The Fractional Laplacian: An adaptive finite difference approach in one and two dimensions, with applications.

Ailbhe Mitchell

Supervisor: Dr. P.A. (Paul) Zegeling
Second reader: Prof. dr. ir. C.W. (Kees) Oosterlee

Master's Thesis 2022

July 4th 2022

Acknowledgements

I'm extremely grateful to Dr. Paul Zegeling for his enthusiastic supervision throughout this thesis and valuable feedback and guidance. A special thanks also to my parents, siblings and Andrew for their support and encouragement.

Abstract

The fractional Laplacian can be viewed as a generalisation of the ordinary Laplacian and likewise can be used to model systems in sectors such as engineering, hydrology and biology. Super diffusion, a consequence of Lévy flights, can be described using the fractional Laplacian, $-(-\Delta)^{\alpha/2}$, of order $\alpha \in (0,2)$. The fractional Laplacian has numerous definitions and here we will use the Riesz potential definition in one and two dimensions (which is equal to the Riesz derivative in one dimension). Methods exist to approximate the fractional Laplacian on fixed uniform grids including the L1 and L2 methods derived from the Caputo fractional derivative definitions. In this thesis, the new L1(NU) and L1(NU)-2D methods for $\alpha \in (0,1)$, and the L2(NU) and L2(NU)-2D methods for $\alpha \in (1,2)$ will be presented to approximate the fractional Laplacian on non-uniform adaptive finite difference meshes in one and two dimensions. The L1(NU) and L1(NU)-2D methods for $\alpha \in (0,1)$ display some unstable behaviours and therefore require a higher number of grid points, and less adaptive grids to work well. The L2(NU) and L2(NU)-2D methods for $\alpha \in (1,2)$ are successfully applied to space-fractional heat, advection-diffusion and Fisher's PDEs where the solutions and adaptive grids behave as expected.

Contents

Introduction

The notion of fractional derivatives was first considered in September 1695 in a letter between two leading mathematicians, Leibniz and L'Hôpital, where L'Hôpital asked Leibniz if his notation for integer order derivatives could be extended to fractional orders, specifically 1/2. With no known use for such quantities, it was difficult for Leibniz to know where to begin in defining a fractional differential and this eventually lead him to so-called "paradoxical" results [32] [47]. It wasn't until 1823 when Niels Henrik Abel published a paper studying the Tautochrone problem that fractional differentials were used to elegantly solve a problem [47], and Leibniz's belief that such objects would have "*fundamentum un re*" was proved to be true [33]. During the 19^{th} century many different definitions for fractional derivatives appeared in texts; Laplace in 1812, Lacroix in 1819, Fourier in 1822, Liouville in 1832, but it wasn't until 1974 that the first text entirely dedicated to fractional calculus was published by Oldham and Spanier [47].

The theory of fractional calculus and its application in science and engineering has developed extensively over the last 50 years [39] [22] [49], and much of the focus has been on finding numerical methods for approximating fractional derivatives [34]. There exists a multitude of methods for approximating fractional derivatives on uniform finite difference grids [34] [18], and using finite element methods [34]. However, there are far fewer existing methods which can be applied to non-uniform finite difference grids [34] and there are no signs in the literature of space-fractional derivatives being approximated on adaptive finite difference meshes, in one or two dimensions. In this thesis we will consider partial differential equations (PDEs) in one and two dimensions which involve the fractional Laplacian - a space-fractional partial differential operator. Such PDEs will be solved numerically using finite difference methods on an adaptive grid.

We will draw our focus to fractional derivatives of order less than 2, and particularly of order $\alpha \in (0, 2)$. These will define a fractional Laplacian which will govern super diffusive behaviour - a common application of fractional derivatives in the physical world [22]. Some examples of these are Lévy flights applied to modern epidemic models [27], finance [13] and reaction diffusion models [9]. Conveniently, for lower orders of α, many of the definitions for fractional derivatives are simpler in that they also involve lower order integer derivatives which are often simpler (at least in a book keeping sense) to approximate on finite difference grids, particularly in the non-uniform case. It is also notable that the fractional Laplacian holds the property that in the limit $\alpha \to 2$ it equals the ordinary Laplacian [46], so by choosing this range for the order of the fractional derivatives, α, we can check that this property holds for the numerical methods we develop.

Just as ordinary diffusion governed by the ordinary Laplacian can be derived via Brownian motion, super diffusion which is governed by the fractional Laplacian for $\alpha \in (0, 2)$ can be derived via Lévy flights. Such a foundational derivation is striking and it can help motivate us to see that fractional calculus is a deep-rooted part of our universe - although it has its origin linked to the extension of meaning to symbolic notation, it is in fact useful and can describe behaviours such as Lévy flights which are fundamentally different to Brownian motions.

There are many different ways of defining fractional derivatives [34] and while each definition has its advantages in certain situations [22], in general they are not equivalent. These different possible ways of defining the differential quantities is one of the things that makes the study of fractional calculus so interesting, but it also means we have to make a decision as to which definition we use. The fractional

Laplacian is defined via the different definitions for fractional derivatives, with each definition preserving different properties of the ordinary Laplacian. In this thesis we will focus on the fractional Laplacian defined via the Riesz derivative which is a combination of the left and right Riemann-Liuoville derivatives, and the Riesz potential which is a natural extension of the ordinary Laplacian [11]. These are equivalent in one dimension, but not in two dimensions [43].

There are many known ways to approximate the various definitions for fractional derivatives on uniform finite difference grids in one and two dimensions for $\alpha \in (0, 2)$ [34] [18]. Not all of these definitions are as easy as others to work with: some involve singular integrals, and others are undefined for particular orders of derivatives, for example $\alpha = 1$. Throughout this thesis we study three different existing methods which approximate fractional derivatives on uniform finite difference grids in one dimension, namely: "A novel and accurate finite difference method for the fractional Laplacian" [18], approximations of the Grünwald-Letikov definitions [34], and the "L1" and "L2" methods for approximating Caputo derivatives [34]. We will choose one of these methods to motivate an approach to find approximations to the fractional Laplacian on non-uniform grids in one and two dimensions.

Finding an approximation for the fractional Laplacian on a non-uniform grid is a crucial step to being able to solve a PDE on an adaptive grid. Once a method for discretising the space-fractional part of the PDE is found on a non-uniform grid, we can then implement existing adaptive grid methods used for first order derivative in time PDEs. The methods used in this thesis were developed by Verwer et al. [52] for one dimension and taken from papers by Zegeling et al. [55] [57] [58] for two dimensions. We will consider PDEs of the form,

$$u_t(\underline{x}, t) = F(u, \underline{x}, t), \quad x \in \mathbb{R} \text{ or } \mathbb{R}^2, \tag{1}$$

where F is a function including the space-fractional Laplacian. Thus, once we have a finite difference approximation for F, and have chosen an adaptive grid method, we can use a stiff differential algebraic equation (DAE) solver to solve the resulting system of differential equations. We will use two solvers, Assimulo's Radau5DAE, a fifth-order, three-stage method with step-size control based on the FORTRAN code RADAU5 [2], and a python wrapper for Sundials DAE integrator IDA which is written in C. This method is adaptive in time and uses variable-order, variable-coefficient BDF (Backward Differentiation Formula) [40].

These solvers will allow us to run numerical experiments solving PDEs of type (1) on adaptive finite difference grids in one and two dimensions. When using adaptive grids we expect to see that the density of grid points will change as the solution evolves according to where the solution has a greater rate of change. We will consider three applications: (i) space-fractional heat equations, (ii) space-fractional advection-diffusion equations and (iii) space-fractional heat equations with Fisher's term. These three applications will allow us to observe the degree of success of the adaptive grids. In theory using adaptive grids is advantageous compared to fixed grids as we can potentially decrease the number of grid points we need to use as they will automatically move to where they are needed most. This reduction of grid points can lead to increased efficiency in solving PDEs [8] [29].

Chapter 1

Origins of the Fractional Laplacian and Important Definitions

In this chapter we will discuss the origins of the fractional Laplacian and the reasons why it took a long time to properly understand and utilise the theory. Just as the ordinary Laplacian can be derived from studying the global dynamics of point particles admitting Brownian motion, so too can the fractional Laplacian be uncovered by studying the somewhat similar behaviour of Lévy flights leading to super-diffusion. The ideas from these derivations can provide motivation for solving PDEs involving the fractional Laplacian in later chapters as we will see that systems that can be described in this way exist intrinsically within our world. We will also present the important definitions of fractional derivatives that we will use throughout this thesis, and discuss some interesting properties of the fractional Laplacian, including the effects that boundary conditions have on solutions to space-fractional PDEs. Finally, we will introduce the standard ways we will approximate the ordinary Laplacian in one and two dimensions.

1.1 Origins of fractional calculus

Many branches of mathematics have their origins in finding solutions to specific problems - whether they be theoretical or physical questions. This is not the case for fractional calculus which arose due to a question L'Hôpital had for Leibniz in 1695 regarding the limits of the symbolic notation Leibniz used to describe a n^{th} order integer derivative. "What if n equals 1/2?" asked L'Hôpital in a letter to Leibniz, which set the study of fractional calculus in motion [47]. In Leibniz's reply he tried to answer this question by assuming that the rule for differentiating exponential functions would be preserved for fractional derivatives also [32]; that is,

$$\frac{d^q}{dx^q}e^{ax} = a^q e^{ax}, \text{ for } q \in \mathbb{Q}^{>0}. \tag{1.1}$$

Leibniz refers to his result as a paradox, and this could be for a variety of reasons. Although he doesn't state any contradictions, his result can lead to perplexing results when his product rule for differentials is applied to it [44]; we in fact now use a generalised product rule when working with fractional derivatives [48]. Using generalised factorials in the form of Gamma functions and expressing e^x as an infinite sum, we reach a contradiction to Leibniz's result [7]. Given that Gamma functions were introduced after Leibniz's time this is the least likely reason for him regarding his findings a paradox, although it is possible he may have considered another form of a generalised factorial, perhaps via interpolations, that could have led him to see this contradiction. This contradiction can be solved by employing Weyl fractional derivatives which were defined in 1917 [23]. Lastly, given the use of the world "paradox" during the 17^{th} century [42], it is likely Leibniz saw no contradiction in his workings but referred to his results as paradoxical because with no mathematical problem to apply fractional differentials to, he could neither make sense of nor understand them.

Whatever Leibniz meant when he called these fractional derivatives "paradoxical", it is true that fractional differentials were poorly understood for quite some time. Contributions were made by Euler in 1730 [21] who considered what a curve defined by,

$$yd^{1/2}x = d\sqrt{dy}, \tag{1.2}$$

would look like, and Lagrange helped the cause in 1772 via his law of exponents for (integer) derivatives [47]. The first definition appeared in 1812 when Laplace used integrals to define fractional differentiation [22]. Lacroix studied a method to generalise integer order derivatives in 1812, yielding a result which agrees with the now used Riemann-Liouville derivatives [47]. In 1822, Fourier also defined these derivatives via integrals, but besides Euler's curves, possible applications remained evasive [22].

This lack of practical use for fractional calculus explains why between 1695 and 1822 the topic was not quickly evolving. In 1823, Abel published a paper which included the first use of fractional derivatives to solve a problem, namely the tautochrone problem [45]. Following this there was an increase of activity in the field: Liouville published three memoirs in 1832 where he applied his definitions to potential theory and was the first to consider fractional differential equations [47]. At this point there were now two very different definitions for fractional derivatives and these became known as Riemann class and Liouville class derivatives [47]. Riemann defined his version of fractional derivatives in a publication in 1876 via Taylor series [22]. N. Ya. Sonin (1869) and later Letnikov (1868-72) defined fractional derivatives via Cauchy's integral formula [22]. This work was continued by Laurent in 1884 whose results are now known as the Riemann-Liouville derivatives as in different limiting cases they preserve the properties of either Riemann or Liouville class derivatives [47]. We will study Riemann-Liouville derivatives in this thesis as well as Grünwald-Letnikov (1868) and Caputo derivatives (1967) [22]. These are not the only definitions that exist and are used today, we also have for example Weyl derivatives (1917) and many others [5] [22] [47] [34].

It is interesting to note that this origin story is the reason why the study of fractional calculus is given its name - in the letters between Leibniz and L'Hôpital, only fractional order derivatives are considered. However, the study of fractional calculus is not limited only to rationals: irrational and complex numbers can also be considered [3] [37].

1.2 Random walks and the fractional Laplacian

To understand the foundational way in which the fractional Laplacian exists within our real world, we will first look at how the ordinary Laplacian can be derived via observable particles exhibiting Brownian motion. The phenomenon known as Brownian motion was first observed by a Scottish botanist, Robert Brown, in 1828 when he noticed the "random walks" taken by pollen floating on the surface of a liquid.

> Having found motion in the particles of the pollen of all the living plants which I had examined, I was led next to inquire whether this property continued after the death of the plant, and for what length of time it was retained.
>
> *Robert Brown, 1828* [10]

And so began the investigation of Brownian particles which was later treated by Einstein in 1905 who used statistical thermodynamics to describe the movement of small particles suspended in a fluid [20]. Einstein found that the high frequency of collisions of the water molecules exhibiting Brownian motion was enough to influence the movement of the pollen, thus explaining why Brown was able to observe movement, even from dead cells.

Following a statistical approach for the diffusive regime over long timescales we can predict the mean-square displacement to be proportional to the time and diffusion constant, $\mathcal{D}$ [41];

$$\langle \Delta X^2 \rangle = 2\mathcal{D}t, \quad \text{where} \quad \mathcal{D} = \frac{k_B T}{6\pi\eta R}. \tag{1.3}$$

Here, R is the radius of the Brownian particle, η is the viscosity of the fluid in which the particle is suspended, T is the temperature and k_B the Boltzmann constant.

This proportionality result between the mean-squared displacement of the particle and time is central to understanding the physical differences between ordinary diffusion and super diffusion. The former is governed by the ordinary Laplacian, Δ, and can be derived via Brownian motion where the probability density function for displacement is of Gaussian type. The latter is governed by the fractional Laplacian, $-(-\Delta)^{\alpha/2}$ for $\alpha \in (0, 2)$, and can be derived by studying Lévy flights in which case the probability density function for displacement is no longer normally distributed.

Lévy flights are Markov processes in which the random walk of particles has spatial step sizes governed by a Lévy distribution [15]. These distributions differ to normal distributions as they are heavy tailed. The random walk is influenced by the Lévy distribution and now particles admit "random" jumps where the spatial step size has increased. Figure 1.1 displays this behaviour where we see ordinary Brownian motion ($\alpha = 2$) as well as random walks governed by Lev́y flights ($\alpha = 1, 5$ and 1.75) in two dimensions. We can see that for $\alpha = 1.5$ the behaviour of the random walk is more erratic and the variation of spatial step sizes is more pronounced. As $\alpha \to 2$, we observe Brownian motion.

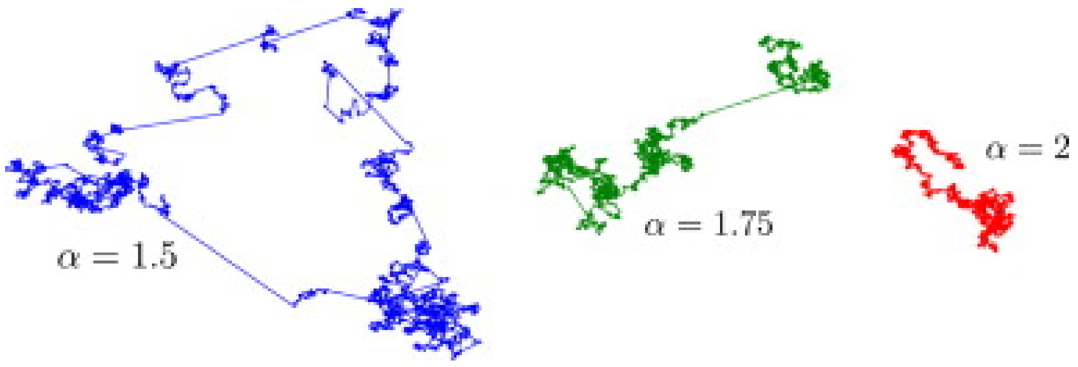


Figure 1.1: Random walks in two dimensions for Brownian motion ($\alpha = 2$), and Lévy flights ($\alpha = 1.5$ and 1.75). Figure is taken from [26].

It can in fact be shown that when we consider diffusion equations given by,

$$u(x,t)_t = -(-\Delta)^{\alpha/2} u(x,t), \tag{1.4}$$

the mean square displacement varies with time as follows [22] [28],

$$\langle \Delta X^2 \rangle \propto t^{2/\alpha}. \tag{1.5}$$

For $\alpha = 2$ we have ordinary diffusion, present in homogeneous media; for $\alpha > 2$ we have sub or anomalous diffusion, present in disordered solids and fractal or porus media; and for $\alpha < 2$ we have super diffusion [28] [22]. Super diffusion is governed by the fractional Laplacian for $\alpha \in (0, 2)$ and gets its name from the more rapid diffusive regime we observe. We will focus on super diffusion throughout this thesis which appears in the presence of Lévy flights, transport in polymer and turbulent plasma [28], modern epidemic models [27], finance models [13] and reaction diffusion models [9].

For details of how we can begin with a Lévy distribution and reach the fractional diffusion equation see [28]. The derivation starts with a Fourier-Laplace transformed equation for a continuous time random walk and Fourier transforms of the Lévy distribution; after some substitution, the inverse Fourier transform of the Riesz derivative [28] [34] can be applied to give us the fractional diffusion equation. This derivation highlights the natural existence of fractional calculus and we can see that the fractional Laplacian appears as a consequence of nature as opposed to a derived tool.

1.3 Important definitions

There are many different definitions for fractional derivatives, which are useful in different situations, and are equivalent to each other under certain conditions. In this section we will introduce all of the definitions that are used within this thesis and discuss when they are equal to one another.

1.3.1 One dimension

The Riemann–Liouville derivatives are a more general form of the definition Lacroix used in 1819, and they are named after both Riemann and Liouville as in different limiting cases they agree with their earlier definitions [22]. This definition is widely used today.

Definition 1. The left and right Riemann-Liouville derivatives of order $\alpha > 0$ for a function $u(x)$, where $x \in [a, b]$ are given by [34],

$$ {}_{RL}D^{\alpha}_{a,x}u(x) = \frac{1}{\Gamma(m-\alpha)} \frac{d^m}{dx^m} \int_a^x (x-s)^{m-\alpha-1} u(s) ds, \tag{1.6}$$

$$ {}_{RL}D^{\alpha}_{x,b}u(x) = \frac{(-1)^m}{\Gamma(m-\alpha)} \frac{d^m}{dx^m} \int_x^b (s-x)^{m-\alpha-1} u(s) ds, \tag{1.7}$$

respectively. $m \in \mathbb{Z}^{>0}$ such that $m - 1 \leq \alpha < m$.

Specifically this means that for $\alpha \in (0, 1),\ m = 1$,

$$ {}_{RL}D^{\alpha}_{a,x}u(x) = \frac{1}{\Gamma(1-\alpha)} \frac{d}{dx} \int_a^x (x-s)^{-\alpha} u(s) ds, \tag{1.8}$$

$$ {}_{RL}D^{\alpha}_{x,b}u(x) = \frac{(-1)}{\Gamma(1-\alpha)} \frac{d}{dx} \int_x^b (s-x)^{-\alpha} u(s) ds. \tag{1.9}$$

For $\alpha \in [1, 2),\ m = 2$,

$$ {}_{RL}D^{\alpha}_{a,x}u(x) = \frac{1}{\Gamma(2-\alpha)} \frac{d^2}{dx^2} \int_a^x (x-s)^{1-\alpha} u(s) ds, \tag{1.10}$$

$$ {}_{RL}D^{\alpha}_{x,b}u(x) = \frac{1}{\Gamma(2-\alpha)} \frac{d^2}{dx^2} \int_x^b (s-x)^{1-\alpha} u(s) ds. \tag{1.11}$$

The Caputo definitions for fractional derivatives were introduced in 1967 and are related to the Riemann-Liouville definitions [12] [22].

Definition 2. The left and right Caputo derivatives of order $\alpha > 0$ for a function $u(x)$, where $x \in [a, b]$ are given by [34],

$$ {}_C D^{\alpha}_{a,x} u(x) = \frac{1}{\Gamma(m-\alpha)} \int_a^x (x-s)^{m-\alpha-1} u^{(m)}(s) ds, \tag{1.12}$$

$$ {}_C D^{\alpha}_{x,b} u(x) = \frac{(-1)^m}{\Gamma(m-\alpha)} \int_x^b (s-x)^{m-\alpha-1} u^{(m)}(s) ds, \tag{1.13}$$

respectively. $m \in \mathbb{Z}^{>0}$ such that $m - 1 \leq \alpha < m$. For $\alpha \in (0, 1),\ m = 1$, for $\alpha \in [1, 2),\ m = 2$.

Note that the integer differential operators in the Riemann-Liouville definitions have been moved inside the integral for the Caputo derivatives. These two definitions for fractional derivatives are related by the following relations,

$$ {}_{RL} D^{\alpha}_{a,x} u(x) = \sum_{k=0}^{m-1} \frac{u^{(k)}(a)(x-a)^{k-a}}{\Gamma(k+1-\alpha)} + {}_C D^{\alpha}_{a,x} u(x), \tag{1.14}$$

$$ {}_{RL} D^{\alpha}_{x,b} u(x) = \sum_{k=0}^{m-1} \frac{u^{(k)}(b)(b-x)^{k-a}}{\Gamma(k+1-\alpha)} + {}_C D^{\alpha}_{x,b} u(x). \tag{1.15}$$

Hence throughout this thesis we will impose the conditions that: for $\alpha \in (0, 1)$, $x \in [a, b]$, $u(a) = u(b) = 0$, and for $\alpha \in [1, 2),\ x \in [a, b],\ u(a) = u'(a) = u(b) = u'(b) = 0$. Thus for the problems we consider the Caputo and Riemann-Liouville derivatives will agree.

Another alternative definition for fractional derivatives are the Grünwald-Letnikov definitions, inspired by Liouville's work and introduced by Grünwald in 1867 and Letnikov in 1868 [22]. These are found via a simple extension of the general n^{th} integer derivative, to general fractional orders.

For a function $u(x)$, $x \in (a, b)$, the first derivative can be defined by,

$$\frac{d}{dx} u(x) = \lim_{h \to 0} \frac{u(x) - u(x-h)}{h},$$

and a backward difference approximation by,

$$u'\big|_{x_n} \simeq \frac{u_n - u_{n-1}}{h},$$

where $u'\big|_{x_n}$ is the first derivative of $u(x)$ at the n^{th} spatial grid point x_n, $n = 0, ..., N + 1$ and $h = (b - a)/(N + 1)$ is constant. Using $\frac{d^{q+1}}{dx^{q+1}} = \frac{d}{dx}\frac{d^q}{dx^q}$, we can find expressions for the second and third derivatives as follows.

$$\begin{aligned}
\frac{d}{dx} u(x) &= \lim_{h \to 0} \frac{u(x) - u(x-h)}{h}, & u'\big|_{x_n} &\simeq \frac{u_n - u_{n-1}}{h}, \\
\frac{d^2}{dx^2} u(x) &= \lim_{h \to 0} \frac{\frac{d}{dx} u(x) - \frac{d}{dx} u(x-h)}{h}, & u''\big|_{x_n} &\simeq \frac{u'_n - u'_{n-1}}{h}, \\
&= \lim_{h \to 0} \frac{1}{h}\left[\frac{u(x) - u(x-h)}{h} - \frac{u(x-h) - u(x-2h)}{h}\right], & &= \frac{1}{h}\left[\frac{u_n - u_{n-1}}{h} - \frac{u_{n-1} - u_{n-2}}{h}\right], \\
&= \lim_{h \to 0} \frac{u(x) - 2u(x-h) + u(x-2h)}{h^2}. & &= \frac{u_n - 2u_{n-1} + u_{n-2}}{h^2}. \\
\frac{d^3}{dx^3} u(x) &= \lim_{h \to 0} \frac{\frac{d^2}{dx^2} u(x) - \frac{d^2}{dx^2} u(x-h)}{h}, & u'''\big|_{x_n} &\simeq \frac{u''_n - u''_{n-1}}{h}, \\
&= \lim_{h \to 0} \frac{u(x) - 3u(x-h) + 3u(x-2h) - u(x-3h)}{h^3}. & &= \frac{u_n - 3u_{n-1} + 3u_{n-2} - u_{n-3}}{h^3}.
\end{aligned}$$

Hence we arrive at a general formula for the q^{th} integer derivative.

Theorem 1. *The q^{th} derivative, $q \in \mathbb{N}$ for a function $u(x)$, $x \in (a,b)$, is given by,*

$$D^q u(x) = \frac{d^q}{dx^q} u(x) = \lim_{h\to 0} \frac{1}{h^q} \sum_{k=0}^{q} (-1)^k \binom{q}{k} u(x-hk). \tag{1.16}$$

The backward, finite difference approximation for the q^{th} derivative of $u(x)$ at the n^{th} spatial grid point x_n, $n = 0,...,N+1$, $h = (b-a)/(N+1)$ is constant is given by,

$$u^{(q)}(x)\Big|_{x_n} \simeq \frac{1}{h^q} \sum_{k=0}^{q} (-1)^k \binom{q}{k} u_{n-k}. \tag{1.17}$$

Proof. We will only show the proof for the backward finite difference approximation as the proof for the general q^{th} derivative given in equation (1.16) is the same. We have shown already that this expression holds for $n = 1,2,3$, so using proof by induction now let us assume the expression holds for $q = r$, $r \in \mathbb{N}$. Then we have,

$$\begin{aligned}
u^{(r)}(x) &\simeq \frac{1}{h^r} \sum_{k=0}^{r} (-1)^k \binom{r}{k} u_{i-k}.\\
u^{(r+1)}(x) = \left(u^{(r)}(x)\right)' &\simeq \left(\frac{1}{h^r} \sum_{k=0}^{r} (-1)^k \binom{r}{k} u_{i-k}\right)',\\
&= \frac{1}{h^r} \sum_{k=0}^{r} (-1)^k \binom{r}{k} \left(\frac{u_{i-k} - u_{i-(k+1)}}{h}\right),\\
&= \frac{1}{h^{r+1}} \left[\sum_{k=0}^{r} (-1)^k \binom{r}{k} u_{i-k} - \sum_{s=1}^{r+1} (-1)^{s-1} \binom{r}{s-1} u_{i-s}\right] \quad \text{(where } s = k+1\text{)},\\
&= \frac{1}{h^{r+1}} \left[\sum_{k=1}^{r} \left((-1)^k \binom{r}{k} - (-1)^{k-1} \binom{r}{k-1}\right) u_{i-k} + (-1)^0 \binom{r}{0} u_i - (-1)^r \binom{r}{r} u_{i-(r+1)}\right],\\
&= \frac{1}{h^{r+1}} \left[\sum_{k=1}^{r} \left((-1)^k \frac{r!}{k!(r-k)!} - (-1)^{k-1} \frac{r!}{(k-1)!(r-k+1)!}\right) u_{i-k} + u_i + (-1)^{r+1} u_{i-(r+1)}\right],\\
&= \frac{1}{h^{r+1}} \sum_{k=0}^{r+1} (-1)^k \binom{r+1}{k} u_{i-k}
\end{aligned}$$

□

Now we can extend equation (1.16) to find the left Grünwald-Letikov derivative. The right fractional derivative is found similarly.

Definition 3. The left and right Grünwald-Letnikov derivatives of order $\alpha > 0$ for a function $u(x)$, where $x \in [a,b]$ and $h = \frac{x-a}{N}$, are given by [34],

$${}_{GL}D^{\alpha}_{a,x} u(x) = \lim_{h\to 0} h^{-\alpha} \sum_{j=0}^{N} (-1)^j \binom{\alpha}{j} u(x - jh), \text{ where } h = \frac{x-a}{N} \tag{1.18}$$

$${}_{GL}D^{\alpha}_{x,b} u(x) = \lim_{h\to 0} h^{-\alpha} \sum_{j=0}^{N} (-1)^j \binom{\alpha}{j} u(x + jh), \text{ where } h = \frac{b-x}{N}. \tag{1.19}$$

Here we make use of the general binomial coefficient where $\alpha \in \mathbb{R}$ and $k \in \mathbb{N}$ which is defined as,

$$\binom{\alpha}{k} = \frac{\alpha(\alpha-1)\dots(\alpha-k-1)}{k!} = \frac{\Gamma(\alpha-1)}{\Gamma(k+1)\Gamma(\alpha-k-1)}. \tag{1.20}$$

In general, the Riemann–Liouville and Grünwald-Letnikov definitions for fractional derivatives are not equivalent, although, for smooth functions, $u(x) \in C^m[a,b]$, the left and right Riemann-Liouville derivatives equal the left and right Grünwald Letnikov derivatives. All functions we consider in this thesis will have this property.

Now, we can consider the general definition for the fractional Laplacian in one dimension for $\alpha \in (0,2)$ [11].

Definition 4. For a smooth function $u(x)$ and for $\alpha \in (0,2)$, the fractional Laplacian is defined by;

$$(-\Delta)^{\alpha/2} u(x) = \left(\int_{\mathbb{R}} \frac{1-\cos(\zeta)}{|\zeta|^{n+\alpha}} d\zeta\right)^{-1} P.V. \int_{\mathbb{R}} \frac{u(x)-u(y)}{|x-y|^{n+\alpha}} dy, \tag{1.21}$$

$$= -\frac{1}{2}\left(\int_{\mathbb{R}} \frac{1-\cos(\zeta)}{|\zeta|^{n+\alpha}} d\zeta\right)^{-1} \int_{\mathbb{R}} \frac{u(x+y)-2u(x)+u(x-y)}{|y|^{n+\alpha}} dy. \tag{1.22}$$

The singular integral can make approximating the fractional Laplacian difficult. Alternatively, we can consider the Riesz derivative which can be defined as a weighted sum of left and right Riemann-Liouville derivatives for one dimensional functions [34].

Definition 5. The Riesz derivative of order $\alpha > 0,\ \alpha \neq 2k+1,\ k \in \mathbb{Z}^{\geq 0}$, for a function $u(x)$, where $x \in [a,b]$ is given by,

$${}_{RZ}D_x^{\alpha} u(x) = -\frac{1}{2\cos(\alpha\pi/2)}\left({}_{RL}D_{a,x}^{\alpha} u(x) + {}_{RL}D_{x,b}^{\alpha} u(x)\right). \tag{1.23}$$

In one dimension we can use the result that, for non-integer derivatives, the fractional Laplacian and Riesz derivatives coincide (for the proof refer to [43]).

Lemma 1. *For a smooth function $u(x)$, $x \in \mathbb{R}$, and for $\alpha \in (0,1) \cup (1,2)$ we have,*

$$-(-\Delta)^{\alpha/2} u(x) = {}_{RZ}D_x^{\alpha} u(x). \tag{1.24}$$

It is important to note that although the Riesz derivative is often taken to be the definition of the fractional derivative in one dimension, it doesn't hold for when α is an odd integer as it is undefined. The Riesz definition of the fractional Laplacian is much easier to approximate as we need only approximate Caputo or Grünwald-Letnikov derivatives under the conditions stated above (sufficient homogeneous boundary conditions and smooth functions). We can't, however, treat the special case of $\alpha = 1$ using this definition. In this special case we can find $-(-\Delta)^{1/2}$ by considering the Hilbert transform of the first derivative [14]. We do find that in the limit of $\alpha \to 2$ the Riesz derivative approaches the ordinary Laplacian [46].

Lemma 2. *For a suitably smooth function $u(x),\ x \in \mathbb{R}$,*

$$\lim_{\alpha \to 2}\ {}_{RZ}D_x^{\alpha} u(x) = u''(x). \tag{1.25}$$

We will check that this property holds when running numerical experiments in Chapter 5. It is important to note that similar relations don't hold for the other relevant integer derivatives [46].

Lemma 3. *For a suitably smooth function $u(x),\ x \in \mathbb{R}$,*

$$\lim_{\alpha \downarrow 0}\ {}_{RZ}D_x^{\alpha} u(x) = -u(x), \tag{1.26}$$

$$\lim_{\alpha \to 1}\ {}_{RZ}D_x^{\alpha} u(x) \neq \pm u'(x). \tag{1.27}$$

1.3.2 Two dimensions

In two dimensions the Riesz derivative and singular integral definitions of the fractional Laplacian given in definitions 6 and 7 below still agree for $\alpha \in (0,2)\backslash\{1\}$, but the definitions become more complex [43]. We also have another definition known as the Riesz potential given in definition 8 which is often used in applications for example to solve space-fractional convection diffusion equations [43] [4].

Definition 6. For a suitably smooth function $u(\underline{x})$, $\underline{x} \in \mathbb{R}^2$ and $\alpha \in (0,2)$, the fractional Laplacian is defined as,

$$
\begin{aligned}
(-\Delta)^{\alpha/2} u(\underline{x}) &= \left(\int_{\mathbb{R}^2} \frac{1-\cos(\zeta_1)}{|\underline{\zeta}|^{n+\alpha}} d\underline{\zeta}\right)^{-1} P.V. \int_{\mathbb{R}^2} \frac{u(\underline{x})-u(\underline{y})}{|\underline{x}-\underline{y}|^{n+\alpha}} d\underline{y}, &(1.28)\\
&= -\frac{1}{2}\left(\int_{\mathbb{R}^2} \frac{1-\cos(\zeta_1)}{|\underline{\zeta}|^{n+\alpha}} d\underline{\zeta}\right)^{-1} \int_{\mathbb{R}^2} \frac{u(\underline{x}+\underline{y})-2u(\underline{x})+u(\underline{x}-\underline{y})}{|\underline{y}|^{n+\alpha}} d\underline{y}. &(1.29)
\end{aligned}
$$

Definition 7. For a suitably smooth function $u(\underline{x})$, $\underline{x} \in \mathbb{R}^2$ and $\alpha \in (0,l)$, the Riesz derivative is defined as,

$$
{}_{RZ}D^{\alpha}_{\underline{x}} u(\underline{x}) = \frac{1}{d_{2,l}(\alpha)} \int_{\mathbb{R}^2} \frac{\left(\Delta^l_y u\right)(\underline{x})}{|\underline{y}|^{2+\alpha}} d\underline{y}, \qquad (1.30)
$$

where $l \in \mathbb{Z}$ such that $l > \alpha$ and,

$$
\left(\Delta^l_y u\right)(\underline{x}) = \sum_{k=0}^{l} (-1)^k \binom{l}{k} u\left(\underline{x} + \left(\frac{l}{2}-k\right)\underline{h}\right), \qquad (1.31)
$$

which represents centered differences with step size $\underline{h}$. The coefficients are given by,

$$
d_{2,l} = \frac{1}{2^{\alpha}\sin(\pi\alpha/2)} \left(\frac{\pi}{\Gamma(1+\alpha/2)}\right)^2 \sum_{k=0}^{l} (-1)^{k-1} \binom{l}{k} k^{\alpha}, \qquad (1.32)
$$

Definition 8. For a suitably smooth function $u(\underline{x})$, $\underline{x} = (x,y) \in [a,b] \times [a,c]$, $a,b,c \in \mathbb{R}$ and $\alpha \in (0,2)\backslash\{1\}$, the Riesz potential is defined as,

$$
\begin{aligned}
-(-\Delta)^{\alpha/2} u(\underline{x}) &= \sum_{j=1}^{2} \frac{\partial^{\alpha} u(\underline{x})}{\partial |x_j|^{\alpha}}, &(1.33)\\
&= \frac{\partial^{\alpha} u(\underline{x})}{\partial |x|^{\alpha}} + \frac{\partial^{\alpha} u(\underline{x})}{\partial |y|^{\alpha}}, &(1.34)\\
&= -\frac{1}{2\cos(\pi\alpha/2)} \left({}_{RL}D^{\alpha}_{a,x} u(\underline{x}) + {}_{RL}D^{\alpha}_{x,b} u(\underline{x}) + {}_{RL}D^{\alpha}_{a,y} u(\underline{x}) + {}_{RL}D^{\alpha}_{y,c} u(\underline{x}) \right). &(1.35)
\end{aligned}
$$

As in one dimension, we can impose an appropriate number of homogeneous boundary conditions such that the Riemann–Liouville and Caputo derivative coincide. Thus we can replace all of the Riemann–Liouville derivatives with Caputo derivatives in definition 8.

The Riesz potential is advantageous for two reasons: it is easier to approximate as we don't need to treat the singular integral, and often it matches applications very well [4]. Additionally, in both one and two dimensions, the Riesz potential preserves properties of Fourier transforms while the fractional Laplacian does not, which is desirable for some applications [43].

1.4 Boundary conditions for PDEs involving the space-fractional Laplacian

Boundary conditions have a large effect on the solutions to any PDE, but due to the non-local nature of the definitions for the fractional Laplacian, boundary conditions can have an even more significant effect [6]. As discussed, taking the fractional Laplacian of order $\alpha < 2$ results in super diffusive behaviour, that is, we expect the solution to diffuse faster. This can be seen in [46] Figure 2.4.1 which is recreated in section 2.1 (Figure 2.4) using a method by Duo et al. [18] outlined in the same section. In this case an infinite domain is assumed, so we do not see the effect of boundary conditions. The smaller the value of α, the faster the solution diffuses.

Throughout this thesis we impose homogeneous boundary conditions such that Grünwald-Letnikov, Riemann-Liouville and Caputo derivatives are equivalent. This choice can have an impact on the qualitative behaviour of the solution if the solution lies near enough the boundaries of the domain, as now we will see that for smaller choices of α, the solutions will diffuse less quickly [16]. To show this, we can solve the space-fractional heat equation on a uniform grid with 100 grid points using the same method as Duo et al. [18], this time with homogeneous boundary conditions and a smaller domain. The results are shown in Figure 1.2. It is clear that, as we increase α, the solution diffuses more quickly.

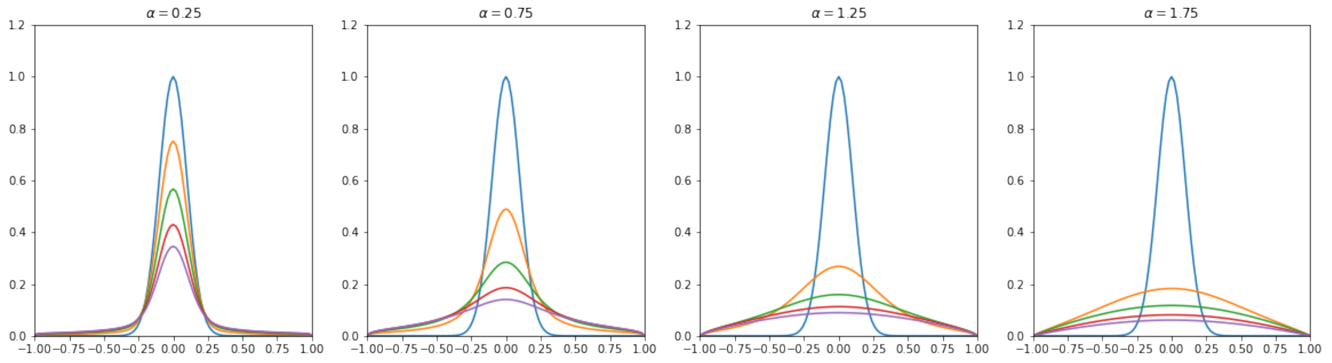


Figure 1.2: Solutions to the space-fractional heat equation generated using the method by Duo et al. [18] for $\alpha = 0.25,\ 0.75,\ 1.25,\ 1.75$. Solutions are shown at times $t = 0,\ 0.175,\ 0.35,\ 0.525,\ 0.7$ in blue, orange, green, red and purple respectively.

1.5 Numerical approximations of the ordinary Laplacian

It will be important to verify that the methods presented in this thesis to approximate the fractional Laplacian approach approximations for the ordinary Laplacian in the limit $\alpha \to 2$. In this section we will introduce the standard methods we will use for approximating the ordinary Laplacian in one and two dimensions.

1.5.1 One dimension

We will use the standard three-point central approximation for the Laplacian in one dimension,

$$\left.\frac{\partial^2 u(x)}{\partial x^2}\right|_{x=x_n} \approx \frac{u(x_{n+1}) - 2u(x_n) + u(x_{n-2})}{h^2}, \tag{1.36}$$

where $x \in [a, b]$ and there are $N + 2$ spatial grid points $x_0 = a,\ x_1,\ ...,\ x_N,\ x_{N+1}$. h is the step size on the uniform grid, so $h = (b - a)/(N + 1)$. We can write this in vector and matrix notation as,

$$U_{xx} = \frac{1}{h^2} D_2 U, \tag{1.37}$$

where $U = [u(x_0), ..., u(x_{N+1})]^T$ and D_2 is the $(N + 1) \times (N + 1)$ matrix given by,

$$\begin{pmatrix} -2 & 1 & 0 & \cdots & \cdots & \cdots & 0 \\ 1 & -2 & 1 & \ddots & & & \vdots \\ 0 & 1 & -2 & \ddots & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & -2 & 1 & 0 \\ \vdots & & & \ddots & 1 & -2 & 1 \\ 0 & \cdots & \cdots & \cdots & 0 & 1 & -2 \end{pmatrix}. \tag{1.38}$$

On a non-uniform grid, using the method of undetermined coefficients and a three-point central stencil, we find that that the Laplacian can be approximated by,

$$\left.\frac{\partial^2 u(x)}{\partial x^2}\right|_{x=x_n} \approx \frac{2u(x_{n+1})}{\Delta X_n(\Delta X_n + \Delta X_{n-1})} - \frac{u(x_n)}{\Delta X_n \Delta X_{n-1}} + \frac{u(x_{n-1})}{\Delta X_{n-1}(\Delta X_n + \Delta X_{n-1})}, \tag{1.39}$$

where $\Delta X_k = x_{k+1} - x_k$. We find the accuracy and efficiency of the central difference method sufficient for use within this thesis where it is used to make a direct comparison between solutions to space-fractional PDEs in the limit $\alpha \to 2$ where the fractional Laplacian approaches the ordinary Laplacian.

