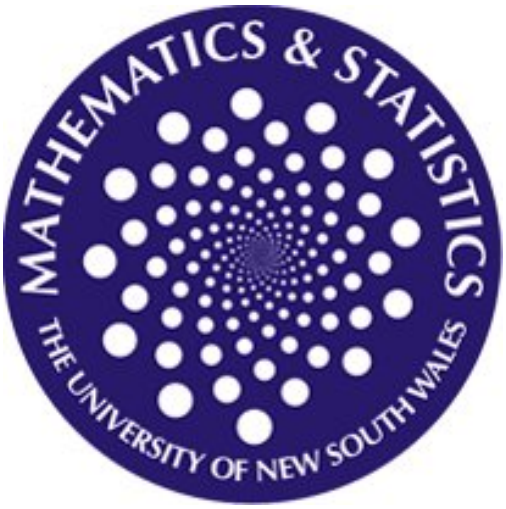
UNSW
AUSTRALIA



University of New South Wales

School of Mathematics and Statistics

Honours Thesis

Fractional Differential Equations

Author:
Adam J. Gray

Student Number:
3329798

Supervisor:
Dr Chris Tisdell

Acknowledgements

I have many people to thank, especially after so long at UNSW. I would like to thank all the people, both staff and students of the School of Mathematics and Statistics for their support, friendship and guidance throughout my degree. These are years that I will always remember and cherish as they have enriched me in ways I could never had imagined.

I would like to specifically thank Dr Chris Tisdell for officially supervising me throughout my Honours year. Dr Chris Tisdell was one of my first lecturers in mathematics and he has been a formative influence on my mathematical view of the world. I would also like to thank Dr Chris Angstmann for his extensive comments, feedback and guidance on work I have been doing throughout the year. Without the talks I have had with him, my perspective on fractional calculus would be significantly narrower than it is today. I would also like to thank Dr Michael Hirschhorn and Dr Bill McLean for their assistance and feedback on a number of technical questions throughout the year.

I also must thank my colleagues for their kindness and understanding throughout the year. A special thanks must go To Ed McDonald for his mathematical assistance on countless problems throughout my degree. It is hard to overestimate how helpful Ed has been on so many occasions.

Finally, I would like to thank my mother and father, because without them I almost certainly would not have had the opportunities I do today, and I would have not gone to University. I will remain eternally indebted to them for their sacrifices and my mother's unwaivering support and inspiration.

Thankyou.

Adam J. Gray, 1 November 2014

Abstract

In this thesis I discuss many of the fundamental topics in the field of fractional differential equations. The point of this thesis is not to be extremely detailed in the handling of any particular part (although sometimes I am) but rather to give a thorough and broad account of the field of fractional differential equations.

I start by looking at the history of fractional calculus, then define the operators which are fundamental to the field. In much the same way as ordinary differential equations have results about uniqueness and existence I go through similar results for ordinary fractional differential equations and extend some results. I also go through the analytic solutions to a small number of fractional differential equations before turning to numerical solutions. I go through a fractional Adam's Moulton Bashforth method and explain a multi-threaded implementation on the CPU and a new implementation on the GPU. I also look at some of the functions which are key to fractional calculus and how one might go about numerically evaluating these functions. I finish on the topic of fractional diffusion which is a rapidly growing field. This particular topic is important as it serves to show that fractional calculus is in some sense *not nonsense* and based in some physical reality.

1 Historical Introduction

Fractional calculus is a field which extends all the way back to the birth of *ordinary* calculus. In this section we wish to give an account of the development of this field. Like any historical recount this section will have its flaws, inaccuracies and omissions. One can get bogged down in the historiography of the field and criticise this recount as a story written with whiggish bias, but we seek to tell a story which is relevant to the rest of this thesis an contextualize modern fractional calculus.

In any historical introduction to fractional calculus one almost always starts with the letters exchanged between L'Hopital and Leibniz in the fall of 1695. Upon hearing of Leibniz's calculus and the differential operator

$$\frac{d}{dx} \tag{1}$$

and it's generalisation

$$\frac{d^n}{dx^n} \tag{2}$$

L'Hopital asked Leibniz what it would mean to set $n = \frac{1}{2}$. Leibniz responded saying that

$$d^{\frac{1}{2}}x = x\sqrt{dx : x} \tag{3}$$

and adding that "It will lead to a paradox, from which one day useful consequences will be drawn" (Or at least that what most modern translations say) [1]. The notation is unfamiliar to modern mathematics, but this comment actually doesn't really reveal anything about how one might go about computing

$$\frac{d^{\frac{1}{2}}f}{dx^{\frac{1}{2}}}. \tag{4}$$

Two years latter in letters to J. Wallis and J. Bernoulli Euler proposed a possible approach to dealing with fractional derivatives of exponential functions [1]. He noticed that

$$\frac{d^m}{dx^m}e^n x = n^m e^{nx} \tag{5}$$

and proposed that one could simply consider non-integer values of m in order to define a fraction differential operator. It is perhaps on this note that we should point out that the name *fractional* calculus is a bit of a misnomer. As in the case above we do not require $m \in \mathbb{Q}$, but rather are allowed to select $m \in \mathbb{R}$ or even $m \in \mathbb{C}$.

Figure 1: Bernhard Christoph Francke's portrait of Gottfried Wilhelm von Leibniz

Given the definition / motivation in (5) one might be tempted to use this result and extend it to other functions via the theory of Fourier Analysis. We need to keep in mind, however, that this was the late 17th century and so Fourier had yet to be born, let alone develop the idea of Fourier decomposition and Fourier basis. This meant that at the time there was not *obvious* way to extend this definition to other functions.

