dt \approx y_{n+1} = y_n + hf(t_n, y_n) \tag{3.2.25}$$

We can apply this same expression using an autonomous ODE, such that $y' = F(y)$. Suppose we can partition any ordinary differential equation by its linear

and nonlinear components, such that:

$$y' = Ly + N(y) \tag{3.2.26}$$

Where L is either a scalar or a matrix and $N(y)$ is a nonlinear function. The term $\mathbf{y}$ can be either a vector or a scalar, but for now we will drop the boldface and assume scalar value. Note that we can find the difference of $y_{n+1} - y_n$ using left rectangle integration, such that:

$$y' = Ly + N(y) \tag{3.2.27}$$

$$\implies y_{n+1} - y_n = \int_{t_n}^{t_{n+1}} Ly + N(y)\ dt \tag{3.2.28}$$

$$\implies y_{n+1} - y_n = \int_{t_n}^{t_{n+1}} Ly + \int_{t_n}^{t_{n+1}} N(y)\ dt \tag{3.2.29}$$

Asumme that $N(y(t_n)) \approx N(y_n))$, and therefore will be taken as a constant at the nth step. Additionally, we will approximate the integral of Ly using an implicit Reimann approximation at t_{n+1}, s.t. $\int_{t_n}^{t_{n+1}} Ly = hLy_{n+1}$ [1]. Using $t_{n+1} - t_n = h$, this reduces to:

$$y_{n+1} = y_n + (t_{n+1} - t_n)Ly_{n+1} + (t_{n+1} - t_n)N(y_n) \tag{3.2.30}$$

$$y_{n+1} = y_n + hLy_{n+1} + hN(y_n) \tag{3.2.31}$$

Rearranging terms and solving for y_{n+1} the scheme is:

$$y_{n+1} = (I - hL)^{-1}(y_n + hN(y_n)) \tag{3.2.32}$$

Equation (3.2.30) is the **IMEX Euler** time integrator [1].The term I denotes an identity matrix of dimensions N_x. When dealing with an ODE $N_x = 1$, and $I = 1$. Since we solve for y_{n+1} semi-implicitly, we hope to obtain improve stability, and be able to take larger stepsizes. For Burgers' equation, the IMEX scheme is written

as:

$$\mathbf{u}' = \epsilon D_{xx}\mathbf{u} - D_x\frac{\mathbf{u}^2}{2} \tag{3.2.33}$$

$$\implies \mathbf{u}_{n+1} = (1 - h\epsilon D_{xx})^{-1}(\mathbf{u}_n + D_x\frac{(\mathbf{u}_{n+1})^2}{2}) \tag{3.2.34}$$

When we discuss efficiency of methods, the benefit of IMEX will be seen more. For now, we continue to derive more integrators that only treat the linear term implicitly.

3.2.4 Exponential time integration

Reconsider the problem $y' = Ly + N(y)$. Knowing that L, N are the respective linear and nonlinear components. Since $N(y(t)) = f(t)$ we can rewrite $y' = Ly + N(y)$ as a first order linear non-autonomous ODE, such that:

$$y' - Ly = N(y(t)) \tag{3.2.35}$$

Setting the integrating factor as $e^{\int -Ldt}$, we can write the solution to the ODE as:

$$y' - Ly = N(y(t)) \tag{3.2.36}$$

$$\implies \int_{t_n}^{t_{n+1}} \frac{d}{dt}(e^{-Lt}y) = \int_{t_n}^{t_{n+1}} e^{-Lt}N(y(t))\ dt \tag{3.2.37}$$

$$\implies e^{-Lt_{n+1}}y_{n+1} - e^{-Lt_n}y_n = \int_{t_n}^{t_{n+1}} e^{-Lt}N(y(t))\ dt \tag{3.2.38}$$

$$\implies y_{n+1} = e^{hL}y_n + e^{Lt_{n+1}}\int_{t_n}^{t_{n+1}} e^{-L\hat{t}}N(y(t)\ d\hat{t} \tag{3.2.39}$$

Since we set $N(y) = N(y(t))$, we can make the assumption that $N(y(t_n)) = N(y_n)$ and reduce the equation to:

$$y_{n+1} = e^{hL} y_n + e^{Lt_{n+1}} \int_{t_n}^{t_{n+1}} e^{-L\hat{t}} N(y_n) d\hat{t} \tag{3.2.40}$$

$$\implies y_{n+1} = e^{hL} y_n + e^{Lt_{n+1}} \left[\frac{-1}{L} N(y_n)(e^{-Lt_{n+1}} - e^{-Lt_n}) \right] \tag{3.2.41}$$

$$\implies y_{n+1} = e^{hL} y_n - (1 - e^{hL}) L^{-1} N(y_n) \tag{3.2.42}$$

Equation (3.2.42) is the **exponential time integrator** and the process is called exponential time differencing (ETD) [10]. We now have 4 separate schemes to run on the Viscous Burger's equation. Applying this scheme to the MOL Burger's Equation we have:

$$\mathbf{u}' = \epsilon D_{xx} \mathbf{u} - D_x \frac{\mathbf{u}^2}{2} \tag{3.2.43}$$

$$\implies \mathbf{u}_{n+1} = e^{hD_{xx}} \mathbf{u}_n - (I - e^{hD_{xx}}) L^{-1} N(\mathbf{u}_n) \tag{3.2.44}$$

Exponential time differencing regains more stability, since we compute matrix functions of D_{xx}.

Both IMEX Euler and ETD have $\mathcal{O}(h)$ temporal convergence. We continue by seeking methods with higher-order convergence, so we can reach a suitable error threshold perhaps quicker. [10][1]

3.2.5 IMEX Runge-Kutta methods

While we have discussed both IMEX and exponential time integrators, we only have considered methods with $\mathcal{O}(h)$ temporal convergence. We can devise IMEX RK methods, that can achieve higher order convergence while also only treating the linear term implicitly[1]. Reconsider the convection-diffusion nature of

Burger's equation, and its corresponding spatial discretization:

$$u_t = \epsilon \Delta u - u u_x \tag{3.2.45}$$

$$\mathbf{u}' = \epsilon D_{xx}\mathbf{u} - D_x \frac{\mathbf{u}^2}{2} = L(\mathbf{u}) + \mathbf{N}(\mathbf{u}) \tag{3.2.46}$$

Where F is a nonlinear function. We take G to be a linear operator, such that $L(u) = Lu$. We will assign stage values to both the linear and nonlinear components of the problem. The problem can then be written as $u' = Lu + N(u)$, where $L_{n+1} = Lu_{n+1}$, and $F_{n+1} = F_n$ We cast the Lu problem implicitly, to gain linear stability[1].

The general IMEX RK method can be written as:

$$N_{i+1} = N(Y_i) \tag{3.2.47}$$

$$L_i = LY_i \tag{3.2.48}$$

$$Y_i = u_n + h\sum_{j=1}^{i} a_{ij}L_j + h\sum_{j=1}^{i} \hat{a}_{i+1,j}N_j \qquad i = 1, \ldots, s \tag{3.2.49}$$

$$u_{n+1} = u_n + h\sum_{i=1}^{s} b_i L_i + h\sum_{i=1}^{\sigma} \hat{b}_i N_i \tag{3.2.50}$$

Where a_{ij}, b_i, c_i are given by the DIRK (diagonally implicit Runge Kutta) tableau:

$$\begin{array}{c|ccccc} 0 & 0 & 0 & 0 & \ldots & 0 \\ c_1 & 0 & a_{11} & 0 & \ldots & 0 \\ c_2 & 0 & a_{21} & a_{22} & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ c_s & 0 & a_{s1} & a_{s2} & \ldots & a_{ss} \\ \hline & 0 & b_1 & b_2 & \ldots & b_s \end{array}$$

and the corresponding explicit Runge-Kutta (ERK) tableu for $N(u)$ is given by:

$$
\begin{array}{c|ccccc}
0 & 0 & 0 & 0 & \dots & 0 \\
\hat{c}_1 & \hat{a}_{21} & 0 & 0 & \dots & 0 \\
\hat{c}_2 & \hat{a}_{31} & \hat{a}_{32} & 0 & \dots & 0 \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
\hat{c}_{s-1} & \hat{a}_{s1} & \hat{a}_{s2} & \hat{a}_{s3} & \dots & 0 \\
\hline
 & \hat{b}_1 & \hat{b}_2 & \hat{b}_3 & \dots & \hat{b}_s
\end{array}
$$

The term N_{i+1} is the value of the nonlinear function N at Y_i. The term L_i is the linear operator, denoted L, coupled with the value of Y_i [1]. Note that G_i is cast implicitly. Y_i are the stage values.

3.2.6 IMEX Midpoint method

We look to implement several IMEX RK integrators and test them on Burgers'. The IMEX midpoint method has respective DIRK and ERK tableaus for $u' = Lu + N(u)$ as:

$$
\begin{array}{c|cc}
0 & 0 & 0 \\
\frac{1}{2} & 0 & \frac{1}{2} \\
\hline
 & 0 & 1
\end{array}
\quad , \quad
\begin{array}{c|cc}
0 & 0 & 0 \\
\frac{1}{2} & \frac{1}{2} & 0 \\
\hline
 & 0 & 1
\end{array}
$$

Note that applying the expression (118) for IMEX midpoint we get the scheme:

$$u_{n+1} = u_n + h\sum_{i=1}^{s} b_i L_i + h\sum_{i=1}^{\sigma} \hat{b}_i N_i \tag{3.2.51}$$

$$u_{n+1} = u_n + h(L_1 + N_2) \tag{3.2.52}$$

This method has $\mathcal{O}(h^2)$ temporal convergence, and treats the linear term implicitly. This makes IMEX midpoint an extremely useful method [1].