1.5.2 Two dimensions

When discretising the system in two dimensions, the variables $x,\ y,\ u$, are assigned two indexes i and j where the first references the x coordinate and the second the y coordinate. We will work on a domain where $x \in [a, b],\ y \in [a, c]$ and there are $N + 2$ grid points in the x and y directions. We choose to use a central in space five points finite difference scheme found via the method of undetermined coefficients to approximate the ordinary Laplacian on a non-uniform grid as follows,

$$\begin{aligned} \Delta u|_{i,j} = & \frac{2u_{i+1,j}}{(x_{i+1,j} - x_{i,j})\left((x_{i,j} - x_{i-1,j}) + (x_{i+1,j} - x_{i,j})\right)} - \frac{2u_{i,j}}{(x_{i,j} - x_{i-1,j})(x_{i+1,j} - x_{i,j})} \\ & + \frac{2u_{i-1,j}}{(x_{i,j} - x_{i-1,j})\left((x_{i+1,j} - x_{i,j}) + (x_{i,j} - x_{i-1,j})\right)} \\ & + \frac{2u_{i,j+1}}{(y_{i,j+1} - y_{i,j})\left((y_{i,j} - y_{i,j-1}) + (y_{i,j+1} - y_{i,j})\right)} - \frac{2u_{i,j}}{(y_{i,j} - y_{i,j-1})(y_{i,j+1} - y_{i,j})} \\ & + \frac{2u_{i,j-1}}{(y_{i,j} - y_{i,j-1})\left((y_{i,j+1} - y_{i,j}) + (y_{i,j} - y_{i,j-1})\right)}. \end{aligned} \tag{1.40}$$

Chapter 2

Finite Difference Methods for the Fractional Laplacian in One Dimension

There are many ways of numerically solving PDEs involving the fractional Laplacian via a finite difference scheme on fixed uniform grids. We consider three different methods with a goal of choosing one of these to extend to approximate the fractional Laplacian non-uniform grids. First, we discuss a method by Duo et al. "A novel and accurate finite difference method for the fractional Laplacian" [18], where we start with the singular integral definition for the fractional Laplacian. The method is based on the weighted trapezoidal rule where, by introduction of a so-called splitting parameter, the fractional Laplacian operator can be approximated via multiplication with a matrix. In the limit $\alpha \to 2$, this matrix reduces to the D_2 matrix (1.38) which we can use to approximate the ordinary Laplacian via central differences. This method is appealing because it holds for $\alpha \in (0, 2)$, including $\alpha = 1$. Another advantage to this model is at the point of publication (2018) it is stated to be the most accurate to date [18] [25].

Next we will discuss the Grünwald-Letnikov formula defined in equations (1.18) and (1.19) involving limits for left and right fractional derivatives which can be almost directly applied to approximate solutions on a finite difference grid [34]. For smooth functions, these derivatives are equivalent to the Riemann-Liouville derivatives and thus the fractional Laplacian can be defined via the Riesz derivative and written using equations (1.23) and (1.24). These simple methods for approximation differ only very slightly for $\alpha \in (0, 1)$ and $\alpha \in (1, 2)$, but the main drawback is that we cannot approximate the $\alpha = 1$ case via this method as the Riesz derivative isn't defined in this case.

Finally we will consider the definitions for left and right Caputo derivatives which can, upon combining according to equation (1.24), define the fractional Laplacian. As with the Grünwald-Letnikov derivatives, due to the definition for the Riesz derivative, we cannot treat the case where $\alpha = 1$. The cases $\alpha \in (0, 1)$ and $\alpha \in (1, 2)$ also need to be treated separately as the Caputo derivatives include integer derivatives 1 and 2 respectively for each of these cases, and these derivatives have different approximations. The known methods to approximate left-fractional derivatives on non-uniform grids are known as the L1 and L2 methods respectively [34]. Despite us considering two separate methods for these two cases, we will see that generalising these L1 and L2 methods to non-uniform grids proves to be the most viable option, and thus from here we will develop the L1(NU) and L2(NU) methods.

2.1 Singular integral definition approach

In this section we will apply the method proposed by Duo et al. [18] which is second order accurate and, in the limit $\alpha \to 2$, tends to the central difference method we saw for the ordinary Laplacian.

The main difficulty when solving the space-fractional heat equation is the singularity in the definition of the fractional derivative. Many numerical methods treat this by removing the singularity as a small interval, to be treated separately from the tails of the integral [18] [30]. We take the following definition for the fractional derivative in one dimension,

$$(-\Delta)^{\alpha/2}u(\mathbf{x}) = c_{1,\alpha}\ \text{P.V.}\int_{\mathbb{R}} \frac{u(\mathbf{x})-u(\mathbf{y})}{|\mathbf{x}-\mathbf{y}|^{n+\alpha}}d\mathbf{y}, \qquad \text{for } \alpha\in(0,2), \quad c_{1,\alpha} = \frac{2^{\alpha-1}\alpha\Gamma\left(\frac{\alpha}{2}\right)}{\sqrt{\pi}\Gamma\left(1-\frac{\alpha}{2}\right)}. \tag{2.1}$$

Note that P.V. denotes a principal value integral and this definition is equivalent to Definition 4. If we let $\xi = |x-y|$ this can be rewritten as [19],

$$(-\Delta)^{\alpha/2}u(x) = -c_{1,\alpha}\int_0^\infty \frac{u(x-\xi)-2u(x)+u(x+\xi)}{\xi^{1+\alpha}}d\xi. \tag{2.2}$$

If we consider an interval $[-L/2, L/2] = [l, l]$ where $u(x,t) = 0$ outside this interval, then we can split this integral into two parts and solve one integral exactly. Here we use that for $\xi > L$, $u(x\pm\xi) = 0$.

$$(-\Delta)^{\alpha/2}u(x) = -c_{1,\alpha}\left(\int_0^L \frac{u(x-\xi)-2u(x)+u(x+\xi)}{\xi^{1+\alpha}}d\xi + \int_L^\infty \frac{u(x-\xi)-2u(x)+u(x+\xi)}{\xi^{1+\alpha}}d\xi\right), \tag{2.3}$$

$$= -c_{1,\alpha}\left(\int_0^L \frac{u(x-\xi)-2u(x)+u(x+\xi)}{\xi^{1+\alpha}}d\xi - 2u(x)\int_L^\infty \frac{1}{\xi^{1+\alpha}}d\xi\right), \tag{2.4}$$

$$= -c_{1,\alpha}\left(\int_0^L \frac{u(x-\xi)-2u(x)+u(x+\xi)}{\xi^{1+\alpha}}d\xi - \frac{2}{\alpha L^\alpha}u(x)\right). \tag{2.5}$$

The remaining integral is treated numerically. We choose to have $K+1$ spatial grid points, resulting in a stepsize $h = 2l/K = L/K$. We can then denote spatial grid points by $\xi_k = kh$, $0\le k\le K$. Then a so called splitting parameter γ is chosen such that $\gamma\in(0,2]$. There is a detailed analysis on how to best choose this splitting parameter in [18] but for the purposes of this thesis we will assign $\gamma = \alpha/2+1$.

$$\int_0^L \frac{u(x-\xi)-2u(x)+u(x+\xi)}{\xi^{1+\alpha}}d\xi = \int_0^L \frac{u(x-\xi)-2u(x)+u(x+\xi)}{\xi^\gamma}\xi^{\gamma-(1+\alpha)}d\xi, \tag{2.6}$$

$$= \int_0^L \phi_\gamma(x,\xi)\xi^{\gamma-(1+\alpha)}d\xi, \tag{2.7}$$

$$= \sum_{k=1}^K \int_{\xi_{k-1}}^{\xi_k} \phi_\gamma(x,\xi)\xi^{\gamma-(1+\alpha)}d\xi. \tag{2.8}$$

The weighted trapezoidal rule is then employed to solve these integrals over the intervals (ξ_{k-1},ξ_k), and the calculation of for $k=1$ is treated as follows (note the dependence on the choice of γ).

$$\int_{\xi_{k-1}}^{\xi_k} \phi_\gamma(x,\xi)\xi^{\gamma-(1+\alpha)}d\xi \approx \frac{1}{2}\left(\phi_\gamma(x,\xi_{k-1})+\phi_\gamma(x,\xi_k)\right)\int_{\xi_{k-1}}^{\xi_k}\xi^{\gamma-(1+\alpha)}d\xi, \tag{2.9}$$

$$= \frac{1}{2(\gamma-\alpha)}\left(\xi_k^{\gamma-\alpha}-\xi_{k-1}^{\gamma-\alpha}\right)\left(\phi_\gamma(x,\xi_{k-1})+\phi_\gamma(x,\xi_k)\right), \tag{2.10}$$

$$\int_{\xi_0}^{\xi_1} \phi_\gamma(x,\xi)\xi^{\gamma-(1+\alpha)}d\xi \approx \frac{\kappa_\gamma}{2(\gamma-\alpha)}\xi_1^{\gamma-\alpha}\phi_\gamma(x,\xi_1), \tag{2.11}$$

where $\kappa_\gamma = 2$ if $\gamma = 2$, or 1 otherwise. The complete system can be written as,

$$(-\Delta)_{h,\gamma}^{\alpha/2}u(x) = -\frac{c_{1,\alpha}}{2(\gamma-\alpha)}\left[\kappa_\gamma\xi_1^{\gamma-\alpha}\phi_\gamma(x,\xi_1) + \sum_{k=2}^K\left(\xi_k^{\gamma-\alpha}-\xi_{k-1}^{\gamma-\alpha}\right)\left(\phi_\gamma(x,\xi_{k-1})+\phi_\gamma(x,\xi_k)\right)\right] + \frac{2c_{1,\alpha}}{\alpha L^\alpha}u(x). \tag{2.12}$$

Then we can rewrite the system in matrix form, $(-\Delta)^{\alpha/2}_{h,\gamma}U = AU$, where $U = \big(u(x_1), ..., u(x_{K-1}))\big)^T$, and A is a $(K-1)\times(K-1)$ matrix given by,

$$A_{ij} = C^h_{\alpha,\gamma}\begin{cases} \sum_{k=2}^{K-1}\frac{(k+1)^v-(k-1)^v}{k^\gamma} + \frac{K^v-(K-1)^v}{K\gamma} + \left(2^v+\kappa_\gamma-1\right) + \frac{2v}{\alpha K^\alpha}, & j = i, \\ -\frac{(|j-i|+1)^v-(|j-i|-1)^v}{2|j-i|^\gamma}, & j \neq i, i\pm 1, \\ -\frac{1}{2}\left(2^v+\kappa_\gamma-1\right), & j = i\pm 1, \end{cases} \tag{2.13}$$

according to [18]. Note that using $C^h_{\alpha,\gamma} = \frac{c_{1,\alpha}}{vh^\alpha}$, and $v = \gamma-\alpha$, is simple to find,

$$A_{ii} = C^h_{\alpha,\gamma}\Bigg[\frac{2v}{\alpha k^\alpha} + k_\gamma + \frac{L^\gamma}{K^\alpha(K-1)^\gamma h^\gamma} + \frac{1}{K^\alpha} - \frac{h^\alpha}{(K-1)^\alpha h^\alpha} - \frac{h^\alpha(K-1)^v h^v}{L^\gamma} + \sum_{k=2}^{K-1}\left(\frac{k^v-(k-1)^v}{(k-1)^\gamma} + \frac{k^v-(k-1)^v}{k^\gamma}\right)\Bigg], \tag{2.14}$$

which upon inspection does agree with the expression in [18] and numerical experiments produce the same results. We can implement the matrix in Python to first check if Figure 3 in [18] could be generated. This is shown in Figure 2.1 where we take $u(x) = \exp(-x^2)$, and use the system $(-\Delta)^{\alpha/2}_{h,\gamma}U = AU$ to plot the derivatives of this function for different choices of α. The results here match those of the paper perfectly, and we should note that, as seen in Figure 2.2, in the limit $\alpha \to 2$, the fractional derivative approaches the ordinary second derivative. This is consistent with the convergence property in equation (1.25).

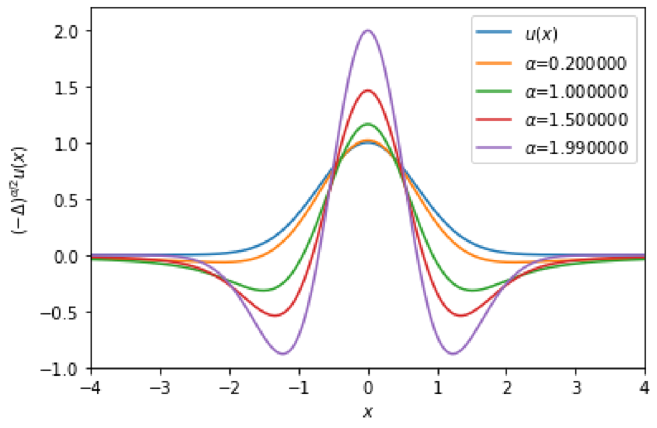


Figure 2.1: Fractional derivatives for $u(x) = e^{-x^2}$, $L = 20$, $K = 640$, with different values of α. This matches Figure 3 in [18].

Now that the implementation of this method has been shown to match the results from the paper, we can apply this method to solve the space-fractional heat equation,

$$\frac{\partial u(x,t)}{\partial t} = -\left(-\Delta\right)^{\alpha/2}u(x,t) \qquad u(x,0) = \exp(-x^{20}),\ u(-L,t) = u(L,t) = 0, \tag{2.15}$$

We use Euler backward for the time derivative and the method of lines,

$$\frac{dU}{dt} = -(-\Delta)^{\alpha/2}_{h,\gamma}U = -AU, \tag{2.16}$$

$$\implies \frac{U_n - U_{n-1}}{\Delta t} = -AU_n. \tag{2.17}$$

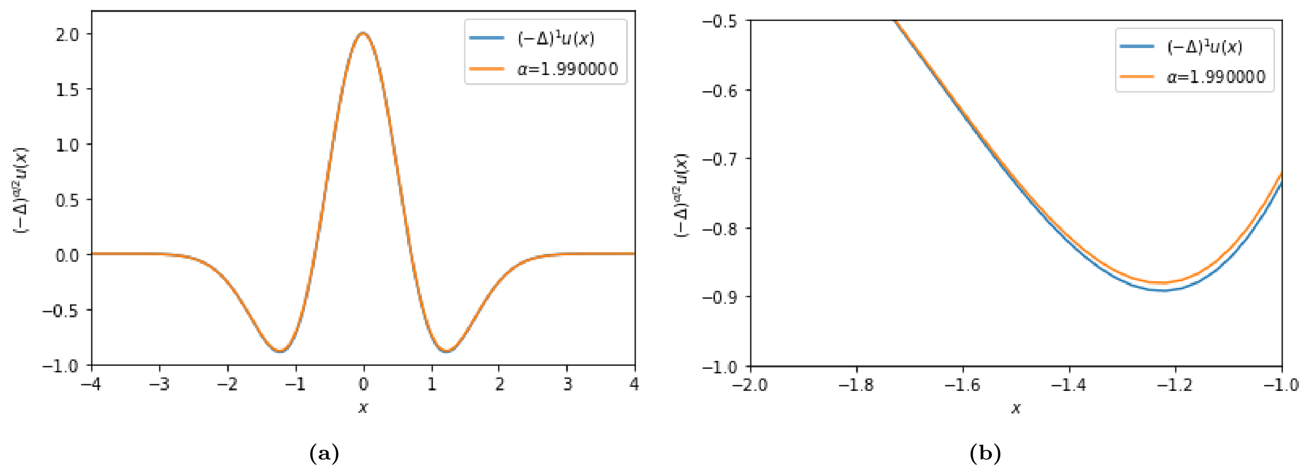


Figure 2.2: In the limit $\alpha \to 2$ the fractional derivative approaches the ordinary Laplacian.

Thus by rearranging we find,

$$U_n = (I + A\Delta t)^{-1} U_{n-1}. \tag{2.18}$$

Figure 2.4 displays some examples of these numerical experiments where method (2.13) is used to solve the space-fractional heat equation with $L = 10$ for different values of α. Here we can observe that the way in which the solution profile "diffuses" is different depending on the choice of α. For smaller values of α, we observe thin tails and the base of the solution profile is less spread out, for larger choices of α we see fatter tails and the bell-shaped curve of the solution profile is more thin at early times. This figure reproduces the behaviours seen in [46] Figure 2.4.1 pp. 71. Upon comparison with the solution to the ordinary heat equation, it can be shown that the solution shown in the top left of Figure 2.4 where $\alpha = 1.99$ is almost equivalent to the solution to the ordinary heat equation (see Figure 2.3).

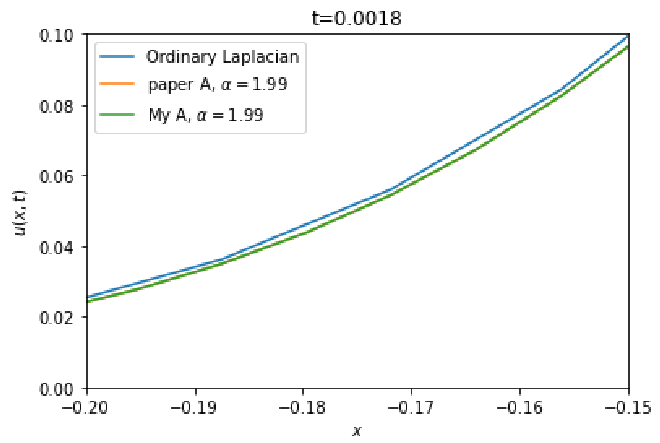


Figure 2.3: Method (2.13) applied to 2.15, L=20, solution shown at $t = 0.0018$. We can see that the solutions for the expression for matrix A in [18] labelled "paper A" gives identical results to the expression given in (2.14) labelled "My A" (orange line is under blue line). It is also clear that the solution for $\alpha = 1.99$ approaches the solution to the ordinary heat equation where $\alpha = 2$.

Although this method produces good results to approximating space-fractional derivatives, and uses only one method for $\alpha \in (0, 2)$ including the special case of $\alpha = 1$, we choose not to extend this method to work on non-uniform grids as even when working on a uniform grid, the computations leading to the final matrix in the method were already lengthy. The complexity of these relations would only increase on a

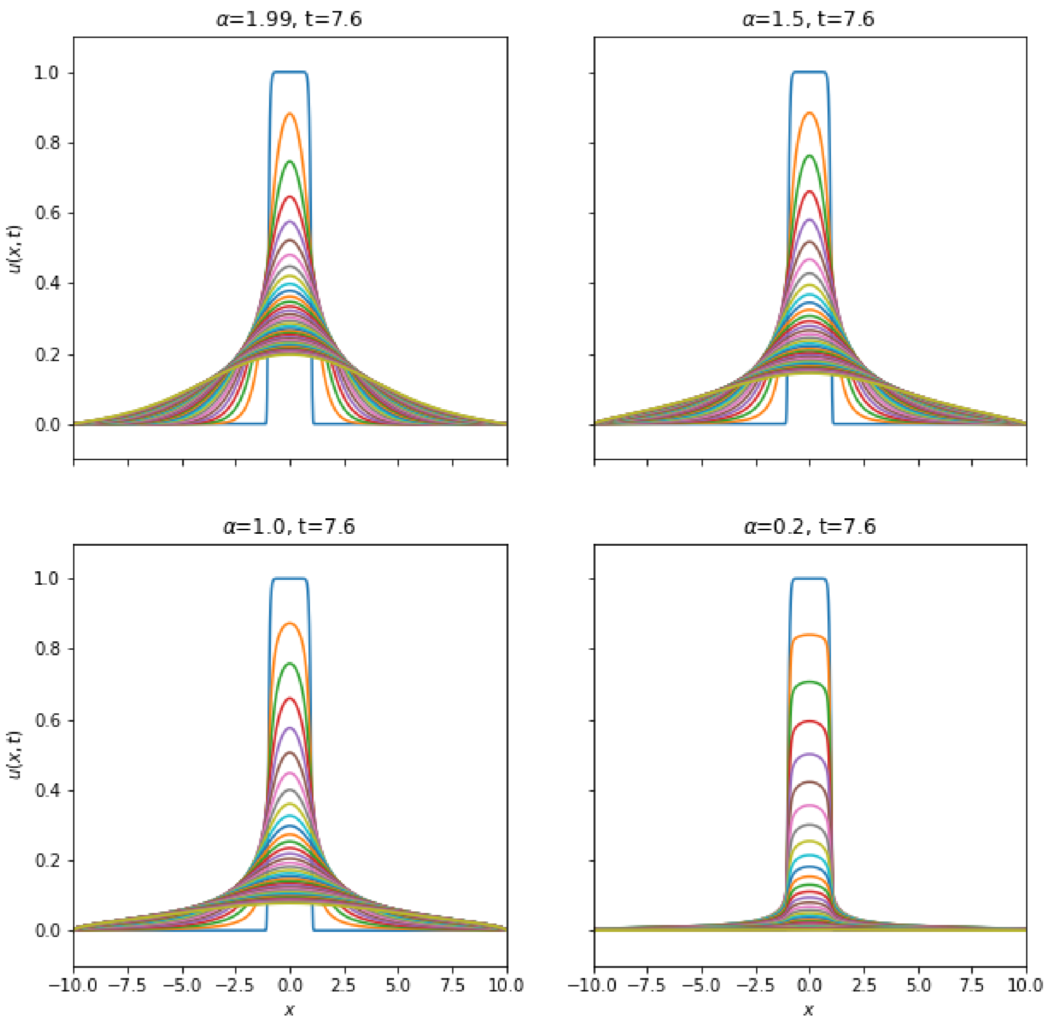


Figure 2.4: Solution profiles to problem (2.15) with Gaussian initial condition $u(x,0) = \exp(-20x^2)$, $L = 20$, for difference values of α. solutions $u(x,t)$ are shown every 0.2 seconds until $t = 7.6s$. Approximations to fractional Laplacian found using method from section (2.1).

non-uniform grid resulting in a very complex matrix which the system would have to compute at each stage of the time integration when solving a PDE.

2.2 Grünwald-Letnikov definition approach

In the same way that equation (1.16) can be extended to derivatives of fractional order to obtain the left Grünwald-Letnikov fractional derivative given in equation (1.18), we can extend equation (1.17) to obtain an approximation to the left Grünwald-Letnikov fractional derivative on a uniform finite difference grid with step size h [34] [17].

Theorem 2 (Grünwald-Letnikov derivatives on uniform grids). *For a suitably smooth function $u(x)$, $x \in [a,b]$, we can approximate the left and right Grünwald-Letnikov derivatives on uniform grid with $N+2$ grid points $x_0 = a$, $x_{N+1} = b$, $h = (b-a)/(N+1)$, by [34]:*

For $\alpha \in (0,1)$ the standard Grünwald-Letnikov formulas are given by,

$$ {}_{GL}D^{\alpha}_{a,x}u(x)\Big|_{x_n} \approx \frac{1}{h^{\alpha}} \sum_{j=0}^{n} (-1)^j \binom{\alpha}{j} u(x_{n-j}), \tag{2.19}$$

$$ {}_{GL}D^{\alpha}_{x,b}u(x)\Big|_{x_n} \approx \frac{1}{h^{\alpha}} \sum_{j=0}^{N-n} (-1)^j \binom{\alpha}{j} u(x_{n+j}). \tag{2.20}$$

For $\alpha \in (1,2)$ the shifted Grünwald-Letnikov formulas are given by,

$$ {}_{GL}D^{\alpha}_{a,x}u(x)\Big|_{x_n} \approx \frac{1}{h^{\alpha}} \sum_{j=0}^{n+p} (-1)^j \binom{\alpha}{j} u(x_{n-j+p}), \tag{2.21}$$

$$ {}_{GL}D^{\alpha}_{x,b}u(x)\Big|_{x_n} \approx \frac{1}{h^{\alpha}} \sum_{j=0}^{N-n-p} (-1)^j \binom{\alpha}{j} u(x_{n+j-p}). \tag{2.22}$$

where the p-shifts, $p \in \mathbb{N}$, are included to help with stability as the standard Grünwald-Letnikov formulas can lead to unstable numerical schemes for $\alpha \in (1,2)$. p is often chosen to be equal to one for $\alpha \in (1,2)$ [34].

Thus under the assumption that we have a smooth function, the fractional Laplacian can be approximated by,

$$\begin{aligned} -\left(-\Delta\right)^{\alpha/2} u(x) &= -\frac{1}{2\cos(\alpha\pi/2)} \left({}_{RL}D^{\alpha}_{a,x}u + {}_{RL}D^{\alpha}_{x,b}u \right), &(2.23)\\ &= -\frac{1}{2\cos(\alpha\pi/2)} \left({}_{GL}D^{\alpha}_{a,x}u + {}_{GL}D^{\alpha}_{x,b}u \right). &(2.24) \end{aligned}$$

So for $\alpha \in (0,1)$,

$$ -\left(-\Delta\right)^{\alpha/2} u(x_n) \approx -\frac{1}{2h^{\alpha}\cos(\alpha\pi/2)} \left(\sum_{j=0}^{n} (-1)^j \binom{\alpha}{j} u(x_{n-j}) + \sum_{j=0}^{N-n} (-1)^j \binom{\alpha}{j} u(x_{n+j}) \right). \tag{2.25}$$

And for $\alpha \in (1,2)$,

$$ -\left(-\Delta\right)^{\alpha/2} u(x_n) \approx -\frac{1}{2h^{\alpha}\cos(\alpha\pi/2)} \left(\sum_{j=0}^{n+p} (-1)^j \binom{\alpha}{j} u(x_{n-j+p}) + \sum_{j=0}^{N-n-p} (-1)^j \binom{\alpha}{j} u(x_{n+j-p}) \right). \tag{2.26}$$

We don't consider the special case where $\alpha = 1$ as we can't express the fractional Laplacian via Grünwald-Letnikov derivatives in this case.

These formulas inspire an approach for finding an approximation to Grünwald-Letnikov derivatives on non-uniform grids. The aim was to first produce a general formula similar to equation (1.17) which would approximate any integer derivative on a non-uniform grid using backward differences. From here there were two possible approaches;

1. Try to generalise this formula to non-integer derivatives giving an approximation for the left Grünwald-Letnikov derivative on a non-uniform grid. The approximation for the right Grünwald-Letnikov derivative could be found in a similar way.

2. Use the definition for fractional derivatives in terms of an infinite sum of integer order derivatives to find an approximation for the left Grünwald-Letnikov derivative. The formula for the fractional derivative in terms of integer derivatives on a uniform grid is given by [24],

$$ {}_{GL}D^{\alpha}_{a,x}u(x) = \lim_{\substack{h\to 0\\ N\to\infty}} \sum_{k=0}^{N-1} \frac{(-1)^{N-k+1}(N-k)h^{k-\alpha}}{\alpha-k}\binom{N}{k}\binom{\alpha}{N}\frac{d^k}{dx^k}u(x), \tag{2.27}$$

where h is the grid spacing and N the number of grid points. This expression would need to be slightly adapted for a non-uniform grid. Again the right Grünwald-Letnikov derivative could be approximated in a similar way.

A general expression for approximating integer derivatives on a non-uniform grid was found by a repeated application of backward differences as follows:

Theorem 3. *On a non-uniform grid where $\Delta X_i = x_i - x_{i-1}$, the n^{th} derivative of a function $u(x)$, evaluated at point x_i, $u^n|_{x_i}$, can be approximated by,*

$$\sum_{k=0}^{n}(-1)^k u_{i-k}\left\{\sum_{\substack{i_1=\ldots=i_{n-(k+1)}=0,\\ i_{n-k}\geq 0,\\ i_{n-(k-1)}\geq 1,\\ \vdots\\ i_{n-1}\geq 1,\\ i_{n-1}+\ldots+i_{n-k}\leq n-1}} \left(\frac{1}{\Delta X_{i-(n-1)}}\right)^{i_1}\left(\frac{1}{\Delta X_{i-(n-2)}}\right)^{i_2}\cdots\left(\frac{1}{\Delta X_{i-1}}\right)^{i_{n-1}}\left(\frac{1}{\Delta X_i}\right)^{n-i_1-\ldots-i_{n-1}}\right\}, \tag{2.28}$$

where we make the choice to repeatedly apply a backwards difference approximation.

Proof. We use a proof by induction. First we can show that the formula holds for $n=1$ and $n=2$. For $n=1$ we have,

$$u'\big|_{x_i} = \frac{u_i - u_{i-1}}{\Delta X_i}. \tag{2.29}$$

And (2.28) gives us,

$$u'\big|_{x_i} = u_i\frac{1}{\Delta X_i} - u_{i-1}\frac{1}{\Delta X_i} = \frac{u_i - u_{i-1}}{\Delta X_i}. \tag{2.30}$$

For $n=2$ we have,

$$u''\big|_{x_i} = \frac{u'_i - u'_{i-1}}{\Delta X_i} = \frac{1}{\Delta X_i}\left(\frac{u_i-u_{i-1}}{\Delta X_i} - \frac{u_{i-1}-u_{i-2}}{\Delta X_{i-1}}\right), \tag{2.31}$$

$$= \frac{u_i-u_{i-1}}{(\Delta X_i)^2} - \frac{u_{i-1}-u_{i-2}}{\Delta X_i \Delta X_{i-1}}. \tag{2.32}$$

And (2.28) gives us,

$$u''\big|_{x_i} = u_i\left(\frac{1}{\Delta X_i}\right)^2 - u_{i-1}\left(\left(\frac{1}{\Delta X_i}\right)^2 + \frac{1}{\Delta X_{i-1}}\frac{1}{\Delta X_i}\right) + u_{i-2}\frac{1}{\Delta X_{i-1}}\frac{1}{\Delta X_i}. \tag{2.33}$$

So (2.28) holds true for the base cases. Now let $u^{(q)}\Big|_{x_i}$ be approximated by (2.28) where $n = q$ for some integer $q \geq 2$, and consider $u^{(q+1)}\Big|_{x_i}$.

$$u^{(q+1)}\Big|_{x_i} = \frac{d}{dx} u^{(q)}\Big|_{x_i}, \tag{2.34}$$

$$= \frac{d}{dx} \sum_{k=0}^{q} (-1)^k u(x_{i-k}) \mathcal{B}_q \tag{2.35}$$

where $\mathcal{B}_q$ is given by,

$$\mathcal{B}_q = \left\{ \sum_{\substack{i_1 = \ldots = i_{q-(k+1)} = 0, \\ i_{q-k} \geq 0, \\ i_{q-(k-1)} \geq 1, \\ \vdots \\ i_{q-1} \geq 1, \\ i_{q-1} + \ldots + i_{q-k} \leq q-1}} \left(\frac{1}{\Delta X_{i-(q-1)}}\right)^{i_1} \left(\frac{1}{\Delta X_{i-(q-2)}}\right)^{i_2} \cdots \left(\frac{1}{\Delta X_{i-1}}\right)^{i_{q-1}} \left(\frac{1}{\Delta X_i}\right)^{q-i_1-\ldots-i_{q-1}} \right\}. \tag{2.36}$$

So we have,

$$u^{(q+1)}\Big|_{x_i} = \sum_{k=0}^{q} (-1)^k \frac{u_{i-k} - u_{i-(k+1)}}{\Delta X_{i-k}} \mathcal{B}_q, \tag{2.37}$$

$$= \sum_{k=0}^{q} (-1)^k \frac{1}{\Delta X_{i-k}} u_{i-k} \mathcal{B}_q - \sum_{k=0}^{q} (-1)^k \frac{1}{\Delta X_{i-k}} u_{i-(k+1)} \mathcal{B}_q, \tag{2.38}$$

$$\begin{aligned} = u_i \frac{1}{\Delta X_i} \left(\frac{1}{\Delta X_i}\right)^q + \sum_{k=1}^{q} (-1)^k \frac{1}{\Delta X_{i-k}} u_{i-k} \mathcal{B}_q - \sum_{k=1}^{q} (-1)^{k-1} u_{i-k} \frac{1}{\Delta X_{i-k}} \mathcal{B}_q \\ - (-1)^q u_{i-(q+1)} \frac{1}{\Delta X_{i-q}} \frac{1}{\Delta X_{i-(q-1)}} \cdots \frac{1}{\Delta X_{i-1}} \frac{1}{\Delta X_i}, \end{aligned} \tag{2.39}$$

$$\begin{aligned} = u_i \left(\frac{1}{\Delta X_i}\right)^{q+1} + 2 \sum_{k=1}^{q} (-1)^k \frac{1}{\Delta X_{i-k}} u_{i-k} \mathcal{B}_q \\ + (-1)^{q+1} u_{i-(q+1)} \frac{1}{\Delta X_{i-q}} \frac{1}{\Delta X_{i-(q-1)}} \cdots \frac{1}{\Delta X_{i-1}} \frac{1}{\Delta X_i}, \end{aligned} \tag{2.40}$$

$$= \sum_{k=0}^{q+1} (-1)^k u_{i-k} \mathcal{B}_{q+1}. \tag{2.41}$$

where $\mathcal{B}_{q+1}$ is given by,

$$\mathcal{B}_{q+1} = \left\{ \sum_{\substack{i_1 = \ldots = i_{q-k} = 0, \\ i_{q+1-k} \geq 0, \\ i_{q-k} \geq 1, \\ \vdots \\ i_q \geq 1, \\ i_q + \ldots + i_{q+1-k} \leq q}} \left(\frac{1}{\Delta X_{i-(q)}}\right)^{i_1} \left(\frac{1}{\Delta X_{i-(q-1)}}\right)^{i_2} \cdots \left(\frac{1}{\Delta X_{i-1}}\right)^{i_q} \left(\frac{1}{\Delta X_i}\right)^{q+1-i_1-\ldots-i_q} \right\}. \tag{2.42}$$

□

This formula does not generalise well to non-integer derivatives as the number indexes $i_1, ..., i_n$ are dependent on the integer order of differentiation. Approach 2 is still viable, however we decide not to proceed in this manner due to an expected large computational cost associated with computing a truncated infinite sum of integer derivatives. It is also suspected that such a formula would not accurately approximate high integer derivatives due to the repeated application of backwards difference approximation. This could be improved by choosing a central difference approximation of the derivatives. It is also not immediately clear how to extend the formulation of the fractional derivative in terms of integer derivatives (2.27) to work on a non-uniform grid.

2.3 Caputo definitions approach

In this section we study the L1, for $\alpha \in (0,1)$, and L2, for $\alpha \in (1,2)$, methods which are found using the Caputo definitions for the left and right fractional derivatives. Currently the methods stated in the literature [34] [1] [54] are applied to a domain $x \in [0,b]$, and only for the left derivatives, so we will make these methods more general on domains $x \in [a,b],\ a < b,\ a,b \in \mathbb{R}$ and also consider the right derivatives. The L1 method has already been shown to extend easily to non-uniform grids [34], and some work has been done on extending temporal fractional derivatives to non-uniform grids for the L2 method [38], $t \in [0,l]$. We will also extend our adapted L1 and L2 methods to approximate space-fractional derivatives on non-uniform grids $x \in [a,b]$.

2.3.1 L1 method

The L1 method for approximating the left Caputo derivative at $x = x_n$ on a uniform grid is found by applying a forward difference approximation to the first derivative in the definition for the Caputo derivative [34]. We work with $N+2$ grid points denoted $x_0, ..., x_{N+1}$ where $x_0 = a$, $x_{N+1} = b$. We denoted the grid spacings by $h = \frac{b-a}{N+1}$.

$$
\begin{aligned}
{}_CD^{\alpha}_{a,x}u(x)\Big|_{x=x_n} &= \frac{1}{\Gamma(1-\alpha)}\int_{x_0}^{x_n}(x_n-s)^{-\alpha}u'(s)ds, && (2.43)\\
&\approx \frac{1}{\Gamma(-\alpha)}\sum_{k=0}^{n-1}\int_{x_k}^{x_{k+1}}(x_n-s)^{1-\alpha}\frac{u(x_{k+1})-u(x_k)}{h}ds, && (2.44)\\
&= \frac{h^{-\alpha}}{\Gamma(2-\alpha)}\sum_{k=0}^{n-1}\Big((n-k)^{1-\alpha}-(n-k-1)^{1-\alpha}\Big)\big(u(x_{k+1})-u(x_k)\big), && (2.45)
\end{aligned}
$$

where we have used the property of the gamma function $x\Gamma(x) = \Gamma(x+1)$. Similarly, the L1 method for approximating the right Caputo derivative at $x = x_n$ is found to be,

$$
{}_CD^{\alpha}_{x,b}u(x)\Big|_{x=x_n} \approx -\frac{h^{-\alpha}}{\Gamma(2-\alpha)}\sum_{k=n}^{N}\Big((k+1-n)^{1-\alpha}-(k-n)^{1-\alpha}\Big)\big(u(x_{k+1})-u(x_k)\big), \qquad (2.46)
$$

where we use the same approximation for the first derivative. Hence using equation (1.23) to define the fractional Laplacian, and assuming the function $u(x)$ is zero on the boundaries of the domain, we can approximate the fractional Laplacian for $\alpha \in (0,1)$ by,

$$- (-\Delta)^{\alpha/2} u(x)\Big|_{x=x_n} \approx -\frac{1}{2\cos(\alpha\pi/2)} \frac{h^{-\alpha}}{\Gamma(2-\alpha)} \left(\sum_{k=0}^{n-1} \Big((n-k)^{1-\alpha} - (n-k-1)^{1-\alpha} \Big) \big(u(x_{k+1}) - u(x_k) \big) \right.$$
$$\left. - \sum_{k=n}^{N} \Big((k+1-n)^{1-\alpha} - (k-n)^{1-\alpha} \Big) \big(u(x_{k+1}) - u(x_k) \big) \right). \tag{2.47}$$

This L1 method for the left derivative has error estimate given by [34] [31] [36] [50],

$$\left| \frac{u(a)h^{\alpha}}{\Gamma(1-\alpha)} + \sum_{k=0}^{n-1} \frac{h^{-\alpha}}{\Gamma(2-\alpha)} \Big((k+1)^{1-\alpha} - k^{1-\alpha} \Big) \big(u(x_{k+1}) - u(x_k) \big) - {}_{RL}D^{\alpha}_{a,x} u(x)\Big|_{x=x_n} \right| \leq C h^{2-\alpha}, \tag{2.48}$$

for some constant C, so we can see that the method has lower error for small α. The left approximation has already been extended to work on non-uniform grids, so this make this approach a good candidate for finding a non-uniform grid approximation to the fractional Laplacian.

2.3.2 L2 method

This is a method from [34], commonly used and adapted in the literature [38] [1], which approximates the left Caputo derivative for $\alpha \in (1,2)$ on an interval $x \in [0,b]$ on a uniform grid. In the limit $\alpha \to 2$ the method reduces to the central 3-point approximation for the second derivative. The accuracy of the method is dependent on the chosen value for α and is most accurate for $\alpha \in (1,1.5)$. There exists a slightly adapted method known as L2C which performs better for $\alpha \in (1.5,2)$ [34], for simplicity we will only consider the L2 method in this thesis. Here, we adapt the L2 method to work over a general interval $x \in [a,b]$ where $a,b \in \mathbb{R}$ and $a < b$. We also use parallel reasoning to find an approximation of the right Caputo derivative on the same uniform grid and interval with $N+2$ grid points as used in the L1 method.

The method starts with the Caputo integral definition and then applies an approximation of the second derivative. The original method applies a three point central approximation, but we use a three point forward stencil. The reason for this was to avoid any problems near the boundary but the central approximation would in fact also work as throughout this thesis we always let derivatives on the boundary be equal to zero for $\alpha \in (1,2)$.

$${}_{C}D^{\alpha}_{a,x} u(x)\Big|_{x=x_n} = \frac{1}{\Gamma(2-\alpha)} \int_{x_0}^{x_n} (x_n - s)^{1-\alpha} u''(s) ds, \tag{2.49}$$

$$= \frac{1}{\Gamma(2-\alpha)} \sum_{k=0}^{n-1} \int_{x_k}^{x_{k+1}} (x_n - s)^{1-\alpha} u''(s) ds, \tag{2.50}$$

$$\approx \frac{1}{\Gamma(2-\alpha)} \sum_{k=0}^{n-1} \frac{u(x_k) - 2u(x_{k+1}) + u(x_{k+2})}{h^2} \int_{x_k}^{x_{k+1}} (x_n - s)^{1-\alpha} ds, \tag{2.51}$$

$$= \frac{h^{-2}}{\Gamma(3-\alpha)} \sum_{k=0}^{n-1} \big(u(x_k) - 2u(x_{k+1}) + u(x_{k+2}) \big) \Big((x_n - x_k)^{2-\alpha} - (x_n - x_{k+1})^{2-\alpha} \Big), \tag{2.52}$$

$$= \frac{h^{-\alpha}}{\Gamma(3-\alpha)} \sum_{k=0}^{n-1} \big(u(x_k) - 2u(x_{k+1}) + u(x_{k+2}) \big) \Big((n-k)^{2-\alpha} - (n-k-1)^{2-\alpha} \Big). \tag{2.53}$$

Now, when we group terms by the functions $u(x_k)$, we get,

$$ {}_C D^{\alpha}_{a,x} u(x)\Big|_{x=x_n} \approx \frac{h^{-\alpha}}{\Gamma(3-\alpha)} \sum_{k=0}^{n+1} u(x_k) W_L(k), \tag{2.54}$$

where the coefficients are given by,

$$ W_L(k) = \begin{cases} 1 & \text{if } k = n+1 \\ 2^{2-\alpha} - 3 & \text{if } k = n \\ (n-k+2)^{2-\alpha} - 3(n-k+1)^{2-\alpha} + 3(n-k)^{2-\alpha} - (n-k-1)^{2-\alpha} & \text{if } 2 \le k \le n-1 \\ -2(n^{2-\alpha} - (n-1)^{2-\alpha}) + (n-1)^{2-\alpha} - (n-2)^{2-\alpha} & \text{if } k = 1 \\ n^{2-\alpha} - (n-1)^{2-\alpha} & \text{if } k = 0. \end{cases} \tag{2.55}$$

We can approximate the right Caputo derivative in a similar way using a three point central stencil for the second derivative.

$$ {}_C D^{\alpha}_{x,b} u(x)\Big|_{x=x_n} = \frac{1}{\Gamma(2-\alpha)} \int_{x_n}^{x_{N+1}} (s-x_n)^{1-\alpha} u''(s) ds, \tag{2.56}$$

$$ = \frac{1}{\Gamma(2-\alpha)} \sum_{k=n}^{N} \int_{x_k}^{x_{k+1}} (s-x_n)^{1-\alpha} u''(s) ds, \tag{2.57}$$

$$ \approx \frac{1}{\Gamma(2-\alpha)} \sum_{k=n}^{N} \frac{u(x_{k+1}) - 2u(x_k) + u(x_{k-1})}{h^2} \int_{x_k}^{x_{k+1}} (s-x_n)^{1-\alpha} ds, \tag{2.58}$$

$$ = \frac{h^{-2}}{\Gamma(3-\alpha)} \sum_{k=n}^{N} \left(u(x_{k+1}) - 2u(x_k) + u(x_{k-1})\right) \Big((x_{k+1}-x_n)^{2-\alpha} - (x_k - x_n)^{2-\alpha}\Big), \tag{2.59}$$

$$ = \frac{h^{-\alpha}}{\Gamma(3-\alpha)} \sum_{k=n}^{N} \left(u(x_{k+1}) - 2u(x_k) + u(x_{k-1})\right) \Big((k+1-n)^{2-\alpha} - (k-n)^{2-\alpha}\Big). \tag{2.60}$$

Then, when we group terms by the functions $u(x_k)$, we get,

$$ {}_C D^{\alpha}_{x,b} u(x)\Big|_{x=x_n} \approx \frac{h^{-\alpha}}{\Gamma(3-\alpha)} \sum_{k=n-1}^{N+1} u(x_k) W_R(k), \tag{2.61}$$

where the coefficients are given by,

$$ W_R(k) = \begin{cases} 1 & \text{if } k = n-1 \\ 2^{2-\alpha} - 3 & \text{if } k = n \\ -(k-1-n)^{2-\alpha} - 3(k+1-n)^{2-\alpha} + 3(k-n)^{2-\alpha} + (k+2-n)^{2-\alpha} & \text{if } n+1 \le k \le N-1 \\ -(N-1-n)^{2-\alpha} - 2(N+1-n)^{2-\alpha} + 3(N-n)^{2-\alpha} & \text{if } k = N \\ (N+1-n)^{2-\alpha} - (N-n)^{2-\alpha} & \text{if } k = N+1. \end{cases} \tag{2.62}$$

Both the L1 and L2 methods originate from the Caputo definitions for fractional derivatives where we can use standard approximations for integer derivatives. This makes extending these methods a good option for working with non-uniform grids as we only need to adapt the approximation for the integer derivatives. This has already been done for the left Caputo derivative for $\alpha \in (0, 1)$ [34] so in the following sections we will generalise the L1 and L2 methods to form the L1(NU) and L2(NU) methods to approximate the left and right Caputo derivatives on non-uniform grids.