Jumping forward a quarter of a century we arrive at the contributions of Euler. Euler made extremely important contributions in field of special functions, especially the Gamma function which turns out to be crucial in defining fractional integrals and fractional derivatives.

Figure 2: 1753 portrait of Leonhard Euler by Emanuel Handmann

The Gamma function is essentially a solution to an interpolation problem, that is how to extend the factorial function to non-integer values of the argument.

Although the problem of extending the factorial function had been considered by Daniel Bernoulli and Christian Goldbach in the 1720s, it was eventually Euler who in a two letters, dated 13th October 1729 and 8th January 1730 respectively, gave two different representations of the factorial which could easily be extended to non-integer values. These were

$$n! = \prod_{k=1}^{\infty} \frac{\left(1+\frac{1}{k}\right)^n}{1+\frac{n}{k}} \tag{6}$$

and

$$n! = \int_0^1 (-\ln(s))^n ds. \tag{7}$$

Upon finding the first representation Euler could have simply stopped. He had found a formula which extended the factorial function to non-integer values. However he noticed that a special case of his product had already been calculated by Wallis. If we set $n = \frac{1}{2}$ we get

$$\prod_{k=1}^{\infty} \frac{(1+\frac{1}{k})^{\frac{1}{2}}}{1+\frac{1}{2k}} \tag{8}$$

and squaring and multiplying by 2 we get

$$2\left[\prod_{k=1}^{\infty} \frac{(1+\frac{1}{k})^{\frac{1}{2}}}{1+\frac{1}{2k}}\right]^2 = 2\prod_{k=1}^{\infty} \frac{1+\frac{1}{k}}{(1+\frac{1}{2k})^2} \tag{9}$$

$$= 2\prod_{k=1}^{\infty} \frac{k^2+1}{(k+\frac{1}{2})^2} \tag{10}$$

$$= 2\frac{2}{(1+\frac{1}{2})^2} \cdot \frac{5}{(2+\frac{1}{2})^2} \cdot \frac{10}{(3+\frac{1}{2})^2} \cdots \tag{11}$$

$$= \frac{2}{1} \cdot \frac{2}{3} \cdot \frac{4}{3} \cdot \frac{4}{5} \cdots \tag{12}$$

$$= \prod_{k=1}^{\infty} \frac{4n^2}{4n^2-1} \tag{13}$$

and this last product had previously been calculated by Wallis and was known to be equal to $\frac{\pi}{2}$. This meant that Euler had the result

$$\frac{1}{2}! = \frac{\sqrt{\pi}}{2} \tag{14}$$

We should point out that Wallis had calculated this result by considering

$$\int_0^{\pi} \sin^n(x) dx \tag{15}$$

but this result is actually more easily obtained as a special case of Euler's infinite product for the sine function,

$$\sin(x) = x \prod_{k=1}^{\infty} \left(1 - \frac{x^2}{k^2\pi^2}\right) \tag{16}$$

but Euler had not yet developed this.

The fact that Euler had $\frac{1}{2}! = \frac{\sqrt{\pi}}{2}$ piqued his curiosity and on nothing more than what appears to have been a hunch he went looking for an integral representation of $n!$. He arrived at (7) and it is here that our story returns to the history of fractional calculus. Aware of the fact that a meaning for $\frac{d^{\frac{1}{2}}}{dx^{\frac{1}{2}}}$ was sought he noted that

$$\frac{d^m}{dx^m}x^n = \frac{n!}{(n-m)!}x^{n-m} \tag{17}$$

and then suggested that using either (6) or (7) could give the necessary extension of meaning in order to define the derivative of the power function for non-integer orders of differentiation. In fact using Wallis' result along with a slight extension he was able to suggest that

$$\frac{d^{\frac{1}{2}}}{dx^{\frac{1}{2}}}x = \sqrt{\frac{4x}{\pi}}. \tag{18}$$

Although technically the modern notation we use for the Gamma function and the fact that $\Gamma(n) = (n-1)!$ is due to Legendre, and were developed some considerable time after these ideas were first worked on by Euler, we will adopt the modern gamma function notation so as to make our discussions from this point more readable to those with some background knowledge.

We would like to point out just how *good* this idea for the definition of the fractional derivative by Euler was. Firstly note that if we take $n = 1$ in (17) we get that

$$\frac{d^{-1}}{dx^{-1}}x^m = \frac{\Gamma(m+1)}{\Gamma(m+2)}x^{m+1} \tag{19}$$

$$= \frac{1}{m+1}x^{m+1} \tag{20}$$

$$= \int_0^x t^m dt \tag{21}$$

which is consistent with the fundamental theorem of calculus. We discuss a modern version of this result in section 2 of this thesis.

Also in some formal sense this definition of the fractional derivative is consistent with that of Leibniz in that if we take a Taylor expansion of the exponential function we get that

$$e^{mx} = \sum_{k=0}^{\infty} \frac{m^k}{k!}x^k \tag{22}$$

and so without regard for convergence or any other technicalities one might be tempted to write

$$\frac{d^r}{dx^r}e^{mx} = \sum_{k=r}^{\infty} \frac{d^r}{dx^r}\frac{m^k}{k!}x^k \tag{23}$$

$$= \sum_{k=r}^{\infty} \frac{m^k}{\Gamma(k+1)}\frac{\Gamma(k+1)}{\Gamma(k-r+1)}x^{k-r} \tag{24}$$

$$= \sum_{k=r}^{\infty} \frac{m^k}{\Gamma(k-r+1)}x^{k-r} \tag{25}$$

$$= m^r \sum_{k=r}^{\infty} \frac{m^{k-r}}{\Gamma(k-r+1)}x^{k-r}. \tag{26}$$

Letting $j = k - r$ we have

$$\begin{aligned} m^r \sum_{k=r}^{\infty} \frac{m^{k-r}}{\Gamma(k-r+1)}x^{k-r} &= m^r \sum_{j=0}^{\infty} \frac{m^j}{\Gamma(j+1)}x^j \\ &= m^r \sum_{j=0}^{\infty} \frac{m^j}{j!}x^j \\ &= m^r e^{mx}. \end{aligned}$$

and so in some formal sense these two definitions would at first glance appear to be consistent with each other.