3.2.7 MATLAB implementation

Shown below is a MATLAB implementation of the IMEX Euler time integrator:

Listing 3.2: IMEX midpoint time integrator

```
function [ys,cpu_time] = IMEXmidpoint(L,Nl,tspan,y0,N)

ys = zeros(length(y0),N+1);
ys(:,1) = y0;
y = y0;
dt = diff(tspan)/N;

if(issparse(L))
    I = speye(length(y0));
else
    I = eye(length(y0));
end

tic
for i = 1:N
    k1_hat = Nl(y);
    k1 = L*((I-dt/2*L)\(y+dt/2*k1_hat));
    k2_hat = Nl(y+dt/2*(k1+k1_hat));
    y = y+dt*(k1+k2_hat);
    ys(:,i+1) = y;
end
cpu_time = toc;
```

3.2.8 L-stable, 2-stage, 2-ordered DIRK method

Consider a method where two stages are taken for DIRK, and three taken for ERK, with corresponding tableaus:

$$\begin{array}{c|cc} \gamma & \gamma & 0 \\ 1 & 1-\gamma & \gamma \\ \hline & 1-\gamma & \gamma \end{array} \quad , \quad \begin{array}{c|ccc} 0 & 0 & 0 & 0 \\ \gamma & \gamma & 0 & 0 \\ 1 & \delta & 1-\delta & 0 \\ \hline & 0 & 1-\gamma & \gamma \end{array}$$

Note that $b \in \mathbb{R}^s$ and $\hat{b} \in \mathbb{R}^\sigma$, with $A \in \mathbb{R}^{s\times s}$, $A \in \mathbb{R}^{\sigma\times\sigma}$. This yields the scheme:

$$u_{n+1} = u_n + h\Big[(1-\gamma)(L_2+N_2) + \gamma(L_3+N_3)\Big] \tag{3.2.53}$$

As previously mentioned, both the IMEX midpoint and DIRK methods retain $O(h^2)$ temporal convergence, as well as the unconditional stability provided from the implicit linear solve. While a method like Heun also has $O(h^2)$ convergence, it is unfortunately restricted due to its entirely explicit stability region [1].

3.2.9 Exponential Runge-Kutta integrators

Recall that another method of time integration is *exponential time differencing* (ETD). For a function $y : \mathbb{R} \to \mathbb{R}$, and a partioned differential equation as $y' = Ly + N(y)$, the time integrator is:

$$y_{n+1} = y_n e^{Lh} + e^{Lh} \int_{t_n}^{t_{n+1}} e^{-Lh} N(y) d\tau \tag{3.2.54}$$

$$\implies y_{n+1} = y_n e^{Lh} + L^{-1} N(y_n)(e^{Lh} - 1) \tag{3.2.55}$$

This method has $\mathcal{O}(h)$ convergence. We can overcome limitation introducing a second order ETD RK scheme [10]. We begin by calculating intermediary a_n on $t_n \leq t \leq t_{n+1}$, such that:

$$a_n = u_n e^{Lh} + L^{-1} N(y_n)(e^{Lh} - 1) \tag{3.2.56}$$

Then write the final step of u_{n+1} as:

$$u_{n+1} = a_n + (N(a_n) - N(u_n)(e^{Lh} - 1 - Lh)h^{-1}L^{-2} \tag{3.2.57}$$

This is an example of the second ordered ETD scheme. Shown on the next page is the ETD RK scheme.

3.2.10 MATLAB implementation

Shown below is a MATLAB implementation of the ETD RK scheme:

Listing 3.3: ETD Runge-Kutta time integrator

```
function [ys,cpu_time] = expRK(L,Nl,tspan,y0,N)
```

```
ys = zeros(length(y0),N+1);
ys(:,1) = y0;
y = y0;
dt = diff(tspan)/N;

if(issparse(L))
    I = speye(length(y0));
else
    I = eye(length(y0));
end



tic

p0 = expm(dt*L);
p1 = L\(expm(dt*L)-I);
p2 = (L\(L\((expm(dt*L)-I-dt*L))))/dt;

for i = 1:N
    a_n = p0*y + p1*Nl(y);
    y = a_n + p2*(Nl(a_n)-Nl(y));
    ys(:,i+1) = y;
end
cpu_time = toc;
```

3.2.11 Temporal error for Viscous Burger's Equation

We will begin our evaluation of methods by only measuring temporal error for Burger's equation. Our error expression for Burgers's is:

$$\text{error} = ||\mathbf{u}_N - \mathbf{u}_{N_{\text{ref}}}||_2 \tag{3.2.58}$$

We choose choose $N_x = 100$ spatial steps, with $t \in [0, .1]$, $x \in [-1, 1]$. We choose $u(x, 0) = -\sin(\pi x)$, the initial position of the wave, and diffusion

constant $\epsilon = .02$, in order to not make the problem too rigid. Our vector of $N_t = (10^1, \ldots, 10^4)$, and $N_{\text{tref}} = 10^5$, one degree of magnitude above the final time step. The convergence diagram and subsequent precision diagram are shown below:

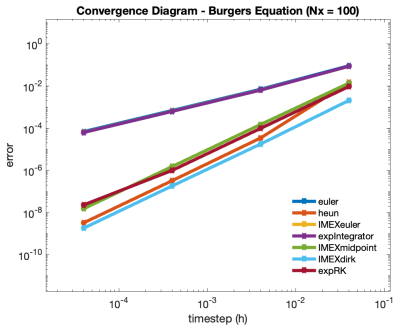

Figure 3.11: Temporal convergence diagram for Viscous Burger's equation

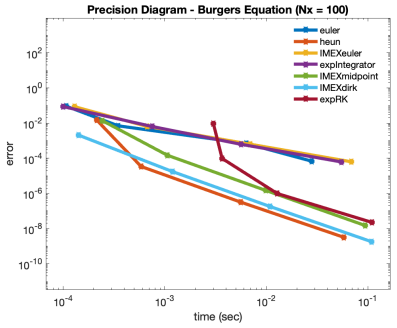

Figure 3.12: Temporal precision diagram for Viscous Burger's equation

We can see that the temporal convergence for forward Euler, IMEX Euler, and exponential integrator is $O(h)$, referenced by their lines. They are overlaid so it may be difficult to see. IMEX midpoint, IMEX DIRK, and ETD RK all have $O(h^2)$ convergence, due to $s = 2$ stages.

The precision diagram is rather interesting for Burger's equation. IMEX midpoint reaches $e_{t_f} < .01$ in nearly $T = .0001$ seconds, an extremely effecient method for solving Burger's using these parameters. If we choose $e_{t_f} \approx .01$ as our threshold, we'd actually choose Heun, because it reaches the threshold in a nearly incomprehensibly small amount of time. The exponential integrators fail here, due to the N_t amount of matrix exponentials it must compute. Either Heun or IMEX midpoint is a good choice of method here. As for the Heat equation, we can do

away with temporal refinement, favoring a simultaneous refinement instead. On the following page is the algorithm used for these diagrams.

Now, consider a case in which $N_x = 350$, stiffening the Burgers' equation. We load the same parameters as before. Note the convergence and precision diagram below:

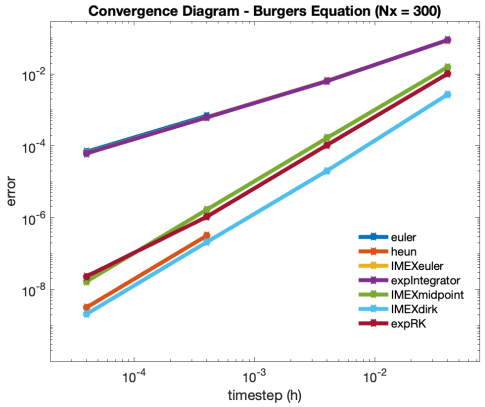

Figure 3.13: Temporal convergence diagram for viscous Burgers' equation

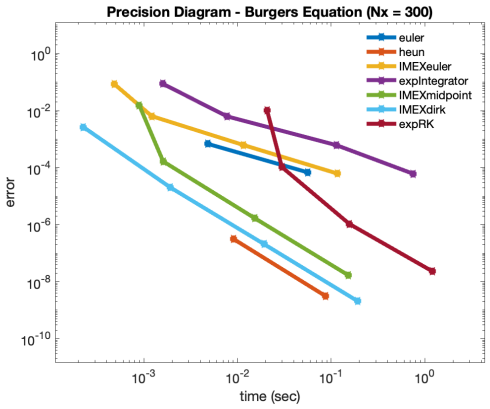

Figure 3.14: Temporal precision diagram for viscous Burgers' equation

Note now due to stiffness, the explicit methods require much large time steps to achieve a certain accuracy. IMEX midpoint is able to achieve a reasonable accuracy much quicker than a strictly explicit method is.

3.2.12 Restriction for variant spatial meshes

Eventually, when we refine in time and space, we will be dealing with numerical solutions that do not have the same size vectors at $\mathbf{u}_{N_{\text{ref}}}$, since we numerically

compute $\mathbf{u}_{N\text{ref}}$, and we will use different discretizations for the reference and current solution. Suppose we are measuring error between two numerically computed solutions for Burger's equation, $u_1(x,t)$ and $u_2(x,t)$ on meshes with dimensions M_x, M_t and N_x, N_t. The discretizations can be written as:

$$u_1(x,t) = \begin{bmatrix} a_{21} & a_{22} & \dots & a_{2M_{t+1}} \\ a_{31} & a_{32} & \dots & a_{3M_{t+1}} \\ \vdots & \vdots & \ddots & \vdots \\ a_{M_{x+1}1} & a_{M_{x+1}2} & \dots & a_{M_{x+1}M_{t+1}} \end{bmatrix} \tag{3.2.59}$$

(3.2.60)

$$u_2(x,t) = \begin{bmatrix} a_{21} & a_{22} & \dots & a_{2N_{t+1}} \\ a_{31} & a_{32} & \dots & a_{3N_{t+1}} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N_{x+1}1} & a_{M_{x+1}2} & \dots & a_{N_{x+1}N_{t+1}} \end{bmatrix} \tag{3.2.61}$$

We begin in the $N_x = 2$ row and end at N_{x+1} because of homogenous boundary conditions, as previously mentioned. Note that $M_x \neq N_x$. To measure error we take:

$$\text{error} = ||U_{M_t} - U_{N_t}||_\infty \tag{3.2.62}$$

$$\implies \text{error} = || \begin{bmatrix} a_{2M_{t+1}} \\ a_{3M_{t+1}} \\ \vdots \\ a_{M_{x+1}M_{t+1}} \end{bmatrix} - \begin{bmatrix} a_{2N_{t+1}} \\ a_{3N_{t+1}} \\ \vdots \\ a_{N_{x+1}M_{t+1}} \end{bmatrix} ||_\infty \tag{3.2.63}$$

Knowing that $M_x \neq N_x$, this implies that the dimensions of the two spatial meshes are different, and we can not take the norm of two different sized vectors. To solve this, we restrict ourselves to spatial grids with 2^{n-1} spatial points. By removing

the appropriate points we can restrict the fine solution to the coarse grid.

$$\gamma_3 = \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \end{pmatrix} \gamma_2 = \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} \gamma_1 = \begin{pmatrix} u_4 \end{pmatrix} \tag{3.2.64}$$

This algorithm allows us to restrict the solution defined on $N_x > M_x$, so we can now find error across different spatial domains with restriction condition $N_x, M_x \in \mathbb{R}^{2^n-1}$. Shown on the next page is the algorithm used for restricting a vector of size $\mathbb{R}^n$ to domain $\mathbb{R}^m$, and returning error using the 2-norm.