2.4 L1(NU) method

We now work on a non-uniform grid with $N+2$ grid points denoted $x_0, ..., x_{N+1}$ where $x_0 = a$, $x_{N+1} = b$. We denote the grid spacings by $\Delta X_k,\ k = 0, ..., N$ where $\Delta X_k = x_{k+1} - x_k$. We first slightly adapt the method from [34] for a general interval $x \in [a, b]$,

$$
\begin{aligned}
{}_C D^{\alpha}_{a,x}\Big|_{x=x_n} &= \frac{1}{\Gamma(1-\alpha)} \int_{x_0}^{x_n} (x_n - s)^{-\alpha} u'(s) ds, && (2.63)\\
&\approx \frac{1}{\Gamma(1-\alpha)} \sum_{k=0}^{n-1} \int_{x_k}^{x_{k+1}} (x_n - s)^{-\alpha} \left(\frac{u(x_{k+1}) - u(x_k)}{\Delta X_k} \right) ds, && (2.64)\\
&= \frac{1}{\Gamma(1-\alpha)} \sum_{k=0}^{n-1} \left(\frac{u(x_{k+1}) - u(x_k)}{\Delta X_k} \right) \frac{1}{1-\alpha} \left(\left(\sum_{j=k}^{n-1} \Delta X_j \right)^{1-\alpha} - \left(\sum_{j=k+1}^{n-1} \Delta X_j \right)^{1-\alpha} \right), && (2.65)\\
&= \frac{1}{\Gamma(2-\alpha)} \sum_{k=0}^{n-1} W_{1,L}(k) \left(u(x_{k+1}) - u(x_k) \right). && (2.66)
\end{aligned}
$$

The coefficients are given by,

$$
W_{1,L}(k) = \frac{1}{\Delta X_k} \left(\left(\sum_{j=k}^{n-1} \Delta X_j \right)^{1-\alpha} - \left(\sum_{j=k+1}^{n-1} \Delta X_j \right)^{1-\alpha} \right). \qquad (2.67)
$$

Analysis shows that when $a = 0$ (and some other conditions are satisfied as described in [34]), the magnitude of the error is bounded by $C\Delta X_{max}^{2-\alpha}$ for some constant C, where ΔX_{max} is the largest step size on the non-uniform grid [59]. Note that this is similar to the error bound on the uniform grid shown in equation (2.48). Similarly the right Caputo derivative of $u(x)$ approximated at x_n is given by,

$$
\begin{aligned}
{}_C D^{\alpha}_{x,b}\Big|_{x=x_n} &= -\frac{1}{\Gamma(1-\alpha)} \int_{x_n}^{x_N} (s - x_n)^{-\alpha} u'(s) ds, && (2.68)\\
&= -\frac{1}{\Gamma(1-\alpha)} \sum_{k=n}^{N} \int_{x_k}^{x_{k+1}} (s - x_n)^{-\alpha} u'(s) ds, && (2.69)\\
&\approx -\frac{1}{\Gamma(2-\alpha)} \sum_{k=n}^{N} \left(\frac{u(x_{k+1}) - u(x_k)}{\Delta X_k} \right) \left(\left(\sum_{j=n}^{k} \Delta X_j \right) - \left(\sum_{j=n}^{k-1} \Delta X_j \right) \right), && (2.70)\\
&= -\frac{1}{\Gamma(2-\alpha)} \sum_{k=n}^{N} W_{1,R}(k) \left(u(x_{k+1}) - u(x_k) \right). && (2.71)
\end{aligned}
$$

The coefficients are given by,

$$
W_{1,R}(k) = \frac{1}{\Delta X_k} \left(\left(\sum_{j=n}^{k} \Delta X_j \right) - \left(\sum_{j=n}^{k-1} \Delta X_j \right) \right). \qquad (2.72)
$$

Thus the Riesz derivative (equivalent to the fractional Laplacian) can be approximated at $x = x_n$ by,

$$- (-\Delta)^{\alpha/2} u(x)\Big|_{x=x_n} = {}_{RZ}D^{\alpha} u(x)\big|_{x=x_n} \approx \frac{-1}{2\cos(\pi\alpha/2)\Gamma(2-\alpha)} \left(\sum_{k=0}^{n-1} W_{1,L}(k) \left(u(x_{k+1}) - u(x_k)\right) - \sum_{k=n}^{N} W_{1,R}(k) \left(u(x_{k+1}) - u(x_k)\right) \right), \tag{2.73}$$

Where we assume $u(a) - u(b) = 0$. We should note that in the case $\Delta X_k = \Delta X_{k-1} = (a-b)/(N+1) = h$ for all $k = 1, ..., N+1$, i.e a uniform grid, this method reduces to the L1 method in equation (2.47) as in this case,

$$\begin{aligned} W_{1,L}(k) &= \frac{1}{h}\left(((n-k)h)^{1-\alpha} - (n-k-1)^{1-\alpha} \right), \\ &= h^{-\alpha}\left[(n-k)^{1-\alpha} - (n-k-1)^{1-\alpha} \right], \\ W_{1,R}(k) &= \frac{1}{h}\left(((k+1-n)h)^{1-\alpha} - (k-n)^{1-\alpha} \right), \\ &= h^{-\alpha}\left[(k+1-n)^{1-\alpha} - (k-n)^{1-\alpha} \right]. \end{aligned}$$

2.5 L2(NU) method

We now extend the L2 method for $\alpha \in (1,2)$ to a non-uniform grid using a non-uniform approximation for the second derivative. We work with $N+2$ grid points denoted $x_0, ..., x_{N+1}$ where $x_0 = a$, $x_{N+1} = b$. We denote the grid spacings by $\Delta X_k,\ k = 0, ..., N$ where $\Delta X_k = x_{k+1} - x_k$. We start with the left Caputo derivative where we apply a 3 point forward approximation of the second derivative, but this time on a non-uniform gird. This approximation was found using the method of undetermined coefficients.

$$\begin{aligned} {}_{C}D^{\alpha}_{a,x} u(x)\Big|_{x=x_n} &= \frac{1}{\Gamma(2-\alpha)} \sum_{k=0}^{n-1} \int_{x_k}^{x_{k+1}} (x_n - s)^{1-\alpha} u''(s) ds, & (2.74) \\ &\approx \frac{2}{\Gamma(2-\alpha)} \sum_{k=0}^{n-1} \left(\frac{u(x_k)}{\Delta X_k(\Delta X_k + \Delta X_{k+1})} - \frac{u(x_{k+1})}{\Delta X_{k+1}\Delta X_k} + \frac{u(x_{k+2})}{\Delta X_{k+1}(\Delta X_{k+1} + \Delta X_k)} \right) \\ &\qquad \times \int_{x_k}^{x_{k+1}} (x_n - s)^{1-\alpha} ds, & (2.75) \\ &= \frac{2}{\Gamma(3-\alpha)} \sum_{k=0}^{n-1} \left(\frac{u(x_k)}{\Delta X_k(\Delta X_k + \Delta X_{k+1})} - \frac{u(x_{k+1})}{\Delta X_{k+1}\Delta X_k} + \frac{u(x_{k+2})}{\Delta X_{k+1}(\Delta X_{k+1} + \Delta X_k)} \right) \\ &\qquad \times \left(\left(\sum_{j=k}^{n-1} \Delta X_j \right)^{2-\alpha} - \left(\sum_{j=k+1}^{n-1} \Delta X_j \right)^{2-\alpha} \right). & (2.76) \end{aligned}$$

A valid alternative to using the forward approximation for the derivative is the central approximation which would give a similar expression. Then if we collect terms in $u(x_k)$ we get,

$${}_{C}D^{\alpha}_{a,x}\Big|_{x=x_n} \approx \frac{2}{\Gamma(3-\alpha)} \sum_{k=0}^{n+1} u(x_k) W_{2,L}(k), \tag{2.77}$$

where the coefficients are given by,

$$W_{2,L}(k) = \begin{cases} \frac{\left(\sum_{j=0}^{n-1}\Delta X_j\right)^{2-\alpha}-\left(\sum_{j=1}^{n-1}\Delta X_j\right)^{2-\alpha}}{\Delta X_0(\Delta X_0+\Delta X_1)} & \text{if } k=0 \\ -\frac{\left(\sum_{j=0}^{n-1}\Delta X_j\right)^{2-\alpha}-\left(\sum_{j=1}^{n-1}\Delta X_j\right)^{2-\alpha}}{\Delta X_0\Delta X_1} + \frac{\left(\sum_{j=1}^{n-1}\Delta X_j\right)^{2-\alpha}-\left(\sum_{j=2}^{n-1}\Delta X_j\right)^{2-\alpha}}{\Delta X_1(\Delta X_1+\Delta X_2)} & \text{if } k=1 \\ \frac{\left(\sum_{j=k-2}^{n-1}\Delta X_j\right)^{2-\alpha}-\left(\sum_{j=k-1}^{n-1}\Delta X_j\right)^{2-\alpha}}{\Delta X_{k-1}\left(\Delta X_{k-2}+\Delta X_{k-1}\right)} - \frac{\left(\sum_{j=k-1}^{n-1}\Delta X_j\right)^{2-\alpha}-\left(\sum_{j=k}^{n-1}\Delta X_j\right)^{2-\alpha}}{\Delta X_{k-1}\Delta X_k} & \\ \qquad + \frac{\left(\sum_{j=k}^{n-1}\Delta X_j\right)^{2-\alpha}-\left(\sum_{j=k+1}^{n-1}\Delta X_j\right)^{2-\alpha}}{\Delta X_k\left(\Delta X_k+\Delta X_{k+1}\right)} & \text{if } 2\le k\le n-1 \\ \frac{(\Delta X_{n-2}+\Delta X_{n-1})^{2-\alpha}-(\Delta X_{n-1})^{2-\alpha}}{\Delta X_{n-1}(\Delta X_{n-2}+\Delta X_{n-1})} - \frac{(\Delta X_{n-1})^{2-\alpha}}{\Delta X_{n-1}\Delta X_n} & \text{if } k=n \\ \frac{\Delta X_{n-1}^{2-\alpha}}{\Delta X_n(\Delta X_{n-1}+\Delta X_n)} & \text{if } k=n+1. \end{cases} \tag{2.78}$$

We follow a similar procedure for the right derivative, but this time apply a central approximation for the second derivative of $u(x)$ on the non-uniform grid.

$$ {}_{C}D_{x,b}^{\alpha}u(x)\Big|_{x=x_n} = \frac{1}{\Gamma(2-\alpha)}\sum_{k=n}^{N}\int_{x_k}^{x_{k+1}}(s-x_n)^{1-\alpha}u''(s)ds, \tag{2.79}$$

$$\approx \frac{2}{\Gamma(3-\alpha)}\sum_{k=n}^{N}\left(\frac{u(x_{k-1})}{\Delta X_{k-1}(\Delta X_k+\Delta X_{k-1})} - \frac{u(x_k)}{\Delta X_k\Delta X_{k-1}} + \frac{u(x_{k+1})}{\Delta X_k(\Delta X_k+\Delta X_{k-1})}\right) \tag{2.80}$$

$$\times\left(\left(\sum_{j=n}^{k}\Delta X_j\right)^{2-\alpha} - \left(\sum_{j=n}^{k-1}\Delta X_j\right)^{2-\alpha}\right). \tag{2.81}$$

Then we collect terms in $u(x_k)$ to get the approximation,

$$ {}_{C}D_{x,b}^{\alpha}u(x)\Big|_{x=x_n} \approx \frac{2}{\Gamma(3-\alpha)}\sum_{k=n-1}^{N+1} W_{2,R}(k)u(x_k), \tag{2.82}$$

$$W_{2,R}(k) = \begin{cases} \frac{(\Delta X_n)^{2-\alpha}}{\Delta X_{n-1}(\Delta X_n+\Delta X_{n-1})} & \text{if } k=n-1 \\ -\frac{(\Delta X_n)^{2-\alpha}}{\Delta X_n\Delta X_{n-1}} + \frac{(\Delta X_n+\Delta X_{n+1})^{2-\alpha}-(\Delta X_n)^{2-\alpha}}{\Delta X_n(\Delta X_{n+1}+\Delta X_n)} & \text{if } k=n \\ \frac{\left(\sum_{j=n}^{k-1}\Delta X_j\right)^{2-\alpha}-\left(\sum_{j=n}^{k-2}\Delta X_j\right)^{2-\alpha}}{\Delta X_{k-1}\left(\Delta X_{k-1}-\Delta X_{k-2}\right)} - \frac{\left(\sum_{j=n}^{k}\Delta X_j\right)^{2-\alpha}-\left(\sum_{j=n}^{k-1}\Delta X_j\right)^{2-\alpha}}{\Delta X_k\Delta X_{k-1}} & \\ \qquad + \frac{\left(\sum_{j=n}^{k+1}\Delta X_j\right)^{2-\alpha}-\left(\sum_{j=n}^{k}\Delta X_j\right)^{2-\alpha}}{\Delta X_k\left(\Delta X_{k+1}-\Delta X_k\right)} & \text{if } n+1\le k\le N-1 \\ \frac{\left(\sum_{j=n}^{N-1}\Delta X_j\right)^{2-\alpha}-\left(\sum_{j=n}^{N-2}\Delta X_j\right)^{2-\alpha}}{\Delta X_{N-1}\left(\Delta X_{N-1}-\Delta X_{N-2}\right)} - \frac{\left(\sum_{j=n}^{N}\Delta X_j\right)^{2-\alpha}-\left(\sum_{j=n}^{N-1}\Delta X_j\right)^{2-\alpha}}{\Delta X_{N-1}\Delta X_N} & \text{if } k=N \\ \frac{\left(\sum_{j=n}^{N}\Delta X_j\right)^{2-\alpha}-\left(\sum_{j=n}^{N-1}\Delta X_j\right)^{2-\alpha}}{\Delta X_N\left(\Delta X_{N-1}-\Delta X_N\right)} & \text{if } k=N+1. \end{cases} \tag{2.83}$$

We can now combine the left and right derivatives using the definition for the Riesz derivative representation of the fractional Laplacian, equations (1.23) and (1.24), hence the L2(NU) approximation to the fractional Laplacian in one dimension when $u(a) = u(b) = u'(a) = u'(b) = 0$ is given by,

$$-\left(-\Delta\right)^{\alpha/2} u(x)\Big|_{x=x_n} = {}_{RZ}D^{\alpha} u(x)\big|_{x=x_n} \approx -\frac{1}{\Gamma(3-\alpha)\cos(\pi\alpha/2)}\left(\sum_{k=0}^{n+1} W_{2,L}(k)u(x_k) + \sum_{k=n-1}^{N+1} W_{2,R}(k)u(x_k)\right). \quad (2.84)$$

Chapter 3

Finite Difference Methods for the Fractional Laplacian in Two Dimensions

In higher dimensions, the Riemann-Liouville and Riesz derivatives no longer agree with each other [43], so depending on the application of the fractional operator, a choice needs to be made as to which of these definitions we use. Each definition preserves properties of the traditional Laplacian. In this thesis we choose to define the fractional Laplacian in two dimensions via the Riesz potential as introduced in Chapter 1,

$$-\left(-\Delta\right)^{\alpha/2} u(x,y) = \frac{\partial^\alpha u(x,y)}{\partial |x|^\alpha} + \frac{\partial^\alpha u(x,y)}{\partial |y|^\alpha}. \tag{3.1}$$

When discretising the system in two dimensions, the variables $x,\ y,\ u,$ are assigned two indexes i and j where the first references the x coordinate and the second the y coordinate. We will work on a domain where $x \in [a,b],\ y \in [a,c]$ and there are $N+2$ grid points in the x and y directions. We use the Riesz potential and the L1(NU) and L2(NU) methods to approximate the fractional Laplacian for $\alpha \in (0,1)$ and $\alpha \in (1,2)$ respectively.

3.1 L1(NU)-2D method

To approximate the fractional Laplacian for $\alpha \in (0,1)$ in two dimensions we start with the definition of the Riesz potential, equation (1.35) in Chapter 1,

$$\begin{aligned}
-\left(-\Delta\right)^{\alpha/2} u\Big|_{x=x_n,y=y_m} &= \left.\frac{\partial^\alpha u(x,y)}{\partial |x|^\alpha}\right|_{x=x_n,y=y_m} + \left.\frac{\partial^\alpha u(x,y)}{\partial |y|^\alpha}\right|_{x=x_n,y=y_m}, &(3.2)\\
&= -\frac{1}{2\cos(\pi\alpha/2)}\Big({}_CD^\alpha_{a,x} u(x_n,y_m) + {}_CD^\alpha_{x,b} u(x_n,y_m) \\
&\qquad + {}_CD^\alpha_{a,y} u(x_n,y_m) + {}_CD^\alpha_{y,c} u(x_n,y_m)\Big), &(3.3)
\end{aligned}$$

where we assume that $u(x,y)$, equals zero on the boundary. From here we can apply the L1(NU) method from the previous section to find the L1(NU)-2D method,

$$\begin{aligned}
-\left(-\Delta\right)^{\alpha/2} u\Big|_{x=x_n,y=y_m} \approx -\frac{1}{2\Gamma(2-\alpha)\cos(\pi\alpha/2)} &\left(\sum_{k=0}^{n-1} W_{1,L}(k)\left(u(x_{k+1},y_m) - u(x_k,y_m)\right)\right. \\
&- \sum_{k=n}^{N} W_{1,R}\left(u(x_{k+1},y_m) - u(x_k,y_m)\right) + \sum_{k=0}^{n-1} W_{1,L}(k)\left(u(x_n,y_{k+1}) - u(x_n,y_k)\right) \\
&\left.- \sum_{k=n}^{N} W_{1,R}\left(u(x_n,y_{k+1}) - u(x_n,y_k)\right)\right). \qquad (3.4)
\end{aligned}$$

3.2 L2(NU)-2D method

To approximate the fractional Laplacian for $\alpha \in (1,2)$ in two dimensions we start with the definition of the Riesz potential, equation (1.35) in Chapter 1,

$$\begin{aligned} -\left(-\Delta\right)^{\alpha/2} u\Big|_{x=x_n,y=y_m} &= \frac{\partial^\alpha u(x,y)}{\partial |x|^\alpha}\bigg|_{x=x_n,y=y_m} + \frac{\partial^\alpha u(x,y)}{\partial |y|^\alpha}\bigg|_{x=x_n,y=y_m}, \qquad (3.5)\\ &= -\frac{1}{2\cos(\pi\alpha/2)}\Big({}_C D^\alpha_{a,x} u(x_n,y_m) + {}_C D^\alpha_{x,b} u(x_n,y_m) \\ &\qquad + {}_C D^\alpha_{a,y} u(x_n,y_m) + {}_C D^\alpha_{y,c} u(x_n,y_m)\Big), \qquad (3.6)\end{aligned}$$

where we assume $u(a,y) = u(b,y) = u(x,a) = u(x,c) = 0$ and $u_x(a,y) = u_x(b,y) = u_y(x,a) = u_y(x,c) = 0$. From here we can apply the L2(NU) method from the previous section to find the L2(NU)-2D method,

$$\begin{aligned} -\left(-\Delta\right)^{\alpha/2} u\Big|_{x=x_n,y=y_m} \approx -\frac{1}{\Gamma(3-\alpha)\cos(\pi\alpha/2)} &\left(\sum_{k=0}^{n+1} W_{2,L}(k) u(x_k,y_m) + \sum_{k=n-1}^{N+1} W_{2,R}(k) u(x_k,y_m)\right.\\ &\left. + \sum_{k=0}^{n+1} W_{2,L}(k) u(x_n,y_k) + \sum_{k=n-1}^{N+1} W_{2,R}(k) u(x_n,y_k)\right). \end{aligned} \qquad (3.7)$$

Chapter 4

Adaptive Grid Methods for PDEs in One and Two Dimensions

4.1 One Dimension

We consider PDEs of the form,

$$\frac{\partial u(x,t)}{\partial t} = F(u(x,t),x,t), \tag{4.1}$$

where the right hand side contains a fractional derivative where $\alpha \in (0,1) \cup (1,2)$. We use the L1(NU) or L2(NU) methods described in the previous sections to discretise the fractional Laplacian on the RHS of the PDE and work with $N+2$ grid points denoted $x_0, ..., x_{N+1}$ where $x_0 = a$, $x_{N+1} = b$. We use the method developed by Verwer et al. [52] to determine the adaptive grid. This results in a stiff differential algebraic equation (DAE) system,

$$U' - X' \cdot diag(D) = F, \tag{4.2}$$

$$\tau B X' = g, \tag{4.3}$$

where U' denotes the derivative of U with respect to time, and $U',\ X',\ D,\ F,$ and g are column vectors given by,

$$U' = [U_1', ..., U_N']^T, \qquad D = [D_1, ..., D_N]^T, \qquad F = [F_1, ..., F_N]^T,$$
$$X' = [X_1', ..., X_N']^T, \qquad g = [g_1, ..., g_N]^T.$$

$$D_i = \frac{u_{i+1} - u_{i-1}}{x_{i+1} - x_{i-1}}, \tag{4.4}$$

$$F_i = \text{discretisation of the } i^{th} \text{ component of the RHS of the PDE}, \tag{4.5}$$

$$\begin{aligned} g_i = &\left(-\frac{\mu}{x_{i+2} - x_{i+1}} + \frac{1+2\mu}{x_{i+1} - x_i} - \frac{\mu}{x_i - x_{i-1}}\right)\frac{1}{M_i} \\ &- \left(-\frac{\mu}{x_{i+1} - x_i} + \frac{1+2\mu}{x_i - x_{i-1}} - \frac{\mu}{x_{i-1} - x_{i-2}}\right)\frac{1}{M_{i-1}}, \text{for } 1 < i < N, \end{aligned} \tag{4.6}$$

$$g_1 = 0,\ g_N = 0. \tag{4.7}$$

The monitor function we choose is,

$$M_i = \sqrt{1 + \beta\frac{(u_{i+1} - u_i)^2}{(x_{i+1} - x_i)^2}}. \tag{4.8}$$

If $\beta = 0$, the grid will not adapt and remain fixed, the higher the value of β, the more adaptivity of the grid is allowed [52] [56]. In general $\beta = 1$ are good choices, but the optimal value varies with the PDE and factors such as stability and efficiency [52]. B is an $N \times N$ matrix, $B = [\tilde{B}_1, ...\tilde{B}_N]^T$ where $\tilde{B}_i$ are row vectors $\tilde{B}_i = [\tilde{B}_{i,1}, ..., \tilde{B}_{i,N}]$ with entries given by:

$$\tilde{B}_1 = [2, -1, 0, ..., 0], \tag{4.9}$$

$$\tilde{B}_N = [0, ..., 0, -1, 2], \tag{4.10}$$

$$\text{for } i = 2 : \begin{cases} \tilde{B}_{i,i-1} & = \frac{\mu}{M_i(\Delta X_{i-1})^2} + \frac{1+2\mu}{M_{i-1}(\Delta X_{i-1})^2} + \frac{\mu}{M_{i-1}(\Delta X_{i-2})^2} \\ \tilde{B}_{i,i} & = -\left(\frac{\mu}{M_i(\Delta X_{i-1})^2} + \frac{1+2\mu}{M_i(\Delta X_i)^2} + \frac{1+2\mu}{M_{i-1}(\Delta X_{i-1})^2} + \frac{\mu}{M_{i-1}(\Delta X_i)^2}\right) \\ \tilde{B}_{i,i+1} & = \frac{\mu}{M_i(\Delta X_{i+1})^2} + \frac{1+2\mu}{M_i(\Delta X_i)^2} + \frac{\mu}{M_{i-1}(\Delta X_i)^2} \\ \tilde{B}_{i,i+2} & = -\frac{\mu}{M_i(\Delta X_{i+1})^2} \\ \tilde{B}_{i,j} & = 0 \text{ for } j > i+2, \end{cases} \tag{4.11}$$

$$\text{for } 2 < i < N-1 : \begin{cases} \tilde{B}_{i,j} & = 0 \text{ for } j < i-2 \\ \tilde{B}_{i,i-2} & = -\frac{\mu}{M_{i-1}(\Delta X_{i-2})^2} \\ \tilde{B}_{i,i-1} & = \frac{\mu}{M_i(\Delta X_{i-1})^2} + \frac{1+2\mu}{M_{i-1}(\Delta X_{i-1})^2} + \frac{\mu}{M_{i-1}(\Delta X_{i-2})^2} \\ \tilde{B}_{i,i} & = -\left(\frac{\mu}{M_i(\Delta X_{i-1})^2} + \frac{1+2\mu}{M_i(\Delta X_i)^2} + \frac{1+2\mu}{M_{i-1}(\Delta X_{i-1})^2} + \frac{\mu}{M_{i-1}(\Delta X_i)^2}\right) \\ \tilde{B}_{i,i+1} & = \frac{\mu}{M_i(\Delta X_{i+1})^2} + \frac{1+2\mu}{M_i(\Delta X_i)^2} + \frac{\mu}{M_{i-1}(\Delta X_i)^2} \\ \tilde{B}_{i,i+2} & = -\frac{\mu}{M_i(\Delta X_{i+1})^2} \\ \tilde{B}_{i,j} & = 0 \text{ for } j > i+2, \end{cases} \tag{4.12}$$

$$\text{for } i = N-1 : \begin{cases} \tilde{B}_{i,j} & = 0 \text{ for } j < i-2 \\ \tilde{B}_{i,i-2} & = -\frac{\mu}{M_{i-1}(\Delta X_{i-2})^2} \\ \tilde{B}_{i,i-1} & = \frac{\mu}{M_i(\Delta X_{i-1})^2} + \frac{1+2\mu}{M_{i-1}(\Delta X_{i-1})^2} + \frac{\mu}{M_{i-1}(\Delta X_{i-2})^2} \\ \tilde{B}_{i,i} & = -\left(\frac{\mu}{M_i(\Delta X_{i-1})^2} + \frac{1+2\mu}{M_i(\Delta X_i)^2} + \frac{1+2\mu}{M_{i-1}(\Delta X_{i-1})^2} + \frac{\mu}{M_{i-1}(\Delta X_i)^2}\right) \\ \tilde{B}_{i,i+1} & = \frac{\mu}{M_i(\Delta X_{i+1})^2} + \frac{1+2\mu}{M_i(\Delta X_i)^2} + \frac{\mu}{M_{i-1}(\Delta X_i)^2}. \end{cases} \tag{4.13}$$

The value of μ is defined via an input parameter κ by $\mu = \kappa(\kappa + 1)$. This parameter is related to the smoothing of the grid and the density of clusters of points and in general $\kappa = 1$ or 2 is a good choice [52]. Grid point spacing must obey the local quasi-uniformity relation [52],

$$\frac{\kappa}{\kappa+1} \leq \frac{\Delta X_i}{\Delta X_{i-1}} \leq \frac{\kappa+1}{\kappa}. \tag{4.14}$$

Hence we can see that as we increase the value of κ, the grid becomes more uniform.

We re-write this system as

$$C(Y)Y' = L(Y), \tag{4.15}$$

where,

$$Y = [U_1, U_2, ..., U_N X_1, X_2, ..., X_N]^T, \tag{4.16}$$

$$C(Y) = \begin{pmatrix} I^{N\times N} & -diag(D) \\ 0^{N\times N} & \tau B \end{pmatrix}, \tag{4.17}$$

$$L(Y) = [F_1, ..., F_N, g_1, ..., g_N]^T. \tag{4.18}$$

τ is a temporal grid-smoothing parameter which can act as a delay parameter. For large choices of τ the grid does not move, and for τ too small the grid moves too quickly and not smoothly [52]. Typical choices of τ are $\tau = 1 \times 10^{-4}$ or $\tau = 1 \times 10^{-3}$. We use a Python wrapper for Sundials DAE integrator IDA, which is written in C, to solve the system (4.15). This method is adaptive in time and uses variable-order, variable-coefficient BDF (Backward Differentiation Formula) [40]. It is possible that rearranging the system to let $Y = [X_1, U_1, ..., X_N, U_N]^T$ may help with the efficiency of the method, but since efficiency was not the goal we did not try this.

4.2 Two Dimensions

We consider PDEs of the form,

$$\frac{\partial u(x,y,t)}{\partial t} = F\left(u(x,y,t), x, y, t\right), \tag{4.19}$$

where the right hand side contains a fractional derivative with fractional order $\alpha \in (1, 2)$. We use the L1(NU)-2D or L2(NU)-2D methods described in the previous chapter to discretise the fractional Laplacian on the RHS of the PDE and work with $(N+2)^2$ gridpoints denoted $x_{i,j},\ y_{i,j},\ i,j \in (0, ..., N+1)$ where $x_{0,j} = a,\ x_{N+1,j} = b,\ y_{i,0} = a,\ y_{i,N+1} = c$. We use the method involving a transformation of coordinates used by Zegeling et al. [55] [57] [58] to determine the solution to the PDE on an adaptive grid.

We consider the coordinate transform $u(x,y,t) \mapsto u(\xi, \eta, \theta)$, where $\xi = \xi(x,y,t),\ \eta = \eta(x,y,t),\ \theta = t$ and we choose ξ and η such that they form a fixed uniform grid on a square $[0,1] \times [0,1]$. Thus, there will always be an invertible transformation between x and ξ, and y and η. Applying this transformation to the left hand side of (4.19) we get,

$$u_t = u_\theta \theta_t + u_\xi \xi_t + u_\eta \eta_t. \tag{4.20}$$

Now we make the assumption that there exists an invertible transformation matrix between the coordinates $(x,\ y,\ t)$ and $(\xi, \eta,\ \theta)$, hence the Jacobian of the transform has the property $J^{-1}J = I$ leading us to the following system of equations,

$$\xi_x x_\xi + \xi_y y_\xi = 1, \tag{4.21}$$
$$\eta_x x_\xi + \eta_y y_\xi = 0, \tag{4.22}$$
$$\xi_x x_\eta + \xi_y y_\eta = 0, \tag{4.23}$$
$$\eta_x x_\eta + \eta_y y_\eta = 1, \tag{4.24}$$
$$\xi_x x_\theta + \xi_y y_\theta + \xi_t = 0, \tag{4.25}$$
$$\eta_x x_\theta + \eta_y y_\theta + \eta_t = 0. \tag{4.26}$$

Upon substitution we get,

$$\xi_t = -\frac{x_\theta y_\eta - x_\eta y_\theta}{\mathcal{J}}, \text{ and, } \eta_t = \frac{x_\theta y_\xi - x_\eta y_\theta}{\mathcal{J}}, \tag{4.27}$$

where $\mathcal{J} = x_\xi y_\eta - x_\eta y_\xi$ is the determinant of the Jacobian matrix of the transformation. Hence we can rewrite PDE (4.19) as,

$$u_\theta = -\frac{1}{\mathcal{J}}\left((x_\theta y_\xi - x_\eta y_\theta)u_\eta - (x_\theta y_\eta - x_\eta y_\theta)u_\xi\right) + F(u(x,y,t), x, y, t), \tag{4.28}$$

where all derivatives are approximated by,

$$x_\xi|_{i,j} = \frac{x_{i+1,j} - x_{i-1,j}}{2\Delta\xi}, \qquad x_\eta|_{i,j} = \frac{x_{i,j+1} - x_{i,j-1}}{2\Delta\eta}, \tag{4.29}$$

$$y_\xi|_{i,j} = \frac{y_{i+1,j} - y_{i-1,j}}{2\Delta\xi}, \qquad y_\eta|_{i,j} = \frac{y_{i,j+1} - y_{i,j-1}}{2\Delta\eta}, \tag{4.30}$$

$$u_\xi|_{i,j} = \frac{u_{i+1,j} - u_{i-1,j}}{2\Delta\xi}, \qquad u_\eta|_{i,j} = \frac{u_{i,j+1} - u_{i,j-1}}{2\Delta\eta}, \tag{4.31}$$

$$\Delta\xi = \frac{1}{N+2}, \qquad \Delta\eta = \frac{1}{N+2}. \tag{4.32}$$

The expressions for x_θ and y_θ are determined via the adaptive grid PDEs given by [58].

$$x_\theta = \frac{1}{\tau}\left((\tilde{\omega}x_\xi)_\xi + (\tilde{\omega}x_\eta)_\eta\right), \tag{4.33}$$

$$y_\theta = \frac{1}{\tau}\left((\tilde{\omega}y_\xi)_\xi + (\tilde{\omega}y_\eta)_\eta\right). \tag{4.34}$$

Here $\tilde{\omega}$ is the filtered monitor function and τ controls the adaptivity of the grid. As $\tau \to \infty$, the grid becomes less adaptive and does not move. As in the one-dimensional case, the choice of monitor function determines the adaptivity of the grid. Again, we choose an arc-length type monitor function where $\nabla = \left(\frac{\partial}{\partial\xi}, \frac{\partial}{\partial\eta}\right)^T$,

$$\omega = \sqrt{1 + \beta\nabla u \cdot \nabla u} = \sqrt{1 + \beta\left(u_\xi^2 + u_\eta^2\right)}. \tag{4.35}$$

As in the one-dimensional case, $\beta = 0$ results in a non-adaptive fixed grid, and as we increase β, the grid is more adaptive [58]. Following the approach from [58] and [51] we can obtain a smoother change in the mesh from one time step to the next by applying a filter to the monitoring function as follows,

$$\tilde{\omega}_{i,j} = \frac{1}{4}\omega_{i,j} + \frac{1}{8}\left(\omega_{i+1,j} + \omega_{i-1,j} + \omega_{i,j+1} + \omega_{i,j-1}\right) + \frac{1}{16}\left(\omega_{i+1,j+1} + \omega_{i+1,j-1} + \omega_{i-1,j+1} + \omega_{i-1,j-1}\right). \tag{4.36}$$

Thus the coupled DEA system is,

$$u_\theta = -\frac{1}{\mathcal{J}}\left((x_\theta y_\xi - x_\eta y_\theta)u_\eta - (x_\theta y_\eta - x_\eta y_\theta)u_\xi\right) + F(u(x,y,t),x,y,t), \tag{4.37}$$

$$x_\theta = \frac{1}{\tau}\left((\tilde{\omega}x_\xi)_\xi + (\tilde{\omega}x_\eta)_\eta\right), \tag{4.38}$$

$$y_\theta = \frac{1}{\tau}\left((\tilde{\omega}y_\xi)_\xi + (\tilde{\omega}y_\eta)_\eta\right). \tag{4.39}$$

Note that on the boundaries we impose the condition that the x and y grid points remain fixed; so for $4(N+2) - 4$ equations we have,

$$x_\theta|_{0,j} = x_\theta|_{N+1,j} = y_\theta|_{i,0} = y_\theta|_{i,N+1} = 0. \tag{4.40}$$

The system of $3(N+2)^2$ equations (4.37), (4.38), (4.39) and (4.40) are solved via the Assimulo simulation package [2] on python which uses wrappers from Fortran and C. We choose to use the "Radau5ODE" integrator, a Radau IIA fifth-order three-stages with step-size control and continuous output, based on the FORTRAN code RADAU5 by E.Hairer and G.Wanner [53]. This choice was more optimal than the Sundials DEA integrator IDA used in the one-dimensional case due to Assimulo's capability to deal with larger systems more efficiently. For a numerical experiment in two dimensions with the same parameters, the IDA solver will time-out after 24 hrs, whereas Radau5ODE can reach a stable solution in less than 1hr.

Chapter 5

Adaptive Grid Solutions to Space-Fractional PDEs

Here we provide results of numerical experiments which use the L1(NU), L2(NU), L1(NU)-2D and L2(NU)-2D methods to solve space-fractional partial differential equations for $\alpha \in (0,1) \cup (1,2)$ in one and two dimensions. We consider three PDEs which will display the quality of solutions and adaptive grids.

1. Space-fractional heat equations in which we expect solutions to diffuse to the edges of the domain and lose height.

2. Space-fractional heat equations with advection term in which we expect the solution to lose magnitude while travelling in the direction of the source term.

3. Space-fractional Fisher's equations where we expect the solutions to gain height and approach the edge of the domain in a symmetric fashion.

For one-dimensional PDEs, the systems are solved via the one-dimensional adaptive grid method outlined in Chapter 4 with parameters, $\beta = 1.0,\ \kappa = 2,\ \tau = 1 \times 10^{-4}$, unless stated otherwise. We work on a domain $x \in [-1, 1]$ with the following homogeneous boundary conditions:

$$u(-1,t) = u(1,t) = 0, \qquad u_x(-1,t) = u_x(1,t) = 0. \tag{5.1}$$

For two-dimensional PDEs, the systems are solved via the two-dimensional method outlined in Section 4 with parameters $\tau = 1,\ \beta = 5$ and on an $(N+2) \times (N+2) = 31 \times 31$ grid unless stated otherwise. We work on a square domain $(x, y) \in [a, b] \times [a, c] = [-1, 1] \times [-1, 1],\ t \geq 0$ together with boundary conditions:

$$u(a,y,t) = u(b,y,t) = 0, \qquad u_x(a,y,t) = u_x(b,y,t) = 0, \tag{5.2}$$

$$u(x,a,t) = u(x,c,t) = 0, \qquad u_y(x,a,t) = u_y(x,c,t) = 0. \tag{5.3}$$

All figures are clearly referenced and most are included within this chapter, on occasion we will refer to Appendix A where the supplementary figures show solutions at a greater number of time points, and also some examples of cases in which solutions are almost identical. The code used for these numerical experiments is shown in Appendix B.

5.1 One dimension

There were some problems regarding stability of solutions when implementing the L1(NU) methods for $\alpha \in (0, 1)$. These issues can be avoided if we take the number of grid points to be very large ($N > 199$) or limit the maximum time step that the solver can use. The run time for large N or small time step sizes is very long so we only show solutions using the L1(NU) method on few occasions. It is suspected that these stability issues could be improved by choosing a more stable method than forward differences for the approximation of the first derivative in the definitions for the left and right Caputo derivatives when deriving the L1(NU) methods.

5.1.1 Space-fractional heat equations

In this subsection we consider space-fractional heat equations in one dimension. We consider three different PDEs, one involving only the left space-fractional derivative, one with the right space-fractional derivative, and one with the space-fractional Laplacian. It is interesting to consider the left and right space-fractional heat equations because we can observe limiting behaviour as $\alpha \to 1$ in which the PDEs approach the transport equation.

Left space-fractional heat equation

The left space-fractional heat equation is given by,

$$\frac{\partial}{\partial t} u(x,t) = d \; {}_C D^{\alpha}_{a,x} u(x,t). \tag{5.4}$$

We take the diffusion coefficient $d = 1$ and for the initial condition we take the Gaussian function,

$$u(x,0) = \exp\left(-50x^2\right). \tag{5.5}$$

Figure 5.1 shows how the solution of PDE (5.4) differs for different choices of α. We can see that as α increases, the magnitude of the solution decreases at a faster rate and the tails become thicker. For instance at time $t = 0.025$, the solution for $\alpha = 1.75$ (shown in red) has a peak at $u(x, 0.025) \approx 0.5$, whereas the solution for $\alpha = 1.25$ (shown in green) has a peak at $u(x, 0.025) \approx 0.85$. At the same time we can see that the solution for $\alpha = 1.75$ also has longer and thinner tails than the solution for $\alpha = 1.25$. For more solutions and adaptive grids to PDE (5.4) for $\alpha = 1.25,\ 1.5$ and 1.75 from times $t = 0$ to $t = 0.25$ see A.1.