This analysis has several problems. Firstly although the idea of expressing functions in terms of an infinite series dates back to perhaps the 14th century the formal notion of a Taylor series dates back to 1715 and Brook Taylor, only a few years before Euler's work in this field and it is not clear that such techniques would have been available to Euler.

Secondly, and more importantly from a mathematical point of view, is the slight of hand played in (23) where we essentially assume that if $m > n$, $\frac{d^m}{dx^m}x^n = 0$. This is not the case if $m \notin \mathbb{Z}$. For example, using Euler's definition above we would have that

$$\frac{d^{\frac{3}{2}}}{dx^{\frac{3}{2}}}x = \frac{1}{\sqrt{\pi}}x^{-\frac{1}{2}} \neq 0. \tag{27}$$

One can see that in the integer case we get 0 simply because of singularities in the Gamma function for the non-positive integers. This particular complication of non-zero derivatives for the case where $m > n$ extends even into more modern fractional derivatives and is dealt with in a modern context in section 2 of this thesis.

It was almost a century latter when in 1822 Fourier suggest that using the equality

$$\frac{d^m}{dx^m}f(x) = \frac{1}{2\pi}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(t)u^m \cos\left(ux - tu - \frac{m\pi}{2}\right) dtdu \tag{28}$$

could give meaning to fractional derivative of a function. This appear to have been the first definition of a fractional derivative for a general class of *sufficiently good* functions, in this case, for which (28) is defined.

The history of more modern fractional differential operators really begins with the Abel integral equation which was solved by Abel in papers dating back to 1823 and 1826. It was originally posed and solved in the context of the Tautochrone problem, which is finding the curve for which objects sliding under gravity without the effects of friction take equal times to reach their lowest point, independent of their starting point.

Figure 3: Johan Grbitz's picture of Niels Henrik Abel.

Abel's integral equation (of the first kind) is

$$\frac{1}{\Gamma(\alpha)}\int_0^x \frac{\phi(t)dt}{(x-t)^{1-\alpha}} = f(x) \tag{29}$$

for $x > 0$ and $0 < \alpha < 1$. Finding a *solution* to this equation essentially means making ϕ the subject. We demonstrate the solution method of this integral equation as it turns out to be important in latter formulations of fractional derivatives and integrals. Firstly let's consider the integral

$$I(x) := \int_a^x \frac{f(s)ds}{(x-s)^{1-\alpha}}. \tag{30}$$

Now by substituting (29) into (30) we get

$$I(x) = \frac{1}{\Gamma(\alpha)}\int_a^x \frac{1}{(x-s)^{1-\alpha}}\left(\int_a^s \frac{\phi(t)dt}{(s-t)^{\alpha}}\right)ds \tag{31}$$

$$= \frac{1}{\Gamma(\alpha)}\int_a^x \left(\int_a^s \frac{\phi(t)dt}{(x-s)^{1-\alpha}(s-t)^{\alpha}}\right)ds \tag{32}$$

Now noting that the region of integration in $\mathbb{R}^2$ is just

$$a \leq s \leq x \tag{33}$$

$$a \leq t \leq s \tag{34}$$

which is equivalent to

$$t \leq s \leq x \tag{35}$$

$$a \leq t \leq x \tag{36}$$

we can write

$$\begin{aligned}\frac{1}{\Gamma(\alpha)}\int_a^x \left(\int_a^s \frac{\phi(t)dt}{(x-s)^{1-\alpha}(s-t)^{\alpha}}\right)ds &= \frac{1}{\Gamma(\alpha)}\int_a^x \left(\int_t^x \frac{\phi(t)ds}{(x-s)^{1-\alpha}(s-t)^{\alpha}}\right)dt \\ &= \frac{1}{\Gamma(\alpha)}\int_a^x \phi(t)\left(\int_t^x (x-s)^{\alpha-1}(s-t)^{-\alpha}ds\right)dt. \end{aligned} \tag{37}$$

Now performing the substitution $\tau = \frac{s-t}{x-t}$ yields

$$\int_t^x (x-s)^{\alpha-1}(s-t)^{-\alpha}ds = \int_0^1 \tau^{-\alpha}(1-\tau)^{\alpha-1}d\tau \tag{38}$$

$$= B(1-\alpha,\alpha) \tag{39}$$

$$= \Gamma(1-\alpha)\Gamma(\alpha) \tag{40}$$

and so (37) becomes

$$\frac{1}{\Gamma(\alpha)}\int_a^x \phi(t)\left(\int_t^x (x-s)^{\alpha-1}(s-t)^{-\alpha}ds\right)dt = \frac{1}{\Gamma(\alpha)}\int_a^x \phi(t)\Gamma(\alpha)\Gamma(1-\alpha)dt \tag{41}$$

$$= \Gamma(1-\alpha)\int_a^x \phi(t)dt. \tag{42}$$

So we have that

$$\int_a^x \frac{f(s)ds}{(x-s)^{1-\alpha}} = \Gamma(1-\alpha)\int_a^x \phi(t)dt \tag{43}$$

and by differentiating we get

$$\phi(x) = \frac{1}{\Gamma(1-\alpha)}\frac{d}{dx}\int_a^x \frac{f(s)ds}{(x-s)^{1-\alpha}} \tag{44}$$

and so we have in some sense solved Abel's integral equation. Notice that we have not discussed which functions f are suitable. This is a bit of a tricky question and not really informative in the context of the rest of this thesis so we refer the interested reader to [16].