3.2.13 MATLAB implementation

Shown below is a MATLAB implementation of spatial restriction, from a fine to coarse grid:

Listing 3.4: Spatial restriction algorithm

```
function [s_error] = spatialError(approx,reference)

if ~ isValidGridSize(length(reference))
    error('Invalid reference size');
end

if ~isValidGridSize(length(approx))
    error('Invalid approx size');
end

nr = log(length(reference)+1)/log(2);
na = log(length(approx)+1)/log(2);
gamma = nr - na;
```

```
restrictedRef = reference(2^gamma:2^gamma:end);

s_error = norm(restrictedRef-approx,'inf');
end

function flag = isValidGridSize(n)

exponent = round(log(n+1)/log(2));
flag = 2^exponent == n+1;

end
```

3.2.14 Refining in space and time for viscous Burgers'

Recall that we can refine in both space and time when solving PDEs. The stiffness in Burgers' is predominantly due to the linear term ϵD_{xx} whose eigenvalues grow proportional to $\frac{1}{\Delta x^2}$

As we did for the Heat equation, the following functions give a bound for dt as a function of N_x, such that:

$$\text{Euler} : dt(N_x) = \frac{2}{\epsilon(N_x+1)^2} \ , \ \text{Heun} : dt(N_x) = \frac{2}{\epsilon(N_x+1)^2} \tag{3.2.65}$$

As there is no restriction on stability for the IMEX and ETD integrators, we choose the stepsize to balance spatial and temporal error, giving the following

stepsize restrictors:

$$\text{IMEX Euler} : dt(N_x) = \frac{h}{100(\frac{50}{N_x})^{-2}} \tag{3.2.66}$$

$$\text{ETD non-RK} : dt(N_x) = \frac{h}{100(\frac{50}{N_x})^{-2}} \tag{3.2.67}$$

$$\text{IMEX midpoint} : dt(N_x) = \frac{h}{60(\frac{25}{N_x})^{-2}} \tag{3.2.68}$$

$$\text{IMEX dirk} : dt(N_x) = \frac{h}{40(\frac{25}{N_x})^{-2}} \tag{3.2.69}$$

$$\text{ETD RK} : dt(N_x) = \frac{h}{40(\frac{25}{N_x})^{-2}} \tag{3.2.70}$$

We can now refine in both space and time for the Viscous Burger's equation, and plot the spatial step size dx and computational time vs. error $(\hat{e}_{t_f})$, where the temporal (e_{t_f}) error is found implicitly. This guarantees stability for all methods and allows us to select the most efficient choice. On the following page we present the algorithm,

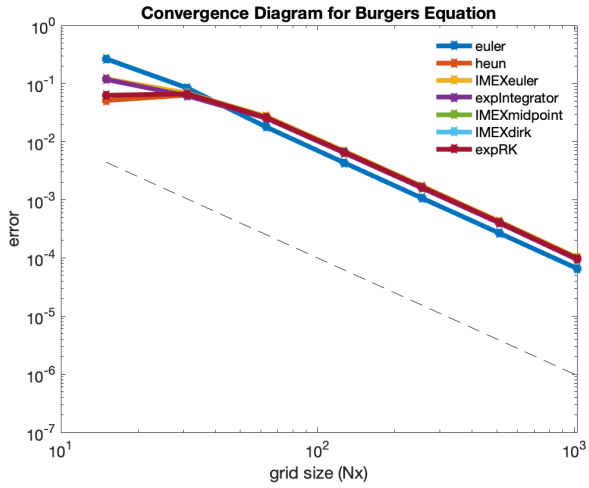

Figure 3.15: Viscous Burger's spatial convergence diagram

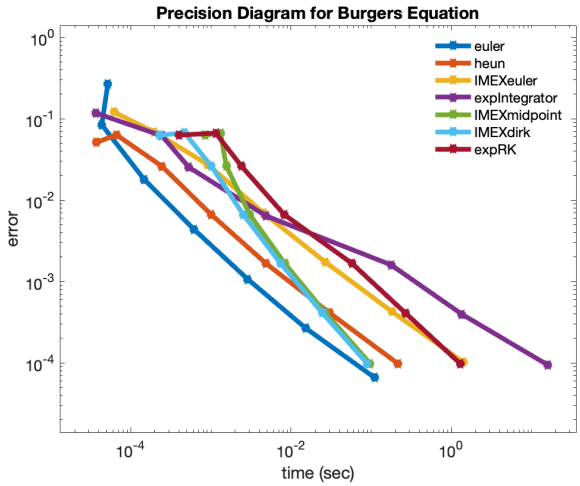

Figure 3.16: Viscous Burger's spatial precision diagram

For the convergence diagram, we now plot the propagation of error vs the amount of spatial steps N_x taken based on our tuning method. Note since we use second order centered difference in space for D_{xx}, each method has the same spatial error convergence, that being $O(dx^2)$. This diagram can be misleading, as we fine tune time and space simultaneously.

Examining the precision diagram, we can conclude a few things. Heun reaches reasonabkle error threshold $e_{t_f} \leq 10^{-1}$ in computational time T $= 10^{-4}$ seconds. If we seek to reach a lower, methods like forward Euler and IMEX midpoint reach $e_{t_f} \leq 10^{-3}$ in quicker computational time than Euler, overtaking it around $T = 10^{-1}$. Note that choice of method entirely depends on the error threshold e_{t_f} we choose. Additionally, since we are refining simultaenously, we need not concern ourselves with stability.

3.2.15 Conclusions of semi-implicit time integration

We have now discussed a total of 9 methods. Shown below is a table comparing each method. While some are outright more valuable than others, they each have their own benefits and drawbacks. Note the nine method table below:

As previously mentioned, each method has advantages on certain problems. Suppose we consider a non-stiff problem, either Dalquist or a MOL equation gov-

Method	Target equation	Stages	Convergence	Simultaneous Expression
Euler	y' = f(t,y)	1	$\mathcal{O}(h)$	$\frac{2}{(Nx+1)^2}$
Heun	y' = f(t,y)	2	$\mathcal{O}(h^2)$	$\frac{2^2(dx)^4}{8(dx)^2+6}$
Backward Euler	y' = f(t,y)	1	$\mathcal{O}(h)$	$\frac{10^4}{\left(\frac{31}{N_x}\right)^2}$
Implicit Midpoint	y' f(t,y)	1	$\mathcal{O}(h^2)$	$\frac{100}{\left(\frac{15}{N_x}\right)}$
IMEX	y' = Ly + N(y)	1	$\mathcal{O}(h)$	$\frac{t_f-t_i}{100(\frac{63}{N_x})^{-2}}$
Exponential Int	y' = Ly + N(y)	1	$\mathcal{O}(h)$	$\frac{t_f-t_i}{100(\frac{63}{N_x})^{-2}}$
IMEX Midpoint	y' = Ly + N(y)	2	$\mathcal{O}(h^2)$	$\frac{t_f-t_i}{60(\frac{30}{N_x})^{-1}}$
IMEX DIRK	y' = Ly + N(y)	2	$\mathcal{O}(h^2)$	$\frac{t_f-t_i}{60(\frac{30}{N_x})^{-1}}$
ETD RK	y' = Ly + N(y)	2	$\mathcal{O}(h^2)$	$\frac{t_f-t_i}{60(\frac{30}{N_x})^{-1}}$

Table 3.1: Comparison of nine time integrators

erned by $\lambda \in \mathbb{R}^n \rightarrow ||\lambda|| < 0$. Since the problem is unstiff, we are not restricted by Euler's stability region. We therefore use an explicit method, which will be more efficient since there are no matrix solves.

Another case, if we seek to reach a certain error threshold e_{t_f}, then we seek to choose a method with rapid error convergence, such as $O(h^2)$. Depending on choice of threshold, the methods with $O(h^2)$ convergence will suit precision better, as they take less computational time to reach said threshold.

For stiff problems, such that $||\lambda|| >>> 1$, explicit integrators require small stepsizes to ensure stability, therefore we must use either an IMEX or fully implicit method, so that $\lambda_{\max}$ lies within the stability region.

Chapter 4

Stokes equations and time integration of velocity fields

4.1 Introduction and notions of fluid dynamics

For the remainder of this paper we will discuss a certain class of equations involved in the study of **fluid dynamics** - or the physical, mechanical, and dynamical properties of fluids. It will be important to note that going forward, we will not thoroughly delve into the physics of fluids. There will be brief words about the physics of various classes of fluids, but we will ultimately maintain our focus on time integration.

4.2 Navier-Stokes equation

The **Navier-Stokes equations** are a class of partial differential equations that govern the velocity and pressure of a fluid [3][5]. We will not analyze Navier-Stokes, nor its solution set, but rather use it as a stepping stone for overall understanding.. The **Incompressible Navier Stokes equation** (INS), in two dimensions is defined as:

$$\rho \frac{D\mathbf{u}}{Dt} = -\nabla p + \mu \Delta \mathbf{u} + \mathbf{f} \tag{4.2.1a}$$

$$\nabla \cdot \mathbf{u} = 0 \tag{4.2.1b}$$

Each quantity defined in the INS in 2-D is described below:

$$\bullet\ \rho = \text{Fluid density} \tag{4.2.2}$$

$$\bullet\ \frac{D}{Dt} = \text{Material derivative} \tag{4.2.3}$$

$$\bullet\ \mathbf{u} = \begin{bmatrix} u_x(x,y) \\ u_y(x,y) \end{bmatrix} = \text{Velocity field} \tag{4.2.4}$$

$$\bullet\ \mathbf{f} = \begin{bmatrix} f_x(x,y) \\ f_y(x,y) \end{bmatrix} = \text{Force term} \tag{4.2.5}$$

$$\bullet\ \mu = \text{viscocity} \tag{4.2.6}$$

$$\bullet\ \Delta = \text{Laplacian in 2-D} \tag{4.2.7}$$

We will solely consider a fluid in two dimensions for the remainder of the paper. The incompressibility of the Navier Stokes equations is $\nabla \mathbf{u} = 0$. This incompressibility condition is divergence free [3][12].