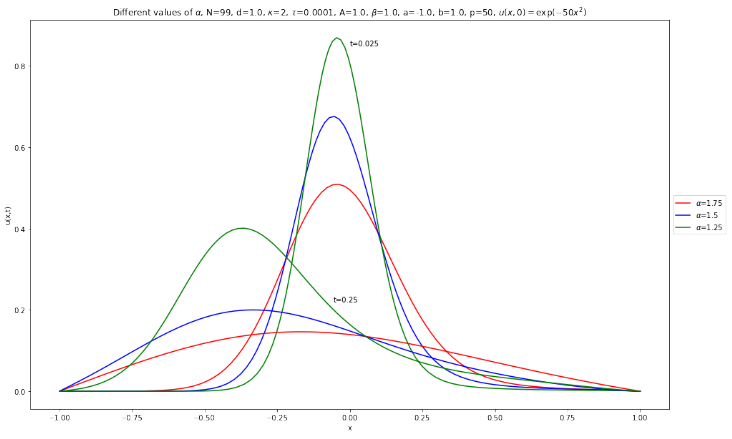
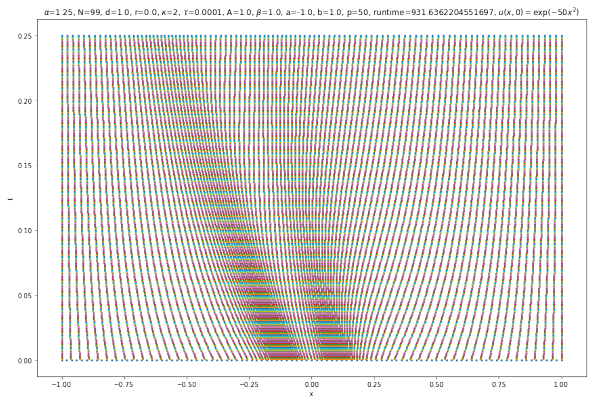
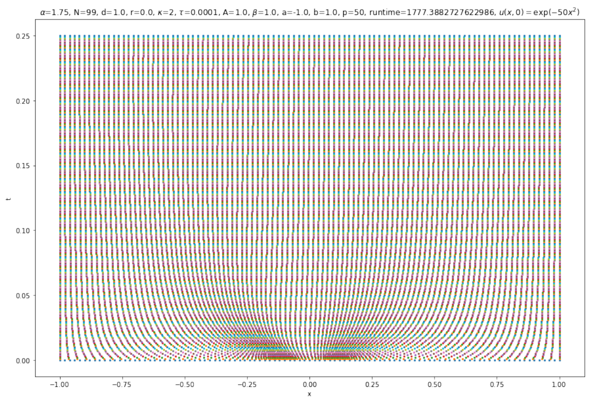


Figure 5.1: Differences to the solution of (5.4) with different values of α. Solutions for $\alpha = 1.75,\ 1.5$ and 1.25 are shown in red, blue and green respectively at times $t = 0.025$ and $t = 0.25$.

Figure 5.2 shows how the adaptive grids for the solutions of PDE (5.4) differ for different choices of α. For a higher value of $\alpha = 1.75$, the adaptive grid reflects the fast diffusive behaviour. At time $t = 0$ the grid is very dense near the center of the domain where the solution is most steep, then as the solution diffuses quickly and soon loses magnitude, the grid also quickly becomes a uniform grid. For a smaller value of $\alpha = 1.25$, the solution diffuses less quickly and the movement of the solution to the left hand side of the domain is more pronounced. The adaptive grid moves to where the solution is steepest so we observe the dense areas of grid points shift to the left as time increases from zero. For later times we see



(a) Adaptive grid for $\alpha = 1.25$



(b) Adaptive grid for $\alpha = 1.75$

Figure 5.2: A comparison of the adaptive grids for solutions to the left space-fractional heat equation (5.4) generated using the L2(NU) method with $\alpha = 1.25$ and $\alpha = 1.75$. Time is shown on the y-axis.

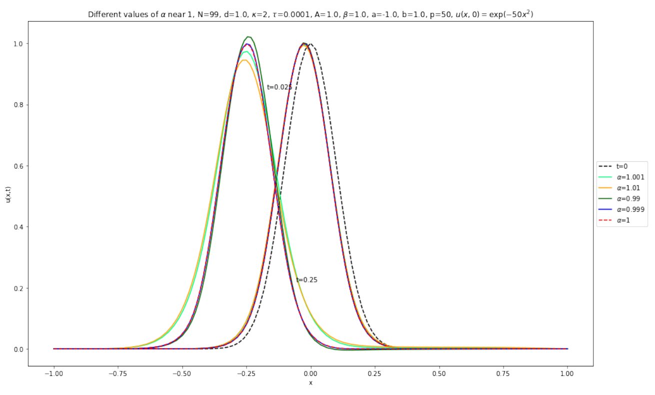
that this grid also approaches a uniform grid as the solution becomes more flat.

We now consider the case where we take α very close to one and compare it with the solution to the transport equation,

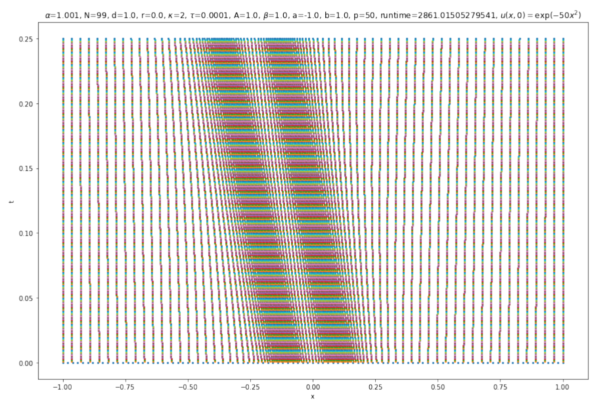
$$\frac{\partial u}{\partial t} = \frac{\partial u}{\partial x}. \tag{5.6}$$

We should note that whilst the Caputo and Riemann-Liouville derivatives are defined for all α, the L2(NU) method for approximating the right or left derivatives applies only to $\alpha \in (1, 2]$. The case for $\alpha = 1$ may be more accurately modelled using the L1(NU) method which is valid for $\alpha \in (0, 1]$. Note that the integer 1 is only contained within these intervals while discussing the Caputo and Riemann-Lioville derivatives, not the Riesz derivative.

Figure 5.3 shows the comparison between the solution to PDE (5.4) compared to PDE (5.6) which is labelled as $\alpha = 1$. We can see that as we take values of α closer to 1, the solution becomes closer to that of the transport equation. The solution to PDE (5.4) with $\alpha = 0.999$ is also shown which was solved using the L1(NU) method, and this solution already almost exactly agrees with the transport equation. The adaptive grid for the case where $\alpha = 1.001$ is also shown.



(a)



(b)

Figure 5.3: A comparison to the transport equation (5.6): Solutions to (5.4) near $\alpha = 1$. The dashed black line represents the initial condition and the solutions for $\alpha = 1.001,\ 1.01,\ 0.99,\ 0.999$ and 1 are shown in light-green, orange, dark-green, blue and red respectively for times $t = 0.025$ and $t = 0.25$. The adaptive grid is shown for $\alpha = 1.001$.

Right space-fractional heat equation

The right space-fractional heat equation is given by,

$$\frac{\partial}{\partial t}u(x,t) = d\ {}_C D^{\alpha}_{x,b} u(x,t). \tag{5.7}$$

Again we take $d = 1$ and same initial condition as before given by equation (5.5).

This PDE behaves in a symmetrical mirrored fashion when compared to the left space-fractional heat equation. The solution profile diffuses while moving to the right hand side of the domain, for higher values of α this diffusive process is faster than for smaller α. Figure 5.4 shows the solutions and adaptive grids to PDE 5.7 for $\alpha = 1.25,\ \alpha = 1.5,\ \alpha = 1.75$. When we compare these to Figures 5.1 and 5.2 we can see that they are mirror images of one another. For full solutions and adaptive grids for each choice of α from $t = 0$ to $t = 0.25$ see Figure A.2.

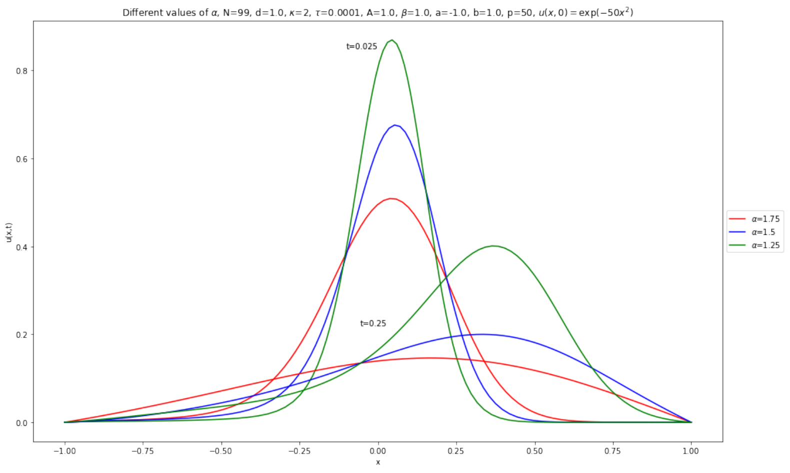
We now consider the case where we take α very close to one and compare it with the solution to the transport equation,

$$\frac{\partial u}{\partial t} = -\frac{\partial u}{\partial x}. \tag{5.8}$$

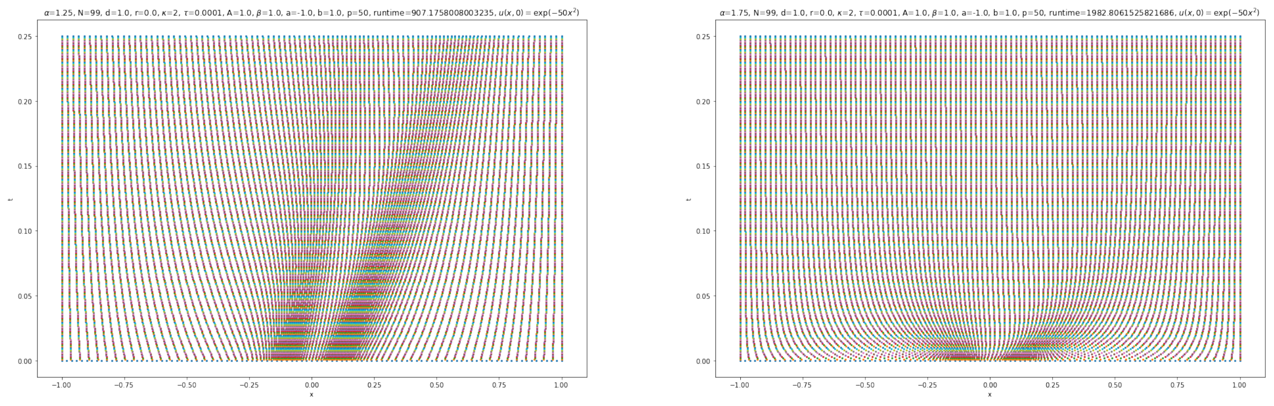
Figure 5.5 shows the comparison between the solution to PDE (5.7) compared to PDE (5.8) which is labelled as $\alpha = 1$. We can see that as we take values of α closer to 1, the solution becomes closer to that of the transport equation. The solution to PDE (5.7) with $\alpha = 0.99$ is also shown which was solved using the L1(NU) method, and this solution already almost exactly agrees with the transport equation. The adaptive grid for the case where $\alpha = 1.001$ is also shown.

Convergence with increasing number of grid points

All numerical presented experiments so far have been run with $N + 2 = 101$ grid points. This number was chosen as it was high enough to get accurate, smooth solutions, but also low enough that the run time was kept to manageable times. We will now show that as we increase the number of grid points, N, the solution converges to one solution. The left and right space-fractional derivatives are symmetrical so we will just show this convergence for the right derivative. Figure 5.6 shows how with increasing N,

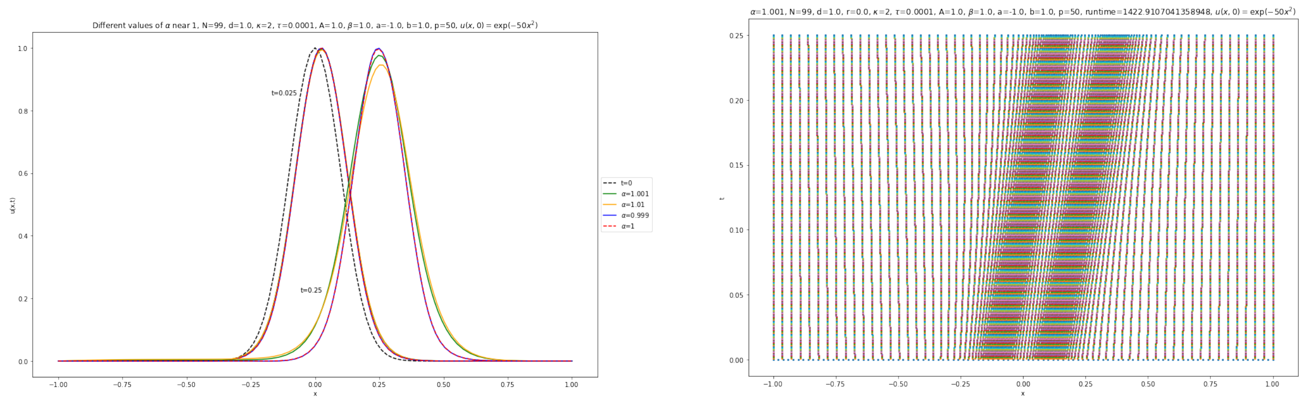


(a) Differences to the solution of (5.7) with different values of α. $\alpha = 1.75,\ 1.5$ and 1.25 are shown in red, blue and green respectively.



(b) $\alpha = 1.25$ **(c)** $\alpha = 1.75$

Figure 5.4: (a) shows the solutions to PDE (5.7) at $t = 0.025$ and $t = 0.25$ for $\alpha = 1.25,\ 1.5,\ 1.75$. Adaptive grids for the solution of (5.7) are shown for (b) $\alpha = 1.25$ and (c) $\alpha = 1.75$.



(a) **(b)**

Figure 5.5: A comparison to the transport equation (5.8): Solutions to (5.7) near $\alpha = 1$.

the solution of PDE (5.7) converges to one solution, we show this at time 0.025 and 0.25 with the same initial condition (5.5). There is very little difference between the blue and green solutions ($N = 99$ and $N = 199$ respectively).

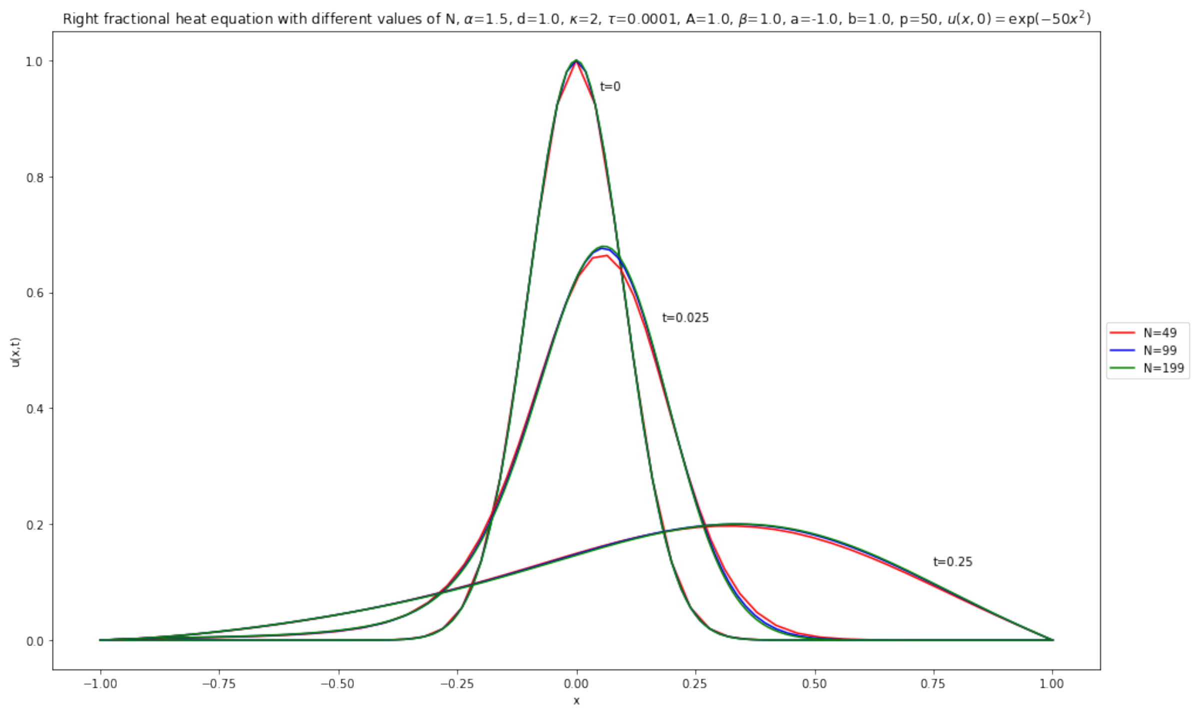


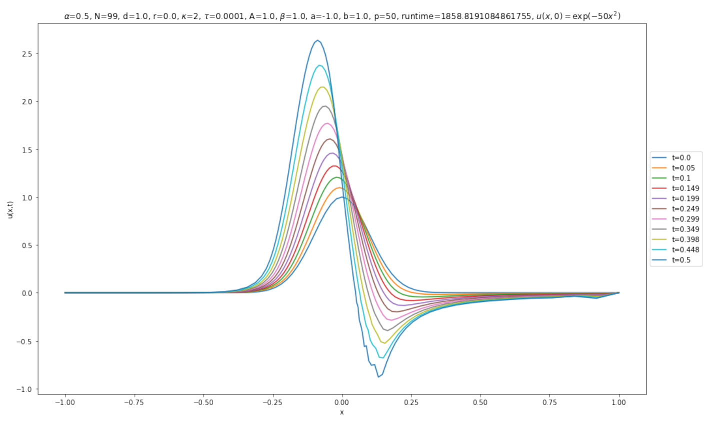
Figure 5.6: Showing the convergence of solutions to PDE (5.7) with $\alpha = 1.5$ as the value of N increases. $N = 49,\ 99$ and 199 are shown in red, blue and green respectively.

Left and right space-fractional heat equations: $\alpha \in (0, 1)$, L1(NU)

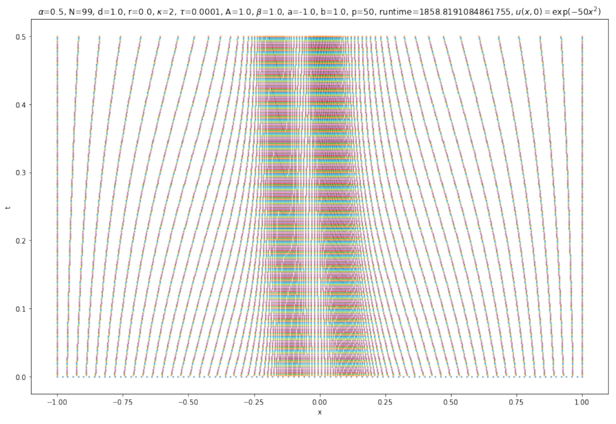
We observe some instability when applying the L1(NU) method to PDEs (5.4) and (5.7). This is perhaps due to the nature of the solutions which become very steep and negative in some parts of the domain as seen in Figure 5.7 for PDE (5.4), the solutions are a mirror image for PDE (5.7). The solutions are generated using the L1(NU) method for $\alpha = 0.5$. These stability issues can be reduced by decreasing the adaptivity of the grid, i.e by increasing $\tau = 1 \times 10^{-4}$ to 1×10^{-3}. We can also increase the number of grid points from $N = 49$ to 99. The results of these changes are also seen in Figure 5.7. When using the usual arc-length monitor function, the adaptive grid doesn't seem to move to become dense at the negative part of the solution with positive gradient; this unstable behaviour is similar to that discussed in [35]. In this paper they are able to improve results by taking a larger number of grid points, and also considering a curvature type monitor function. In Figures 5.7e and 5.7f we consider a curvature type monitor function,

$$M = \sqrt{1 + \left|\frac{\partial^2 u}{\partial x^2}\right|}. \tag{5.9}$$

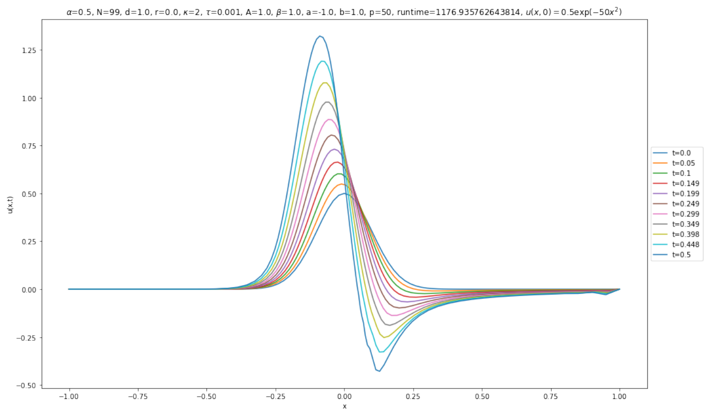
We can see that, although the solution is still unstable, the adaptive grid now considers the curvature of the solution so we also get dense grid points near the minimum of the solution on the right hand side of the domain. This is in contrast to the solutions where we use the arc-length type monitor function where, as the positive peak is more steep than the negative trough, the grid points cluster around the peak on the left side of the domain, leading to unstable solutions near the negative trough on the right hand side of the domain.



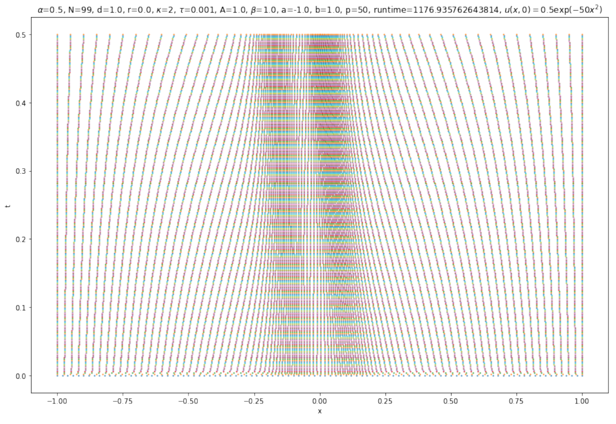
(a) $N = 99,\ \tau = 1 \times 10^{-4}$



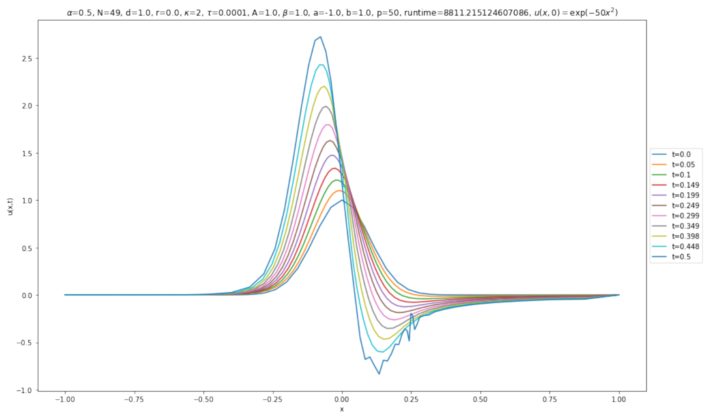
(b) $N = 99,\ \tau = 1 \times 10^{-4}$



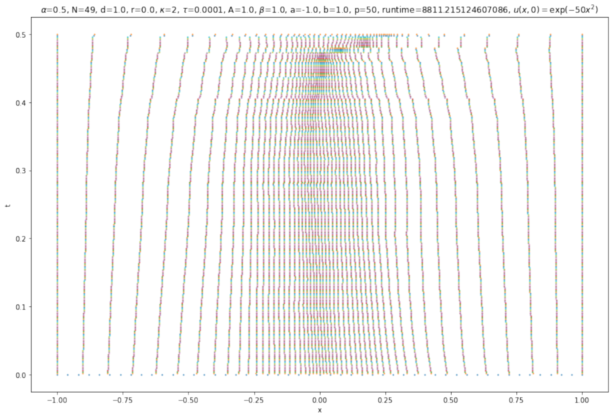
(c) $N = 99,\ \tau = 1 \times 10^{-3}$



(d) $N = 99,\ \tau = 1 \times 10^{-3}$



(e) $N = 49,\ \tau = 1 \times 10^{-3}$ and curvature monitor function



(f) $N = 49,\ \tau = 1 \times 10^{-3}$ and curvature monitor function

Figure 5.7: Solutions and adaptive grids for PDE (5.4) for $\alpha = 0.5$ with different choices for N, τ and monitor function. (a), (b), (c) and (d) use the usual arc-length monitor function whereas (e) and (f) use curvature monitor function. For (a), (c) and (e) solutions are shown at times $t = 0,\ 0.05,\ 0.1,\ 0.149,\ 1.99,\ 0.249,\ 0.349,\ 0.398,\ 0.448$ and 0.5 in dark-blue, orange, green, red, purple, brown, pink, grey, light-green, light-blue and blue respectively.

Space-fractional heat equation

The space-fractional heat equation with the fractional Laplacian is given by,

$$\frac{\partial}{\partial t}u(x,t) = -d\left(-\frac{\partial^2}{\partial x^2}\right)^{\alpha/2} u(x,t). \tag{5.10}$$

Again we take $d = 1$ and the same initial condition given by equation (5.5). This time the solutions to the PDE are symmetric and the solution diffuses without being skewed to the left or right.

We can observe the same patterns in behaviour that for increasing values of α, the solution diffuses and loses magnitude more quickly, as well as having thicker tails. Figure 5.8 shows the solutions for $\alpha \in (1,2)$ computed using the L2(NU) method at $t = 0.025$ and $t = 0.25$. See figure A.3 for the full solutions of PDE (5.10) for $\alpha = 1.25,\ 1.5,\ 1.75$ from $t = 0$ to $t = 0.25$ and the adaptive grids. Figure 5.9 shows the solution and adaptive grid for $\alpha = 0.5$ at times $t = 0,\ t = 0.164, 0.332$ and 0.5. The solutions on uniform grids where $N = 99$ and the L1 and Grünwald-Letnikov methods are also shown. We can see that these solutions are in agreement so the instability of the L1(NU) method with the adaptive mesh equations has not impacted the result.

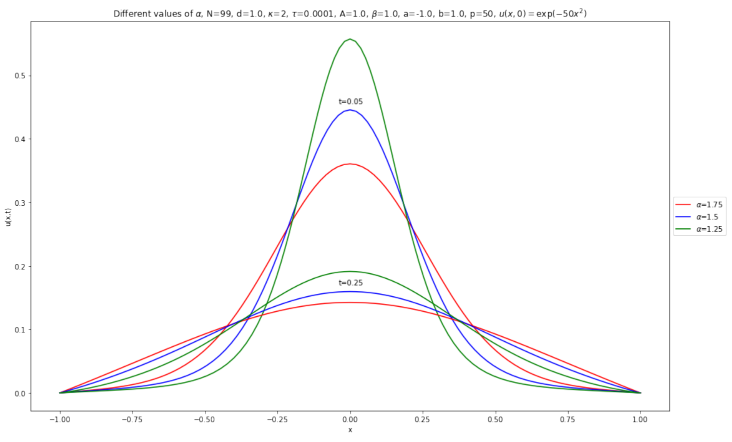


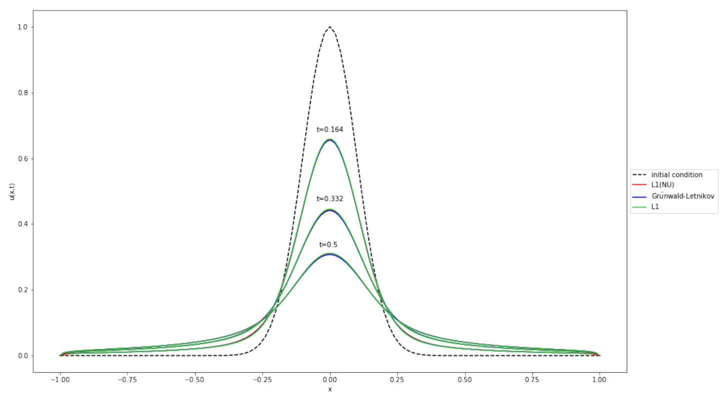
Figure 5.8: Differences to the solution of (5.10) with different values of $\alpha \in (1,2)$. Solutions for $\alpha = 1.75,\ 1.5$ and 1.25 are shown in red, green and blue respectively for times $t = 0.05$ and 0.25.

We can also compare the solutions to PDE (5.10) in the limiting case $\alpha = 1.99$, with the solution to the ordinary heat equation where $\alpha = 2$ and the central difference approximation (1.36) is used to approximate the ordinary Laplacian. Figure 5.10 shows that solutions for $\alpha = 1.99$ approaches the solution for $\alpha = 2$ which shows that the property in equation (1.25) holds for this numerical method as expected. To see the adaptive grids and solutions at more time points which also agree see Figure A.4.

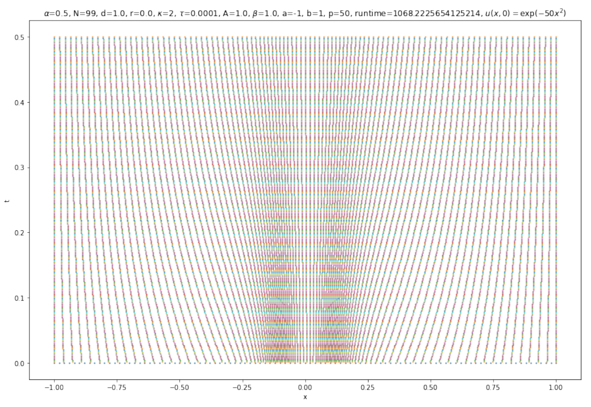
5.1.2 Space-fractional Fisher's equation

In this subsection we consider the space-fractional Fisher's equations in one dimension. We again consider three different PDEs, one involving only the left space-fractional derivative, one with the right space-fractional derivative, and one with the space-fractional Laplacian. In all cases we take the diffusion coefficient $d = 0.05$ and the Fisher's constant $r = 5$. We also always use the same Gaussian type initial condition,

$$u(0,x) = \frac{1}{2}\exp\left(-50x^2\right). \tag{5.11}$$



(a)



(b)

Figure 5.9: (a) shows the solution of (5.10) for $\alpha = 0.5$ generated using the L1(NU), L1 and Grünwald-Letnikov methods (shown in red, green and blue respectively). The dashed black line shows the initial condition and the solutions are shown at times $t = 0.164,\ 0.332$ and 0.5. (b) shows the adaptive grid for the L1(NU) method.

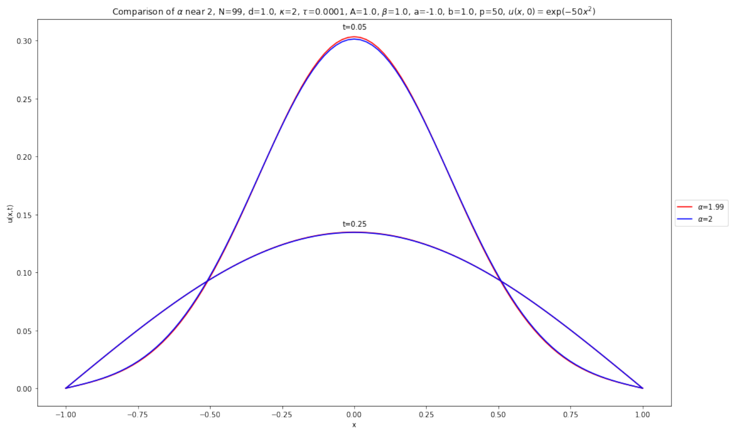


Figure 5.10: A comparison of the solutions generated using the L2(NU) method with $\alpha = 1.99$ (red), with the results where $\alpha = 2$ (blue) and we use the standard central difference approximation for the second derivative of the ordinary Laplacian.

Left space-fractional heat equation with Fisher's term

The left space-fractional heat equation with Fisher's term is given by,

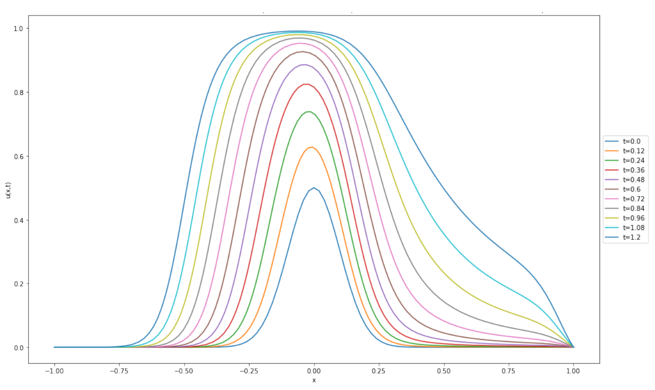
$$\frac{\partial}{\partial t}u(x,t) = d\ {}_C D^{\alpha}_{a,x} u(x,t) + r u(x,t)(1-u(x,t)). \tag{5.12}$$

Figure 5.11 shows how the solution of PDE (5.12) differs for different choices of α. We can see that as α increases, the solution reaches the boundary faster, but takes more time to reach the solution's limiting height of 1. As the diffusive term includes only the left fractional derivative, the solution is also skewed to the left boundary of the domain and thus the solution on the left hand side is steeper than the right hand side. We can see that this behaviour is reflected in the adaptive grids as there is a higher density cluster of grid points on the left sloping side of the solution compared to the right.

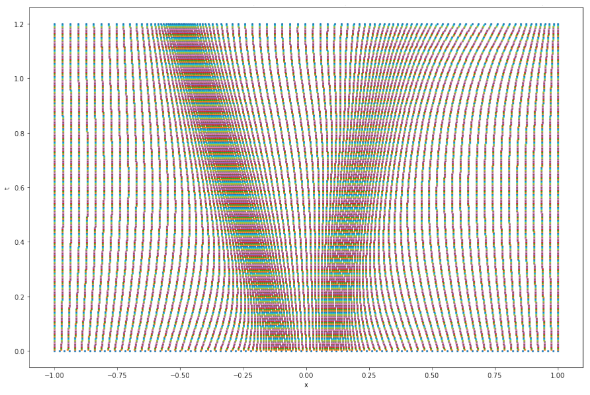
Right space-fractional heat equation with Fisher's term

The right space-fractional heat equation with Fisher's term is given by,

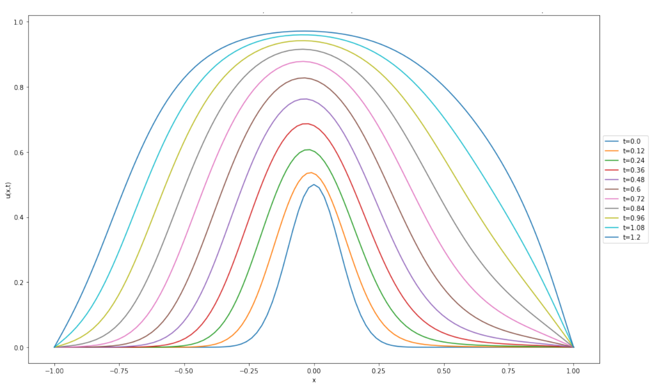
$$\frac{\partial}{\partial t}u(x,t) = d\ {}_C D^{\alpha}_{x,b} u(x,t) + r u(x,t)(1-u(x,t)). \tag{5.13}$$



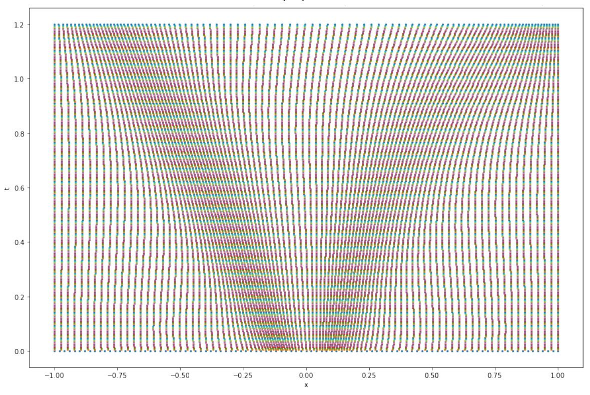
(a)



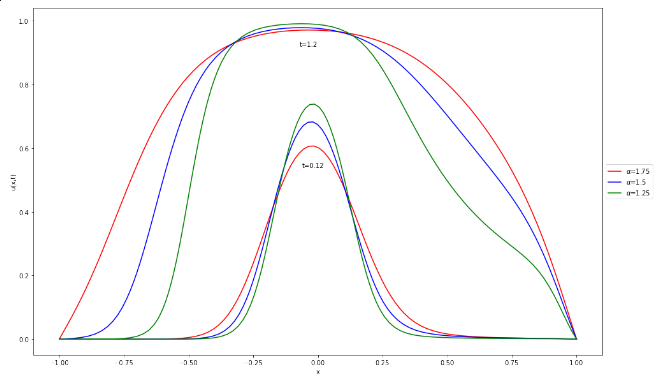
(b)



(c)



(d)



(e)

Figure 5.11: A comparison of the results to the left space-fractional heat equation with Fisher's term generated using the L2(NU) method. Solutions with $\alpha = 1.25$ and 1.75 are shown in (a) and (c) at times $t = 0,\ 0.12,\ 0.24,\ 0.36,\ 0.48,\ 0.6,\ 0.72,\ 0.84,\ 0.96,\ 1.08$ and 1.2 shown in dark-blue, orange, green, red, purple, brown, pink, grey, light-green, light-blue and blue respectively. The respective adaptive grids in (b) and (d). (e) shows the differences in solutions at time points $t = 0.025$ and $t = 0.25$ for $\alpha = 1.25$ (green), $\alpha = 1.5$ (blue) and $\alpha = 1.75$ (red).

The solutions are a mirror image of those of the left space-fractional heat equation with Fisher's term (5.12) and this time the solutions are skewed to the right, see Figure A.5.

Convergence with increasing number of grid points

The L2(NU) method also performs well when we increase the number of grid points N for the space-fractional Fisher's equations. Figure 5.12 shows how the solution of PDE (5.13) changes as we increase

N. We can see that as we increase N the solutions begin to converge. For early points in time, $t = 0.12$, the solutions for $N = 99$ and $N = 199$ almost exactly agree, whereas for later points in time $t = 1.2$ there is a more noticeable difference, but the solution is converging to a more accurate solution as we increase N.

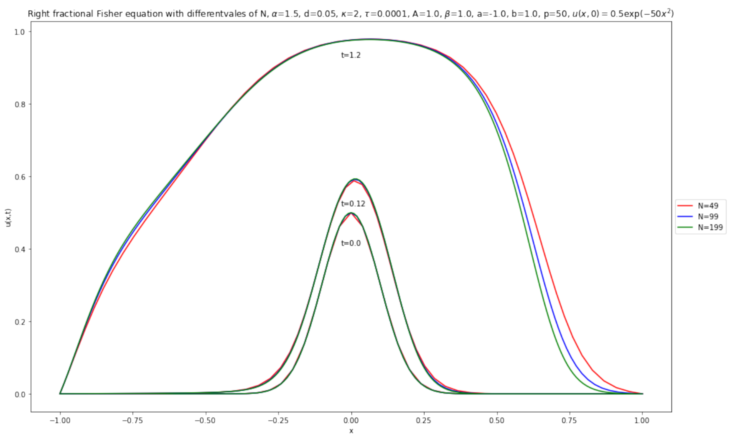


Figure 5.12: The solution to PDE (5.13) with $\alpha = 1.5$ for different choices for N. $N = 49,\ 99$ and 199 are shown in red, blue and green respectively.

Heat equation with space-fractional Laplacian and Fisher's term

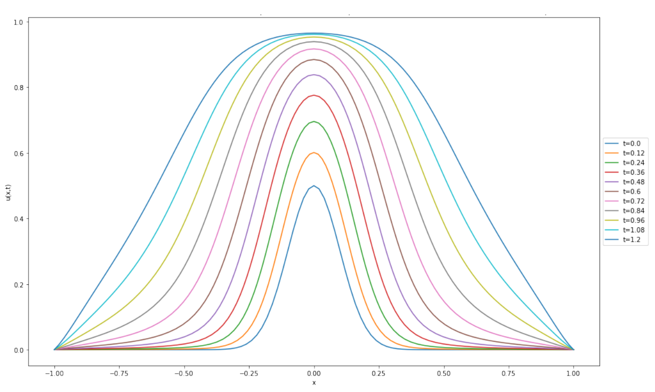
The heat equation with space-fractional Laplacian and Fisher's term is given by,

$$\frac{\partial}{\partial t}u(x,t) = -d\left(-\frac{\partial}{\partial x^2}\right)^{\alpha/2} u(x,t) + ru(x,t)(1-u(x,t)). \tag{5.14}$$

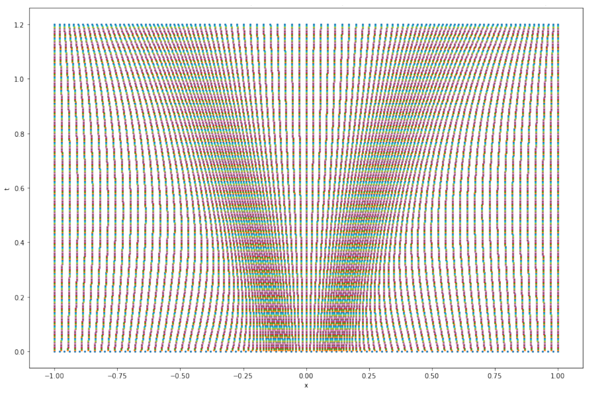
The solutions to this equation are symmetrical as we now include the fractional Laplacian which is a weighted average of the left and right fractional derivatives, and we take the same symmetric Gaussian initial condition (5.11). As before, for higher values of α, the solution diffuses to reach the boundaries of the domain faster, but takes more time to reach the limiting magnitude of $u(x,t) = 1$. For the solutions and adaptive grids for PDE (5.14) for $\alpha = 1.25,\ 1.5$ and 1.75 which display this behaviour, see Figure 5.13.

We see some of the effects of the unstable L1(NU) method when considering the space-fractional Fisher's equation. The solution is symmetric but it takes slightly longer for the grid to adjust to ensure smooth solutions near the left boundary. We can see this clearly in Figure 5.14 where the solution to PDE (5.14) where $\alpha = 0.5$ is shown for $N = 199$ and adaptive grids for $N = 99,\ 149,\ 199$ till $t = 1.5$. Note here we used $\tau = 0.001$ to reduce the speed at which the grid could adapt in an attempt to reduce stability errors. In all cases the adaptive grid ensures there is a smooth solution at the boundary on the left hand side before the right hand side.

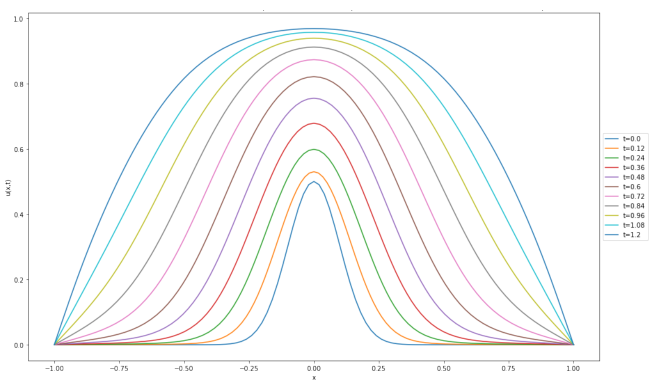
We can also consider again the limiting case as $\alpha \to 2$ to check if the property in equation (1.25) holds. Figure 5.15 shows that the solution to PDE (5.14) where $\alpha = 1.99$ and the L2(NU) method is used to discretise the RHS is very close to the solution of the ordinary Fisher's equation when $\alpha = 2$ and we have the ordinary Laplacian approximated via central differences (1.36). The adaptive grids are almost also identical.