As mentioned earlier Abel solved this integral equation in the context of the Tautochrone problem which corresponded to $\alpha = \frac{1}{2}$. Abel seemingly had no actual desire to define a fractional integral or derivative but it turns out this problem can be recast in terms of fractional integral equation using a Riemann Liouville fractional integral and the result is in some sense a statement of a generalised fundamental theorem of calculus. We'll leave these ideas and turn our attention to the development of the Riemann-Liouville fractional integral and derivative and return to see how these ideas relate.

It was in a collection of letters over six years from 1832 to 1837 that Liouville outlined ideas which hold any reasonable resemblance to modern fraction calculus.

Figure 4: An updated and unattributed early photograph of Joseph Liouville.

The first idea explored for a fractional derivative by Liouville bears someone of a resemblance to that of Fourier and Euler.

If a function f can be expanded as

$$f(x) = \sum_{k=0}^{\infty} c_k e^{a_k x} \tag{45}$$

then the fractional derivative of f of order α could be written as

$$\frac{d^\alpha}{dx^\alpha} f(x) = \sum_{k=0}^{\infty} c_k a_k^\alpha e^{a_k x} \tag{46}$$

Note that by allowing complex values for a_k this is essentially a Fourier series and so the resemblance to Fourier's definition is apparent.

Despite the fact that we could view this series as a Fourier series, its not clear that this was done. Then the convergence of the series is tricky and it makes any consideration of the fractional derivative of f restrictive.

In this series of letters he went on to derive a fractional integral

$$I^{\alpha}f(x) = \frac{1}{(-1)^{\alpha}\Gamma(\alpha)}\int_0^{\infty} f(x+t)t^{\alpha-1}dt \tag{47}$$

Papers in 1832 and 1837 by Liouville were really the first papers which discussed an application of a fractional calculus, specifically to the solution of linear ordinary differential equations.

He also discussed the idea of a fractional difference quotient. Liouville didn't go far with this idea and it was considerably latter in 1867 and 1868 that Grünwald and respectively Letnikov dealt with this idea in much more depth.

Figure 5: An 1863 image of Bernhard Riemann.

As a student in 1847 Riemann arrived at the fractional integral

$$I^{\alpha}f(x) = \frac{1}{\Gamma(\alpha)}\int_0^{x} \frac{f(t)dt}{(x-t)^{1-\alpha}} \tag{48}$$

Interestingly it wasn't until 10 years after his death in 1866 that this definition was published. The astute reader will notice that for positive integral values of α this definition aggrees with the Cauchy formula for repeated integration. We'll touch on this a little more in section 2, but this fact was important in the motivation of the defintion.

Given the similarity of Riemann's and Liouville's fractional integrals these two ideas have been combined and named the Riemann Liouville fractional integral. This particular definition has become extremely significant and it forms the basis for modern fractional calculus.

With a bit of technical fiddling it is quite easy to come up with a Riemann-Liouville fractional derivative. The idea behind this is discussed in section 2.

Although over the next hundred years or so there was significant development of other forms of fractional operators such the Gr"nwald Letnikov, Weyl, Miller Ross fractional derivatives. As we won't really explore these operators wi e will skip forward to Caputo in 1967.

One of the problems with the Riemann-Liouville fractional derivative is that for many fractional differential equations the associated initial or boundary conditions must be of fractional order. Also the Riemann-Liouville fractional derivative of a constant is not zero.

In two papers in 1967 and 1971 Michele Caputo [2] [3], introduced a new fractional differential operator. These papers dealt with linear models of dissipation and interestingly this new operator resolved the issues to do with initial and boundary conditions along with the property that the Caputo fractional derivative of a constant is zero. It is perhaps interesting to note that this operator arose from purely physical considerations.

2 Fractional Operators and Their Properties

There are a variety of ways to define fractional differential operators. Good introductions to the various differential operators can be found in [7, 9, 14, 16]. In this section, and for the rest of the thesis we will focus of three main fractional operators, the Riemann-Liouville integral and derivative and the Caputo derivative. There are an abundance of other operators such that the Miller-Ross sequential fractional derivative [12, 14], the Weyl fractional integral [9, 15, 16], the Caputo fractional integral [9], the Grünwald-Letnikov fractional derivative [14, 16] and fractional operators based on ideas from Fourier analysis [13, 16].

2.1 Riemann Liouville Fractional Integral

In the introduction we considered the historical motivations of the Riemann Liouville fractional integral. Here we will reintroduce it formally and discuss it's properties.

Definition 1 (Riemann Liouville Fractional Integeral)**.** *The Riemann Liouville fractional integral, based at $a \in \mathbb{R}$ and of order $\alpha \in (0, \infty)$ of a function f is given by*

$$\left(I_0^\alpha f\right)(x) := \frac{1}{\Gamma(\alpha)} \int_a^x f(t)(x-t)^{\alpha-1} dt \tag{49}$$

From this point forward we refer to this as the *fractional integral* of the function.