4.3 Non-dimentionalization and Reynolds number

We introduce the concept of the Reynolds number to illustrate the type of flow we will study. The Reynolds number is used to characterize different types of flows. The expression for Reynolds number is:

$$\text{Re} = \frac{\rho L U}{\mu} \tag{4.3.1}$$

Where ρ is the density of the fluid, L is the characteristic length scale, U is the mean velocity of the fluid, and μ is the viscosity of the fluid. A highly viscous fluid, (the main focus of this chapter) follow with $Re \ll 1$. This can be attributed to non-

dense fluids, small length scales, slow velocity, or high viscosity. To understand the role in of the Reynolds number in the INS equation, we choose dimensionless parameters to nondimentionalize the equation:

$$\mathbf{u}' = \frac{\mathbf{u}}{U} \ , \quad p' = p\frac{1}{\rho U^2} \ , \quad \mathbf{f'} = \mathbf{f}\frac{L}{U^2} \ , \quad \frac{\partial}{\partial t'} = \frac{L}{U}\frac{\partial}{\partial t} \ , \quad \nabla' = L\nabla \tag{4.3.2}$$

Applying each expression, and assuming each prime notation is removed, gives us the nondimentionalized INS equation:

$$\mathrm{Re}\,\frac{D\mathbf{u}}{Dt}\Big) = -\nabla p + \frac{\mu}{\rho LU}\Delta\mathbf{u} + \mathbf{f} \tag{4.3.3}$$

For highly viscous flows $\mathrm{Re} \to 0 \implies \frac{1}{\mathrm{Re}} \to \infty$. In the subsequent section we will demonstrate the resulting equation that follows from this assumption.

4.4 Stokes equations in 2-dimensions

4.4.1 Stokes flow

Recall that the Incompressible Navier Stokes equations can be written as:

$$\mathrm{Re}\,\frac{D\mathbf{u}}{Dt}\Big) = -\nabla p + \frac{\mu}{\rho LU}\Delta\mathbf{u} + \mathbf{f} \tag{4.4.1}$$

To reduce the equation to strictly viscous flows, we rewrite it as:

$$\mu\Delta\mathbf{u} - \nabla p + \mathbf{f} = 0 \ , \quad \nabla \cdot \mathbf{u} = 0 \tag{4.4.2}$$

Equation (4.4.2) is called **Stokes flow** [12][5] This system of partial differential equations governs the velocity and pressure of highly viscous flows. We seek to implement time integration to find the streamlines of objects in the fluid with time dependent forcing, through the $(x(t), y(t))$ plane.

4.4.2 Derivation of Stokeslets using 2-dimensional cutoffs

A Stokeslets refers to a fundamental solution to the viscous Stokes equations [5]. A Stokeslet is typically a small 'ball' placed in a viscous fluid with a radius ϵ - in either $\mathbb{R}^2$ or $\mathbb{R}^3$. We will begin with a 2-dimensional case, given by the equation:

$$\phi_\epsilon(\mathbf{x}) = \frac{3\epsilon^2}{2\pi(|\mathbf{x}|^2+\epsilon^2)^{3/2}} \tag{4.4.3}$$

The Stokeslet is then used to find the fundamental solution $\mathbf{u}(\mathbf{x})$ [5]. The equation $\phi_\epsilon(\mathbf{x})$ is commonly referred to as a **cutoff** or **blob**. Each cutoff has a distribution of forces over $\phi_\epsilon(\mathbf{x})$. The cutoff has property $\int_{\mathbb{R}} \phi_\epsilon(\mathbf{x}) = 1$ [5]. Note also that if $\epsilon \to 0$, then $\phi_\epsilon(\mathbf{x})$ approaches a Dirac-Delta, with property $\phi_\epsilon(\mathbf{x} = \mathbf{0}) = \infty$ [5].

We define the following problem to find a regularized Stokeslet velocity and pressure solution (although we won't necessarily need the pressure function). Let $G_\epsilon(\mathbf{x})$ and $B_\epsilon(\mathbf{x})$ represent the solutions to two coupled Poisson problems, such that:

$$\Delta G_\epsilon(\mathbf{x}) = \phi_\epsilon(\mathbf{x}) \tag{4.4.4}$$

$$\Delta B_\epsilon(\mathbf{x}) = G_\epsilon(\mathbf{x}) \tag{4.4.5}$$

$G_\epsilon(\mathbf{x})$ and $B_\epsilon(\mathbf{x})$ are both smooth approximations of Green's functions in infinite space [5]. The regularized Stokeslet pressure and velocity, for cutoffs centered at $x_k{}_{k=1}^N$, the fundamental solutions are:

$$p(\mathbf{x}) = \sum_{k=1}^{N} [\mathbf{f}_k(\mathbf{x} - \mathbf{x}_k)] \left[\frac{G'_\epsilon(r_k)}{r_k} \right] \tag{4.4.6}$$

$$\begin{aligned} \mathbf{u}(\mathbf{x}) = \frac{1}{\mu} \sum_{k=1}^{N} \mathbf{f}_k \left[\frac{B'_\epsilon(r_k)}{r_k} - G_\epsilon(r_k) \right] \\ + [\mathbf{f}_k \dot{(}\mathbf{x} - \mathbf{x}_k)](\mathbf{x} - \mathbf{x}_k) \left[\frac{r_k(B''_\epsilon(r_k) - B'_\epsilon(r_k))}{r_k^3} \right] \end{aligned} \tag{4.4.7}$$

Where $r_k = ||\mathbf{x} - \mathbf{x}_||_2$. Since $\phi_\epsilon(\mathbf{x})$ is radially symmetric across the y-axis, we can find G_ϵ) using:

$$\Delta G_\epsilon(r) = \frac{1}{r}[rG'_\epsilon(r)]' = \phi_\epsilon(r) \tag{4.4.8}$$

$$\implies G'_\epsilon(r) = \frac{1}{r}\int_0^r s\phi_\epsilon(s)\ ds \tag{4.4.9}$$

Using the same radially symmetric property, and the fact that $\Delta B_\epsilon(\mathbf{x}) = G_\epsilon(\mathbf{x})$, we have:

$$\frac{1}{r}[rB'_\epsilon(r)]' = G_\epsilon(r) \tag{4.4.10}$$

$$\implies B'_\epsilon(r) = \frac{1}{r}\int_0^r sG_\epsilon(s)\ ds \tag{4.4.11}$$

If we choose the cutoff (4.4.3), then integrating the following expression gives us the two Green's functions in $\mathbb{R}^n$ dimensional space:

$$G_\epsilon(r) = \frac{1}{2\pi}\left[\ln\left(\sqrt{r^2+\epsilon^2}+\epsilon\right) - \frac{\epsilon}{\sqrt{r^2+\epsilon^2}}\right] \tag{4.4.12}$$

$$B'_\epsilon(r) = \frac{1}{8\pi}\left[2r\ln\left(\sqrt{r^2+\epsilon^2}+\epsilon\right) - r - \frac{2r\epsilon}{\sqrt{r^2+\epsilon^2}+\epsilon}\right] \tag{4.4.13}$$

Note that the r is the instance in which $\mathbf{x}_k = \mathbf{0}$. Using $G_\epsilon(r), B_\epsilon(r)$, and their corresponding derivatives w.r.t to r, we get the final solutions for the regularized Stokeslets, by applying them into (4.4.6-7):

$$p(\mathbf{x}) = \sum_{k=1}^{N}\frac{1}{2\pi}[\mathbf{f}_k\cdot(\mathbf{x}-\mathbf{x}_k)]\left[\frac{r_k^2+2\epsilon^2+\epsilon\sqrt{r_k^2+\epsilon^2}}{(\sqrt{r_k^2+\epsilon^2}+\epsilon)(r_k^2+\epsilon^2)^{3/2}}\right] \tag{4.4.14a}$$

$$\mathbf{u}(\mathbf{x}) = \sum_{k=1}^{N} -\frac{\mathbf{f}_k}{4\pi\mu}\left[\ln\left(\sqrt{r_k^2+\epsilon^2}+\epsilon\right) - \frac{\epsilon(\sqrt{r_k^2+\epsilon^2}+2\epsilon)}{\left(\sqrt{r_k^2+\epsilon^2}+\epsilon\right)\sqrt{r_k^2+\epsilon^2}}\right] + \frac{1}{4\pi\mu}[\mathbf{f}_k\cdot(\mathbf{x}-\mathbf{x}_k)](\mathbf{x}-\mathbf{x}_k)\left[\frac{\sqrt{r_k^2+\epsilon^2}+2\epsilon}{(\sqrt{r_k^2+\epsilon^2}+\epsilon)^2\sqrt{r_k^2+\epsilon^2}}\right] \tag{4.4.14b}$$

Where $r_k = \mathbf{x} - \mathbf{x}_k$. The two above equations are the fundamental solutions to the Stokes equations [5]. These are regularized as $p(\mathbf{x}), \mathbf{u}(\mathbf{x})$ never approach ∞ as $r_k \to 0$. Pressure returns a scalar value at a point $\mathbf{x} \in \mathbb{R}^2$ and velocity returns a bivariate vector in the (x, y). When $\epsilon \to 0$, velocity and pressure $\to \infty$ at $\mathbf{x} = \mathbf{x}_k$, and the Stokeslets become:

$$p(\mathbf{x}) = \sum_{k=1}^{N} \frac{[\mathbf{f}_k(\mathbf{x} - \mathbf{x}_k]}{2\pi r_k^2} \tag{4.4.15a}$$

$$\mathbf{u}(\mathbf{x}) = \sum_{k=1}^{N} \frac{\mathbf{f}_k}{4\pi\mu} \ln(r_k) + [\mathbf{f}_k \cdot (\mathbf{x} - \mathbf{x}_k)] \frac{(\mathbf{x} - \mathbf{x}_k)}{4\pi\mu r_k^2} \tag{4.4.15b}$$

We can easily verify the property of dirac delta forces by noting that both $p(\mathbf{x})$ and $\mathbf{u}(\mathbf{x}) \to \infty$ as $r_k \to 0$ [5].

4.4.3 Time integration of Stokslets velocity fields

Consider a case where forces are crudely assigned, and the fluid is studied across the entire domain. Consider a spatial domain in $\mathbb{R}^2$ on the square $x \in [-1, 1], y \in [-1, 1]$. We set $\mathbf{F}$ and their respective locations $\mathbf{X}$ as:

$$\mathbf{F} = \begin{bmatrix} 1 & -2 & .5 \\ 1 & -2 & .3 \end{bmatrix}, \quad \mathbf{X} = \begin{bmatrix} .1 & .2 & .1 \\ .1 & .4 & .7 \end{bmatrix} \tag{4.4.16}$$

The kth column of $\mathbf{F}$ has the force of $\mathbf{f}_j \in \mathbb{R}^2$ at location $X_j \in \mathbb{R}^2$.