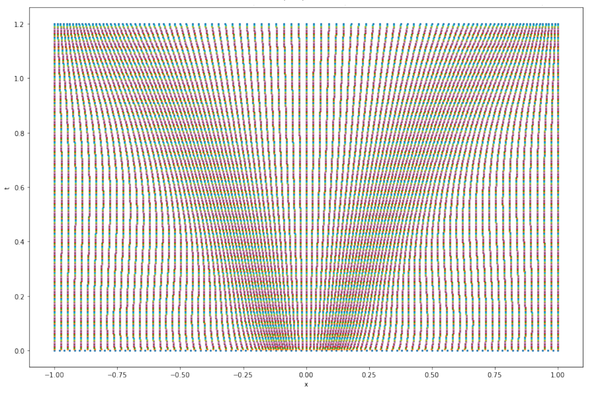
(a)



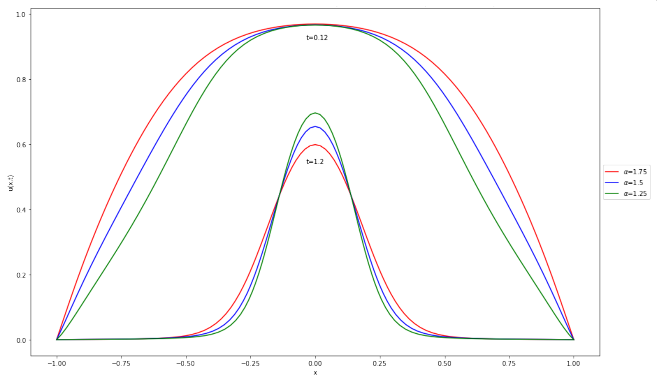
(b)



(c)

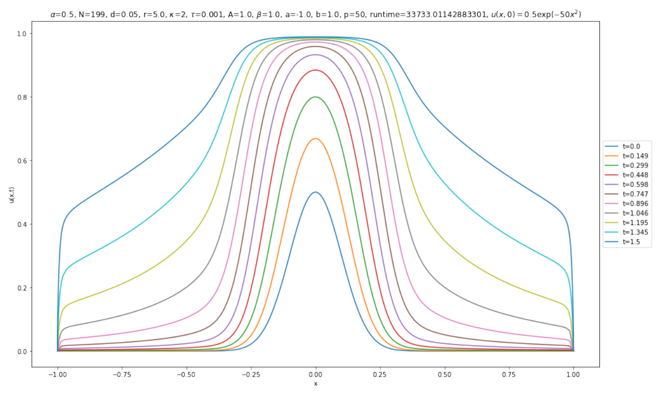


(d)

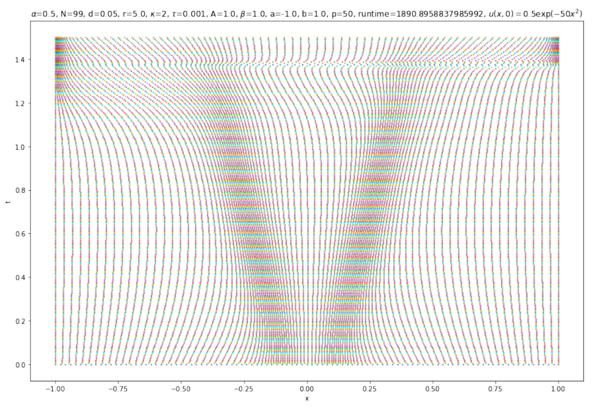


(e) Differences to the solution of (5.14) with different values of α.

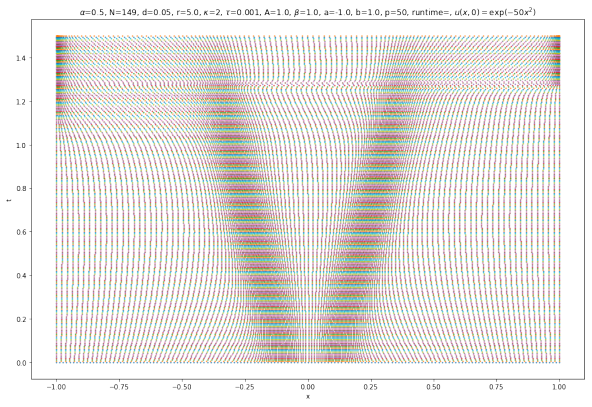
Figure 5.13: A comparison of the results to the space-fractional heat equation with Fisher's term, equation (5.14), generated using the L2(NU) method with $\alpha = 1.25,\ 1.5,\ 1.75$. (a) and (c) show the solutions for $\alpha = 1.25$ and $\alpha = 1.75$ respectively at times $t = 0,\ 0.12,\ 0.24,\ 0.36,\ 0.48,\ 0.6,\ 0.72,\ 0.84,\ 0.96,\ 1.08$ and 1.2. (b) and (d) show the adaptive grids for $\alpha = 1.25$ and $\alpha = 1.75$ respectively. (e) shows how the solutions differ at times $t = 0.12$ and $t = 1.2$ for $\alpha = 1.25,\ 1.5$ and 1.75 shown in green, blue and red respectively.



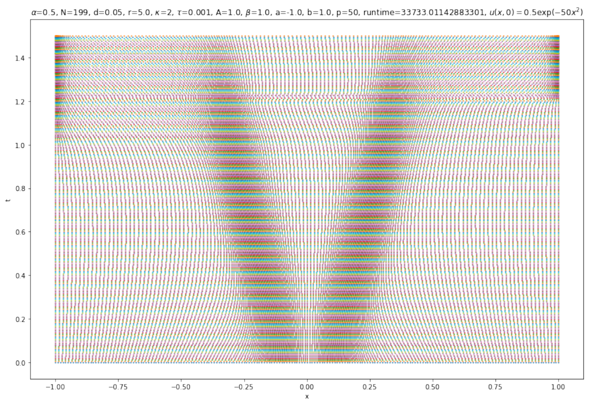
(a) Solutions where $N = 199$.



(b) $N = 99$

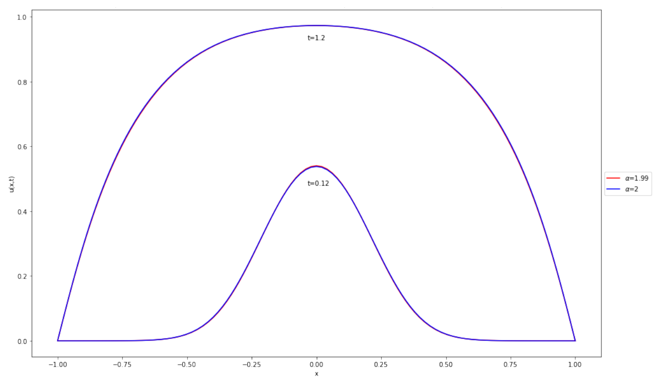


(c) $N = 149$

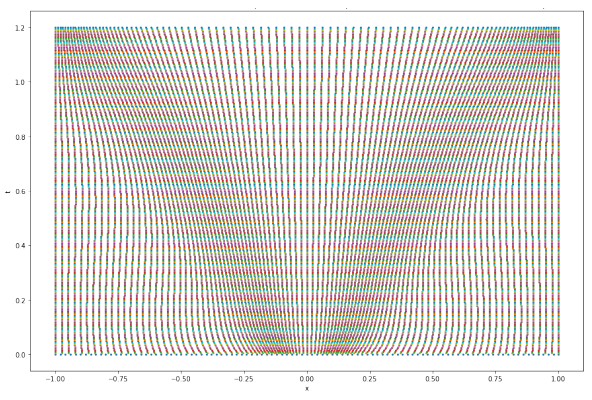


(d) $N = 199$

Figure 5.14: Solutions to PDE (5.14) with $\alpha = 0.5$ generated using the L1(NU) method at times $t = 0,\ 0.149,\ 0.299,\ 0.448,\ 0.598,\ 0.747,\ 0.896,\ 1.049,\ 1.195,\ 1.345$ and 1.5 shown in dark-blue, orange, green, red, purple, brown, pink, grey, light-green, light-blue and blue respectively. The adaptive grids are shown for different values of N.



(a)



(b)

Figure 5.15: A comparison of the results generated using the L2(NU) method with $\alpha = 1.99$ (red), with the results where $\alpha = 2$ (blue) and we use the normal central difference approximation for the second derivative of the ordinary Laplacian. The solutions and adaptive grids are identical.

5.2 Two dimensions

5.2.1 Space-fractional heat equation

In this subsection we consider the space-fractional heat equation in two dimensions,

$$u_t(x,y,t) = -\left(-\Delta\right)^{\alpha/2} u(x,y,t) \tag{5.15}$$

for $\alpha \in (1,2)$, $(x,y) \in [a,b] \times [a,c] = [-1,1] \times [-1,1]$, $t \geq 0$ together with boundary conditions,

$$u(a,y,t) = u(b,y,t) = 0, \qquad u_x(a,y,t) = u_x(b,y,t) = 0, \tag{5.16}$$

$$u(x,a,t) = u(x,c,t) = 0, \qquad u_y(x,a,t) = u_y(x,c,t) = 0, \tag{5.17}$$

and Gaussian type initial condition,

$$u(x,y,0) = \exp\left(-10\left(x^2+y^2\right)\right). \tag{5.18}$$

The space-fractional derivative on the RHS of PDE (5.15) is approximated using the L2(NU)-2D method equation (3.7) for $\alpha \in (1,2)$ and using the L1(NU)-2D method equation (3.4) for $\alpha \in (0,1)$. The system is solved via the two-dimensional method outlined in Section 4 with parameters $\tau = 1$, $\beta = 5$ and on an $(N+2) \times (N+2) = 31 \times 31$ grid unless stated otherwise.

Figure 5.16 shows the solutions and adaptive grids to PDE (5.15) for $\alpha = 0.5$ from time $t = 0$ till $t = 0.18$ seconds. We can see that the long tails that the fractional Laplacian term produces means that we observe a diamond shaped solution at longer times rather than a circular one as the solution is affected by the boundary conditions.

Figure 5.17 shows the solutions to PDE (5.15) for $\alpha = 1.5$ from time $t = 0$ till $t = 0.045$ seconds. We see that we can observe sub-diffusive behaviour as expected and the grid adapts well to the solution, with a higher density of points at steeper parts of the solution. When comparing Figures 5.17b and 5.17c we can see that by increasing β the grid becomes more adaptive. The choice of $\beta = 5$ was found to be optimal as it produced satisfying adaptivity results while keeping the run-time of the numerical experiments manageable. Changing the value of β doesn't change the behaviour of the solution, see Figure A.6.

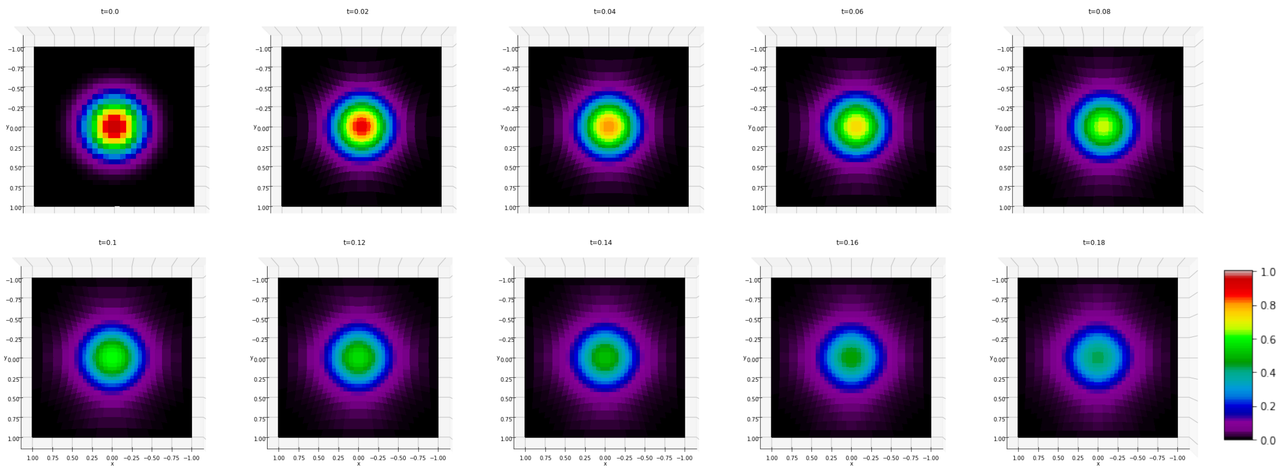
We can see the differences in the solution to PDE (5.15) for different values of α in Figure 5.18 where we see the solutions and grids for $\alpha = 1.25$ and 1.75 at time 0.02. We can see that the solution takes longer to diffuse outwards for smaller α, and the grid points are therefore more dense around the center of the domain in this case when compared to a higher value of α. For $\alpha = 1.75$, the grid points have already started to spread out closer to the boundaries of the domain as the solution is less steep in the center and has spread closer to the boundaries.

We also consider the limiting case as $\alpha \to 2$ and the results behave as expected as we observe no significant visible difference between the solutions or adaptive grids for PDE (5.15) for $\alpha = 1.99$ compared to the ordinary heat equation,

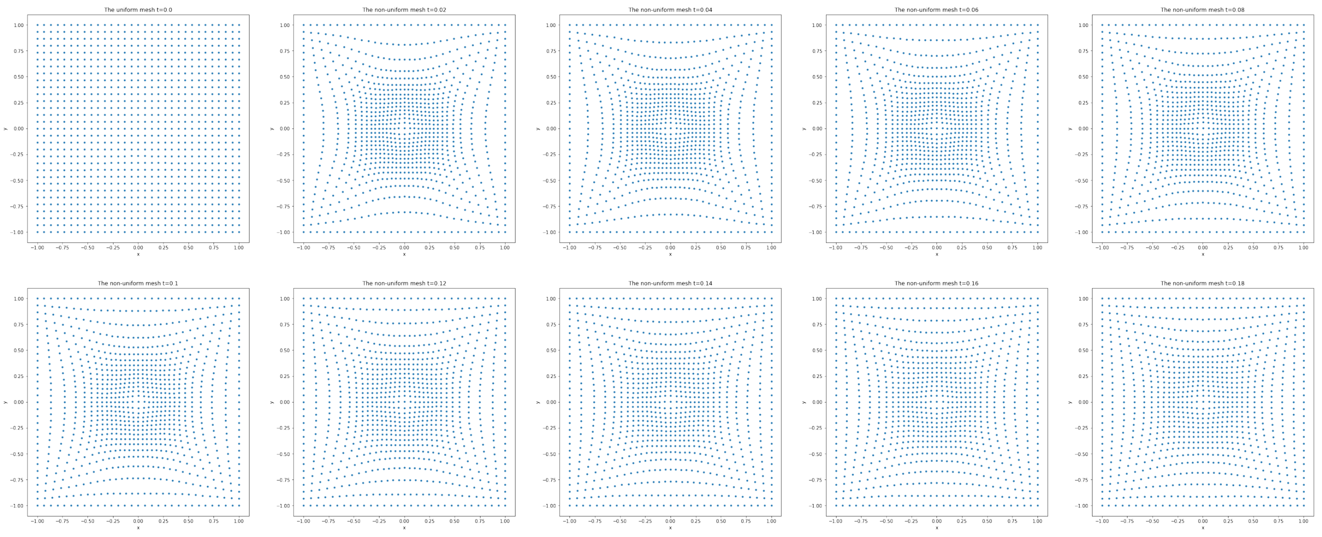
$$u_t(x,y,t) = u_{xx}(x,y,t), \tag{5.19}$$

where we use the central difference approximation given in equation (1.40) to discretise the RHS. This agrees with the expected behaviour that as $\alpha \to 2$ the fractional Laplacian becomes the ordinary Laplacian. See Figure A.7.

Figure 5.19 shows the adaptive grid evolution over a longer time for $\alpha = 1.99$. We take the final time to be $t = 0.3$ and thus the solution at this point is almost zero everywhere. The grid behaves as expected and has returned to the initial uniform form.

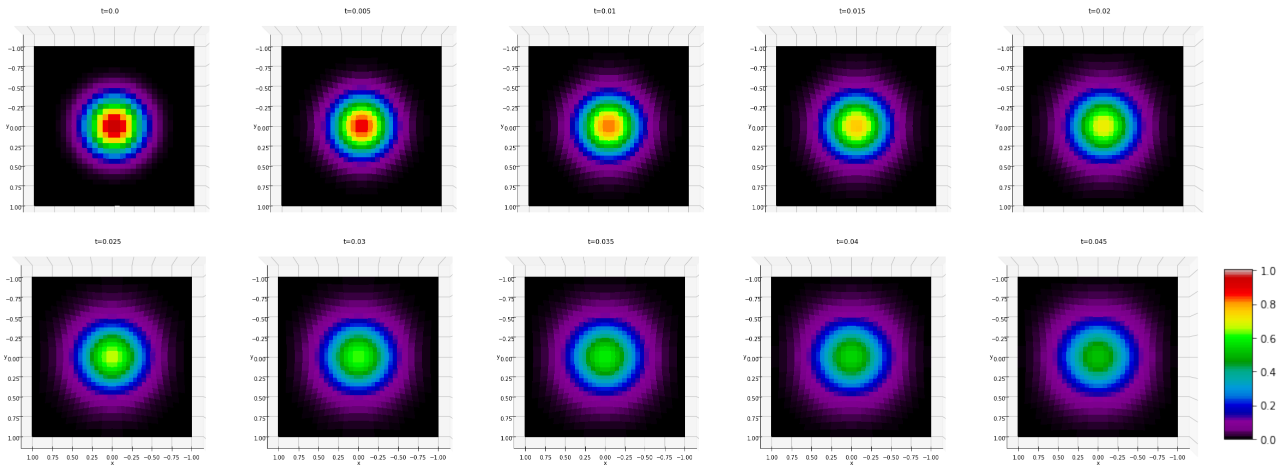


(a) Solutions to the space-fractional heat equation, $\alpha = 1.5$, $\tau = 1$, $\beta = 5$

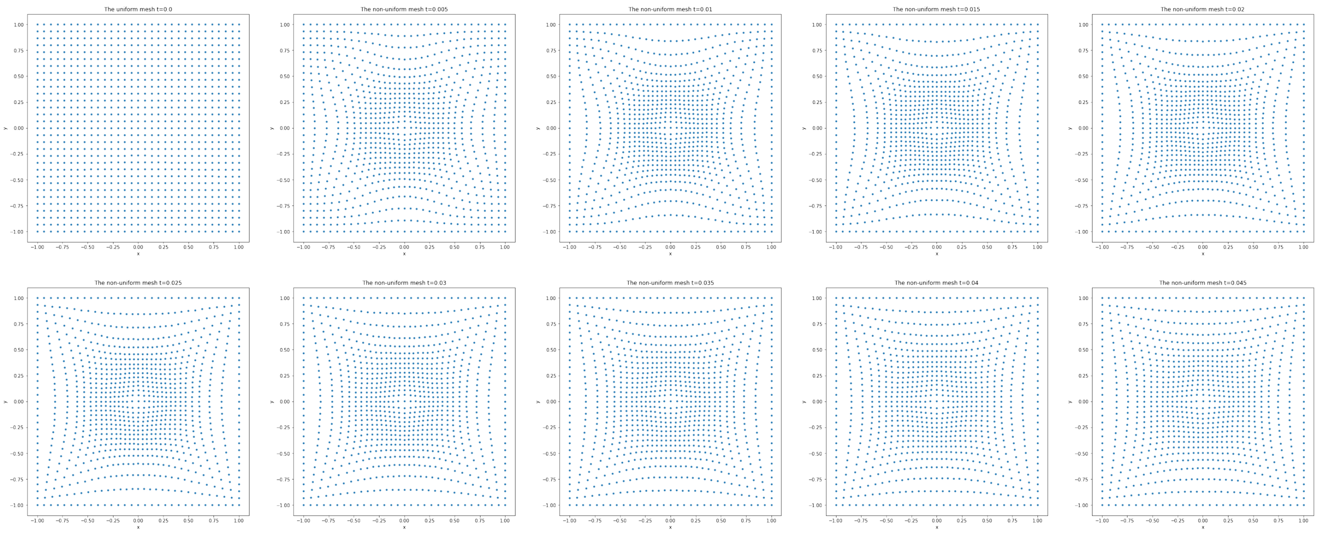


(b) Adaptive grids with $\beta = 5$ for the space-fractional heat equation

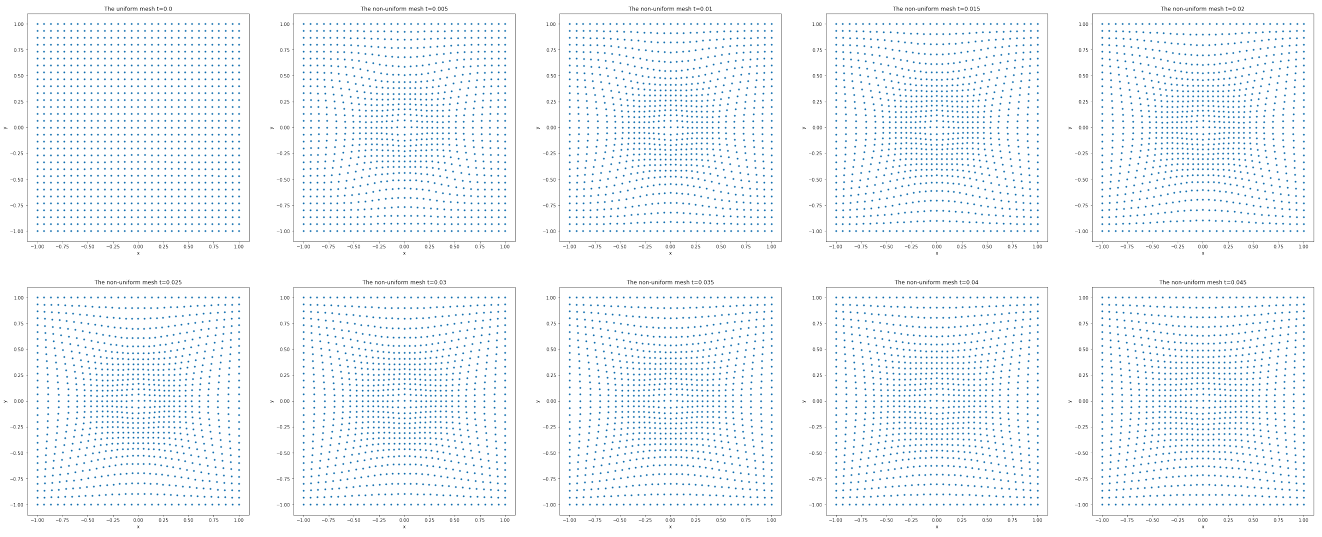
Figure 5.16: Solutions and grids to the space-fractional heat equation, $\alpha = 0.5$, from time $t = 0$ till $t = 0.18$ seconds. Adaptive grid parameters $\tau = 1$, $\beta = 1$ or 5, and an $(N+2) \times (N+2) = 31 \times 31$ grid.



(a) Solutions to the space-fractional heat equation, $\alpha = 1.5$, $\tau = 1$, $\beta = 5$

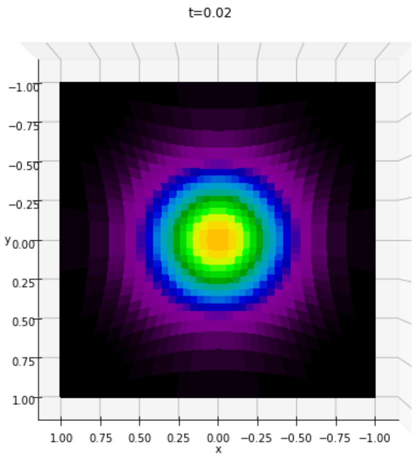


(b) Adaptive grids with $\beta = 5$ for the space-fractional heat equation

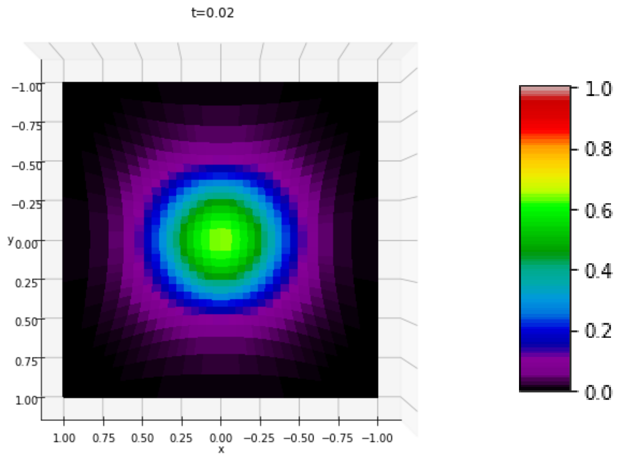


(c) Adaptive grids with $\beta = 1$ for the space-fractional heat equation

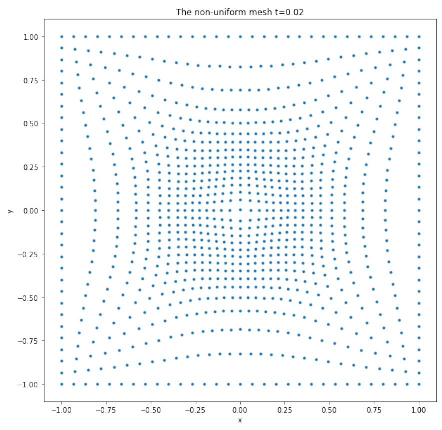
Figure 5.17: Solutions and grids to the space-fractional heat equation, $\alpha = 1.5$, from time $t = 0$ till $t = 0.045$ seconds. Adaptive grid parameters $\tau = 1$, $\beta = 1$ or 5 and an $(N + 2) \times (N + 2) = 31 \times 31$ grid.



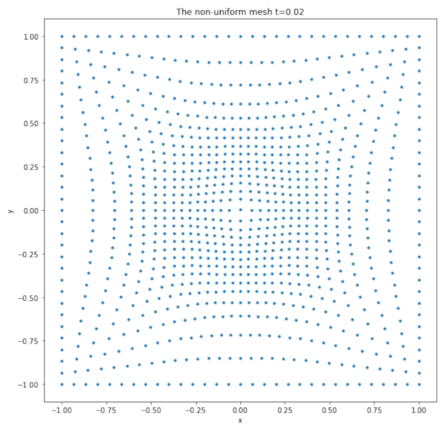
(a) Solution for $\alpha = 1.25$ at $t = 0.02$



(b) Solution for $\alpha = 1.75$ at $t = 0.02$

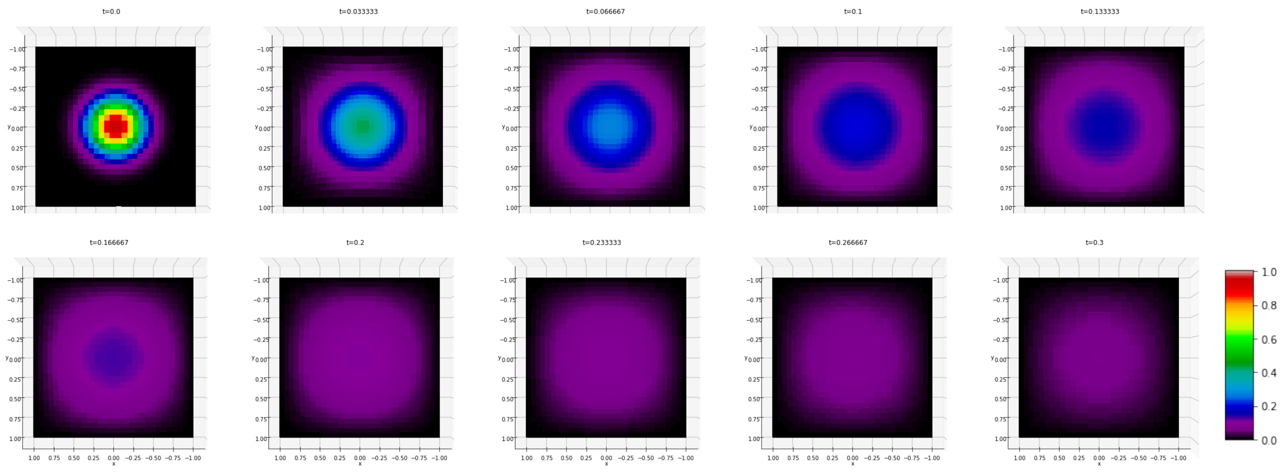


(c) Grid for $\alpha = 1.25$ at $t = 0.02$



(d) Grid for $\alpha = 1.75$ at $t = 0.02$

Figure 5.18: Solutions and adaptive grids for PDE (5.15) at $t = 0.02$ for $\alpha = 1.25$ and $\alpha = 1.75$. The solution takes longer to diffuse outwards for smaller α and the grid is therefore more dense around the center of the domain in this case when compared to a higher value of α.


(a) Solutions to the space-fractional heat equation, $\alpha = 1.99$, $\tau = 1$, $\beta = 5$

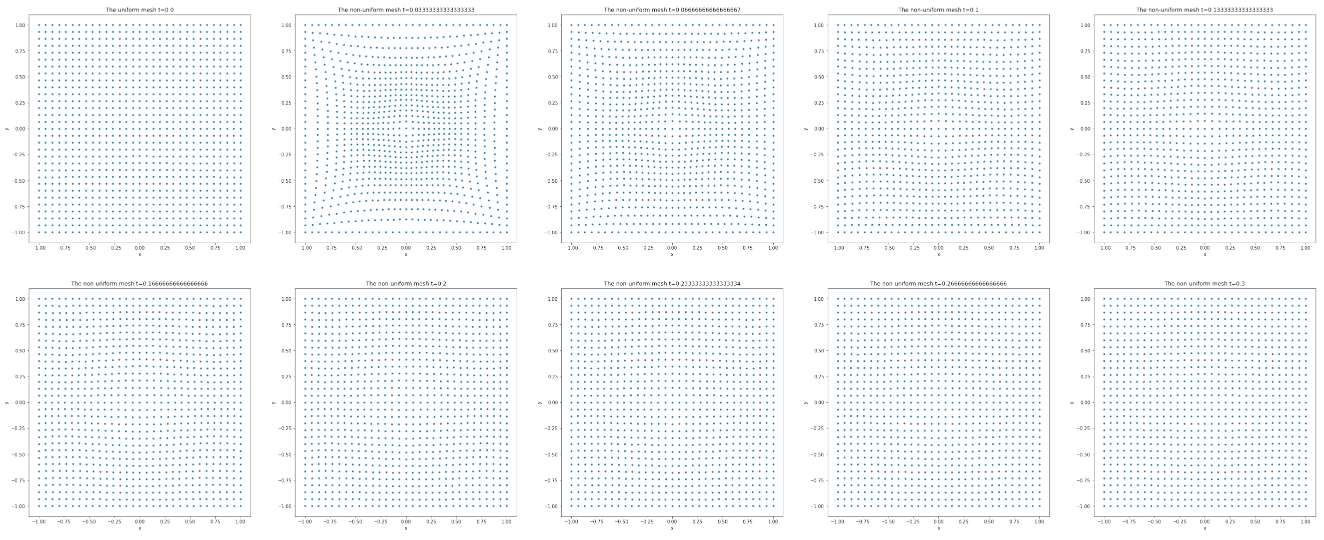

(b) Adaptive grids with $\beta = 5$ for the space-fractional heat equation

Figure 5.19: Solutions and grids to the space-fractional heat equation, $\alpha = 1.99$, from time $t = 0$ till $t = 0.3$ seconds. Adaptive grid parameters $\tau = 1$, $\beta = 1$ or 5 and an $(N + 2) \times (N + 2) = 31 \times 31$ grid.

5.2.2 Space-fractional Fisher's equation

In this section we consider the space-fractional Fisher's equation in two dimensions,

$$u_t(x,y,t) = -d\,(-\Delta)^{\alpha/2}\,u(x,y,t) + ru(1-u), \tag{5.20}$$

for $\alpha \in (1,2)$, $(x,y) \in [a,b] \times [a,c] = [-1,1] \times [-1,1]$, $t \geq 0$ together with boundary conditions,

$$u(a,y,t) = u(b,y,t) = 0, \qquad u_x(a,y,t) = u_x(b,y,t) = 0, \tag{5.21}$$

$$u(x,a,t) = u(x,c,t) = 0, \qquad u_y(x,a,t) = u_y(x,c,t) = 0, \tag{5.22}$$

and Gaussian type initial condition,

$$u(x,y,0) = \frac{1}{2}\exp\left(10\left(x^2+y^2\right)\right). \tag{5.23}$$

The space-fractional derivative on the RHS of PDE (5.20) is approximated using the L2(NU)-2D method equation (3.7). The system is solved via the two-dimensional method outlined in Section 4 with parameters $\tau = 1$, $\beta = 5$ and on an $(N+2) \times (N+2) = 31 \times 31$ grid unless stated otherwise. Figure (5.20) shows the solutions and adaptive grids for PDE (5.20) for $\alpha = 1.5$ from time $t = 0$ to $t = 1.2$. We observe the solutions spreading out towards the edges of the domain while the magnitude of the solution increases until it reaches a magnitude of 1. The adaptive grid moves well throughout the simulation; we initially see a high density of grid points near the middle of the domain where the Gaussian initial condition lies, then as the solution spreads out and becomes steeper near the boundary, we observe that the grid points become less dense in the center of the domain and more dense near the boundaries where the solution is steeper. We should note that as expected both the solution and the adaptive grids behave in a symmetrical way in the x and y directions.

When considering the space-fractional heat equation in two dimensions we observed that increasing the parameter of β from 1 to 5 allowed for a more adaptive grid, but what happens when we change the value of the parameter τ? The choice of this value doesn't affect the solution of the PDE, but it does affect how adaptive the grids are. Figure 5.20b shows the adaptive grids when $\tau = 1$ and Figure 5.20c shows the adaptive grids when $\tau = 0.01$. We can see that choosing the smaller choice for τ does not allow for the grid to adapt as quickly over time. We should note however that the run-time for the experiment where $\tau = 0.01$ was significantly faster than when $\tau = 1$ (38138 compared to 71013 seconds).

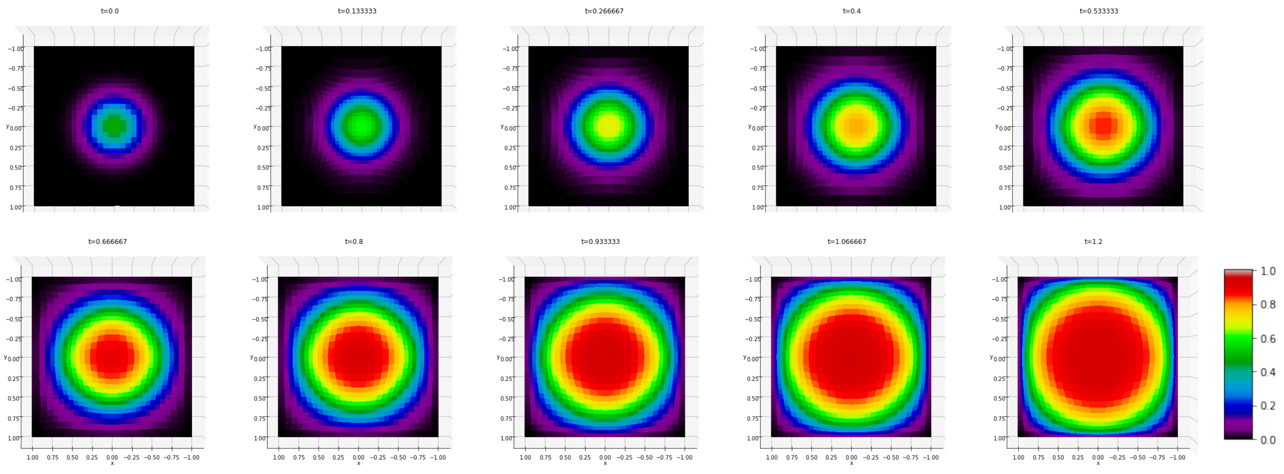
We also consider the differences we can observe in solutions for different values of α. Figure 5.21 shows the solutions and adaptive grids for PDE (5.20) at $t = 1.2$ for $\alpha = 1.25$ and $\alpha = 1.75$. We can see that for a large value of α, at the same point in time the solution has reached a higher magnitude and stretched further out towards the boundaries of the domain.

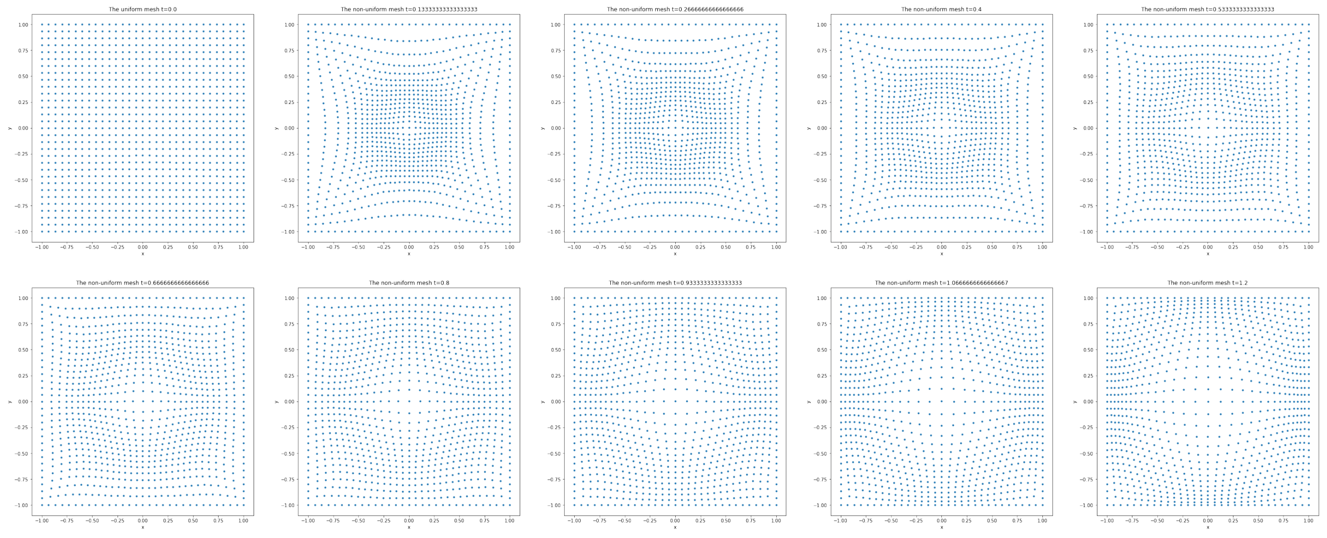
We can also check that behaviour in the limit $\alpha \to 2$ approaches the ordinary Fisher's equation. To do this we run a simulation that solves equation (5.20) with $\alpha = 1.99$, and compare this to the solution for the ordinary Fisher's equation in two dimensions given by,

$$u_t(x,y,t) = d\Delta u(x,y,t) + ru(1-u), \tag{5.24}$$

with the same parameter values, initial condition and boundary conditions. The results (see Figure A.8) show that the solution to the Fisher's equation with the fractional Laplacian does indeed approach the solution to Fisher's equation with the ordinary Laplacian as α approaches 2.

Lastly we show that as $t \to \infty$ the adaptive grid will become very dense around all edges of the domain. The adaptive grid for the solution of (5.20) for $\alpha = 1.99$, $d = 0.05$, $r = 5$ at $t = 5$ is shown in Figure 5.22.


(a) Solutions to the space-fractional Fisher's equation, $\alpha = 1.5$, $\tau = 1$, $\beta = 5$


(b) Adaptive grids with $\tau = 1$ for the space-fractional Fisher's equation

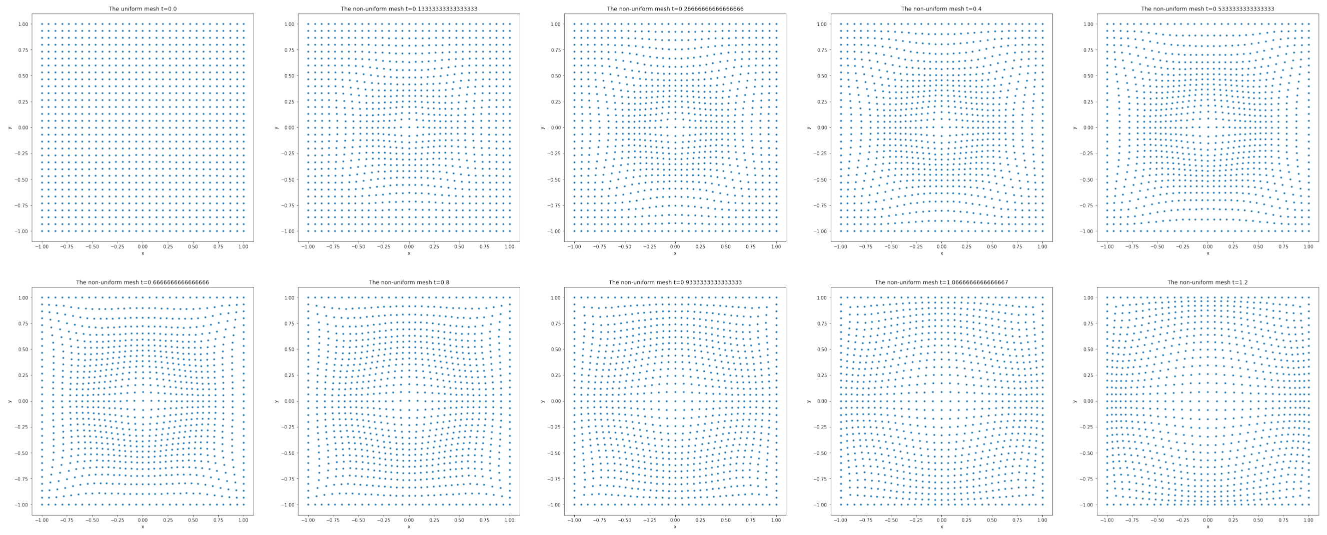
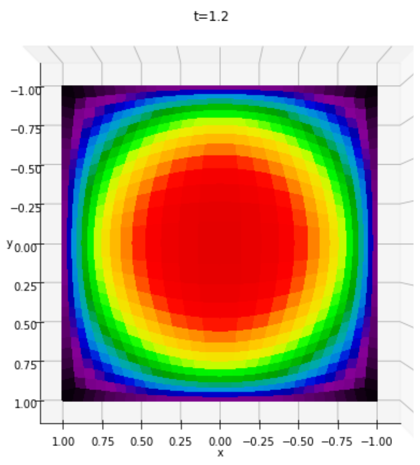
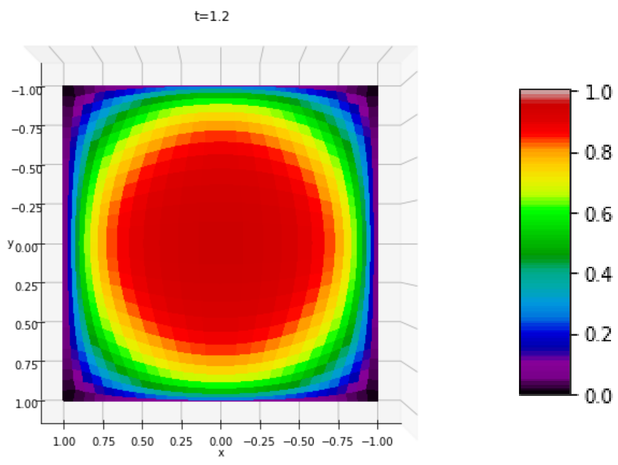

(c) Adaptive grids with $\tau = 0.01$ for the space-fractional Fisher's equation

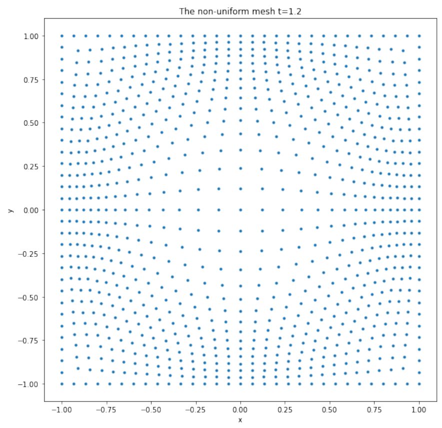
Figure 5.20: Solutions and grids to the space-fractional Fisher's equation, $\alpha = 1.5$, from time $t = 0$ till $t = 1.2$ seconds. Adaptive grid parameters $\tau = 1$ or 0.01, $\beta = 5$ and an $(N + 2) \times (N + 2) = 31 \times 31$ grid.