It is possible to extend to the this definition to $\alpha \in \mathbb{C}$ and $a = \infty$. While we will touch on the $a = -\infty$ case we will neglect discussion of $\alpha \notin \mathbb{R}$ and refer the interested reader to Samko et al. [16]. It is also worthy of note that some authors define a fractional operator with the limits of integration reversed. We will not deal with these operators, but instead refer the interested reader to [14, 16]. The definition given in 1 is the most common in the literature and is compatible with the rest of the theorems and definitions given here.

Although it might seem like a good idea to describe the class of function for which the integral in (49) is well defined, this is a very complicated problem and we refer the interested reader to [16]. Although we don't deal with defining the set of admissible functions, this will not have any significant impact on the rest of our discussions because all the functions we deal with will be admissible.

We firstly wish to show that this coincides with the ordinary integrals when $n \in \mathbb{N}$. That is we wish to show that

$$\int_a^x \int_a^{\xi_1} \cdots \int_a^{\xi_{n-1}} f(\xi_n) d\xi_n \cdots d\xi_1 = \frac{1}{(n-1)!} \int_a^x (x-t)^{n-1} f(t) dt \tag{50}$$

but this is the Cauchy theorem for repeated integration which is a well known result. It follows from applying the fundamental theorem of calculus repeatedly.

What this means is that the the Riemann-Liouville fractional integral makes sense in that it agrees with the ordinary integral. This is obviously a very desirable property.

The second property that we wish to explore is the semigroup property.

Lemma 1. *Two fraction integrals I_0^α and I_0^β have the property that*

$$I_a^\alpha I_a^\beta = I_a^{\alpha+\beta} = I_a^\beta I_a^\alpha. \tag{51}$$

Proof. For a function f we have that

$$(I_a^\alpha I_a^\beta)(f)(t) = \frac{1}{\Gamma(\alpha)} \int_a^t (t-\tau)^{\alpha-1} \frac{1}{\Gamma(\beta)} \int_a^\tau (\tau - z)^{\beta-1} f(z) dz d\tau \tag{52}$$

$$= \frac{1}{\Gamma(\alpha)\Gamma(\beta)} \int_a^t \int_a^\tau (t-\tau)^{\alpha-1} (\tau - z)^{\beta-1} f(z) dz d\tau \tag{53}$$

$$\circledast = \frac{1}{\Gamma(\alpha)\Gamma(\beta)} \int_a^t \int_z^t (t-\tau)^{\alpha-1} (\tau - z)^{\beta-1} f(z) d\tau dz. \tag{54}$$

Now performing the change of variables $x = \frac{\tau - z}{t - z}$ we get that

$$\circledast = \frac{1}{\Gamma(\alpha)\Gamma(\beta)} \int_a^t \int_0^1 (t-z)^{\alpha-1} (1-x)^{\alpha-1} x^{\beta-1} (t-z)^{\beta-1} (t-z) dx dz \tag{55}$$

$$= \frac{1}{\Gamma(\alpha)\Gamma(\beta)} \int_0^1 (1-x)^{\alpha-1} x^{\beta-1} dx \int_a^t (t-z)^{\alpha+\beta-1} f(z) dz \tag{56}$$

$$= \frac{B(\alpha,\beta)}{\Gamma(\alpha)\Gamma(\beta)} \int_a^t (t-z)^{\alpha+\beta-1} f(z) dz \tag{57}$$

$$= \frac{1}{\Gamma(\alpha+\beta)} \int_a^t (t-z)^{\alpha+\beta-1} f(z) dz \tag{58}$$

$$= I_a^{\alpha+\beta}(f)(t). \tag{59}$$

This shows that $I_a^\alpha I_a^\beta = I_a^{\alpha+\beta}$ and hence immediately implies that $I_a^\alpha I_a^\beta = I_a^\beta I_a^\alpha$. □

We now calculate the Laplace transform of the of the Riemann-Liouville fractional integral. While we will not immediately use this result it will prove useful when we attempt to solve fractional differential equations.

Lemma 2. *The laplace transform of $(I_0^\alpha f)$ is given by*

$$\mathcal{L}\{(I_0^\alpha f)\} = \frac{\mathcal{L}\{f\}}{s^\alpha} \tag{60}$$

Proof. Noticing that

$$\int_0^t (t-\tau)^{\alpha-1} f(\tau) d\tau \tag{61}$$

is the Laplace convolution of $t^{\alpha-1}$ and $f(t)$ we can simply use the Laplace convolution theorem to get that

$$\mathcal{L}\{(I_0^\alpha f)\} = \mathcal{L}\{f\}\mathcal{L}\{t^{\alpha-1}\} \tag{62}$$

$$= \frac{\mathcal{L}\{f\}}{s^\alpha}. \tag{63}$$

□

Note that this proof hinges off the Laplace convolution theorem, and so it requires that the base of the fractional derivative, a, has to be 0. In a similar fashion we can deal with the Fourier transform of the fractional integral but with a base, a, of $-\infty$ instead so the Fourier convolution can be invoked. We formalise that with the next lemma.