For a choice of $\epsilon = .3$, we can see the difference between irregularized and regularized Stokeslets. The regularized Stokeslets implement 'cutoffs', losing the Dirac-Delta property that $\phi(\mathbf{x}_0) = \infty$, but instead choosing a smooth radially symmetric function that still possess the property $\int_A \phi(\mathbf{x})\, d\mathbf{x} = 1$. Irregularized stokeslets are stiff in the fact that $p(\mathbf{x} = \mathbf{x}_k) = \infty$.

Suppose we condense $\mathbf{X}$ into a vector where each entry $\mathbf{x}_k \in \mathbb{R}^2$ and $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)$. Since each component of $\mathbf{x}_k = (x_k, y_l)$ has a velocity, we yield a system of linear autonomous ordinary differential equations:

$$\mathrm{d}\frac{}{dt}\begin{bmatrix}\mathbf{x}_1(t)\\ \mathbf{x}_2(t)\\ \mathbf{x}_3(t)\end{bmatrix} = \begin{bmatrix}\mathbf{f}_1(\mathbf{X})\\ \mathbf{f}_2(\mathbf{X})\\ \mathbf{f}31(\mathbf{X})\end{bmatrix} \implies \frac{d\mathbf{X}}{dt} = \mathbf{F}(\mathbf{X})$$

Note that for n amount of cutoffs, where each $\mathbf{x}_k \in mathbbR^2$, it follows that $\mathbf{X} \in \mathbb{R}^{2n}$, and the ODE system has $2n$ equations. For each point on the spatial mesh, we can use that as a starting location, and integrate through time to find the streamline in the $(x(t), y(t))$ parametrized plane. The code for time integrating streamlines is shown below:

4.4.4 MATLAB implementation

Listing 4.1: Streamline integration of cutoffs centers $\mathbf{x}_k$

```
ny = 5;
nx = 5;
[xks,fks,xs,ys] = stokes_parameters(nx,ny);

tspan = [0,1];
f = @(t,X)  velocity(X,xks,fks)';
Nt = 1000;

for i = 1:nx
    for j = 1:ny
        [X,cpu] = euler(f,tspan,[xs(i),ys(j)],Nt);
        plot(X(1,:),X(2,:),color = 'blue');
        hold on;
    end
end
```

Note the plot of the streamlines for the aforementioned simulation:

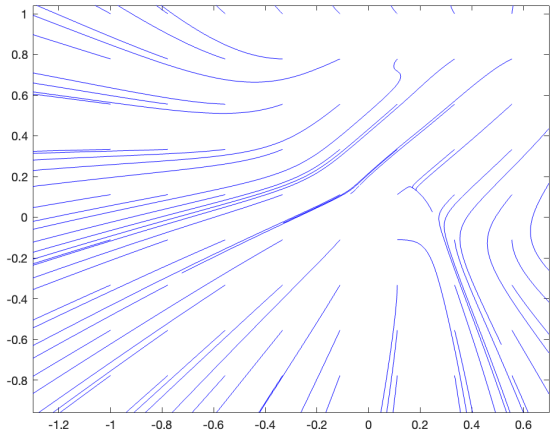

Figure 4.1: Streamlines for 2-dimensional Stokeslets

Applying forces in this way is rather nonsensical, so we turn our focus to a rigid approach in which forces are continuously distributed over $\mathbb{R}^2$.

4.5 Time integrating specific choices of cutoffs in $\mathbb{R}^2$

We can assign each force f_k using a continuous distribution function $\phi_\epsilon(\mathbf{x}_0)$, such that each bivariate force is represented as:

$$\mathbf{F}(\mathbf{x}) = \mathbf{f}_0 \ \phi(\mathbf{x} - \mathbf{x}_0) \tag{4.5.1}$$

One can see that each $\mathbf{F}(\mathbf{x}) \in \mathbb{R}^2$. Such force F is centered around the point $\mathbf{x}_0$. The function $\phi_\epsilon(\mathbf{x})$ is referred to as a 'blob' function, as it's maximum is restrained by the parameter ϵ such that:

$$\phi_\epsilon(\mathbf{x}) = \frac{3\epsilon^3}{2\pi(|\mathbf{x}|^2+\epsilon^2)^{5/2}} \tag{4.5.2}$$

For a each blob centered at point $\mathbf{x}_0$, the forcing function becomes:

$$\phi_\epsilon(\mathbf{x}) = \frac{3\epsilon^3}{2\pi(|\mathbf{x}-\mathbf{x}_0|^2+\epsilon^2)^{5/2}} \tag{4.5.3}$$

Consider a matrix $X \in \mathbb{R}^{2xn}$, for n blob functions on a spatial mesh we can plot them in $\mathbb{R}^3$ with centers $X = [\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_n]$. Note that as $\epsilon \to 0$ each blob becomes a Dirac-Delta function with property $\phi(\mathbf{x}_k) \to \infty$. Shown below are a choice of blob functions with overlapping centers $\mathbf{x}_k$:

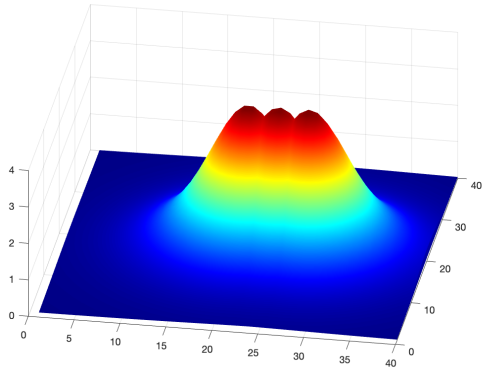

Figure 4.2: Choice of n = 3 cutoffs in $\mathbb{R}^3$

Now, consider a certain set of cutoffs $\phi_\epsilon(\mathbf{x})$, where $\mathbf{x} \in \mathbb{R}^2$. We will assign the centers of each blob $\mathbf{x}_k$ such that we have a sequence of functions $\{\phi_{\epsilon k}(\mathbf{x}_k)\}_{k=1}^n$. For simulation purposes, we will choose $n = 6$. Consider centers $\mathbf{x}_k$ such that:

$$\mathbf{x}_k = \begin{bmatrix} 0 & 0.17 & 0.34 & 0.51 & 0.68 & 0.85 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Where each column represents the kth center. Now consider a time dependent force $\mathbf{f}(t)$, such that $f \in \mathbb{R}^2$:

$$\mathbf{f}(t) = \begin{bmatrix} \frac{1}{2}(\sin(t) + 1) \\ 0 \end{bmatrix}$$

We can apply methods of time integration to plot the position of the cutoffs for a certain temporal interval. Shown below is the algorithm implemented to time integrate the position of the cutoffs and an animated plot of the resulting position due to the force applied for * time interval and time steps *:

4.5.1 MATLAB implementation

Listing 4.2: Script to plot movement of blob centers using streamline integration

```
% Plots movement of blob centers for certain force

nx = 40;
ny = 40;
n = 6;
epsilon = .5;

[xks,fks,xs,ys] = stokes_parameters(nx,ny);
X = init_blob(n);

for i = 1:n
    z = blob_meshes(X(:,i),epsilon,xs,ys);
    surf(xs,ys,z); shading interp; colormap jet;
    hold on;
end

Xv = reshape(X,[2*n,1]);
tspan = [0,1];
Nt = 1000;
[Xvs,cpu] = euler(@f,tspan,Xv,Nt);

figure()
for i = 1:Nt
    Xvi = Xvs(:,i);
    Xvi = reshape(Xvi,[2,length(Xvi)/2]);
    plot(Xvi(1,:),Xvi(2,:),'ko-');
    xlim([-1.5,1.5]); ylim([-1.5,1.5]);
```

```
    drawnow()
end

function Xvp = f(t,Xv)
% Returns velocity for each blob center

fks = @(t) [1;0];
n = length(Xv)/2;
Xvp = zeros(2*n,1);

for i = 1:n
    Xvp(2*i-1:2*i) = velocity_regularized([Xv(2*i-1),Xv
       (2*i)],[0;0],fks(t));
end

end
```

4.5.2 Regularized cutoffs connected to springs

We now consider a simulation that will govern our study of time integration of Stokeslets for the remainder of the paper. Consider again n cutoffs $\phi_\epsilon(\mathbf{x}-\mathbf{x}_k) \in \mathbb{R}^2$. Instead of applying a time dependent force to a single cutoff, we will now attach a spring between each each cutoff. We seek to study the efficiency of time integration for this specific problem.

A simplified diagram below illustrates the mechanics of the problem:

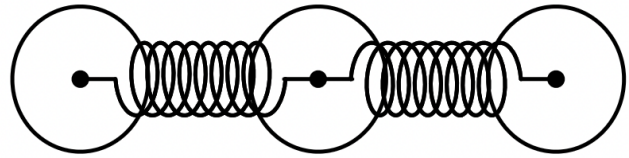

Figure 4.3: 3 spring connected cutoffs in $\mathbb{R}^2$

Suppose we have n cutoffs such that $\{\phi_{\epsilon_k}\}_{k=1}^{n}$, with the aforementioned bivariate sinusoidal force $f(t)$ that is applied to index $k = 1$. Cutoff centered at $\mathbf{x}_k$ will have a force $\mathbf{F}_l$ and $\mathbf{F}_r$, such that the force from the cutoff to the left of $\mathbf{x}_k$ is:

$$\mathbf{F}_l = f_l \mathbf{u} \quad \text{where,} \tag{4.5.4a}$$

$$f_l = -k(||\mathbf{x}_k - \mathbf{x}_{k-1}|| - \delta_0) \quad \text{and} \quad \mathbf{u} = \frac{\mathbf{x}_k - \mathbf{x}_{k-1}}{||\mathbf{x}_k - \mathbf{x}_{k-1}||} \tag{4.5.4b}$$

Similarly, the force applied to the 'right' of the kth cutoff $\mathbf{F}_r$, is:

$$\mathbf{F}_r = f_r \mathbf{u} \quad \text{where,} \tag{4.5.5a}$$

$$f_r = -k(||\mathbf{x}_k - \mathbf{x}_{k+1}|| - \delta_0) \quad \text{and} \quad \mathbf{u} = \frac{\mathbf{x}_k - \mathbf{x}_{k+1}}{||\mathbf{x}_k - \mathbf{x}_{k+1}||} \tag{4.5.5b}$$

The parameter δ_0 is the equilibrium position of the spring. The total force F_k applied at cutoff center $\mathbf{x}_k$ is:

$$\mathbf{F}_k = \mathbf{F}_l + \mathbf{F}_r \tag{4.5.6}$$

We can use these expressions when producing our velocity field. It will become apparent that the governing term of stability for our velocity system is k.