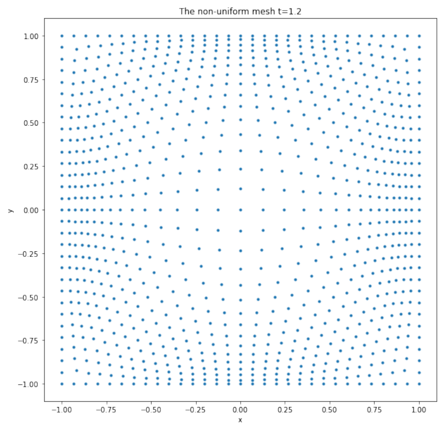
(a) Solution for $\alpha = 1.25$ at $t = 1.2$



(b) Solution for $\alpha = 1.75$ at $t = 1.2$



(c) Grid for $\alpha = 1.25$ at $t = 1.2$



(d) Grid for $\alpha = 1.75$ at $t = 1.2$

Figure 5.21: Solutions and adaptive grids for PDE (5.20) at $t = 1.2$ for $\alpha = 1.25$ and $\alpha = 1.75$.

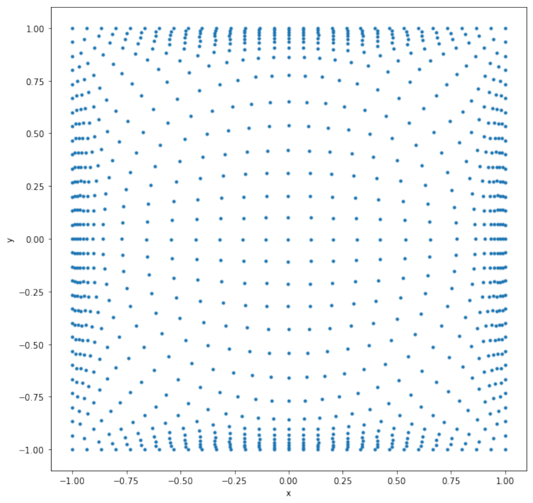


Figure 5.22: The adaptive grid for the solution of (5.20) for $\alpha = 1.99$, $d = 0.05$, $r = 5$ at $t = 5$.

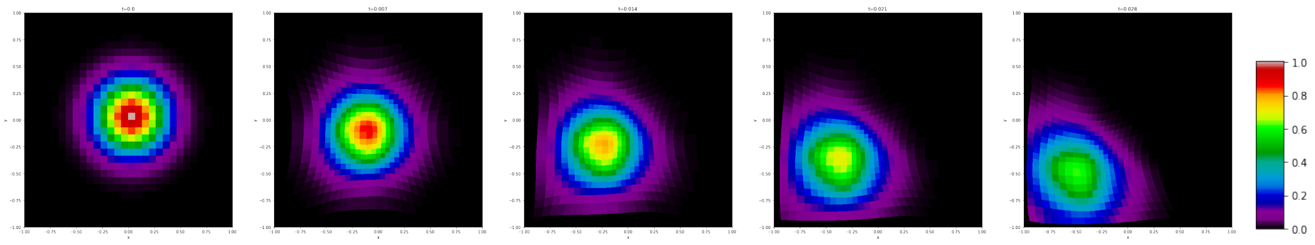
5.2.3 Space-fractional advection-diffusion equation

In this section we consider the space-fractional advection-diffusion equation in two dimensions,

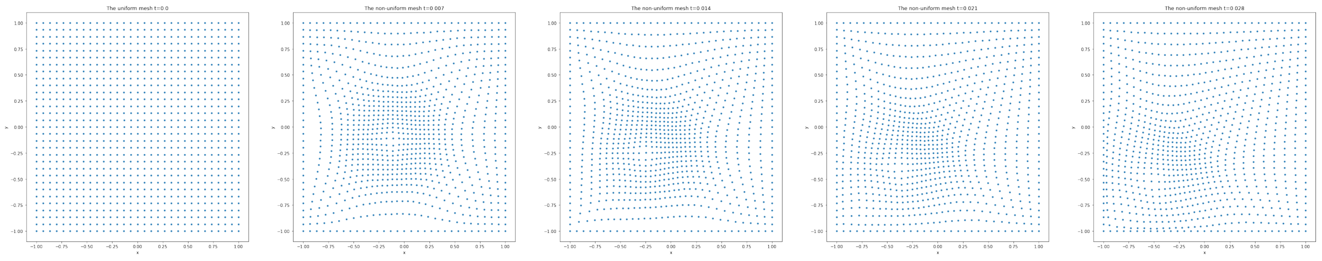
$$u_t(x,y,t) = -d\,(-\Delta)^{\alpha/2}\,u(x,y,t) + \underline{v}\cdot\nabla u, \tag{5.25}$$

for $\alpha \in (1,2)$, $(x,y) \in [a,b]\times[a,c] = [-1,1]\times[-1,1]$, $t \geq 0$ together with boundary conditions (5.16) and (5.17), and Gaussian type initial condition (5.18). The space-fractional derivative on the RHS of PDE (5.20) is approximated using the L2(NU)-2D method equation (3.7), and the first derivatives in the forcing term are approximated using central differences. The system is solved via the two-dimensional method outlined in Section 4 with parameters $\tau = 1$, $\beta = 5$ and on an $(N+2)\times(N+2) = 31\times 31$ grid.

Figure 5.23 shows the solutions and adaptive grids to PDE (5.25) for $\alpha = 1.5$ with diffusion coefficient $d = 1$ and constant velocity $\underline{v} = \binom{-20}{-20}$. The solution behaves as expected and travels in both the negative x and y directions whilst losing magnitude. The boundary conditions are satisfied as the solution remains equal to zero there. The non-uniform grids also adapt as expected and we see dense areas of points move towards the bottom left corner of the domain.



(a) Solutions.



(b) Adaptive grids.

Figure 5.23: Solutions and adaptive grids to PDE (5.25) for $\alpha = 1.5$ with diffusion coefficient $d = 1$ and constant velocity $\underline{v} = \binom{-20}{-20}$ from $t = 0$ to $t = 0.028$ on an $(N+2)\times(N+2) = 31\times 31$ grid.

Chapter 6

Conclusion

After an investigation into the existing methods to approximate fractional derivatives in one dimension on uniform finite difference grids, we chose to extend the L1 ($\alpha \in (0,1)$) and L2 ($\alpha \in (1,2)$) methods, which approximate Caputo derivatives, to non-uniform grids. This choice was made due to the simplicity of the approach which would follow where integer derivatives and the expression inside the integral involving step sizes were generalised to non-uniform grids. This was favoured over the alternatives discussed in Chapter 3 as the method by Duo et al.[18] involved manipulations of the singular integral, and the Grünwald-Letnikov definition approach would have involved calculating a truncated infinite sum of integer derivatives on a non-uniform grid. Once the Caputo derivatives were approximated on a non-uniform grid, these could be used to express the fractional Laplacian via the Riesz derivative in one dimension, and the Riesz potential in two dimensions. This gave us the L1(NU), L2(NU), L1(NU)-2D and L2(NU)-2D methods.

The L2(NU) and L2(NU)-2D methods both perform well when approximating the space-fractional Laplacian in one and two dimensions on adaptive grids. We can see that the solutions they produce for space-fractional heat, advection-diffusion and Fisher's PDEs behave as expected. They appear stable and preserve important properties such as approaching the ordinary Laplacian in the limit $\alpha \to 2$. The adaptive grids also performed as expected. For heat equations, grids were symmetrical and moved to become dense where the solution was most steep, and returned to a uniform grid once the solution had completely diffused. For the advection diffusion PDE the dense parts of the grid moved as dictated by the source term, and for Fisher's equations, the grid became steepest near the boundaries of the domain.

The L1(NU) method displayed some issues regarding stability. A possible cause of this is the low accuracy approximation of the first integer derivative where forward differences were used. Therefore replacing this approximation with a higher order approximation for the first derivative may help eradicate the unstable behaviour. It was found that in one dimension, increasing the parameter τ (which helps determine how quickly the grid adapts) from 1×10^{-4} to 1×10^{-3} helped smooth the solution. Another solution may be changing the monitor function that is used within the adaptive grid generation, or limiting the maximum or minimum time step sizes of the solver. When this stability was not an issue the results generated behaved as expected; we see that solutions to the heat equation have long thin tails and so the solution reaches the boundary more quickly when compared to $\alpha \in (1,2)$. This means that the solutions at the boundary for Fisher's equations will reach the maximum height more quickly for $\alpha \in (0,1)$ compared to $\alpha \in (1,2)$. These behaviours are outlined in [46] and the results of the numerical experiments in this thesis concur. Some small effects of the stability issues were observed to the solutions to Fisher's equations as we saw small differences in the adaptive grids near the boundaries. It is suspected that if a large enough N is chosen, then these effects would not be an issue.

For future studies it would be possible to generalise the L1(NU), L2(NU), L1(NU)-2D and L2(NU)-2D methods to approximate Riemann-Liouville derivatives without making the same assumptions of homogeneous boundary conditions. This would involve the addition of extra terms given by equation (1.14). Additionally the L1(NU) method could be adapted to involve a higher order approximation of the first derivative. The efficiency of the adaptive grid solvers in one dimension could be improved by reordering the system (4.16) as $x_1, u_1, ..., x_N, u_N$. In two dimensions a similar reordering could be performed, and/or the system could be decoupled so that at each time step, the new mesh is first found, and then the solution is determined.

We were successful in presenting the first solutions to PDEs involving the space-fractional Laplacian of order $\alpha \in (0,1) \cup (1,2)$ on adaptive finite difference grids in one and two dimensions. These methods (L1(NU), L2(NU), L1(NU)-2D and L2(NU)-2D - given in equations (2.73), (2.84), (3.4) and (3.7) respectively) can help to open up the study of solving space-fractional PDEs in this way. Alternative methods derived via the Grünwald-Letnikov definition approach combined with the infinite integer derivative series definition for fractional derivatives (discussed in section 2.2) could be derived and the performance compared with the L1(NU), L2(NU), L1(NU)-2D and L2(NU)-2D methods for $\alpha \in (0,1) \cup (1,2)$. A method derived via the singular integral definition for the fractional Laplacian (the uniform approach is shown by Duo et al. [18] and was discussed in section 2.1) would allow us to also consider the fractional Laplacian of order $\alpha = 1$.

Bibliography

[1] A. A. Alikhanov and C. Huang. A high-order L2 type difference scheme for the time-fractional diffusion equation. *Applied Mathematics and Computation*, 411:126545, 2021.

[2] C. Andersson, C. Führer, and J. Åkesson. Assimulo: A unified framework for {ODE} solvers. *Mathematics and Computers in Simulation*, 116(0):26 – 43, 2015.

[3] R. Andriambololona, R. Tokiniaina, and H. Rakotoson. Definitions of complex order integrals and complex order derivatives using operator approach. *Int. J. Latest Res. Sci. Tech*, 1:317–323, 2012.

[4] E. F. Anley and Z. Zheng. Finite Difference Method for Two-Sided Two Dimensional Space Fractional Convection-Diffusion Problem with Source Term. *Mathematics*, 8(11):1878, 2020.

[5] A. Atangana and D. Baleanu. New fractional derivatives with nonlocal and non-singular kernel: theory and application to heat transfer model. *arXiv preprint arXiv:1602.03408*, 2016.

[6] B. Baeumer, M. Kovács, M. M. Meerschaert, and H. Sankaranarayanan. Reprint of: Boundary conditions for fractional diffusion. *Journal of Computational and Applied Mathematics*, 339:414–430, 2018.

[7] J. M. Beach. Fractional derivatives. `https://rdw.rowan.edu/etd/1630`, 2000. [Online; accessed 22-June-2022].

[8] G. Beckett, J. A. Mackenzie, A. Ramage, and D. M. Sloan. Computational solution of two-dimensional unsteady PDEs using moving mesh methods. *Journal of Computational Physics*, 182(2):478–495, 2002.

[9] D. Brockmann and L. Hufnagel. Front propagation in reaction-superdiffusion dynamics: Taming Lévy flights with fluctuations. *Physical review letters*, 98(17):178301, 2007.

[10] R. Brown. On the particles contained in the pollen of plants; and on the general existence of active molecules in organic and inorganic bodies. *Edinburgh New Philosophical Journal*, 5:358–371, 1828.

[11] M. Cai and C. Li. On Riesz derivative. *Fractional Calculus and Applied Analysis*, 22(2):287–301, 2019.

[12] M. Caputo. Linear models of dissipation whose Q is almost frequency independent—II. *Geophysical Journal International*, 13(5):529–539, 1967.

[13] A. Cartea and D. del Castillo-Negrete. Fractional diffusion models of option prices in markets with jumps. *Physica A: Statistical Mechanics and its Applications*, 374(2):749–763, 2007.

[14] J. Cayama, C. M. Cuesta, and F. de la Hoz. A pseudospectral method for the one-dimensional fractional Laplacian on R. *Applied Mathematics and Computation*, 389:125577, 2021.

[15] A. V. Chechkin, R. Metzler, J. Klafter, and V. Y. Gonchar. Introduction to the theory of Lévy flights. *Anomalous transport: Foundations and applications*, pages 129–162, 2008.

[16] Y. Choi and S. Chung. Finite element solutions for the space fractional diffusion equation with a nonlinear source term. In *Abstract and Applied Analysis*, volume 2012. Hindawi, 2012.

[17] Y. Dimitrov. Numerical approximations for fractional differential equations. *arXiv preprint arXiv:1311.3935*, 2013.

[18] S. Duo, H. W. van Wyk, and Y. Zhang. A novel and accurate finite difference method for the fractional Laplacian and the fractional Poisson problem. *Journal of Computational Physics*, 355:233–252, 2018.

[19] S. Duo and Y. Zhang. Computing the ground and first excited states of the fractional Schrödinger equation in an infinite potential well. *Communications in Computational Physics*, 18(2):321–350, 2015.

[20] A. Einstein. *Investigations on the theory of the Brownian movement, edited with notes by R. Furth and translated by A. D. Cowper.* Dover Publications, Inc., New Yok, 1956.

[21] L. Euler. On transcendental progressions that is, those whose general terms cannot be given algebraically. *Commentarii academiae scientiarum Petropolitanae*, 1738(5):36–57, 1999.

[22] L. R. Evangelista and E. K. Lenzi. *Fractional diffusion equations and anomalous diffusion.* Cambridge University Press, 2018.

[23] F. Ferrari. Weyl and Marchaud derivatives: A forgotten history. *Mathematics*, 6(1):6, 2018.

[24] A. Gladkina, G. Shchedrin, U. A. Khawaja, and L. D. Carr. Expansion of fractional derivatives in terms of an integer derivative series: physical and numerical applications. *arXiv preprint arXiv:1710.06297*, 2017.

[25] X.-M. Gu, H.-W. Sun, Y. Zhang, and Y.-L. Zhao. Fast implicit difference schemes for time-space fractional diffusion equations with the integral fractional Laplacian. *Mathematical Methods in the Applied Sciences*, 44(1):441–463, 2021.

[26] E. Hanert. Front dynamics in a two-species competition model driven by Lévy flights. *Journal of theoretical biology*, 300:134–142, 2012.

[27] E. Hanert, E. Schumacher, and E. Deleersnijder. Front dynamics in fractional-order epidemic models. *Journal of theoretical biology*, 279(1):9–16, 2011.

[28] B. I. Henry, T. A. Langlands, and P. Straka. An introduction to fractional diffusion. In *Complex Physical, Biophysical and Econophysical Systems*, pages 37–89. World Scientific, 2010.

[29] W. Huang and R. D. Russell. A high dimensional moving mesh strategy. *Applied Numerical Mathematics*, 26(1-2):63–76, 1998.

[30] Y. Huang and A. Oberman. Numerical methods for the fractional Laplacian Part I: a finite differencequadrature approach. *arXiv preprint arXiv:1311.7691*, 2013.

[31] T. Langlands and B. I. Henry. The accuracy and stability of an implicit solution method for the fractional diffusion equation. *Journal of Computational Physics*, 205(2):719–736, 2005.

[32] G. W. Leibniz. Leibniz, G.W September 30 1695 Letter to G.F.A. L'Hôpital. `https://leibniz.uni-goettingen.de/pages/index`, 1695. [Online; accessed 17-June-2022].

[33] G. W. Leibniz. Leibniz, G.W January 15 1696 Letter to G.F.A. L'Hôpital. `https://leibniz.uni-goettingen.de/pages/index`, 1696. [Online; accessed 17-June-2022].

[34] C. Li and F. Zeng. *Numerical Methods for Fractional Calculus*, volume 24. CRC Press, 2015.

[35] S. Li, L. Petzold, and Y. Ren. Stability of moving mesh systems of partial differential equations. *SIAM Journal on Scientific Computing*, 20(2):719–738, 1998.

[36] Y. Lin and C. Xu. Finite difference/spectral approximations for the time-fractional diffusion equation. *Journal of computational physics*, 225(2):1533–1552, 2007.

[37] E. R. Love. Fractional derivatives of imaginary order. *Journal of the London Mathematical Society*, 2(2):241–259, 1971.

[38] P. Lyu and S. Vong. A nonuniform L2 formula of Caputo derivative and its application to a fractional Benjamin–Bona–Mahony-type equation with nonsmooth solutions. *Numerical Methods for Partial Differential Equations*, 36(3):579–600, 2020.

[39] J. Machado, A. M. Galhano, and J. J. Trujillo. On development of fractional calculus during the last fifty years. *Scientometrics*, 98(1):577–582, 2014.

[40] B. Malengier, P. Kišon, J. Tocknell, C. Abert, F. Bruckner, and M.-A. Bisotti. ODES: a high level interface to ODE and DAE solvers. *The Journal of Open Source Software*, 3(22):165, feb 2018.

[41] P. Nakroshis, M. Amoroso, J. Legere, and C. Smith. Measuring Boltzmann's constant using video microscopy of Brownian motion. *American Journal of Physics*, 71(6):568–573, 2003.

[42] OED. Oxford English Dictionary. *Simpson, JA & Weiner, ESC.–1989*, 1993.

[43] M. D. Ortigueira, T.-M. Laleg-Kirati, and J. T. Machado. Riesz potential versus fractional Laplacian. *Journal of Statistical Mechanics: Theory and Experiment*, 2014(9):P09032, 2014.

[44] T. J. Osler. Leibniz rule for fractional derivatives generalized and an application to infinite series. *SIAM Journal on Applied Mathematics*, 18(3):658–674, 1970.

[45] I. Podlubny, R. L. Magin, and I. Trymorush. Niels Henrik Abel and the birth of fractional calculus. *Fractional Calculus and Applied Analysis*, 20(5):1068–1075, 2017.

[46] C. Pozrikidis. *The Fractional Laplacian.* CRC Press, 2018.

[47] B. Ross. The development of fractional calculus 1695–1900. *Historia Mathematica*, 4(1):75–89, 1977.

[48] B. Ross. *Fractional calculus and its applications: proceedings of the international conference held at the University of New Haven, June 1974*, volume 457. Springer, 2006.

[49] H. Sun, Y. Zhang, D. Baleanu, W. Chen, and Y. Chen. A new collection of real world applications of fractional calculus in science and engineering. *Communications in Nonlinear Science and Numerical Simulation*, 64:213–231, 2018.

[50] Z.-z. Sun and X. Wu. A fully discrete difference scheme for a diffusion-wave system. *Applied Numerical Mathematics*, 56(2):193–209, 2006.

[51] H. Tang and T. Tang. Adaptive mesh methods for one-and two-dimensional hyperbolic conservation laws. *SIAM Journal on Numerical Analysis*, 41(2):487–515, 2003.

[52] J. G. Verwer, J. G. Blom, R. Furzeland, and P. A. Zegeling. *A moving-grid method for one-dimensional PDEs based on the method of lines.* CWI Nampa, ID, 1988.

[53] G. Wanner and E. Hairer. *Solving ordinary differential equations II*, volume 375. Springer Berlin Heidelberg New York, 1996.

[54] Q. Yang, F. Liu, and I. Turner. Numerical methods for fractional partial differential equations with Riesz space fractional derivatives. *Applied Mathematical Modelling*, 34(1):200–218, 2010.

[55] P. Zegeling, W. De Boer, and H. Tang. Robust and efficient adaptive moving mesh solution of the 2-D Euler equations. *Contemporary Mathematics*, 383:375, 2005.

[56] P. A. Zegeling. A dynamically-moving adaptive grid method based on a smoothed equidistribution principle along coordinate lines. In *Proc. 5th International Conference on Numerical Grid Generation in Computational Computational Field Simulation, Starkville*, 1996.

[57] P. A. Zegeling. On resistive MHD models with adaptive moving meshes. *Journal of Scientific Computing*, 24(2):263–284, 2005.

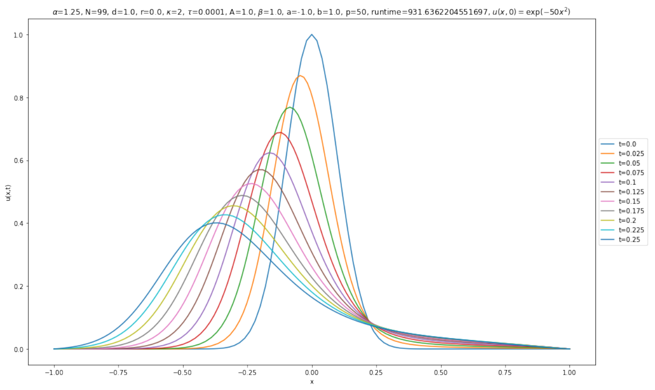
[58] P. A. Zegeling and H. Kok. Adaptive moving mesh computations for reaction–diffusion systems. *Journal of Computational and Applied Mathematics*, 168(1-2):519–528, 2004.

[59] Y. Zhang, Z. Sun, and H. Liao. Finite difference methods for the time fractional diffusion equation on non-uniform meshes. *Journal of Computational Physics*, 265:195–210, 2014.

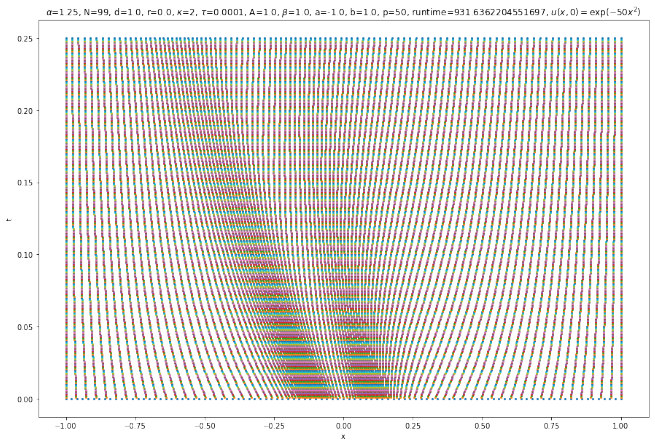
Appendix A

Additional results to numerical experiments

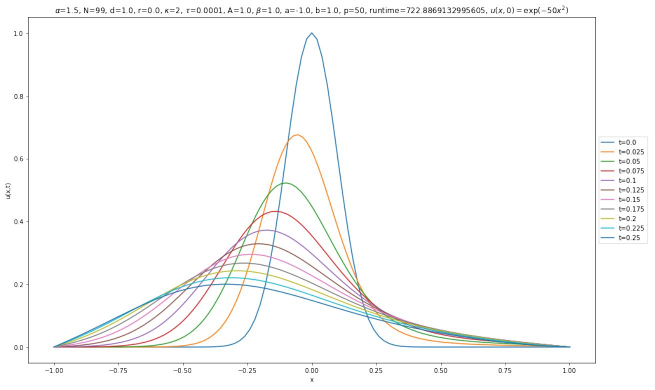
The following pages include supplementary solutions to those presented in Chapter 5. These figures display solutions at a greater number of time points, or solutions that are almost equivalent. References to these figures are made throughout Chapter 5.



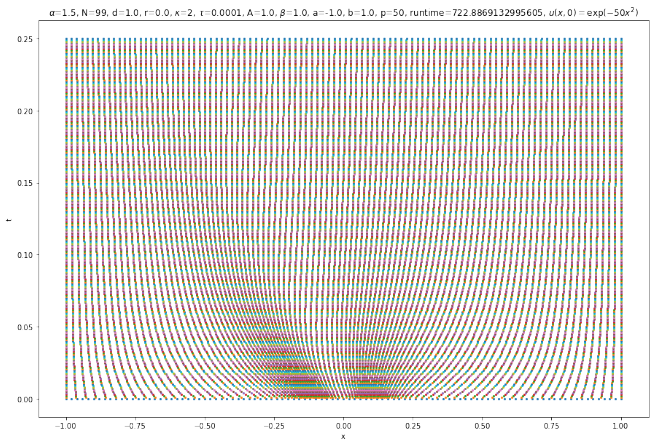
(a)



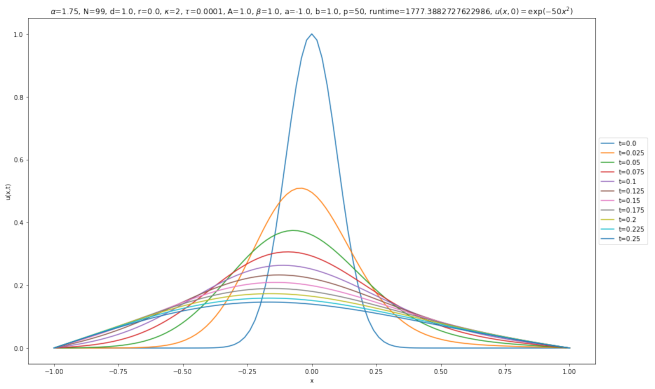
(b)



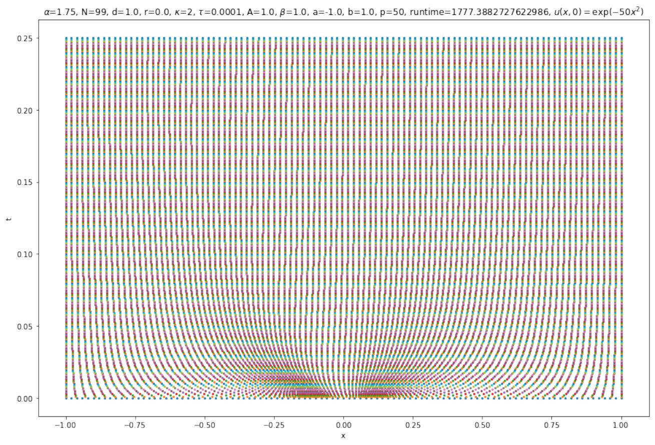
(c)



(d)

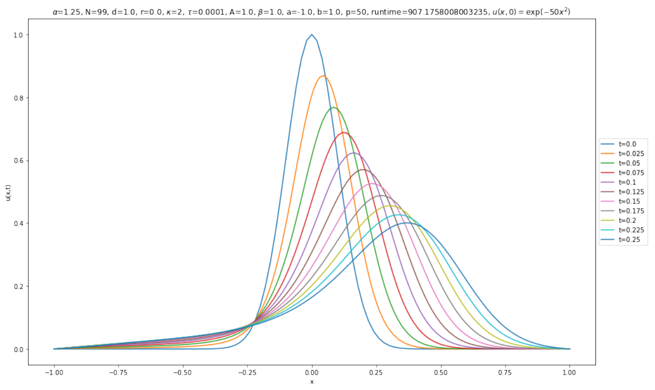


(e)

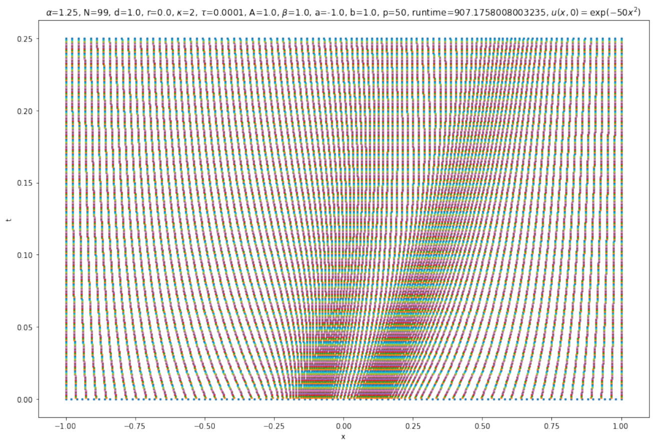


(f)

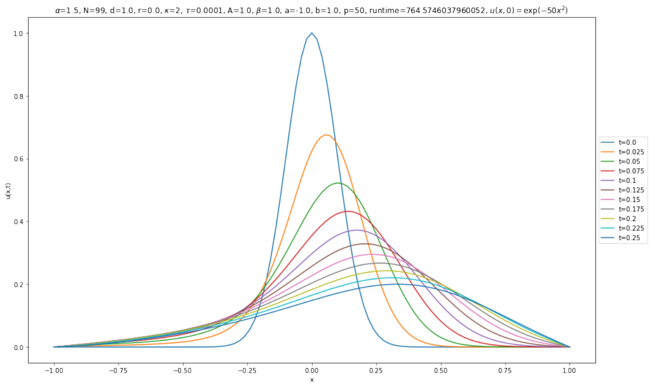
Figure A.1: A comparison of the results to the left space-fractional heat equation (5.4) generated using the L2(NU) method with $\alpha = 1.25,\ 1.5,\ 1.75$. The solutions are shown at times $t = 0, 0.025,\ 0.05,\ 0.075,\ 0.1,\ 0.125,\ 0.15,\ 0.175,\ 0.2,\ 0.225$ and 0.25 in dark-blue, orange, green, red, purple, brown, pink, grey, light-green, light-blue and blue respectively.



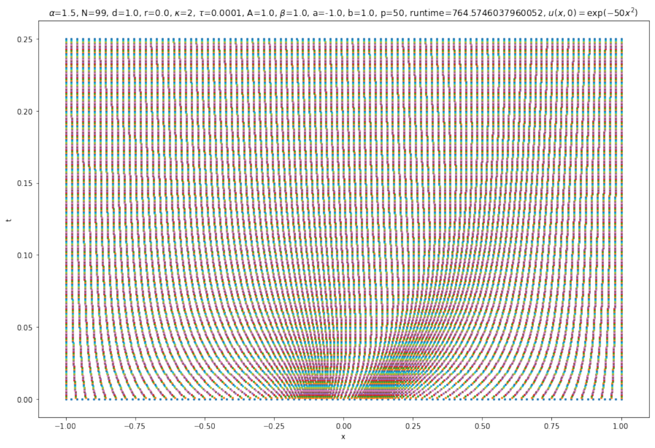
(a)



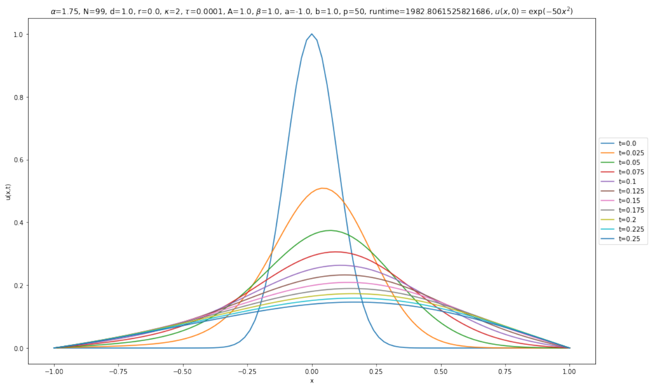
(b)



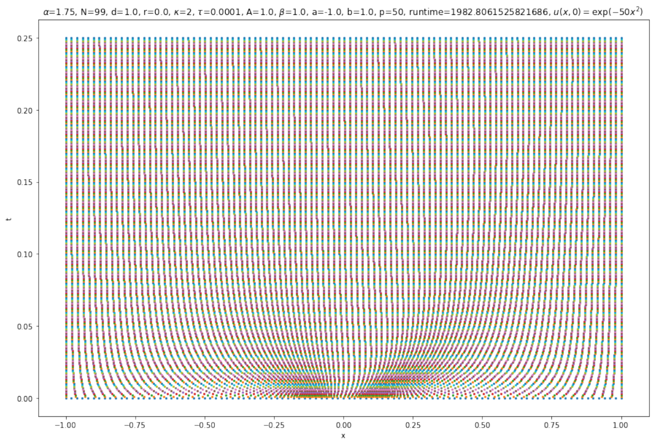
(c)



(d)

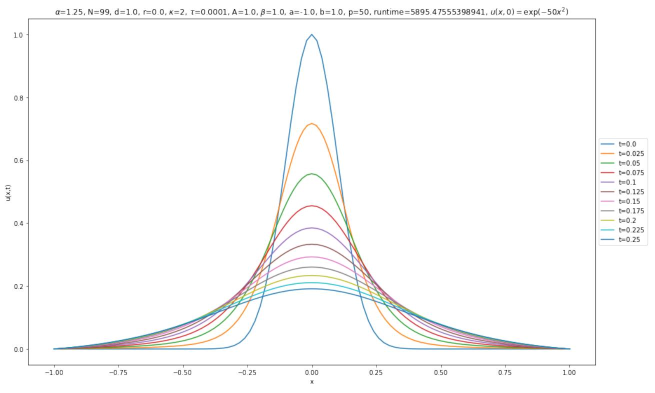


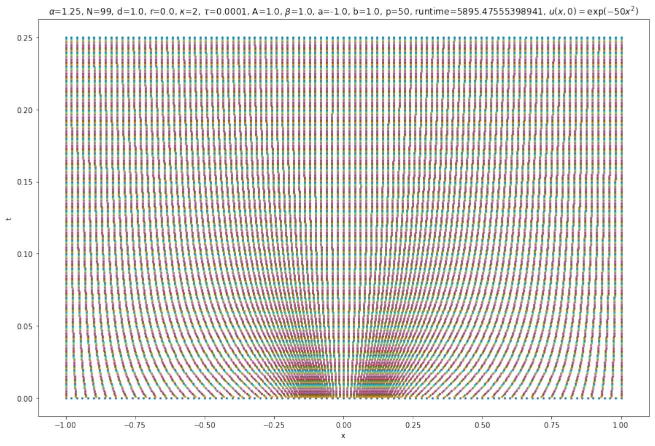
(e)



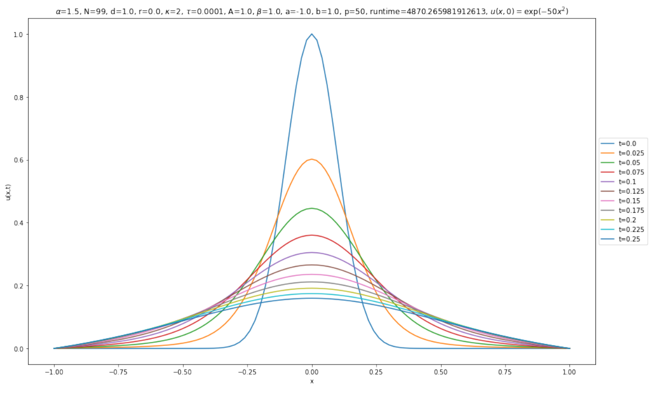
(f)

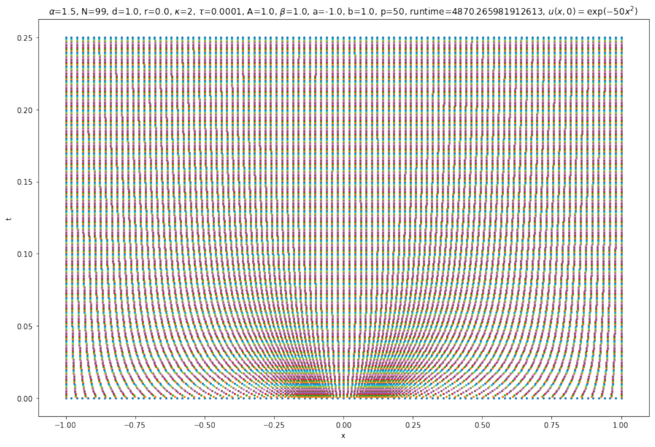
Figure A.2: A comparison of the results to the right space-fractional heat equation (5.7) generated using the L2(NU) method with $\alpha = 1.25,\ 1.5,\ 1.75$. The solutions are shown at times $t = 0, 0.025,\ 0.05,\ 0.075,\ 0.1,\ 0.125,\ 0.15,\ 0.175,\ 0.2,\ 0.225$ and 0.25 in dark-blue, orange, green, red, purple, brown, pink, grey, light-green, light-blue and blue respectively.



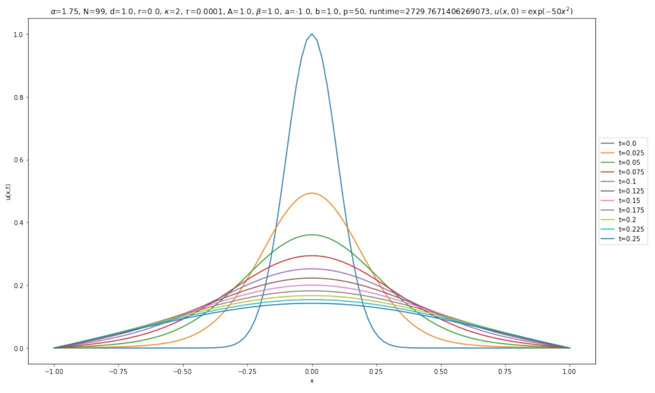


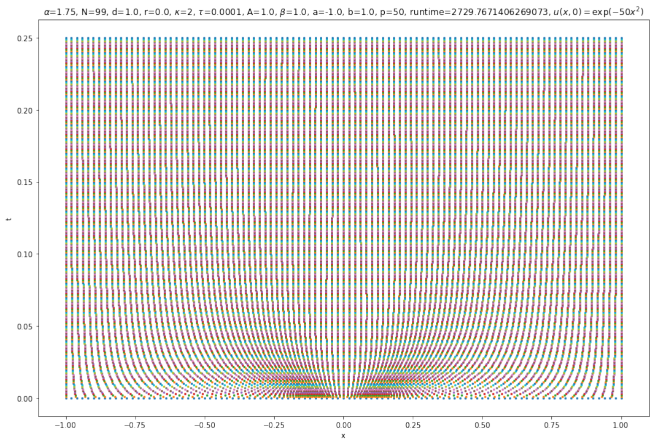
(a) **(b)**





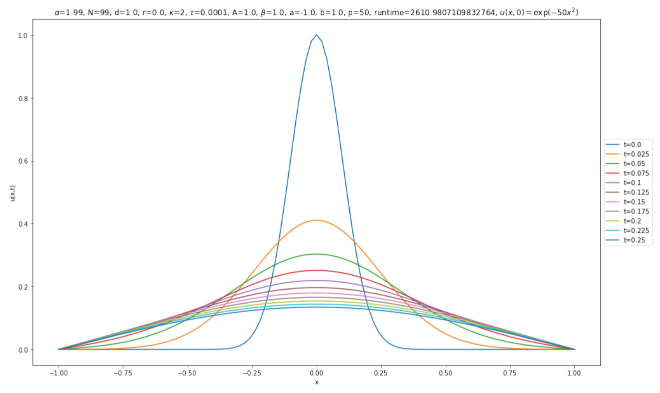
(c) **(d)**



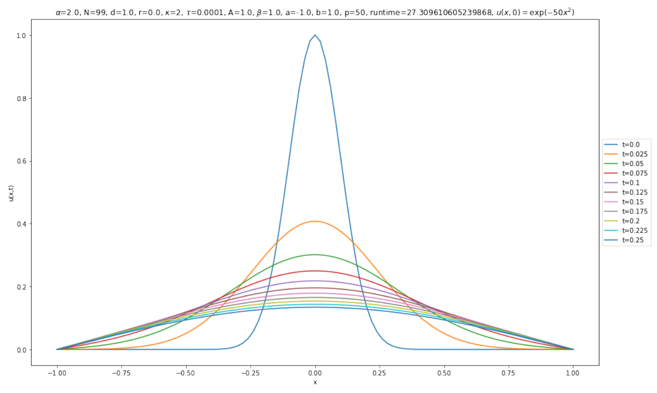


(e) **(f)**

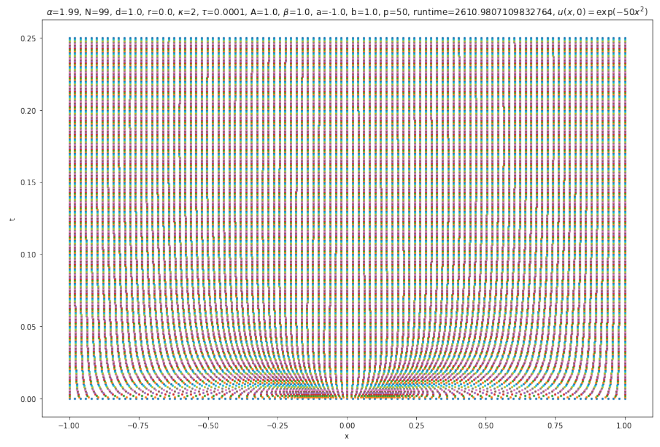
Figure A.3: A comparison of the results to the space-fractional heat equation (5.10) generated using the L2(NU) method with $\alpha = 1.25,\ 1.5,\ 1.75$. The solutions are shown at times $t = 0, 0.025,\ 0.05,\ 0.075,\ 0.1,\ 0.125,\ 0.15,\ 0.175,\ 0.2,\ 0.225$ and 0.25 in dark-blue, orange, green, red, purple, brown, pink, grey, light-green, light-blue and blue respectively.