Lemma 3. *The Fourier transform of $(I_0^\alpha f)$ is given by*

$$\mathcal{F}\{(I_0^\alpha f)\} = \frac{\mathcal{F}\{f\}}{(-i\omega)^\alpha}. \tag{64}$$

Proof. We firstly introduce

$$h_+^\alpha(t) = \begin{cases} \frac{t^{\alpha-1}}{\Gamma(\alpha)} & t > 0 \\ 0 & t \leq 0 \end{cases} \tag{65}$$

and then note that it is clear that

$$\begin{aligned} \left(I_0^\alpha f\right)(t) &= \frac{1}{\Gamma(\alpha)} \int_{-\infty}^{t} (t-\tau)^{\alpha-1} f(\tau) d\tau & (66) \\ &= \int_{-\infty}^{\infty} h_+^\alpha(t-\tau) f(\tau) d\tau & (67) \\ &= (h_+^\alpha * f)(t) & (68) \end{aligned}$$

where $*$ represents the Fourier convolution. By employing the Fourier convolution theorem it follows that

$$\mathcal{F}\{(I_0^\alpha f)\} = \mathcal{F}\{h_+^\alpha\}\mathcal{F}\{f\}. \tag{69}$$

It remains only to show that $\mathcal{F}\{h_+^\alpha\} = (-i\omega)^{-\alpha}$. To see this note that the Laplace transform of h_+^α is given by

$$\mathcal{L}\{h_+^\alpha\} = s^{-\alpha} \tag{70}$$

and so by replacing s with $-i\omega$, as in the Fourier transform we get the result. The convergence of the Fourier integral is guaranteed by Dirichlet's theorem for Fourier integrals as noted in [14]. □

2.2 Riemann-Liouville Fractional Derivative

Although one can approach a fractional derivative from somewhat of a *first principles* approach via the Grünwald-Letnikov derivative [14,16], we will not do that here, mainly because for a large class of functions the Grünwald-Letnikov derivative is actually equivalent to the Riemann-Liouville fractional derivative [14] and the Riemann-Liouville fractional derivative is more tractable from a symbolic manipulation perspective.

The idea behind the Riemann-Liouville fractional derivative is to exploit the Riemann-Liouville fractional integral to give the fractional part and then just use integer order derivatives to get the correct order. For example if one wanted to calculate the 1/2-th derivative of a function you would fractionally integrate by 1/2-th and then differentiate once.

Definition 2 (Riemann-Liouville Fractional Derivative)**.** *We define the Riemann-Liouville fractional derivative based at a and of order $\alpha \in (0, \infty)$ of a function f as*

$$\left({}_a\mathcal{D}^{\alpha} f\right)(x) := \frac{1}{\Gamma(n-\alpha)} \frac{d^n}{dx^n} \int_a^x \frac{f(t)}{(x-t)^{\alpha-n+1}} dt \tag{71}$$

where $n = \lfloor \alpha \rfloor + 1$.

We will not continually specify this condition on n and in future cases the definition of n when dealing with fractional derivatives (both Riemann-Liouville and Caputo) the context should make the condition on n clear.

Again, it should be observed that the precise definition of which functions this transform operates on is not given. As in the fractional integral case this is a complicated question, and we will simply refer to those functions for which (71) is well defined as admissible functions. The interested reader is referred to [16] for a discussion of these issues.

We can also see rather trivially that

$$\left({}_a\mathcal{D}^{\alpha} f\right)(t) = \frac{d^n}{dt^n} \left[I_a^{n-\alpha} f(t) \right] \tag{72}$$

which aligns with our intuitive motivation of a fractional derivative above.

Like in the case of the fractional integral we would like to investigate the semi-group properties of the Riemann-Liouville fractional derivative and the relationship between the Riemann-Liouville fractional integral and derivative.

Lemma 4. *The Riemann-Liouville derivative of order α is the left inverse of the Riemann-Liouville integral of order α in the sense that*

$$\left(I_a^{\alpha}\, {}_a\mathcal{D}^{\alpha} f\right)(t) = f(t) - \sum_{j=1}^{k} \left[\left({}_a\mathcal{D}^{\alpha-j} f\right)(t) \right]_{t=a} \frac{(t-a)^{\alpha-j}}{\Gamma(\alpha-j+1)} \tag{73}$$

where Id *is the identity operator (* $\mathrm{Id}(f) = f$*).*

Proof. For an admissible function f we have that

$$\begin{aligned}
{}_a\mathcal{D}^{\alpha} I_a^{\alpha} f &= \frac{1}{\Gamma(n-\alpha)\Gamma(\alpha)} \frac{d^n}{dt^n} \int_a^t (t-\tau)^{n-\alpha-1} \int_a^{\tau} (\tau - z)^{\alpha-1} f(z) dz d\tau && (74)\\
&= \frac{1}{\Gamma(n-\alpha)\Gamma(\alpha)} \frac{d^n}{dt^n} \int_a^t \int_a^{\tau} (t-\tau)^{n-\alpha-1} (\tau - z)^{\alpha-1} f(z) dz d\tau && (75)\\
\circledast &= \frac{1}{\Gamma(n-\alpha)\Gamma(\alpha)} \frac{d^n}{dt^n} \int_a^t \int_z^t (t-\tau)^{n-\alpha-1} (\tau - z)^{\alpha-1} f(z) d\tau dz. && (76)
\end{aligned}$$

Now with the change of variables $x = \frac{\tau - z}{t - z}$ we have that

$$\circledast = \frac{d^n}{dt^n} \frac{1}{\Gamma(n-\alpha)\Gamma(\alpha)} \int_a^t \int_0^1 (t-z)^{n-\alpha-1}(1-x)^{n-\alpha-1} x^{\alpha-1}(t-z)^{\alpha-1}(t-z) f(z) dx dz \tag{77}$$

$$= \frac{1}{\Gamma(n-\alpha)\Gamma(\alpha)} \int_0^1 (1-x)^{n-\alpha-1} x^{\alpha-1} dx \frac{d^n}{dt^n} \int_a^t (t-z)^{n-1} f(z) dz \tag{78}$$

$$= \frac{B(n-\alpha, \alpha)}{\Gamma(n-\alpha)\Gamma(\alpha)} \frac{d^n}{dt^n} \int_a^t (t-z)^{n-1} f(z) dz \tag{79}$$

$$= \frac{d^n}{dt^n} \frac{1}{(n-1)!} \int_a^t (t-z)^{n-1} f(z) dz \tag{80}$$

$$= f(t) \tag{81}$$

and so the result follows. □

We should note that by selecting $\alpha \in \mathbb{N}$ we could see almost by definition that that the Riemann-Liouville fracitonal derivative coincides with the ordinary derivative, however, we could also use this lemma to see it as well.