4.5.3 Applying time integration to a Stokeslet velocity field

For a certain set of centers $\mathbf{x}_k \in \mathbb{R}^2$, we get a corresponding velocity ordinary differential equation system, where each $\mathbf{x}_k = \mathbf{f}(\mathbf{x})$. The centers in matrix form can be written as:

$$\mathbf{x} = \begin{bmatrix} x_1 & x_2 & \dots & x_n \\ y_1 & y_2 & \dots & y_n \end{bmatrix} \tag{4.5.7}$$

Then the corresponding velocity field can be written as:

$$\frac{d\mathbf{X}}{dt} = \begin{bmatrix} \dot{\mathbf{x}}_1 \\ \dot{\mathbf{x}}_2 \\ \vdots \\ \dot{\mathbf{x}}_n \end{bmatrix} = \mathbf{f}(\mathbf{X}) \tag{4.5.8}$$

Each $\mathbf{x}_k \in \mathbb{R}^2$ as usual. Since the system of ODE's is nonlinear (Stokeslet velocity is nonlinear), we must begin by only using explicit integrators, such as Heun or Euler. The algorithm for returning the velocity field at n centers, centered at $\mathbf{x}_k \in \mathbb{R}^2$ is:

4.5.4 MATLAB implementation

Listing 4.3: Forcing function for velocity of cutoff centers using spring

```
function Xvp = f(t,Xv)
% Returns velocity for each blob center


fks_right = @(t) [0;10*sin(2*t)];
fks_left = @(t) [0;0];
n = length(Xv)/2;
Xvp = zeros(2*n,1);


% spring forces
```

```
% first point, external forcing and last point
for i = 1:n
    Xvp(2*i-1:2*i) = Xvp(2*i-1:2*i) + ...
    velocity_regularized(Bp(Xv,i)',Bp(Xv,n),fks_right(t)
       +fSpring(Xv,n,n-1))+...
         velocity_regularized(Bp(Xv,i)',Bp(Xv,1),fks_left
            (t)+ fSpring(Xv,1,2));

end

% middle points
for i = 2:n-1
    % contribution of forces from ith blob
    for j = 1:n
        % at j locations
        Xvp(2*j-1:2*j) = Xvp(2*j-1:2*j) + ...
        velocity_regularized(Bp(Xv,j)',Bp(Xv,i), ...
            fSpring(Xv,i,i-1)+fSpring(Xv,i,i+1));
    end
end

end
```

The code below *fSpring*, returns the force $\mathbf{f}_k \in \mathbb{R}^2$ acting on cutoff $\mathbf{x}_i$ from cutoff $\mathbf{x}_j$.

4.5.5 MATLAB implementation

Listing 4.4: Algorithm to return forces acting on cut off center i

```
k = 1000; % spring constant
len_rest = .1;
len_curr = norm(Bp(Xv,j)- Bp(Xv,i),2);
f_mag = k*(len_curr-len_rest);
unit_vector = (Bp(Xv,j) - Bp(Xv,i))/len_curr;
F = f_mag * unit_vector;

end
```

4.5.6 Convergence and precision of explicit integrator for Stokes flow

Since the velocity field $\mathbf{x}' = \mathbf{f}(\mathbf{X})$ produced by Stokes is nonlinear, we can only use explicit integrators up to this point. Suppose we use apply $n = 5$ cutoffs to the spring, with $k = 10$, resting length $\delta_0 = .1$. The centers $\mathbf{x}_k$ will have components $(.01(j-1), 0) \in \mathbb{R}^2$, $j = 1, \ldots, 5$. We apply a force on cutoff $\mathbf{x}_1$ such that $\mathbf{f}_r = [0, 10\sin(2t)]^T$, and $\mathbf{f}_l = [0, 0]^T$.

The expression for calculating error will be the $\mathcal{L} - 2$ error between $\mathbf{x}_{t_f}$ and $\mathbf{x}_{\tilde{t_f}}$, where $\mathbf{x}_{\tilde{t_f}}$ is the reference solution. $\mathbf{x}_{t_f}$ represents the positions of n cutoffs in $\mathbb{R}^2$ at the final time step. The error equation is shown below:

$$e_{t_f} = ||\mathbf{x}_{t_f} - \mathbf{x}_{\tilde{t_f}}||_2 \tag{4.5.9}$$

Suppose we measure error using a reference time step $N_{\text{tref}} = 2^{11}$ and a vector of comparison time steps $N_t = [2^6, \ldots, 2^{10}$, $t \in [0, 1]$. We will compute the reference solution using Heun's method. Note the convergence and precision diagram for forward Euler and Heun below:

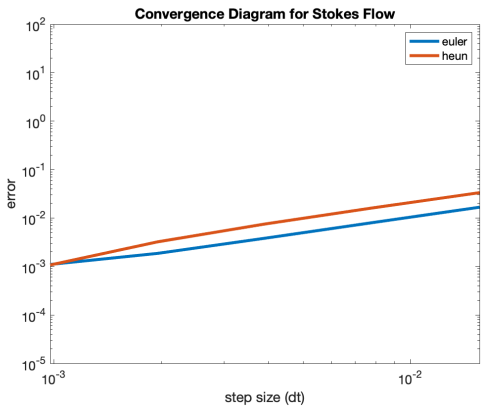

Figure 4.4: Convergence diagram for Stokes flow, $k = 10$

Figure 4.5: Caption

4.6 Semi-linear integrators for spring connected Stokeslets

The central goal of this paper is to be able to implement a semi-linear integrator for the Viscous Stokes equations where we impose high stiffness. We look to use a semi-linear integrator in order to handle this stiffness. Therefore, we can let $k \to \infty$, 'breaking' Euler and Heun while finding an integrator in IMEX that is unconditionally stable.

4.6.1 Implementing IMEX Euler

Recall that for a partioned system, with linear and nonlinear components L and $\mathbf{N}(\mathbf{y})$, the IMEX Euler time integrator is written as:

$$\mathbf{y}' = L\mathbf{y} + N(\mathbf{y}) \tag{4.6.1}$$

$$\mathbf{y}_{n+1} = (I + hL)^{-1}(\mathbf{y} + hN(\mathbf{y})) \tag{4.6.2}$$

Note that we need the differential equation of form $\mathbf{y}' = L\mathbf{y} + N(\mathbf{y})$. Recall that for cutoff centers $\mathbf{X} = \{\mathbf{x}_k\}_{k=1}^n$ where each $\mathbf{x}_k = (x_k, y_k) \in \mathbb{R}^2$, we have a nonlinear

system governed by:

$$\begin{bmatrix} x_1'(t) \\ y_1'(t) \\ \vdots \\ x_n'(t) \\ y_n'(t) \end{bmatrix} = \begin{bmatrix} f_1(\mathbf{X}) \\ f_2(\mathbf{X}) \\ \vdots \\ f_{2n-1}(\mathbf{X}) \\ f_{2n}(\mathbf{X}) \end{bmatrix} = F(\mathbf{X}) \tag{4.6.3}$$

This system is generated from the velocity function that we implement in the cutoff plotter. It retains no closed form expression, and is both autonomous and nonlinear. Its expression $\mathbf{F}(X)$ is not partitioned in the form $L\mathbf{y} + \mathbf{N}(\mathbf{y})$, so we seek to rewrite the expression $F(\mathbf{X})$.

For the sake of notation and synchronicity with code variables, suppose $F(\mathbf{X}) = f(\mathbf{y})$, where $\mathbf{y} \in \mathbb{R}^2$. We can write the system using additive identites, such that:

$$\mathbf{y}' = \mathbf{f}(\mathbf{y}) \tag{4.6.4}$$

$$\mathbf{y}' = J(\mathbf{y})\mathbf{y} + \mathbf{f}(\mathbf{y}) - J(\mathbf{y})\mathbf{y} \tag{4.6.5}$$

The term $J(\mathbf{y})$ is the Jacobian at $\mathbf{y} = \mathbf{X} = \{\mathbf{x}_k\}_{k=1}^n$. The Jacobian $J(\mathbf{y})$ is a $2n$ by $2n$ square matrix. We will discuss the matheamtical interpretation of this in a later section. Our system now has form $\mathbf{y}' = L\mathbf{y} + \mathbf{N}(\mathbf{y})$, where $L = J(\mathbf{y})$, and $\mathbf{N}(\mathbf{y}) = \mathbf{f}(\mathbf{y}) - J(\mathbf{y})\mathbf{y}$. Because of this, we can now apply IMEX methods. Note that the two terms cancel to just reduce to $\mathbf{f}(\mathbf{y})$. Implementing our new system in IMEX Euler yields:

$$\mathbf{y}_{n+1} = (I - hJ(\mathbf{y}_n)^{-1}[\mathbf{y}_n + h(\mathbf{f}(\mathbf{y}_n) - J(\mathbf{y}_n)\mathbf{y}_n)] \tag{4.6.6}$$

Since computing Jacobians can be expensive, we must be careful about implementation. We will now discuss how we can numerically compute the Jacobian at $\mathbf{X}$ cutoff center positions.

4.6.2 Implementation of numerical Jacobian

Since we have the term $J(\mathbf{y})$ in our semi-linear IMEX Euler scheme, we seek to understand how to compute a numerical Jacobian of a non-closed form velocity expression. Consider a general nth ordered dynamical system:

$$x_1' = f_1(x_1, \ldots, x_n) \tag{4.6.7}$$

$$x_2' = f_2(x_1, \ldots, x_n) \tag{4.6.8}$$

$$\vdots \tag{4.6.9}$$

$$x_n' = f_n(x_1, \ldots, x_n) \tag{4.6.10}$$

The Jacobian, $J(\mathbf{x}), \mathbf{x} \in \mathbb{R}^n$, is

$$J(\mathbf{x}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \ddots & \cdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix} \tag{4.6.11}$$

Where $J_{ij} = \frac{\partial f_i}{\partial x_j}$. If $J \in R^{m \times n}$, then the matrix-vector product $J(\mathbf{y}_n)\mathbf{y}_n \in R^n$, is a vector. Since the velocity field has no closed-form interpretation, as we calculated it numerically, we must calculate the Jacobian numerically.