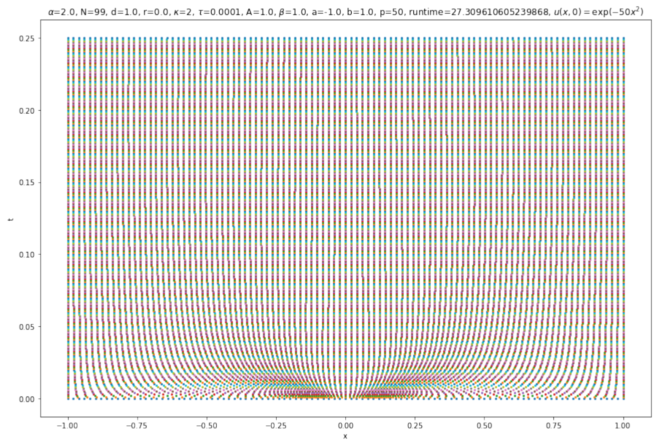
(a)



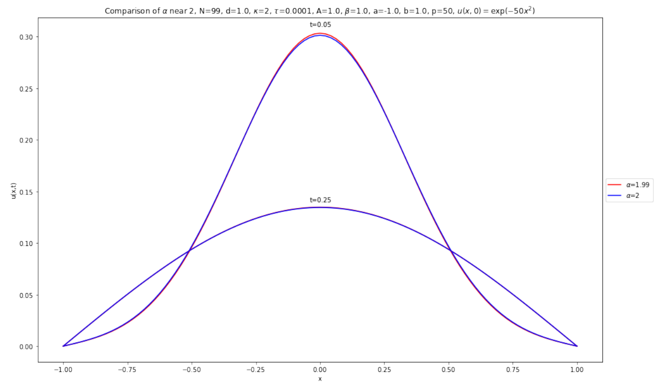
(b)



(c)

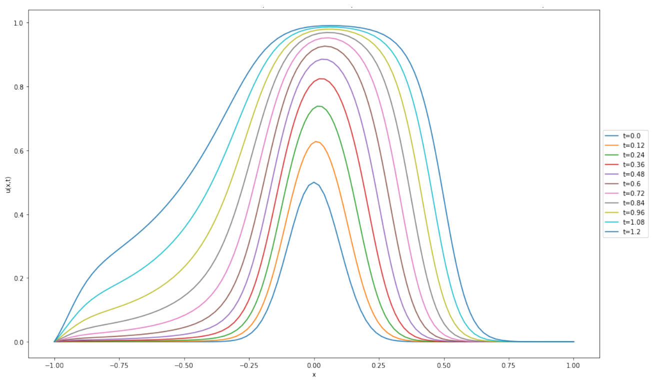


(d)

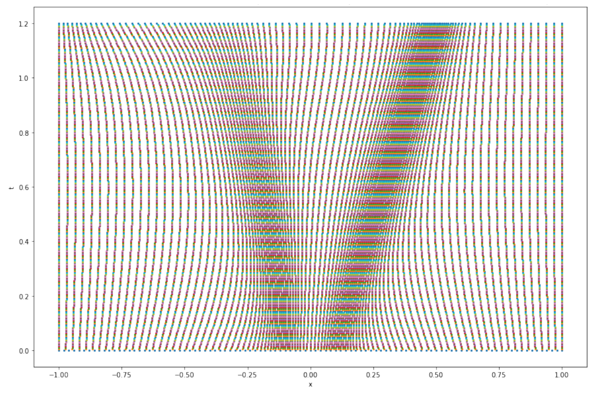


(e)

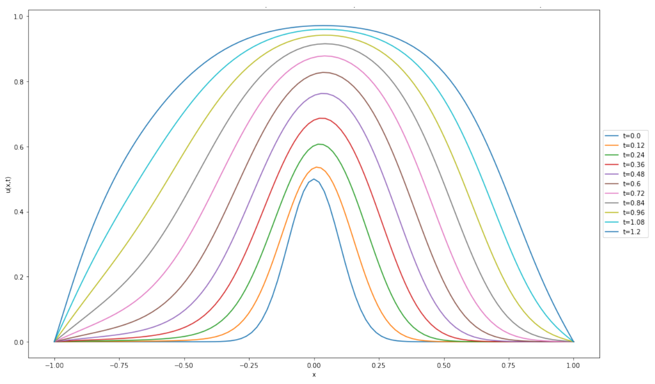
Figure A.4: A comparison of the results generated to PDE (5.10) using the L2(NU) method with $\alpha = 1.99$, with the results where $\alpha = 2$ and we use the standard central difference approximation for the second derivative of the ordinary Laplacian. The solutions in (a) and (b) are shown at times $t = 0, 0.025,\ 0.05,\ 0.075,\ 0.1,\ 0.125,\ 0.15,\ 0.175,\ 0.2,\ 0.225$ and 0.25 in dark-blue, orange, green, red, purple, brown, pink, grey, light-green, light-blue and blue respectively. (e) shows the solutions for $\alpha = 1.99$ (red) and $\alpha = 2$ (blue) at $t = 0.05$ and 0.25.



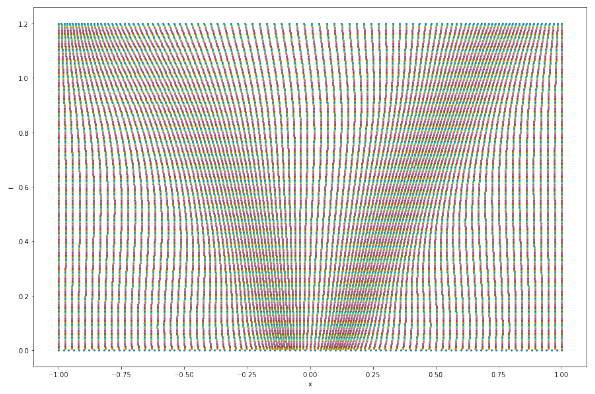
(a)



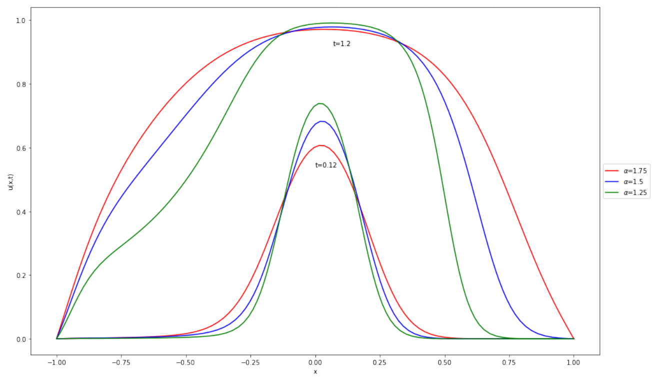
(b)



(c)

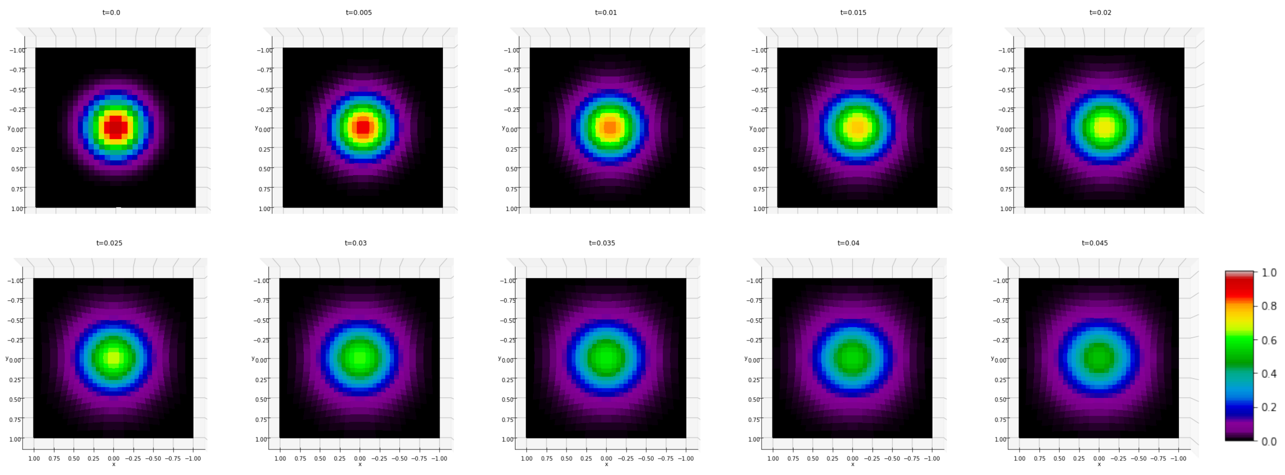


(d)

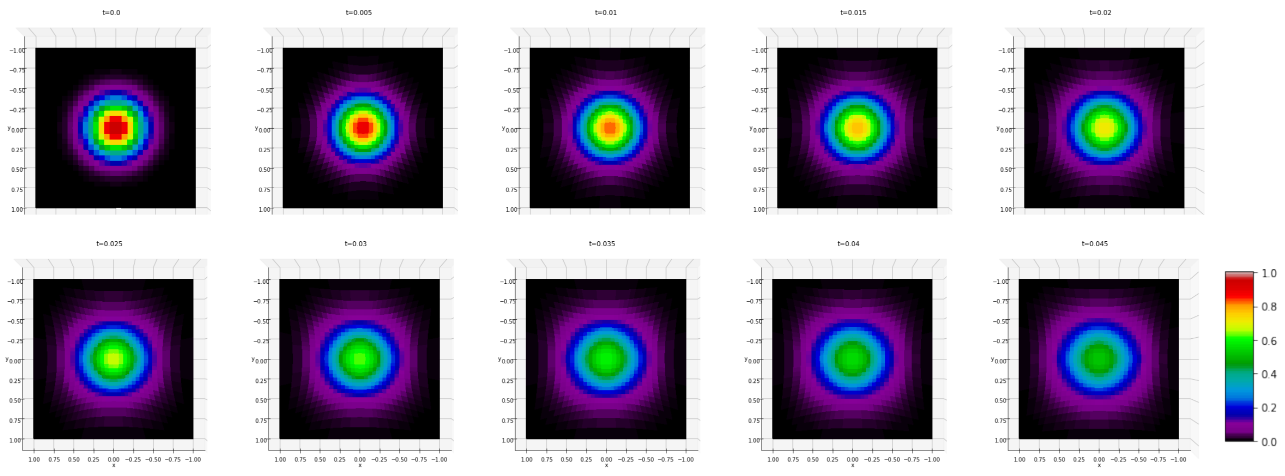


(e) Differences to the solution of (5.13) with different values of α.

Figure A.5: A comparison of the results to the right space-fractional heat equation with Fisher's term (5.13) generated using the L2(NU) method. Solutions with $\alpha = 1.25$ and 1.75 are shown in (a) and (c) at times $t = 0,\ 0.12,\ 0.24,\ 0.36,\ 0.48,\ 0.6,\ 0.72,\ 0.84,\ 0.96,\ 1.08$ and 1.2 shown in dark-blue, orange, green, red, purple, brown, pink, grey, light-green, light-blue and blue respectively. The respective adaptive grids are shown in (b) and (d). (e) shows the differences in solutions at time points $t = 0.025$ and $t = 0.25$ for $\alpha = 1.25,\ 1.5,\ 1.75$ shown in green, blue and red respectively.

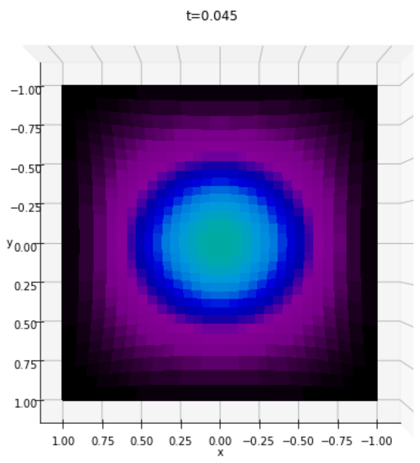


(a) Solutions to the space-fractional heat equation, $\alpha = 1.5,\ \tau = 1,\ \beta = 1$

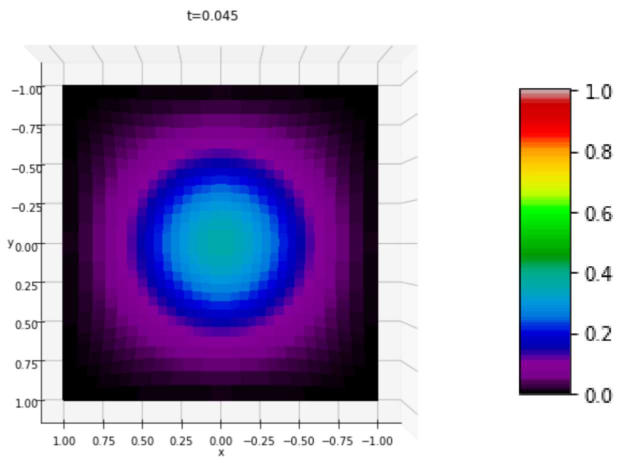


(b) Solutions to the space-fractional heat equation, $\alpha = 1.5,\ \tau = 1,\ \beta = 5$

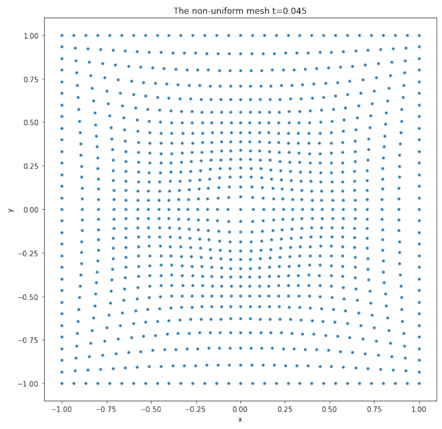
Figure A.6: Solutions to the space-fractional heat equation (5.15), $\alpha = 1.5$, from time $t = 0$ till $t = 0.045$ seconds. Adaptive grid parameters $\tau = 1,\ \beta = 1$ or 5 and an $(N + 2) \times (N + 2) = 31 \times 31$ grid.



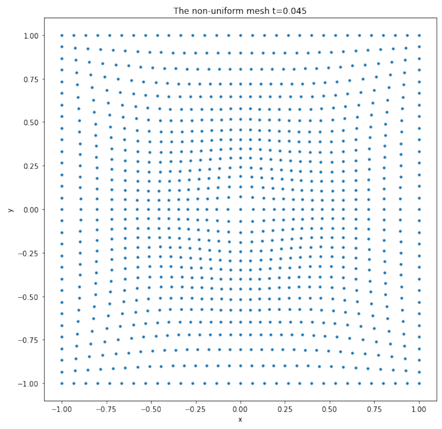
(a) Solution for $\alpha = 1.99$ at $t = 0.045$



(b) Solution for $\alpha = 2$ at $t = 0.045$

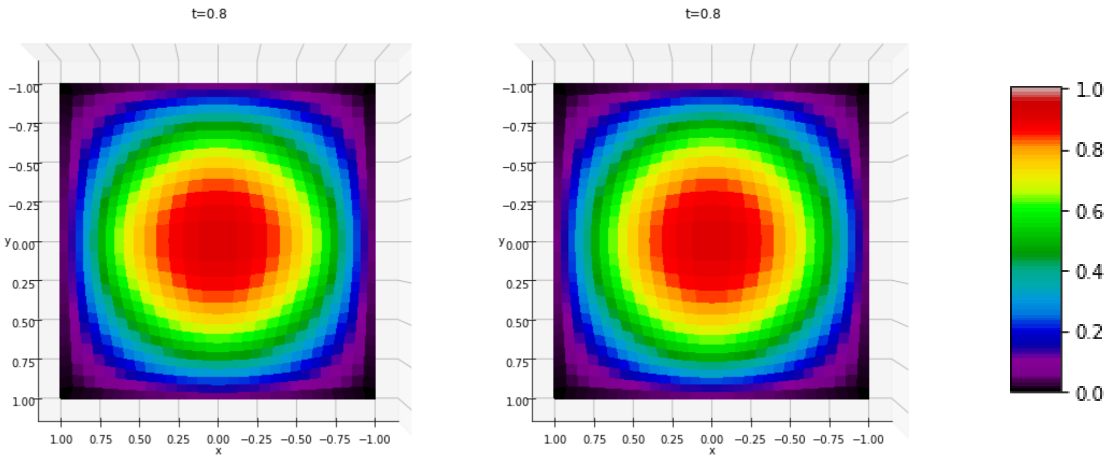


(c) Grid for $\alpha = 1.25$ at $t = 0.02$



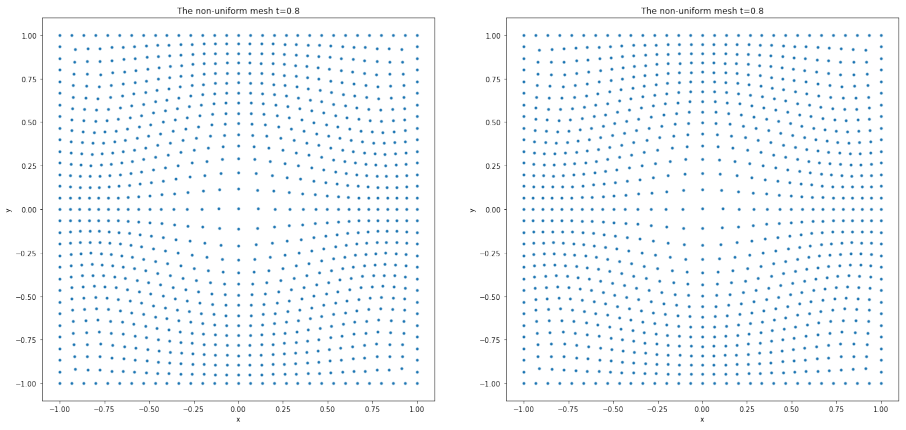
(d) Grid for $\alpha = 1.75$ at $t = 0.02$

Figure A.7: Solutions and adaptive grids for PDE (5.15) at $t = 0.045$ for $\alpha = 199$ and $\alpha = 2$. We can not observe any significant differences in either the solutions or the grids. Adaptive grid parameters $\tau = 1,\ \beta = 5$ and an $(N+2)\times(N+2) = 31 \times 31$ grid.



(a) Solution for $\alpha = 1.99$ at $t = 0.8$

(b) Solution for $\alpha = 2$ at $t = 0.8$



(c) Grid for $\alpha = 1.99$ at $t = 0.8$

(d) Grid for $\alpha = 2$ at $t = 0.8$

Figure A.8: Solutions and adaptive grids for PDE (5.20) for $\alpha = 1.99$, and PDE (5.24) at $t = 0.8$. As expected we see no visible difference in the solutions or grids. Adaptive grid parameters $\tau = 1$, $\beta = 5$ and an $(N + 2) \times (N + 2) = 31 \times 31$ grid.

Appendix B

Python code for numerical methods

The following pages include the code used to generate the figures presented in this thesis. The functions for the L1(NU), L2(NU), L1, L2, Grünwald-Letnikov, shifted Grünwald-Letnikov and the method by Duo et al. [18] are first shown, along with code for some example numerical experiments. Then the functions for the L1(NU)-2D and L2(NU)-2D methods are shown, again with code for some example numerical experiments.

Python code for numerical methods in one dimension:

L1(NU) and L2(NU) for non-uniform grids, L1, L2, Grünwald-Letnikov and method by Duo et al. [18] for uniform grids

[ ]:
```
#%matplotlib notebook
from __future__ import print_function, division
from mpl_toolkits import mplot3d
import matplotlib.pyplot as plt
import numpy as np
## The scikits package an be imported via anaconda: https://anaconda.org/conda-forge/scikits.
↪odes#:~:text=Odes%20is%20a%20scikit%20toolkit,integrade.
from scikits.odes import dae
import math
from time import time
## The Assimulo package can be downloaded from https://jmodelica.org/assimulo/download.html#
from assimulo.solvers import IDA
from assimulo.problem import Implicit_Problem
from assimulo.solvers import Radau5DAE
from scipy.integrate import odeint
from mpl_toolkits.mplot3d import Axes3D
```

[ ]:
```
## functions used for L1(NU) method ##

def W_1_R(k,DeltaX,i):
    aa=0
    bb=0
    for j in range(i,k+1):
        aa+=DeltaX[j]
    for j in range(i,k):
        bb+=DeltaX[j]
    return (1/(DeltaX[k]))*(aa**(1-alpha)-bb**(1-alpha))

def W_1_L(k,DeltaX,i):
    aa=0
    bb=0
    for j in range(k,i):
        aa+=DeltaX[j]
    for j in range(k+1,i):
        bb+=DeltaX[j]
    return (1/(DeltaX[k]))*(aa**(1-alpha)-bb**(1-alpha))

## functions used for L2(NU) method ##

def W_L_NU(k,n,DeltaX,alpha):
    z=2-alpha
    aa=0
    bb=0
    cc=0
    dd=0
    ee=0
    ff=0
    if k==0:
        for j in range(0,n):
            aa+=DeltaX[j]
        for j in range(1,n):
            bb+=DeltaX[j]
        wk=(aa**z-bb**z)/(DeltaX[0]*(DeltaX[0]+DeltaX[1]))
    elif k==1:
        for j in range(0,n):
            aa+=DeltaX[j]
        for j in range(1,n):
            bb+=DeltaX[j]
        cc=bb
        for j in range(2,n):
            dd+=DeltaX[j]
        wk=-(aa**z-bb**z)/(DeltaX[0]*DeltaX[1])+(cc**z-dd**z)/(DeltaX[1]*(DeltaX[1]+DeltaX[2]))
    elif k in range(2,n):
        for j in range(k-2,n):
            aa+=DeltaX[j]
        for j in range(k-1,n):
            bb+=DeltaX[j]
        cc=bb
        for j in range(k,n):
            dd+=DeltaX[j]
        ee=dd
        for j in range(k+1,n):
            ff+=DeltaX[j]
        wk=(aa**z-bb**z)/(DeltaX[k-1]*(DeltaX[k-2]+DeltaX[k-1]))-(cc**z-dd**z)/
↪(DeltaX[k-1]*DeltaX[k])+(ee**z-ff**z)/(DeltaX[k]*(DeltaX[k]+DeltaX[k+1]))
    elif k==n:
        for j in range(n-2,n):
            aa+=DeltaX[j]
        for j in range(n-1,n):
            bb+=DeltaX[j]
        wk=((aa)**z-(bb)**z)/(DeltaX[n-1]*(DeltaX[n-1]+DeltaX[n-2]))-((bb)**z)/
↪(DeltaX[n-1]*DeltaX[n])
    elif k==n+1:
        wk=((DeltaX[n-1])**z)/(DeltaX[n]*(DeltaX[n-1]+DeltaX[n]))
    return wk


def W_R_NU(k,n,DeltaX,alpha,N):
    z=2-alpha
    aa1=0
    bb1=0
    cc1=0
    dd1=0
    ee1=0
    ff1=0
    if k==N+1:
        for j in range(n,N+1):
            aa1+=DeltaX[j]
        for j in range(n,N):
            bb1+=DeltaX[j]
        wk=(aa1**z-bb1**z)/(DeltaX[N]*(DeltaX[N-1]+DeltaX[N]))
    elif k==N:
        for j in range(n,N):
            aa1+=DeltaX[j]
        for j in range(n,N-1):
            bb1+=DeltaX[j]
        for j in range(n,N+1):
            cc1+=DeltaX[j]
        dd1=aa1
        wk=(aa1**z-bb1**z)/(DeltaX[N-1]*(DeltaX[N-1]+DeltaX[N-1]))-(cc1**z-dd1**z)/
↪(DeltaX[N-1]*DeltaX[N])
    elif k in range(n+1,N):
        for j in range(n,k):
            aa1+=DeltaX[j]
        for j in range(n,k-1):
            bb1+=DeltaX[j]
        for j in range(n,k+1):
            cc1+=DeltaX[j]
        dd1=aa1
        for j in range(n,k+2):
            ee1+=DeltaX[j]
        ff1=cc1
        wk=(aa1**z-bb1**z)/(DeltaX[k-1]*(DeltaX[k-1]+DeltaX[k-2]))-(cc1**z-dd1**z)/
↪(DeltaX[k]*DeltaX[k-1])+(ee1**z-ff1**z)/(DeltaX[k]*(DeltaX[k+1]+DeltaX[k]))
    elif k==n:
        wk=-((DeltaX[n])**z)/(DeltaX[n]*DeltaX[n-1])+((DeltaX[n]+DeltaX[n+1])**z-(DeltaX[n])**z)/
↪(DeltaX[n]*(DeltaX[n+1]+DeltaX[n]))
    elif k==n-1:
        wk=((DeltaX[n])**z)/(DeltaX[n-1]*(DeltaX[n]+DeltaX[n-1]))
    return wk

## function to approximate RHS of PDE ##

def F_right_PDE(u,x,N,t):
    u=np.append(bcL,u)
    u=np.append(u,bcR)
    x=np.append(a,x)
    x=np.append(x,b)
    ##########
    # L1(NU) #
    ##########
    if alpha<1:
        dudt = np.zeros(N)
        DeltaX=np.zeros(N+1)      # step sizes
        for i in range(0,N+1):
            DeltaX[i]=x[i+1]-x[i] # DeltaX[0]=x_1-x_0, DeltaX[N]=x_(N+1)-x_N
        for i in range(1, N+1):
            for k in range(0,i):
                dudt[i-1]+=W_1_L(k,DeltaX,i)*(1/math.gamma(2-alpha))*(u[k+1]-u[k])
            for k in range(i,N+1):
                dudt[i-1]+=-W_1_R(k,DeltaX,i)*(1/math.gamma(2-alpha))*(u[k+1]-u[k])
            dudt[i-1]=d*dudt[i-1]*(-1/(2*np.cos(alpha*np.pi/2))) # fractional Laplacian term
            dudt[i-1]+=v*(u[i+1]-u[i-1])/(x[i+1]-x[i-1])         # convection term
            dudt[i-1]+=r*u[i]*(1-u[i])                           # Fisher term
        F=dudt
    ##########
    # L2(NU) #
    ##########
    elif alpha<2:
        dudt = np.zeros(N)
        DeltaX=np.zeros(N+1)      # step sizes
        for i in range(0,N+1):
            DeltaX[i]=x[i+1]-x[i] # DeltaX[0]=x_1-x_0, DeltaX[N]=x_(N+1)-x_N
        for i in range(1, N+1):
            for k in range(0,i+2):
                dudt[i-1] += W_L_NU(k,i,DeltaX,alpha)*(u[k])
            for k in range(i-1,N+2):
                dudt[i-1] += W_R_NU(k,i,DeltaX,alpha,N)*(u[k])
            dudt[i-1]=d*(dudt[i-1]*(2)/(math.gamma(3-alpha)))*(-1/(2*np.cos(alpha*np.pi/2))) #
↪fractional Laplacian term
```

```
            dudt[i-1]+=v*(u[i+1]-u[i-1])/(x[i+1]-x[i-1])         # convection term
            dudt[i-1]+=r*u[i]*(1-u[i])                           # Fisher term
        F=dudt
    ###########
    # alpha=2 #
    ###########
    else:
        F=np.zeros(N)
        for i in range(1,N+1):
            F[i-1]=d*(2*(u[i-1])/((x[i]-x[i-1])*((x[i+1]-x[i])+(x[i]-x[i-1])))-2*(u[i])/
↪((x[i]-x[i-1])*(x[i+1]-x[i]))+2*(u[i+1])/((x[i+1]-x[i])*((x[i]-x[i-1])+(x[i+1]-x[i]))))␣
↪#Laplacian term
            F[i-1]+=v*(1/2)*(u[i+1]-u[i-1])/(x[i+1]-x[i-1])  # convection term
            F[i-1]+=r*u[i]*(1-u[i])                          # Fisher term
    return F #F[0]=F_1,..., F[N-1]=F_N

def D_matrix(u,x,N):
    x=np.append(a,x)
    x=np.append(x,b)
    u=np.append(bcL,u)
    u=np.append(u,bcR)
    D=np.zeros(N)        #D[0]=D_1,..., D[N-1]=D_N
    for i in range(1,N+1):
        D[i-1]=(u[i+1]-u[i-1])/(x[i+1]-x[i-1])
    return D

def monitor_function(A,u,x,N):
    x=np.append(a,x)
    x=np.append(x,b)
    u=np.append(bcL,u)
    u=np.append(u,bcR)
    M=np.zeros(N+1)        #M[0]=M_0,..., M[N]=M_N
    for i in range(0,N+1):
        M[i]=np.sqrt(A+beta*(1/2*u[i+1]-1/2*u[i-1])**2/(1/2*x[i+1]-1/2*x[i-1])**2) # central may␣
↪be better?
    return M

def g_vector(x,M,mu,N):
    g=np.zeros(N)
    x=np.append(a,x)
    x=np.append(x,b)
    for i in range(2,N):
        g[i-1]=(-mu/(x[i+2]-x[i+1])+(1+2*mu)/(x[i+1]-x[i])-mu/(x[i]-x[i-1]))/M[i]-(-mu/
↪(x[i+1]-x[i])+(1+2*mu)/(x[i]-x[i-1])-mu/(x[i-1]-x[i-2]))/M[i-1]
    return g

def B_matrix(x,M,mu,N):
    DeltaX=np.zeros(N+1)
    x=np.append(a,x)
    x=np.append(x,b)
    for i in range(0,N+1):
        DeltaX[i]=x[i+1]-x[i]
    B=np.zeros((N,N))
    for i in range(1,N+1):
        if i==1:
            B[0,0]=2.0
            B[0,1]=-1.0
        elif i==2:
            B[1,0]=mu/(M[i]*(DeltaX[i-1])**2)+(1+2*mu)/(M[i-1]*(DeltaX[i-1])**2)+mu/
↪(M[i-1]*(DeltaX[i-2])**2)
            B[1,1]=-(mu/(M[i]*(DeltaX[i-1])**2)+(1+2*mu)/(M[i]*(DeltaX[i])**2)+(1+2*mu)/
↪(M[i-1]*(DeltaX[i-1])**2)+mu/(M[i-1]*(DeltaX[i])**2))
            B[1,2]=mu/(M[i]*(DeltaX[i+1])**2)+(1+2*mu)/(M[i]*(DeltaX[i])**2)+mu/
↪(M[i-1]*(DeltaX[i])**2)
            B[1,3]=-mu/(M[i]*(DeltaX[i+1])**2)
        elif i in range(3,N-1):
            B[i-1,i-3]=-mu/(M[i-1]*(DeltaX[i-2])**2)
            B[i-1,i-2]=mu/(M[i]*(DeltaX[i-1])**2)+(1+2*mu)/(M[i-1]*(DeltaX[i-1])**2)+mu/
↪(M[i-1]*(DeltaX[i-2])**2)
            B[i-1,i-1]=-(mu/(M[i]*(DeltaX[i-1])**2)+(1+2*mu)/(M[i]*(DeltaX[i])**2)+(1+2*mu)/
↪(M[i-1]*(DeltaX[i-1])**2)+mu/(M[i-1]*(DeltaX[i])**2))
            B[i-1,i]=mu/(M[i]*(DeltaX[i+1])**2)+(1+2*mu)/(M[i]*(DeltaX[i])**2)+mu/
↪(M[i-1]*(DeltaX[i])**2)
            B[i-1,i+1]=-mu/(M[i]*(DeltaX[i+1])**2)
        elif i==N-1:
            B[N-2,N-4]=-mu/(M[i-1]*(DeltaX[i-2])**2)
            B[N-2,N-3]=mu/(M[i]*(DeltaX[i-1])**2)+(1+2*mu)/(M[i-1]*(DeltaX[i-1])**2)+mu/
↪(M[i-1]*(DeltaX[i-2])**2)
            B[N-2,N-2]=-(mu/(M[i]*(DeltaX[i-1])**2)+(1+2*mu)/(M[i]*(DeltaX[i])**2)+(1+2*mu)/
↪(M[i-1]*(DeltaX[i-1])**2)+mu/(M[i-1]*(DeltaX[i])**2))
            B[N-2,N-1]=mu/(M[i]*(DeltaX[i+1])**2)+(1+2*mu)/(M[i]*(DeltaX[i])**2)+mu/
↪(M[i-1]*(DeltaX[i])**2)
        elif i==N:
            B[N-1,N-2]=-1.0
            B[N-1,N-1]=2.0
    return B

##### RESIDUAL FUNCTIONS ######

# ## If using the sundials IDA integrator use this residual function ##
# def residual(t,x,xdot,result):
#     DD=D_matrix(x[0:N],x[N:2*N],N)
#     AA=F_right_PDE(x[0:N],x[N:2*N],N,t)
#     M=monitor_function(A,x[0:N],x[N:2*N],N)
#     g=g_vector(x[N:2*N],M,mu,N)
#     B=B_matrix(x[N:2*N],M,mu,N)
#     for i in range(0,N):
#         result[i]=-xdot[i]+AA[i]+DD[i]*xdot[i+N]
#     for i in range(N,2*N):
#         result[i]=g[i-N]-tau*B[i-N,:].dot(xdot[N:2*N])
#     ## print time so that we can track the progress ##
#     print(t)
#     #return result

## If using the Assimulo integrator use this residual function ##

def residual(t,x,xdot):#,result):
    result=np.zeros(2*N)
    DD=D_matrix(x[0:N],x[N:2*N],N)
    AA=F_right_PDE(x[0:N],x[N:2*N],N,t)
    M=monitor_function(A,x[0:N],x[N:2*N],N)
    g=g_vector(x[N:2*N],M,mu,N)
    B=B_matrix(x[N:2*N],M,mu,N)
    for i in range(0,N):
        result[i]=-xdot[i]+AA[i]+DD[i]*xdot[i+N]
    for i in range(N,2*N):
        result[i]=g[i-N]-tau*B[i-N,:].dot(xdot[N:2*N])
    ## print time so that we can track the progress ##
    print(t)
    return result
```

[ ]:
```
## functions for Grünwald-Letnikov methods

def generalized_binomial(x,y):
    return math.gamma(x+1) / (math.gamma(y+1) * math.gamma(x-y+1))
import scipy.special

#########################################################################
# GL left derivative (2.51) in Numerical methods for fractional calculus #
#########################################################################
# works well for alpha < 1

def GL_Backwards(u,t):
    dudt = np.zeros(u0.shape)
    h = (b-a)/(N-1) #step size
    u[0]=0
    u[N-1]=0
    for i in range(1, N-1):
        for k in range(0,i+1):
            dudt[i] += (-1)**k*generalized_binomial(alpha,k)*u[i-k]
        dudt[i]=dudt[i]*(1/h**alpha)
    return dudt

#######################
# GL right derivative #
#######################
# works well for alpha < 1

def GL_Forwards(u,t):
    dudt = np.zeros(u0.shape)
    h = (b-a)/(N-1) #step size
    u[0]=0
    u[N-1]=0
    for i in range(1, N-1):
        for k in range(0,N-i):
            dudt[i] += (-1)**k*generalized_binomial(alpha,k)*u[i+k]
        dudt[i]=dudt[i]*(1/h**alpha)
    return dudt #was -

##############################################
# GL left and right to find Reiss Derivative #
##############################################
# works well for alpha < 1

def GL_RD(u,t):
    dudt = np.zeros(u0.shape)
    #step size
    h = (b-a)/(N-1)
    u[0]=0
    u[N-1]=0
    for i in range(1, N-1):
        for k in range(0,i+1):
            dudt[i] += (-1)**k*scipy.special.binom(alpha,k)*u[i-k]
        for k in range(0,N-i):
            dudt[i] += (-1)**k*scipy.special.binom(alpha,k)*u[i+k]
        dudt[i]=dudt[i]*(1/h**alpha)
        dudt[i]=dudt[i]*(-1/(2*np.cos(alpha*np.pi/2)))
        dudt[i]+=v*(u[i+1]-u[i-1])/(2*h)
    return dudt

#######################
# initial conditions  #
#######################

def IC(x,N,p):
    u=np.zeros(N)
    for i in range(1,N-1):
        u[i]=np.exp(-p*x[i]**2)
    return u


#########################################################################
################################### SHIFTED #############################
# GL left derivative (2.52) in Numerical methods for fractional calculus #
#########################################################################
# works well for 1 < alpha < 2

def GLM_Backwards(u,t):
    dudt = np.zeros(u0.shape)
    h = (b-a)/(N-1)
    u[0]=0
    u[N-1]=0
    for i in range(1, N-1):
        for k in range(0,i+shift_p+1):
            dudt[i] += (-1)**k*generalized_binomial(alpha,k)*u[i-k+shift_p]
        dudt[i]=dudt[i]*(1/h**alpha)
    return dudt

#######################
####### SHIFTED #######
# GL right derivative #
#######################
# works well for 1 < alpha < 2

def GLM_Forwards(u,t):
    dudt = np.zeros(u0.shape)
    h = (b-a)/(N-1)
    u[0]=0
    u[N-1]=0
    for i in range(1, N-1):
        for k in range(0,N-i-shift_p+1):
            dudt[i] += (-1)**k*generalized_binomial(alpha,k)*u[i+k-shift_p]
        dudt[i]=dudt[i]*(1/h**alpha)
    return dudt

##############################################
########## SHIFTED ###########################
# GL left and right to find Riesz Derivative #
```

```
##############################################
# works well for 1 < alpha < 2

def GLM_RD(u,t):
    dudt = np.zeros(u0.shape)
    h = (b-a)/(N-1) #step size
    u[0]=0
    u[N-1]=0
    for i in range(1, N-1):
        for k in range(0,i+shift_p+1):
            dudt[i] += (-1)**k*scipy.special.binom(alpha,k)*u[i-k+shift_p]
        for k in range(0,N-i-shift_p+1):
            dudt[i] += (-1)**k*scipy.special.binom(alpha,k)*u[i+k-shift_p]
        dudt[i]=dudt[i]*(h**(-alpha))
        dudt[i]=dudt[i]*(-1/(2*np.cos(alpha*np.pi/2)))
    return dudt
```

```
[ ]: ######################################################################
     # L1 method for C (2.63) in Numerical methods for fractional calculus #
     ######################################################################

     ## coefficient functions

     def b_k(k):
         h = (b-a)/(N-1)
         return (h**(-alpha))*((k+1)**(1-alpha)-k**(1-alpha))/(math.gamma(2-alpha))
     def a_k(k):
         h = (b-a)/(N-1)
         return (h**(-alpha))*((k+1)**(1-alpha)-(k)**(1-alpha))/(math.gamma(2-alpha))

     ## Approximate left Caputo derivative

     def L1_method_left(u,t):
         dudt = np.zeros(u0.shape)
         h = (b-a)/(N-1)
         u[0]=0
         u[N-1]=0
         for i in range(1, N-1):
             dudt[i]=0
             for k in range(0,i):
                 dudt[i] += b_k(i-k-1)*(u[k+1]-u[k])
         return dudt

     ## Approximate right Caputo derivative

     def L1_method_right(u,t):
         dudt = np.zeros(u0.shape)
         h = (b-a)/(N-1) #step size
         u[0]=0
         u[N-1]=0
         for i in range(1, N-1):
             dudt[i]=0
             for k in range(i,N-1):
                 dudt[i] += a_k(i-k-1)*(u[k+1]-u[k])
         return dudt

     ## Caputo left and right combined to find Riesz derivative
```

```
def L1_method(u,t):
    dudt = np.zeros(u0.shape)
    h = (b-a)/(N-1) #step size
    u[0]=0
    u[N-1]=0
    for i in range(1, N-1):
        for k in range(0,i):
            dudt[i] += b_k(i-k-1)*(u[k+1]-u[k])
        for k in range(i,N-1):
            dudt[i] += -a_k(k-i)*(u[k+1]-u[k])
        dudt[i]=dudt[i]*(-1/(2*np.cos(alpha*np.pi/2)))
        dudt[i]+=v*(u[i+1]-u[i-1])/(2*h)
    return dudt
```

```
[ ]: ######################################################################
     # L2 method for C (2.72) in Numerical methods for fractional calculus #
     ######################################################################

     ## coefficient functions

     def W_L2_left(i,k,h):
         return (h**(-alpha))*((i-k)**(2-alpha)-(i-k-1)**(2-alpha))/math.gamma(3-alpha)

     def W_L2_right(i,k,h):
         return h**(-alpha)*((k-i+1)**(2-alpha)-(k-i)**(2-alpha))/math.gamma(3-alpha)

     ## Caputo left and right combined to find Riesz derivative

     def L2_method(u,t):
         dudt = np.zeros(u0.shape)
         h = (b-a)/(N-1) #step size
         u[0]=0
         u[N-1]=0
         for i in range(1, N-1):
             ## The left derivative
             for k in range(0,i):
                 dudt[i] += W_L2_left(i,k,h)*(u[k]-2*u[k+1]+u[k+2])
             ## The right derivative
             for k in range(i,N-1):
                 dudt[i] += W_L2_right(i,k,h)*(u[k+1]-2*u[k]+u[k-1])
             dudt[i]=dudt[i]*(-1/(2*np.cos(alpha*np.pi/2)))
             dudt[i]+=v*(u[i+1]-u[i-1])/(2*h)
         return dudt
```

```
[ ]: ##############################################################################################
     ## Method from "A novel and accurate finite difference method for the fractional Laplacian" ##
     ######################################### by  Duo et. al #####################################
     ##############################################################################################

     ## functio for coefficient

     def C_h_alpha_gamma(alpha,h,nu):
         c_1_alpha=(2**(alpha-1)*alpha*math.gamma((alpha+1)/2))/(np.sqrt(np.pi)*math.gamma(1-alpha/2))
         return c_1_alpha/(nu*h**alpha)

     ## The matrix A as given in the paper

     def MatrixA(K,nu, gamma, alpha, kappa_gamma,h):
```

```
    A=np.zeros((int(K)-1,int(K)-1))
    for i in range(1,int(K)):
        for j in range(1,int(K)):
            if j==i:
                for k in range(2, int(K)-1):
                    A[i-1,j-1]+=((k+1)**nu-(k-1)**nu)/(k**gamma)
                A[i-1,j-1]+=2*nu/(alpha*(K**alpha))+kappa_gamma+2**nu-1+(K**nu-(K-1)**nu)/
↪(K**gamma)
            elif j==i-1:
                A[i-1,j-1]=-1/2*(2**nu+kappa_gamma-1)
            elif j==i+1:
                A[i-1,j-1]=-1/2*(2**nu+kappa_gamma-1)
            else:
                A[i-1,j-1]=-((np.abs(j-i)+1)**nu-(np.abs(j-i)-1)**nu)/(2*np.abs(j-i)**gamma)
    A=C_h_alpha_gamma(alpha,h,nu)*A
    return A

## The matrix A using eq (2.4) from thesis (equivalent to the paper)

def MyMatrixA(K,nu, gamma, alpha, kappa_gamma,h):
    A=np.zeros((int(K)-1,int(K)-1))
    for i in range(1,int(K)):
        for j in range(1,int(K)):
            if j==i:
                for k in range(2, int(K)-1):
                    A[i-1,j-1]+=((k)**nu-(k-1)**nu)/(k**gamma)+((k)**nu-(k-1)**nu)/((k-1)**gamma)
                A[i-1,j-1]+=2*nu/(alpha*(K**alpha))+kappa_gamma+(K**nu-(K-1)**nu)/
↪(K**gamma)+(K**nu-(K-1)**nu)/((K-1)**gamma)
            elif j==i-1:
                A[i-1,j-1]=-1/2*(2**nu+kappa_gamma-1)
            elif j==i+1:
                A[i-1,j-1]=-1/2*(2**nu+kappa_gamma-1)
            else:
                A[i-1,j-1]=-((np.abs(j-i)+1)**nu-(np.abs(j-i)-1)**nu)/(2*np.abs(j-i)**gamma)
    A=C_h_alpha_gamma(alpha,h,nu)*A
    return A

## Function to generate Figure 3 in the original paper

def frompaper(L, alpha, gamma, K):
    l=L/2
    nu=gamma-alpha
    h=L/K           # step size
    if gamma<2:
        kappa_gamma=1
    else:
        kappa_gamma=2
    x=np.zeros(int(K)-1)
    for i in range(0,int(K)-1):
        x[i]=-l+(i+1)*h
    u=np.zeros(int(K)-1)
    for i in range(0,int(K)-1): # implement the initial condition
        u[i]=np.exp(-x[i]**2)
    ## To use expression from paper mor matrix A
    A=MatrixA(K,nu, gamma, alpha, kappa_gamma,h)
    ## To use the expression in eq (2.4) of thesis for matrix A (equivalent to paper)
    #A=MyMatrixA(K,nu, gamma, alpha, kappa_gamma,h)
    du=np.zeros(int(K)-1)
    for i in range(0,int(K)-1):
        du[i]=np.dot(A[i,:],u)
    return plt.plot(x, du, label=r'from paper $\alpha$=%f' %alpha)

## Function to determine right hand side of space-fractional heat equation

def matrix_method(u,t):
    l=L/2
    nu=gamma-alpha
    h=2*l/K           # step size
    if gamma<2:
        kappa_gamma=1
    elif gamma==2:
        kappa_gamma=2
    ## To use the expression in eq (2.4) of thesis for matrix A (equivalent to paper)
    A=MyMatrixA(K,nu, gamma, alpha, kappa_gamma,h)
    ## To use expression from paper mor matrix A
#     A=MatrixA(K,nu, gamma, alpha, kappa_gamma,h)
    dudt = np.zeros(u0.shape)
    dudt[1:N-1]=-A.dot(u[1:N-1])
    return dudt
```