The Riemann-Liouville fractional derivative is *not* a right inverse of the fractional integral. The relationship is considerably more subtle and is formalized in the following lemma.

Lemma 5. *For an admissable function f we have that*

$$\left(I_a^\alpha \, {}_a\mathcal{D}^\alpha f\right)(t) = f(t) - \sum_{j=1}^{n} \left[{}_a\mathcal{D}^{\alpha-j} f(t)\right]_{t=a} \frac{(t-a)^{\alpha-j}}{\Gamma(\alpha-j+1)} \tag{82}$$

The following proof follows closely to that of Podlubny [14].

Proof. See that

$$\left(I_a^\alpha \, {}_a\mathcal{D}^\alpha f\right)(t) = \frac{1}{\Gamma(\alpha)} \int_a^t (t-\tau)^{\alpha-1} \, {}_a\mathcal{D}^\alpha f(\tau) d\tau \tag{83}$$

$$= \frac{d}{dt} \left[\underbrace{\frac{1}{\Gamma(\alpha+1)} \int_a^t (t-\tau)^\alpha \, {}_a\mathcal{D}^\alpha f(\tau) d\tau}_{\circledast} \right]. \tag{84}$$

So we turn our attention to computing $\circledast$, and see that

$$\circledast = \frac{1}{\Gamma(\alpha+1)} \int_a^t (t-\tau)^p \frac{d}{dt^n} \left(I_a^{n-\alpha} f(\tau)\right) d\tau \tag{85}$$

and by repeated integration by parts we get

$$\circledast = \frac{1}{\Gamma(\alpha-n+1)}\int_a^t (t-\tau)^{\alpha-n}\left(I_a^{n-\alpha}f(\tau)\right)d\tau \tag{86}$$

$$-\sum_{j=1}^{n}\left[\frac{d^{n-j}}{dt^{n-j}}\left(I_a^{n-\alpha}f(t)\right)\right]_{t=a}\frac{(t-a)^{\alpha-j-1}}{\Gamma(2+\alpha-j)} \tag{87}$$

$$= I_a^{n-\alpha-1}\left(I_a^{\alpha-n}f(t)\right) - \sum_{j=1}^{n}\left[{}_a\mathcal{D}^{\alpha-j}f(t)\right]_{t=a}\frac{(t-a)^{\alpha-j+1}}{\Gamma(2+\alpha-j)} \tag{88}$$

$$= I_a^1 f(t) - \sum_{j=1}^{n}\left[{}_a\mathcal{D}^{\alpha-j}f(t)\right]_{t=a}\frac{(t-a)^{\alpha-j+1}}{\Gamma(2+\alpha-j)} \tag{89}$$

and then by applying the derivative from (84) we get the result. □

The astute reader might have noticed that we have abused notation in (88) and (89) where we have permitted a negative order of differentiation in the last term of the sums. This is ok though, because $\alpha - n < 1$ and we formalise this in the next lemma.

Lemma 6. *If we permit $-1 < \alpha < 0$ for an order of differentiation then formally we have that*

$$ {}_a\mathcal{D}^{\alpha} = I_a^{-\alpha}. \tag{90}$$

The proof is trivial and follows immediately from the definitions.

What lemmas 4 and 5 essentially amount to is an *extended fundamental theorem of calculus*. While these results generalise tha FTC they not more *fundamental* than the ordinary fundamental theorem of calculus in that they cannot be used to prove the FTC. This is because implicitly in the definition of the Riemann-Liouville fractional integral and integral and in the proofs of 4 and 5 we have used the FTC.

It turns out that the semi-group property that we formulated for the fractional integral does not hold for the Riemann-Liouville fractional derivative. That is, it is not neseccerily true that ${}_0\mathcal{D}^{\alpha}\,{}_0\mathcal{D}^{\beta} = {}_0\mathcal{D}^{\alpha+\beta} = {}_0\mathcal{D}^{\beta}\,{}_0\mathcal{D}^{\alpha}$. Gorenflo and Mainardi give excelent examples of where these equalitities break down [9].

Due to the utilitiy of the Laplace transform of the Riemann-Liouville fractional derivative we calculate it here as it will prove useful for latter results.

Lemma 7. *The Laplace transform the of the Riemann-Liouville derivative of an admissable function f is given by*

$$\mathcal{L}\left\{({}_0\mathcal{D}^{\alpha}f)\right\} = s^{\alpha}\mathcal{L}\left\{f(x)\right\} - \sum_{k=0}^{n-1} s^k\left({}_0\mathcal{D}^{\alpha-k-1}f\right)(0). \tag{91}$$

Proof. See that

$$\mathcal{L}\left\{({}_0\mathcal{D}^{\alpha}f)\right\} = \mathcal{L}\left\{\frac{d^n}{dt^n}\left(I_0^{n-\alpha}f\right)\right\} \tag{92}$$

$$= s^n\mathcal{L}\left\{\left(I_0^{n-\alpha}f\right)\right\} - \sum_{k=0}^{n-1} s^k\frac{d^{n-k-1}}{dt^{n-k-1}}\left(I_0^{n-\alpha}f\right)(0) \tag{93}$$

and by applying the result of lemma 2 we get

$$\mathcal{L}\left\{\left({}_0\mathcal{D}^{\alpha}f\right)\right\} = s^{\alpha}\mathcal{L}\left\{f\right\} - \sum_{k=0}^{n-1} s^{k}\left({}_0\mathcal{D}^{\alpha-k-1}f\right)(0). \tag{94}$$

□

We also consider the Fourier transform of the Riemann-Liouville fractional derivative as it will also prove useful for latter results.