Recall that for a function $f : \mathbb{R} \to \mathbb{R}$, we can find an approximation of the derivative using:

$$f'(x) \approx \frac{f(x+h) - f(x)}{h} \ , \quad h <<< 1 \tag{4.6.12}$$

Suppose we have a vector of functions that are evaluated at vectors such that $\mathbf{F} : \mathbb{R}^n \to \mathbb{R}^n$. Where $\mathbf{F} = \mathbf{F}(\mathbf{X})$, $\mathbf{X} \in \mathbb{R}^n$. We can write J_{ij} as a finite difference approximation, such that:

$$\frac{\partial F_i}{\partial x_j} = \frac{F_i(\mathbf{X} + h\mathbf{e}_j) - F_i(\mathbf{X})}{h} \tag{4.6.13}$$

Where $\mathbf{e}_j = (0, \ldots, 1, \ldots, 0)$ such that the jth entry is 1. This returns a scalar value at $J_i j$. If we wanted to compute the Jacobian column wise, we could simply this expression as:

$$\mathbf{J}_j = \frac{\mathbf{F}((\mathbf{X} + h\mathbf{e}_j) - \mathbf{F}(\mathbf{X}}{h} \tag{4.6.14}$$

Where J_j is $\frac{\partial \mathbf{F}}{x_j} \in \mathbb{R}^m$. This will be the expression we implement using MATLAB. If $\mathbf{X} = \{\mathbf{x}_k\}_{k=1}^n$, then we have $J \in \mathbb{R}^{2n \times 2n}$. The code for the numerical Jacobian is shown below:

4.6.3 MATLAB implementation

Listing 4.5: Numerical Jacobian algorithm

```
function J = jac_FD(F,X,h)


J = zeros(length(X),length(X));


for i = 1:length(X)
    e = zeros(length(X),1);
    e(i) = 1;
    J(:,i) = (F(0,X+h*e) - F(0,X))/h;
end
end
```

We will use this when writing the semi-linear IMEX Euler integrator.

4.6.4 Speeding up using GMRES

Recall the IMEX Euler for our tricking scheme for $\mathbf{y}' = J(\mathbf{y})\mathbf{y} + \mathbf{f}(\mathbf{y}) - J(\mathbf{y})\mathbf{y}$ is:

$$\mathbf{y}_{n+1} = (I + hJ(\mathbf{y}_n))^{-1}[\mathbf{y}_n + h((\mathbf{y}_n) - J(\mathbf{y}_n)\mathbf{y}_N)] \tag{4.6.15}$$

If $A = I + hJ(\mathbf{y}_n)$ and $b = \mathbf{y}_n + h(f(\mathbf{y}_n) - J(\mathbf{y}_n)\mathbf{y}_N)$, then we are computing a linear solve of form $A\mathbf{x} = \mathbf{b}$ or $A^{-1}\mathbf{b}$. The solution $\mathbf{x}$ solves $A\mathbf{x} = \mathbf{b}$. Instead of computing this using the backslash operator in MATLAB, we can implement iterative methods for linear systems to speed up our algorithm.

The generalized minimum residual method (GMRES) is an interative method for solving square linear systems such that $A\mathbf{x} = \mathbf{b}$ [15]. We are less concerned with the derivation of the algorithm, but will present the basic notions. Beginning with an initial guess $\mathbf{x}_0$, the r_0 residual is $r_0 = ||\mathbf{b} - A\mathbf{x}_0||_2 \implies r_n = ||\mathbf{b} - A\mathbf{x}_n||_2$ [15]. We seek to find a solution $\mathbf{x}_n$ such that $\mathbf{x}_n \in K_n(A, r_0)$, the nth Kyrlov subspace, such that:

$$\mathbf{x}_n \in K(A, r_0) = \text{span}\{r0, Ar_0, A^2r_0, \ldots, A^{n-1}r_0\} \tag{4.6.16}$$

We iterate this algorithm until $r_n = ||b - A\mathbf{x}_n|| < \epsilon$ for some small tolerance. The iterative algorithm is written as:

$$\mathbf{x}_n = \mathbf{x}_0 + Q_n\mathbf{y}_n \tag{4.6.17}$$

The term Q_n is a matrix formed by orthogonal vectors $\mathbf{q}_1, \ldots, \mathbf{q}_n$, and $\mathbf{y}_n \in \mathbb{R}^n$ [15]. Again, we are not concerned with the algorithm beyond this. MATLAB has a built in $gmres(A, b)$ function that we will use. We can transform the system $A\mathbf{y}_{n+1} = \mathbf{b}$ where $A = I - hJ(\mathbf{y}_n)$ $\mathbf{b} = \mathbf{y}_n + h(\mathbf{f}(\mathbf{y}_n) - J(\mathbf{y}_n)\mathbf{y}_n)$. Transforming this system using $gmres()$, would change the algorithm such that:

$$\mathbf{y}_{n+1} = (I - hJ(\mathbf{y}_n))^{-1}[\mathbf{y}_n + h(f(\mathbf{y}_n) - J(\mathbf{y}_n)\mathbf{y}_n)] \tag{4.6.18}$$

$$\mathbf{y}_{n+1} = \text{gmres}\Big(I - hJ(\mathbf{y}_n), \mathbf{y}_n + h(f(\mathbf{y}_n) - J(\mathbf{y}_n)\mathbf{y}_n)\Big) \tag{4.6.19}$$

Unfortunately, using GMRES in this way instead of the backslash operator will not save us much computational time. Luckily, MATLAB allows an anonymous function handle to be passed through, such that $gmres(@(x)\ Ax, b)$ will compute

matrix vector products, instead of forming the entire matrix. This is much quicker, and will save us a lot of time compared to using the backslash. The algorithm for this uniquely defined integrator is shown below:

4.6.5 MATLAB implementation

Listing 4.6: Semi-linear IMEX method using Jacobians

```
function [ys,cpu_time] = IMEXSemiLinearEulerGMRES(F,
   tspan,y0,N)


ys(:,1) = y0;
y = y0;
dt = diff(tspan)/N;
t = tspan(1);
dim = length(y0);
tol = 10^-4;
h = 10^-7;


tic
for i = 1:N
    Jv = jac_ProdFD(F,y,y,h)';
    [y,flag] = gmres(@(X) X-dt*jac_ProdFD(F,y,X,h)', y +
        dt*(F(t,y) - Jv),...
        dim,tol,dim);
    ys(:,i+1) = y;
    t = t + dt;
end
cpu_time = toc;
```

Note that we multiply $I - hJ(\mathbf{y}_n)$ with $\mathbf{y}_n$, allowing for only matrix vector products

to be computed throughout the entirety of the scheme. The term X in the code can be thought of the solution to $A\mathbf{y}_{n+1} = \mathbf{b}$.

4.6.6 Convergence and precision of time integrators for spring connected Stokeslets in 2-dimensions

For the analysis of convergence and precision for Stokes equations, we proceed similarly as we did in previous sections. Suppose $\mathbf{X} \in \mathbb{R}^2$ where n represents the amount of cutoffs. If we compute a 'fine' solution such that N_{tref} is large, then we denote $\tilde{\mathbf{X}}$ the position of the n cutoff centers at time step N_{t_i}. The error expression between these two vectors:

$$e_{t_f} = ||\mathbf{X}_{t_f} - \tilde{\mathbf{X}}_{t_f}||_2 \tag{4.6.20}$$

We will introduce 3 methods for this analysis - Euler, Heun, and our new semi-implicit integrator. Suppose we first measure error using time steps $N_t \in [2^6, \ldots, 2^10]$ with a reference time step $N_{tref} = 2^11$. For simplicity of convergence we will use $m = 5$ cutoffs, a less rigid spring constant $k = 10^3$, and a resting length $\delta_0 = .1$. We will measure error along the temporal interval $t \in [0, .5]$. Note the convergence and precision diagrams below:

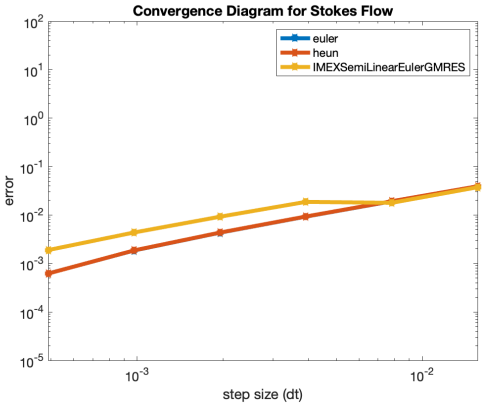

Figure 4.6: Convergence diagram for Stokes flow in 2-dimensions

Note that we are getting the expected convergence for forward Euler and what we will now refer to as GMIME (GMRES IMEX Euler) - that being $O(h)$. Unfor-

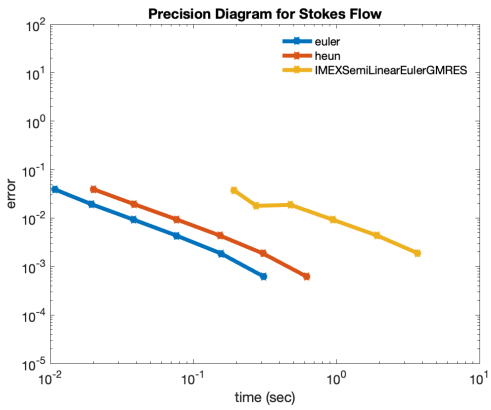

Figure 4.7: Precision diagram for Stokes flow in 2-dimensions

tunately, Heun is not converging. This could be due to problem stiffness, or other factors that are currently unknown to the authors.

The precision diagram tells us that forward Euler is the best choice for an unstiff system. GMIME takes several degrees of magnitude to start becoming precise. Yet this is not the full scope of the problem. As discussed with the Heat equation, when stiffness becomes an issue, we will need the benefits of GMIME's linearly implicit stability.