[ ]:
```
############################################
## Generate solutions on an adaptive grid ##
############################################

N=99          # number of grid points
tau=1e-3      # adaptive grid parameter, the smaller it is the more adaptive the grid
kappa=2       # adaptive grid parameter
A=1.0         # value in monitor function
beta=1.0      # value in monitor function
mu=kappa*(kappa+1)


    ## the right hand side of the PDE takes the following form:
    ##    (    d^2 ) ^(alpha/2)               d
    ## - (- -----)            u(x,t)*d + v*---- u(x,t) + r*u(x,t)*(1-u(x,t))
    ##    (    dx^2)                          dx
    ## where d, v and r are constants

## data of the system ##

a=-1.0      # x_0
b=1.0       # x_(N+1)
alpha=0.5   # order of fractional derivative, choose 2 for ordinary heat
d=1.0       # diffusion coefficient
v=0.0       # advection constant
r=0.0       # Fisher constant
p=100        # IC constant

## Boundary conditions ##

bcL=0       # value at left boundary
bcR=0       # value at right boundary

## initial condition x and u ##

def ivp(x0, N):
    u0=np.zeros(N+2)
    for i in range(1,N+2):
        u0[i]=1.0*np.exp(-p*(x0[i])**2) ## Gaussian type on symmetric interval around 0
    return u0
x0= np.linspace(a, b, N+2)

## initial step sizes

DeltaX=np.zeros(N+1)
for i in range(0,N+1):
    DeltaX[i]=x0[i+1]-x0[i] ## DeltaX[0]=x_1-x_0, DeltaX[N]=x_(N+1)-x_N

## create the initial vector Y

u0=ivp(x0,N)
y0=np.append(u0[1:N+1],x0[1:N+1]) ## y0=[u1,...,uN,x1,...xN]

## create the inital Y' vector (the integrator also makes this better) ##

xp0=1/tau*np.linalg.inv(B_matrix(x0[1:N+1],monitor_function(A,u0[1:N+1],x0[1:N+1],N),mu,N)).
 ↪dot(g_vector(x0[1:N+1],monitor_function(A,u0[1:N+1],x0[1:N+1],N),mu,N))
up0=np.diag(D_matrix(u0[1:N+1],x0[1:N+1],N)).dot(xp0)+F_right_PDE(u0[1:N+1],x0[1:N+1],N,0)
yp0=np.append(up0,xp0)

## Now choose to use sundials IDA integrator or Assimulo Radau5DAE ##

##########################
# For using sundials IDA #
##########################
# start = time()
# solver = dae('ida', residual,
#               first_step_size=1e-6,
#               atol=1e-5,                # tolernaces
#               rtol=1e-5,
#               compute_initcond='yp0',
#               max_steps=500,
#               ontstop=True)

# # define the times for the solution
# times=np.linspace(0,0.5,251)#251
# solution = solver.solve(times, y0, yp0)
# print(f'Time taken to run: {time() - start} seconds')
# runtime=time() - start

#############################
# For using Assimulo solver #
#############################
start = time()
mod = Implicit_Problem(residual, y0, yp0, 0)
sim=Radau5DAE(mod)
sim.atol=1e-4      # tolerances
sim.rtol=1e-4
sim.thet=0.5       # larger value makes te system compute new jacobian less often a speed up␣
 ↪runtime
t, u, yd= sim.simulate(tfinal=0.5, ncp=251) # ncp is the number of returned solutions at␣
 ↪different times
runtime=time() - start
print(f'Time taken to run: {time() - start} seconds')
```

```
[ ]: ## plot a sample of the solutions (10 equally spaced) for sundials IDA ##

plt.figure(figsize=(15,10))
for j in range(0,11):
    i=int(j*(len(times)-1)/10)
    plt.plot(np.pad(solution.values.y[i,:][N:2*N],(1,1), 'constant',constant_values=(a, b)),np.
 ↪pad(solution.values.y[i,:][0:N],(1,1), 'constant',constant_values=(bcL, bcR)),␣
 ↪label='t='+str(round(times[i],3)))#+str(round(c*tf/(no_time-1),3))
plt.xlabel('x')
plt.ylabel('u(x,t)')
plt.legend()
plt.title(r'$\alpha $='+str(alpha)+', N='+str(N)+', d='+str(d)+', r='+str(r)+r',␣
 ↪$\kappa$='+str(kappa)+r', $\tau$='+str(tau)+', A='+str(A)+r', $\beta$='+str(beta)+',␣
 ↪a='+str(a)+', b='+str(b)+', p='+str(p)+', runtime='+str(runtime)+',␣
 ↪$u(x,0)=\exp(-$'+str(p)+r'$x^2)$')

plt.show()

## plot the adaptive grid ##

plt.figure(figsize=(15,10))
for i in range(0,len(times)):
    plt.plot(np.pad(solution.values.y[i,:][N:2*N],(1,1), 'constant',constant_values=(a, b)),(np.
 ↪ones(N+2)*solution.values.t[i]),'o',markersize=2, label='grid at t='+str(times[i]))
plt.xlabel('x')
plt.ylabel('t')
plt.title(r'$\alpha$='+str(alpha)+', N='+str(N)+', d='+str(d)+', r='+str(r)+r',␣
 ↪$\kappa$='+str(kappa)+r', $\tau$='+str(tau)+', A='+str(A)+r', $\beta$='+str(beta)+',␣
 ↪a='+str(a)+', b='+str(b)+', p='+str(p)+', runtime='+str(runtime)+',␣
 ↪$u(x,0)=\exp(-$'+str(p)+r'$x^2)$')
plt.show()
```

```
[ ]: ## plot a sample of the solutions (10 equally spaced) for Assimulo Radau5DAE ##

plt.figure(figsize=(15,10))
for j in range(0,11):
    i=int(j*(len(t)-1)/10)
    plt.plot(np.pad(u[i,:][N:2*N],(1,1), 'constant',constant_values=(a, b)),np.pad(u[i,:][0:
 ↪N],(1,1), 'constant',constant_values=(bcL, bcR)),␣
 ↪label='t='+str(round(t[i],3)))#+str(round(c*tf/(no_time-1),3))
plt.xlabel('x')
plt.ylabel('u(x,t)')
plt.legend()
plt.title(r'$\alpha $='+str(alpha)+', N='+str(N)+', d='+str(d)+', r='+str(r)+r',␣
 ↪$\kappa$='+str(kappa)+r', $\tau$='+str(tau)+', A='+str(A)+r', $\beta$='+str(beta)+',␣
 ↪a='+str(a)+', b='+str(b)+', p='+str(p)+', runtime='+str(runtime)+',␣
 ↪$u(x,0)=\exp(-$'+str(p)+r'$x^2)$')

plt.show()

## plot the adaptive grid ##

plt.figure(figsize=(15,10))
for i in range(0,len(t)):
    plt.plot(np.pad(u[i,:][N:2*N],(1,1), 'constant',constant_values=(a, b)),(np.
 ↪ones(N+2)*t[i]),'o',markersize=1, label='grid at t='+str(t[i]))
plt.xlabel('x')
plt.ylabel('t')
plt.title(r'$\alpha$='+str(alpha)+', N='+str(N)+', d='+str(d)+', r='+str(r)+r',␣
↪$\kappa$='+str(kappa)+r', $\tau$='+str(tau)+', A='+str(A)+r', $\beta$='+str(beta)+',␣
↪a='+str(a)+', b='+str(b)+', p='+str(p)+', runtime='+str(runtime)+',␣
↪$u(x,0)=\exp(-$'+str(p)+r'$x^2)$')
plt.show()
```

[ ]:
```
## Plot solutions using Grünwald-Letnikov methods on uniform grids ##

N = 101     # number of gridpoints
alpha=0.5  # order of fractional deriavtives
a=-1       # x_0
b=1        # x_{N-1}
t0 = 0     # initial time
tf = 0.5   # final time
v=0        # velocity
shift_p=1  # shifting parameter for shifted GL fomulas
p=100

x = np.linspace(a, b, N) # uniform x grid
u0 = IC(x,N,p)           # initial condition

tspan = np.linspace(t0, tf, 251)

## find solutions

if alpha<1:
    sol_RD = odeint(GL_RD,u0,tspan)
elif alpha<2:
    sol_RD = odeint(GLM_RD,u0,tspan)

## plot solutions

plt.figure(figsize=(15,10))
for j in range(0,11):
    i=int(j*(len(tspan)-1)/10)
    plt.plot(x,sol_RD[i,:], label='t='+str(round(tspan[i],3)))
plt.xlabel('x')
plt.ylabel('u(x,t)')
plt.legend()
plt.show()
```

[ ]:
```
## Plot solutions using L1 or L2 methods on uniform grids ##

N = 101     # number of gridpoints
alpha=0.5  # order of fractional deriavtives
a=-1       # x_0
b=1        # x_{N-1}
t0 = 0     # initial time
tf = 0.5   # final time
v=0        # velocity
p=100

x = np.linspace(a, b, N) # uniform x grid
u0 = IC(x,N,p)           # initial condition

tspan = np.linspace(t0, tf, 251)
```

```
## find solutions

if alpha<1:
    sol_L = odeint(L1_method,u0,tspan)
elif alpha<2:
    sol_L = odeint(L2_method,u0,tspan)

## plot solutions

plt.figure(figsize=(15,10))
for j in range(0,11):
    i=int(j*(len(tspan)-1)/10)
    plt.plot(x,sol_L[i,:], label='t='+str(round(tspan[i],3)))
plt.xlabel('x')
plt.ylabel('u(x,t)')
plt.legend()
plt.show()
```

[ ]:
```
## Plot solutions using method from Duo et. al on uniform grids ##

a=-1        # x_0
b=1         # x_{K}
t0 = 0      # initial time
tf = 0.5    # final time
K=100       # K+1 gridpoints
N=K+1       # number of gridpoints
p=100
x = np.linspace(a, b, N)  # spatial grid
u0 = IC(x,N,p)            # initial condition
alpha=0.5                 # order of fractional Laplacian
L=2                       # width of symmetric interval
gamma=1+alpha/2           # value of gamma in method
tspan = np.linspace(t0, tf, 11)  # choices of times to return solutions

## calculate the solution
sol_RD_175=odeint(matrix_method, u0,tspan)

## plot the solution
plt.figure(figsize=(15,10))
for i in range(0,len(tspan)):
    plt.plot(x,sol_RD_175[i,:], label='t='+str(round(tspan[i],3)))
plt.legend()
plt.xlabel('x')
plt.ylabel('u(x,t)')
plt.show()
```

[ ]:
```
## To generate figure similar to Figure 3 in paper bu Duo et. al.

L=20
alpha=0.25
gamma=1+alpha/2
K=500
frompaper(L, alpha, gamma, K)
alpha=0.5
gamma=1+alpha/2
frompaper(L, alpha, gamma, K)
alpha=0.75
gamma=1+alpha/2
```

```
frompaper(L, alpha, gamma, K)
alpha=1
gamma=1+alpha/2
frompaper(L, alpha, gamma, K)
alpha=1.25
gamma=1+alpha/2
frompaper(L, alpha, gamma, K)
alpha=1.5
gamma=1+alpha/2
frompaper(L, alpha, gamma, K)
alpha=1.75
gamma=1+alpha/2
frompaper(L, alpha, gamma, K)
limits = [ -4,4, -1, 2.2]
plt.axis(limits)
```

Python code for numerical methods in two dimensions:

L1(NU)-2D and L2(NU)-2D for non-uniform grids

[ ]:
```python
from __future__ import division
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import math
from scipy.integrate import solve_bvp
from mpl_toolkits import mplot3d
from matplotlib import cm
from time import time
## The Assimulo package can be downloaded from https://jmodelica.org/assimulo/download.html#
from assimulo.solvers import IDA
from assimulo.problem import Implicit_Problem
from assimulo.solvers import Radau5DAE
```

[ ]:
```python
#######################
# initial conditions  #
#######################

def IC(x,y,N):
    u=np.zeros((N+2,N+2))
    for i in range(1,N+1):
        for j in range(1,N+1):
            u[i,j]=mag*np.exp(-p*((x[i,j]-0)**2+(y[i,j]-0)**2))
    return u

###############################
# coefficients for L2(NU)-2D  #
###############################

def W_L_NU(k,n,DeltaX,alpha):
    z=2-alpha
    aa=0
    bb=0
    cc=0
    dd=0
    ee=0
    ff=0
    if k==0:
        for j in range(0,n):
            aa+=DeltaX[j]
        for j in range(1,n):
            bb+=DeltaX[j]
        wk=(aa**z-bb**z)/(DeltaX[0]*(DeltaX[0]+DeltaX[1]))
    elif k==1:
        for j in range(0,n):
            aa+=DeltaX[j]
        for j in range(1,n):
            bb+=DeltaX[j]
        cc=bb
        for j in range(2,n):
            dd+=DeltaX[j]
        wk=-(aa**z-bb**z)/(DeltaX[0]*DeltaX[1])+(cc**z-dd**z)/(DeltaX[1]*(DeltaX[1]+DeltaX[2]))
    elif k in range(2,n):
        for j in range(k-2,n):
            aa+=DeltaX[j]
        for j in range(k-1,n):
            bb+=DeltaX[j]
        cc=bb
        for j in range(k,n):
            dd+=DeltaX[j]
        ee=dd
        for j in range(k+1,n):
            ff+=DeltaX[j]
        wk=(aa**z-bb**z)/(DeltaX[k-1]*(DeltaX[k-2]+DeltaX[k-1]))-(cc**z-dd**z)/
↪(DeltaX[k-1]*DeltaX[k])+(ee**z-ff**z)/(DeltaX[k]*(DeltaX[k]+DeltaX[k+1]))
    elif k==n:
        for j in range(n-2,n):
            aa+=DeltaX[j]
        for j in range(n-1,n):
            bb+=DeltaX[j]
        wk=((aa)**z-(bb)**z)/(DeltaX[n-1]*(DeltaX[n-1]+DeltaX[n-2]))-((bb)**z)/
↪(DeltaX[n-1]*DeltaX[n])
    elif k==n+1:
        wk=((DeltaX[n-1])**z)/(DeltaX[n]*(DeltaX[n-1]+DeltaX[n]))
    return wk


def W_R_NU(k,n,DeltaX,alpha,N):
    z=2-alpha
    aa1=0
    bb1=0
    cc1=0
    dd1=0
    ee1=0
    ff1=0
    if k==N+1:
        for j in range(n,N+1):
            aa1+=DeltaX[j]
        for j in range(n,N):
            bb1+=DeltaX[j]
        wk=(aa1**z-bb1**z)/(DeltaX[N]*(DeltaX[N-1]+DeltaX[N]))
    elif k==N:
        for j in range(n,N):
            aa1+=DeltaX[j]
        for j in range(n,N-1):
            bb1+=DeltaX[j]
        for j in range(n,N+1):
            cc1+=DeltaX[j]
        dd1=aa1
        wk=(aa1**z-bb1**z)/(DeltaX[N-1]*(DeltaX[N-2]+DeltaX[N-1]))-(cc1**z-dd1**z)/
↪(DeltaX[N-1]*DeltaX[N])
    elif k in range(n+1,N):
        for j in range(n,k):
            aa1+=DeltaX[j]
        for j in range(n,k-1):
            bb1+=DeltaX[j]
        for j in range(n,k+1):
            cc1+=DeltaX[j]
        dd1=aa1
        for j in range(n,k+2):
            ee1+=DeltaX[j]
        ff1=cc1
        wk=(aa1**z-bb1**z)/(DeltaX[k-1]*(DeltaX[k-1]+DeltaX[k-2]))-(cc1**z-dd1**z)/
↪(DeltaX[k]*DeltaX[k-1])+(ee1**z-ff1**z)/(DeltaX[k]*(DeltaX[k+1]+DeltaX[k]))
    elif k==n:
        wk=-((DeltaX[n])**z)/(DeltaX[n]*DeltaX[n-1])+((DeltaX[n]+DeltaX[n+1])**z-(DeltaX[n])**z)/
↪(DeltaX[n]*(DeltaX[n+1]+DeltaX[n]))
    elif k==n-1:
        wk=((DeltaX[n])**z)/(DeltaX[n-1]*(DeltaX[n]+DeltaX[n-1]))
    return wk

###############################
# coefficients for L1(NU)-2D  #
###############################

def W_1_R(k,DeltaX,i):
    aa=0
    bb=0
    for j in range(i,k+1):
        aa+=DeltaX[j]
    for j in range(i,k):
        bb+=DeltaX[j]
    return (1/(DeltaX[k]))*(aa**(1-alpha)-bb**(1-alpha))

def W_1_L(k,DeltaX,i):
    aa=0
    bb=0
    for j in range(k,i):
        aa+=DeltaX[j]
    for j in range(k+1,i):
        bb+=DeltaX[j]
    return (1/(DeltaX[k]))*(aa**(1-alpha)-bb**(1-alpha))

#######################################
# function to approximate RHS of PDE  #
#######################################

def twodrhs(u,t,x,y):
    u=u.reshape((N+2, N+2))
    x=x.reshape((N+2, N+2))
    y=y.reshape((N+2, N+2))
    dudt=np.zeros((N+2,N+2))
    #######################
    # ordinary Laplacian  #
    #######################
    if alpha==2:
        for i in range(1,N+1):
            for j in range(1,N+1):
                # diffusion term
                dudt[i,j]=(d*(2*(u[i,j-1])/
↪((x[i,j]-x[i,j-1])*((x[i,j+1]-x[i,j])+(x[i,j]-x[i,j-1])))
                              -2*(u[i,j])/((x[i,j]-x[i,j-1])*(x[i,j+1]-x[i,j]))
                              +2*(u[i,j+1])/
↪((x[i,j+1]-x[i,j])*((x[i,j]-x[i,j-1])+(x[i,j+1]-x[i,j])))
                              +2*(u[i-1,j])/
↪((y[i,j]-y[i-1,j])*((y[i+1,j]-y[i,j])+(y[i,j]-y[i-1,j])))
                              -2*(u[i,j])/((y[i,j]-y[i-1,j])*(y[i+1,j]-y[i,j]))
                              +2*(u[i+1,j])/
↪((y[i+1,j]-y[i,j])*((y[i,j]-y[i-1,j])+(y[i+1,j]-y[i,j]))))
                # fisher term
                dudt[i,j]+=r*u[i,j]*(1-u[i,j])
                # advection term
                dudt[i,j]+=vel*(u[i+1,j]-u[i-1,j])/(y[i+1,j]-y[i-1,j])+vel*(u[i,j+1]-u[i,j-1])/
↪(x[i,j+1]-x[i,j-1])
    ############################################################
    # fractional Laplacian of order 1<alpha<2 using L2(NU)-2D  #
    ############################################################
    elif alpha<2 and alpha>1:
        for i in range(1,N+1):
            # calculate the step sizes
            DeltaX=np.zeros(N+1)
            for k in range(0,N+1):
                DeltaX[k]=x[i,k+1]-x[i,k] # DeltaX[0]=x_1-x_0, DeltaX[N]=x_(N+1)-x_N
            DeltaY=np.zeros(N+1)
            for k in range(0,N+1):
                DeltaY[k]=y[k+1,i]-y[k,i]
            for j in range(1,N+1):
                ## left derivative in x direction
                for k in range(0,i+2):
                    dudt[i,j] += W_L_NU(k,i,DeltaX,alpha)*(u[k,j])
                ## right derivative in x direction
                for k in range(i-1,N+2):
                    dudt[i,j] += W_R_NU(k,i,DeltaX,alpha,N)*(u[k,j])
                ## left derivative in y direction
                for k in range(0,j+2):
                    dudt[i,j] += W_L_NU(k,j,DeltaY,alpha)*(u[i,k])
                ## right derivative in y direction
                for k in range(j-1,N+2):
                    dudt[i,j] += W_R_NU(k,j,DeltaY,alpha,N)*(u[i,k])
                ## add weighting for fractional Laplacian and diffusion coefficient
                dudt[i,j]=dudt[i,j]*d*(2)/(math.gamma(3-alpha))*(-1/(2*np.cos(alpha*np.pi/2)))
                ## Fisher term
                dudt[i,j]+=r*u[i,j]*(1-u[i,j])
                ## Advection term
                dudt[i,j]+=vel*(u[i,j+1]-u[i,j-1])/(x[i,j+1]-x[i,j-1])+vel*(u[i+1,j]-u[i-1,j])/
↪(y[i+1,j]-y[i-1,j])
    ############################################################
    # fractional Laplacian of order 0<alpha<1 using L1(NU)-2D  #
    ############################################################
    elif alpha<1:
        for i in range(1,N+1):
            ## calculate the stepsizes
            DeltaX=np.zeros(N+1)
            for k in range(0,N+1):
                DeltaX[k]=x[i,k+1]-x[i,k]
            DeltaY=np.zeros(N+1)
            for k in range(0,N+1):
                DeltaY[k]=y[k+1,i]-y[k,i]
            for j in range(1,N+1):
                ## left derivative in x direction
                for k in range(0,i):
                    dudt[i,j] += W_1_L(k,DeltaX,i)*(u[k+1,j]-u[k,j])
                ## right derivative in x direction
                for k in range(i,N+1):
                    dudt[i,j] += -W_1_R(k,DeltaX,i)*(u[k+1,j]-u[k,j])
                ## left derivative in y direction
                for k in range(0,j):
                    dudt[i,j] += W_1_L(k,DeltaX,j)*(u[i,k+1]-u[i,k])
                ## right derivative in y direction
                for k in range(j,N+1):
                    dudt[i,j] += -W_1_R(k,DeltaX,j)*(u[i,k+1]-u[i,k])
                ## add weighting for fractional Laplacian and diffusion coefficient
                dudt[i,j]=dudt[i,j]*d*(1)/(math.gamma(2-alpha))*(-1/(2*np.cos(alpha*np.pi/2)))
                ## Fisher term
                dudt[i,j]+=r*u[i,j]*(1-u[i,j])
    return dudt.flatten()

##############################################################################
## Function for initial x and y grids, choose ccc close to zero for uniform ##
####### ccc larger skews gridpoints to be dense in center of the domain ######
##############################################################################

def x_grid(ccc):
    x=np.zeros((N+2, N+2))
    cc=np.ones(N+2)*0.01
    cc=np.linspace(0.01,ccc,int(np.floor((N+2)/2)))
    cc=np.append(cc,ccc)
    cc=np.append(cc,np.linspace(ccc,0.01,int(np.floor((N+2)/2))))
    for i in range(0,N+2):
        x[i,:]=np.linspace(a,b,N+2)
        for j in range(1,N+1):
            x[i,j]=np.sinh(x[i,j]*cc[i])/np.sinh(cc[i])
    return x.flatten()

def y_grid(ccc):
    y=np.zeros((N+2, N+2))
    cc=np.ones(N+2)*0.01
    cc=np.linspace(0.01,ccc,int(np.floor((N+2)/2)))
    cc=np.append(cc,ccc)
    cc=np.append(cc,np.linspace(ccc,0.01,int(np.floor((N+2)/2))))
    for i in range(0,N+2):
        y[:,i]=np.linspace(a,c,N+2)
        for j in range(1,N+1):
            y[j,i]=np.sinh(y[j,i]*cc[i])/np.sinh(cc[i])
    return y.flatten()

####################################################################
## Functions to calculate first derivatives in x and y directions ##
####################################################################

def DX(u,x):
    u=u.reshape((N+2, N+2))
    x=x.reshape((N+2, N+2))
    Dx=np.zeros((N+2,N+2))
    for i in range(0,N+2):
        for j in range(0,1):
            Dx[i,j]=(u[i,j+1]-u[i,j])/(x[i,j+1]-x[i,j])
        for j in range(1,N+1):
            Dx[i,j]=(u[i,j+1]-u[i,j-1])/(x[i,j+1]-x[i,j-1])
        for j in range(N+1,N+2):
            Dx[i,j]=(u[i,j]-u[i,j-1])/(x[i,j]-x[i,j-1])
    return Dx.flatten()

def DY(u,y):
    u=u.reshape((N+2, N+2))
    y=y.reshape((N+2, N+2))
    Dy=np.zeros((N+2,N+2))
    for i in range(0,1):
        for j in range(0,N+2):
            Dy[i,j]=(u[i+1,j]-u[i,j])/(y[i+1,j]-y[i,j])
    for i in range(1,N+1):
        for j in range(0,N+2):
            Dy[i,j]=(u[i+1,j]-u[i-1,j])/(y[i+1,j]-y[i-1,j])
    for i in range(N+1,N+2):
        for j in range(0,N+2):
            Dy[i,j]=(u[i,j]-u[i-1,j])/(y[i,j]-y[i-1,j])
    return Dy.flatten()

######################
## Monitor Function ##
######################

def Omega(a,b):
    a=a.reshape((N+2, N+2))
    b=b.reshape((N+2, N+2))
    aa=np.zeros((N+2,N+2))
    ba=aa
    Omega=np.zeros((N+2, N+2))
    for i in range(0,N+2):
        for j in range(0,N+2):
            Omega[i,j]=np.sqrt(1+beta*(a[i,j]**2+b[i,j]**2))
    ## Filter the monitor function
    for i in range(1,N+1):
        for j in range(1,N+1):
            Omega[i,j]=(1/4*Omega[i,j]+1/8*(Omega[i+1,j]+Omega[i-1,j]+Omega[i,j+1]+Omega[i,j-1])
                        +1/16*(Omega[i+1,j+1]+Omega[i-1,j-1]+Omega[i-1,j+1]+Omega[i+1,j-1]))
    return Omega

#######################
## Residual function ##
#######################

def residual(t,v, vp):
    result=np.zeros((N+2)**2*3)
    # calculate the right hand side
    AA=twodrhs(v[2*(N+2)**2:3*(N+2)**2].flatten(),t,v[0:1*(N+2)**2],v[1*(N+2)**2:2*(N+2)**2])
    # split vector v into peices of x coords, y coords and solution values u
    x=v[0:(N+2)**2]
    y=v[1*(N+2)**2:2*(N+2)**2]
    uu=v[2*(N+2)**2:3*(N+2)**2]
    # calculate necessary deriatives
    duxi=DX(uu,xi)
    dueta=DY(uu,eta)
    dxxi=DX(x,xi)
    dxeta=DY(x,eta)
    dyxi=DX(y,xi)
    dyeta=DY(y,eta)
    # calculate monitor function
    omega = Omega(duxi,dueta)
    # calculate components of x_theta and y_theta equations
    domegaxxixi=DX(np.multiply(omega.flatten(), dxxi),xi)
    domegaxetaeta=DY(np.multiply(omega.flatten(),dxeta),eta)
    domegayxixi=DX(np.multiply(omega.flatten(),dyxi),xi)
    domegayetaeta=DY(np.multiply(omega.flatten(),dyeta),eta)
    vpx=vp[0:(N+2)**2].reshape((N+2,N+2))
    vpy=vp[(N+2)**2:2*(N+2)**2].reshape((N+2,N+2))
    deltaxi=xi.reshape((N+2,N+2))[3,2]-xi.reshape((N+2,N+2))[3,1]
    deltaeta=eta.reshape((N+2,N+2))[3,2]-eta.reshape((N+2,N+2))[2,2]
    x=v[0:(N+2)**2].reshape((N+2,N+2))
    y=v[1*(N+2)**2:2*(N+2)**2].reshape((N+2,N+2))
    xp=vp[0:(N+2)**2].reshape((N+2,N+2))
    yp=vp[1*(N+2)**2:2*(N+2)**2].reshape((N+2,N+2))
    uu=v[2*(N+2)**2:3*(N+2)**2].reshape((N+2,N+2))
    ## find the residuals of x_theta equations
    for i in range(0,(N+2)**2):
        result[i] = +taux*(domegaxxixi[i] + domegaxetaeta[i])-vp[i]
    resbox=result[0:(N+2)**2].reshape((N+2,N+2))
    ## impose boundary conditions
    for i in range(0,N+2):
        for j in range(0,N+2):
            if i==0 or i==N+1 or j==0 or j==N+1:
                resbox[i,j]=-vp[0:(N+2)**2].reshape((N+2,N+2))[i,j]
    result[0:(N+2)**2]=resbox.flatten()
    ## find the residuals of y_theta equations
    for i in range((N+2)**2,2*(N+2)**2):
        ii=i-1*(N+2)**2
        result[i] = +tauy*(domegayxixi[ii] + domegayetaeta[ii])-vp[i]
    resbox=result[(N+2)**2:2*(N+2)**2].reshape((N+2,N+2))
    ## impose boundary conditions
    for i in range(0,N+2):
        for j in range(0,N+2):
            if i==0 or i==N+1 or j==0 or j==N+1:
                resbox[i,j]=-vp[(N+2)**2:2*(N+2)**2].reshape((N+2,N+2))[i,j]
    result[(N+2)**2:2*(N+2)**2]=resbox.flatten()
    ## find the residuals of the u_theta equations
    for i in range(2*(N+2)**2,3*(N+2)**2):
        ii=i-2*(N+2)**2
        result[i]=0
        result[i]+=-vp[i]
    resbox=result[2*(N+2)**2:3*(N+2)**2].reshape((N+2,N+2))
    AA=AA.reshape((N+2,N+2))
    for i in range(0,N+2):
        for j in range(0,N+2):
            if i==0 or i==N+1 or j==0 or j==N+1:
                resbox[i,j]=-vp[2*(N+2)**2:3*(N+2)**2].reshape((N+2,N+2))[i,j]+AA.
↪reshape((N+2,N+2))[i,j]
            else:
                xxi=(x[i,j+1]-x[i,j-1])/(2*deltaxi)
                yeta=(y[i+1,j]-y[i-1,j])/(2*deltaeta)
                xeta=(x[i+1,j]-x[i-1,j])/(2*deltaeta)
                yxi=(y[i,j+1]-y[i,j-1])/(2*deltaxi)
                J=xxi*yeta-xeta*yxi
                resbox[i,j]+=(AA[i,j])
                resbox[i,j]+=-(1/J*((xp[i,j]*yxi-yp[i,j]*xxi)*((uu[i+1,j]-uu[i-1,j])/(2*deltaxi))
                                  -(xp[i,j]*yeta-yp[i,j]*xeta)*((uu[i,j+1]-uu[i,j-1])/
↪(2*deltaeta))))
    result[2*(N+2)**2:3*(N+2)**2]=resbox.flatten()
    print(t)
    return result
```

[ ]:
```
##########################
##### Solve the PDE ######
```

```
##########################

## coefficients of the system
a=-1.0      # lower x and y limit
b=1.0       # upper x limit
c=1.0       # upper y limit
d=0.05      # diffusion coefficient
r=5.0       # Fisher's coefficient
p=10        # parameter for initial condition (determines width of Gaussian curve)
alpha=0.5   # order of fractional derivative
vel=0.0     # advection coefficient
beta=5.0    # monitor function parameter
taux=1.0    # adaptive mesh PDE parameter
tauy=taux
mag=1.0     # parameter for initial condition (determines height of Gaussian curve)

## the (non-uniform) mesh

N=15
ccc=0.1 ### 0.00001 for uniform
x=x_grid(ccc).reshape((N+2, N+2))
y=y_grid(ccc).reshape((N+2, N+2))

## define xi and eta grid

xi=np.zeros((N+2,N+2))
eta=xi
for i in range(0,N+2):
        xi[i,:]=np.linspace(0,1,N+2)
        eta[i,:]=np.linspace(0,1,N+2)
eta=eta.T
xi=xi.flatten()
eta=eta.flatten()

## initial conditions

u=IC(x,y,N)

## define vector v=[x,y,u]

v=np.append(x.flatten(), y.flatten())
v=np.append(v,u.flatten())

## determine initial guess for vector v_theta=[x_theta,y_theta,u_theta]
## The solver improves this guess

xp=x.flatten()
yp=y.flatten()
dxix= DX(xi,x.flatten())
dxiy= DY(xi,y.flatten())
detax=DX(eta,x.flatten())
detay = DY(eta, y.flatten())
duxi=DX(u.flatten(),xi)
dueta = DY(u.flatten(), eta)
dxxi=DX(x.flatten(),xi)
dxeta = DY(x.flatten(), eta)
dyxi=DX(y.flatten(),xi)
dyeta = DY(y.flatten(), eta)
```

```
omega = Omega(duxi,dueta)
domegaxxixi= DX(np.multiply(omega.flatten(),dxxi),xi)
domegaxetaeta = DY(np.multiply(omega.flatten(),dxeta), eta)
domegayxixi= DX(np.multiply(omega.flatten(),dyxi),xi)
domegayetaeta = DY(np.multiply(omega.flatten(),dyeta), eta)
J=(np.multiply(dxxi,dyeta)-np.multiply(dxeta,dyxi))

for i in range(0,(N+2)**2):
    xp[i]=taux*(domegaxxixi[i]+domegaxetaeta[i])
    yp[i]=tauy*(domegayxixi[i]+domegayetaeta[i])
vpx=xp.reshape((N+2,N+2))
vpy=yp.reshape((N+2,N+2))
jacobian_term=(np.multiply(1/J,(np.multiply((np.multiply(xp,dyxi))-(np.multiply(yp,dxxi)),dueta))
                               -(np.multiply((np.multiply(xp,dyeta))-(np.
 ↪multiply(yp,dxeta)),duxi))))
up=twodrhs(u.flatten(),0., x.flatten(), y.flatten())
for i in range(0,(N+2)**2):
    up[i]+=-jacobian_term.flatten()[i]

vp=np.append(xp,yp)
vp=np.append(vp,up)

## plot initial condition ##

fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
surf = ax.plot_surface(x, y, u, cmap=cm.nipy_spectral,
                       linewidth=0, antialiased=False, vmax=mag+0.01, vmin=0)
ax.set_zlabel('u')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_title('t=0')
ax.axes.set_xlim3d(left=-1, right=1)
ax.axes.set_ylim3d(bottom=-1, top=1)
ax.axes.set_zlim3d(bottom=0, top=mag)
fig.colorbar(surf, shrink=0.75, aspect=6, boundaries=np.linspace(0,mag+.01,101), ticks=[0,0.2, 0.
 ↪4, 0.6, 0.8, 1.0])

## plot initial mesh ##

x_list=np.zeros((N+2)**2)
y_list=np.zeros((N+2)**2)
for i in range(0,N+2):
    for j in range(0,N+2):
        x_list[i*(N+2)+j]=x[i,j]
        y_list[i*(N+2)+j]=y[i,j]
plt.figure(figsize=(10,10))
plt.plot(x_list,y_list, '.')
plt.xlabel('x')
plt.ylabel('y')
plt.title('The non-uniform mesh')
plt.show()

### SOLVE ###

start = time()
mod = Implicit_Problem(residual, v, vp, 0)
sim=Radau5DAE(mod)
sim.atol=1e-4       ## tolerances
sim.rtol=1e-4
#sim.inith=0.000001 ## initial stepsize (optional)
sim.thet=0.5        ## larger makes the solver compute new jacobian less often
t, u, yd= sim.simulate(tfinal=1.5, ncp=9) ## ncp is the number of times the solution is recorded

## print the run time
print(f'Time taken to run: {time() - start} seconds')
```

[ ]:
```
############################################
##### Plot the solutions as a 3D plot ######
## The same colour bar is used throughout ##
############################################

plt.figure(figsize=(20,10))
for i in range(0,10):
    fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
    X = u[i,0:1*(N+2)**2].reshape((N+2,N+2))
    Y = u[i,1*(N+2)**2:2*(N+2)**2].reshape((N+2,N+2))
    surf = ax.plot_surface(X, Y, u[i,2*(N+2)**2:3*(N+2)**2].reshape((N+2, N+2)), cmap=cm.
↪nipy_spectral,#turbo,#cmap=cm.coolwarm,
                       linewidth=0, antialiased=False, vmax=1.01, vmin=0)
    ax.set_zlabel('u')
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_title('t='+str(round(t[i],6)))
    ax.axes.set_xlim3d(left=-1, right=1)
    ax.axes.set_ylim3d(bottom=-1, top=1)
    ax.axes.set_zlim3d(bottom=0, top=1)
    fig.colorbar(surf, shrink=0.75, aspect=6, boundaries=np.linspace(0,1.01,101), ticks=[0,0.2,␣
↪0.4, 0.6, 0.8, 1.0])


plt.show()
```

[ ]:
```
####################################
##### Plot the adaptive grids ######
####################################

fig, axs = plt.subplots(2,5,figsize=(50,20))
for i in range(0,10):
    if i<5:
        axs[0,i].plot(u[i,0:1*(N+2)**2].flatten(),u[i,1*(N+2)**2:2*(N+2)**2].flatten(), '.')
        if i==0:
            axs[0,i].set_title('The uniform mesh t='+str(t[i]))
        else:
            axs[0,i].set_title('The non-uniform mesh t='+str(t[i]))
    else:
        axs[1,i-5].plot(u[i,0:1*(N+2)**2].flatten(),u[i,1*(N+2)**2:2*(N+2)**2].flatten(), '.')
        axs[1,i-5].set_title('The non-uniform mesh t='+str(t[i]))
for ax in axs.flat:
    ax.set(xlabel='x', ylabel='y')
```

[ ]:
```
############################################
##### Plot the solutions as heat maps ######
## The same colour bar is used throughout ##
############################################

fig, axs = plt.subplots(2,5,figsize=(50,20))
```

```
for i in range(0,10):
    if i<5:
        X = u[i,0:1*(N+2)**2].reshape((N+2,N+2))
        Y = u[i,1*(N+2)**2:2*(N+2)**2].reshape((N+2,N+2))
        surf = axs[0,i].pcolor(X, Y, u[i,2*(N+2)**2:3*(N+2)**2].reshape((N+2,␣
↪N+2)),shading='auto',cmap=cm.nipy_spectral,linewidth=0, antialiased=False, vmax=1.01,␣
↪vmin=0)#,#turbo,#cmap=cm.coolwarm,
        axs[0,i].title.set_text('t='+str(round(t[i],6)))
    else:
        X = u[i,0:1*(N+2)**2].reshape((N+2,N+2))
        Y = u[i,1*(N+2)**2:2*(N+2)**2].reshape((N+2,N+2))
        surf = axs[1,i-5].pcolor(X, Y, u[i,2*(N+2)**2:3*(N+2)**2].reshape((N+2,␣
↪N+2)),shading='auto', cmap=cm.nipy_spectral,#,#turbo,#cmap=cm.coolwarm,
                       linewidth=0, antialiased=False, vmax=1.01, vmin=0)
        axs[1,i-5].title.set_text('t='+str(round(t[i],6)))
limits=[-1,1,-1,1]
for ax in axs.flat:
    ax.set(xlabel='x', ylabel='y')
    ax.axis(limits)
plt.show()
```