Suppose we impose a large value of $k = 10^5$. This is where the help of GMIME comes in. With such a high stiffness, forward Euler and Heun completely break down, while GMIME is still able to solve the problem. Using time steps $N_t = [10, \ldots, 10^4]$ with $N_{\text{tref}} = 10^5$, we can see the benefit of our method below:

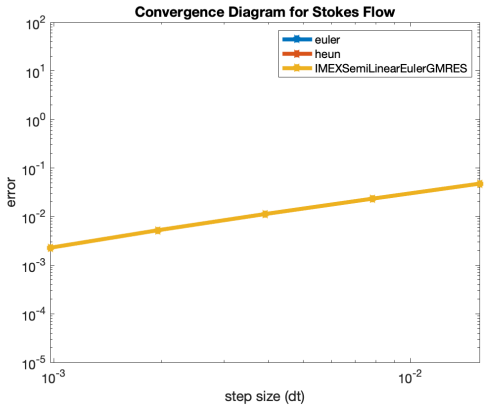

Figure 4.8: Convergence diagram for Stokes flow in 2-dimensions, k = 10^5

We conclude for 2-dimensional Stokeslets connected by a highly stiff spring,

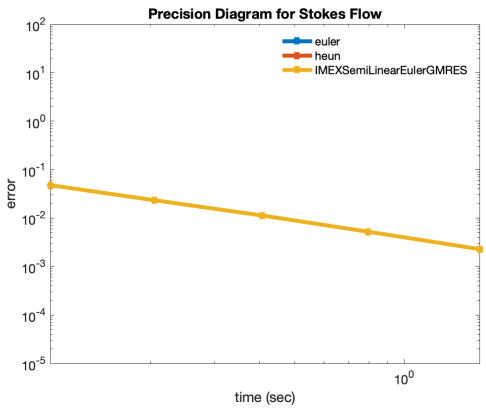

Figure 4.9: Precision diagram for Stokes flow in 2-dimensions, k = 10^5

that GMIME is the most suitable method because of its linearly implicit stability.

4.7 Time integration for Stokeslets in 3-dimensions

4.7.1 Velocity solutions for a cutoff in 3-dimensions

We now choose to analyze our spring connected Stokeslets in such that each center $\mathbf{x}_k \in \mathbb{R}^3$. We will choose a cutoff, $\phi_\epsilon(\mathbf{x})$, such that:

$$\phi_\epsilon(\mathbf{x}) = \frac{15\epsilon^4}{8\pi(|x|^2+\epsilon^2)^{7/2}} \tag{4.7.1}$$

Again where $\mathbf{x} \in \mathbb{R}^3$ [5]. Using the radially symmetric properties, i.e. $G'_\epsilon(r) = \frac{1}{r}\int_0^r s\phi_\epsilon(s)$, and the same for $B'_\epsilon(r)$, we have:

$$G_\epsilon(r) = \frac{-2r^2-3\epsilon^2}{8\pi\sqrt{r^2+\epsilon^2}} \tag{4.7.2}$$

$$B'_\epsilon(r) = -\frac{r}{8\pi\sqrt{r^2+\epsilon^2}} \tag{4.7.3}$$

The following Stokeslet velocity solution is:

$$\mathbf{u}(\mathbf{x}) = \frac{1}{\mu}\sum_{k=1}^{N}\mathbf{f}_k\left[\frac{r^2+2\epsilon^2}{8\pi\sqrt{r^2+\epsilon^2}}\right] + [\mathbf{f}_k\cdot(\mathbf{x}-\mathbf{x}_k)](\mathbf{x}-\mathbf{x}_k) - \frac{1}{8\pi\sqrt{r^2+\epsilon^2}} \tag{4.7.4}$$

We modify our code to appropriately fit such parameters. We now proceed with the same analysis as done for Stokeslets in $\mathbb{R}^2$. [5]

4.7.2 Convergence and precision analysis

We conclude our analysis into the method of regularized Stokeslets in 3-dimensions by loading a certain set of parameters to measure convergence and precision. Suppose that we have $n = 5$ cutoff centers in 3 dimensions, in matrix form $\mathbf{X}$, such that:

$$\mathbf{X} = \begin{bmatrix} 0 & .01 & \dots & .04 \\ 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 \end{bmatrix} \tag{4.7.5}$$

Each column, $\mathbf{X}_j$ represents a cutoff center $\mathbf{x}_k \in \mathbb{R}^3$. We only perturb the x location in the 3-dimensional space. Let $\delta_0 = .1$ represent the resting length of the spring, with constant $k = 10^4$. Recall that the stiffness of the system is drive by the value of k. Suppose we measure temporal error using a reference time step of $N_{\text{tref}} = 2^13$ with coarse time steps $N_t = [2^6, \dots, 2^{12}]$, with $t \in [0, 1]$. We anticipate both Euler and Heun to fail, as the system becomes too numerically stiff to handle a spring constant of such magnitude. Shown below is the convergence and precision diagram for this specific simulation:

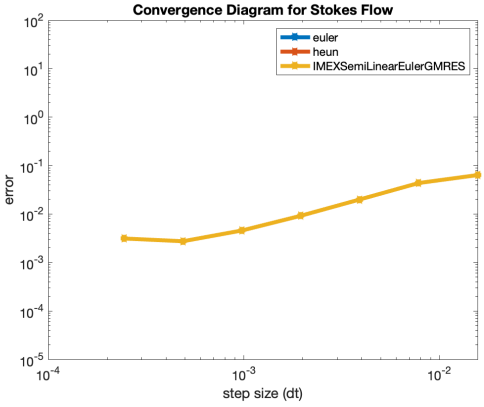

Figure 4.10: Convergence diagram for Stokes equations in $\mathbb{R}^3$

For temporal convergence, we confirm that the GMIME method has $O(h)$ convergence. We could improve this by adding an additional stage, perhaps by

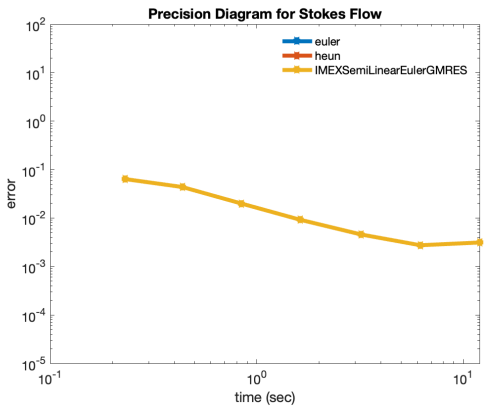

Figure 4.11: Precision diagram for Stokes equations in $\mathbb{R}^3$

scheming the method to parallel the IMEX midpoint method. Both Euler and Heun do not show up, as the fully explicit methods are not equipped to handle stiff systems.

The precision diagram gives us the conclusion that we seek. We reach an error threshold of 10^{-1} in computational time $T = .1$ seconds. It would take an inconceivable amount of time steps for Heun and Euler to converge. We conclude that the GMIME method is successful in solving stiff systems for spring-connected Stokeslets. By employing some implicit stability, it is able to handle the robust eigenvalues of the system, while Euler and Heun are unsuccessful.

Chapter 5

Conclusions and further work

In this paper we sought to introduce and analyze the concept of time integration for several PDEs, as well as solve a stiff spring connected Stokeslet system by removing some of the stiffness through implicit-explicit time integration schemes. In 2 and 3-dimensions, we hypothesized and confirmed that $k = 10^4$ was too numerically stiff for Euler and Heun, while our 'GMIME' method was able to handle it. Therefore we conclude that the method was both stable, and succesful at finding a velocity integrated solution to the Stokes equations in 3-dimensions.

Although our experiment was successful, we look to continue our work further in a number of ways. This problem is rather specific. For general Stokeslets that aren't imposed stiffly, Euler is suitable enough to solve the subsequent system of differential equations. If we compare GMIME to the two explicit methods, it will be significantly slower in reaching a low error tolerance, due to the solves computed by GMRES. We seek to continue our study in this research field by looking for ways to speed up the GMIME method in solving the method of regularized Stokeslets. We will proceed by examining other integrators and their efficiencies, and hopefully reach a point where we can find an integrator faster than Euler.

Additionally, we will look to implement fully implicit solvers for the Stokes equations, by using the Jacobian method as well as Newton solvers. In addition to time integration schemes, we will look further into the world of high performance

computing and deep learning to study the underlying mechanics of these methods and how they can improve the method of regularized Stokeslets. For example, the Parareal algorithm allows us to compute numerical solutions across several parallel systems. This will be studied further.

Bibliography

[1] Uri M. Ascher, Steven J. Ruuth, and Raymond J. Spiteri. "Implicit-explicit Runge-Kutta methods for time-dependent partial differential equations". In: *Applied Numerical Mathematics* (1997).

[2] Kendall Atkinson, Weimin Han, and David Stewart. *Numerical Solution of Ordinary Differential Equations*. 1st ed. John Wiley and Sons, 2009.

[3] Andrew Benitez. "Fluid Dynamics: The Navier Stokes Flow". In: *UNKNOWN* (2021).

[4] Fan Chung and S.-T. b Yau. "Discrete Green's Functions". In: *Journal of Computational Theory* ().

[5] Ricardo Cortez. "THE METHOD OF REGULARIZED STOKESLETS". In: *Journal of COmputational Physics* (2000).

[6] C. Edwards Henry and David Penney E. *Differential Equations and Boundary Value Problems*. 5th ed. Pearson, 2015.

[7] Richard Haberman. *Elementary Applied Partial Differential Equations*. 2nd ed. Prentice Hall, 1987.

[8] Seongjai Kim. "Numerical Methods for Partial Differential Equations". In: *Mississippi State University* ().

[9] Tai-Ping Liu. "Hopf-Cole Transformation". In: *Academia Sinica* (2017).

[10] P.C. Matthews and S.M. Cox. "Exponential Time Differencing for Stiff Systems". In: *Journal of COmputational Physics* (2000).

[11] Cleve Moler. *Numerical Computing with MATLAB*. 1st ed. SIAM, 2004.

[12] NYU. “Stokes Flow”. In: *UNKNOWN* (2023).

[13] Anthony Ralston. “Runge-Kutta methods with minimum error bounds”. In: *Mathematics of Computation* ().

[14] Gerald Recktendwald. “Finite-Difference Approximations to the Heat Equation”. In: *UNKNOWN* ().

[15] Youcef Saad and Martin H. Schultz. “GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems”. In: *SIAM Journal on Scientific and Statistical Computing* 7.3 (1986).

[16] Bashar Zogheib et al. “Method of lines for multi-dimensional coupled viscous Burgers’ equations via nodal Jacobi spectral collocation method”. In: *Journal of Computational Theory* ().

About the author

The author completed his undergraduate education at Tulane University, graduating in May 2022 with a Bachelor's of Science in Mathematics, with a minor in Chemistry. They conclude, for the foreseeable future, their education by receiving their Master's of Science in Applied Mathematics in May 2023. Moving forward, the author will use many of the skills they learned in their academic career in industry - namely data and computer science. They will continue their research into the field of numerical computing and applied mathematics as time allows.