Lemma 8. *For an admissable function f the Fourier transform of the Riemann-Liouville fractional derivative with $a = -\infty$ and of order α is given by*

$$\mathcal{F}\left\{\left({}_{-\infty}\mathcal{D}^{\alpha}f\right)(t)\right\} = (-i\omega)^{\alpha}\mathcal{F}\left\{f(t)\right\}. \tag{95}$$

Proof. Since it is possible to regard the Riemann-Liouville fractional derivative in terms of the composition of integer order derivatives with the fractional integral this proof follows quickly from the result of lemma 3. Firstly we have that

$$\left({}_{-\infty}\mathcal{D}^{\alpha}f\right)(t) = \frac{d^n}{dt^n}\left[I_{-\infty}^{n-\alpha}f(t)\right] \tag{96}$$

and so by elementry results for Fourier transforms we have that

$$\mathcal{F}\left\{\left({}_{-\infty}\mathcal{D}^{\alpha}f\right)(t)\right\} = \mathcal{F}\left\{\frac{d^n}{dt^n}\left[I_{-\infty}^{n-\alpha}f(t)\right]\right\} \tag{97}$$

$$= (-i\omega)^n\mathcal{F}\left\{I_{-\infty}^{n-\alpha}f(t)\right\} \tag{98}$$

and then by using the result of lemma 3 we have that

$$(-i\omega)^n\mathcal{F}\left\{I_{-\infty}^{n-\alpha}f(t)\right\} = (-i\omega)^n(-i\omega)^{\alpha-n}\mathcal{F}\left\{f(t)\right\} \tag{99}$$

$$= (-i\omega)^{\alpha}\mathcal{F}\left\{f(t)\right\} \tag{100}$$

□

Another very useful result is the fractional derivative of a power function.

Lemma 9. *The Riemann-Liouville fractional derivative of a power function is as follows*

$${}_0\mathcal{D}^{\alpha}z^{\nu} = \frac{\Gamma(\nu+1)}{\Gamma(\nu+1-\alpha)}z^{\nu-\alpha}. \tag{101}$$

Proof. Using lemma 7 we have that

$$\mathcal{L}\left\{{}_0\mathcal{D}^{\alpha}z^{\nu}\right\} = s^{\alpha}\mathcal{L}\{z^{\nu}\} \tag{102}$$

but it is well known that

$$\mathcal{L}\{z^{\nu}\} = \frac{\Gamma(\nu+1)}{s^{\nu+1}} \tag{103}$$

and so we have that

$$\mathcal{L}\left\{{}_0\mathcal{D}^\alpha z^\nu\right\} = \frac{\Gamma(\nu+1)}{s^{\nu+1-\alpha}} \tag{104}$$

$$= \frac{\Gamma(\nu+1)}{\Gamma(\nu+1-\alpha)}\frac{\Gamma(\nu+1-\nu)}{s^{\nu+1-\alpha}} \tag{105}$$

and so by inverting the Laplace transform we get

$${}_0\mathcal{D}^\alpha z^\nu = \frac{\Gamma(\nu+1)}{\Gamma(\nu+1-\alpha)} z^{\nu-\alpha}. \tag{106}$$

□

2.3 Caputo Fractional Derivative

We now turn our attention to the last fractional differential operator that we will study in detail, the Caputo fractional derivative. The Caputo derivative first turned up in a 1967 paper by Caputo on linear models of dissipation [2] and again in 1971 in a paper by Caputo and Mainardi [3]. When specifying differential equations (both of the ordinary and partial type) Riemann-Liouville derivatives typically lead to intial conditions and boundary values of fractional order [3,11,14,16], however, physical intuition for what sensible values for these initial and boundary values can be is somewhat hard to come by [3, 9, 14, 16]. The Caputo derivative, however, leads to systems where the initial and boundary values are of integer order, which leads to much greater physical usefulness [9, 14, 16]

Despite the fact that fractional differential equations with Riemann-Liouville operators in them often lead to initial and boundary conditions which are fractional, recent work has been done on interpreting these systems physically. c.f. [11]

We will now make formal a definition of the Caputo fractional derivative.

Definition 3. *We define the Caputo fractional derivative based at a and of order $\alpha \in (0,\infty)$ of an admissable function f, as*

$$\left({}_a^C\mathcal{D}^\alpha f\right)(x) := \frac{1}{\Gamma(n-\alpha)}\int_a^x \frac{\frac{d^n}{dt^n}f(t)}{(x-t)^{\alpha-n+1}}dt$$

where $n = \lfloor\alpha\rfloor + 1$.

Here an admissable function is taken to have the same general meaning as described in the Riemann-Liouville fractional integral and derivative cases.

We can easily see that

$$\left({}_a^C\mathcal{D}^\alpha f\right)(x) = I_a^{n-\alpha}\frac{d^n}{dx^n}f(x) \tag{107}$$

and when we compare this observation with that in (72) we see that the Caputo derivative essentially amounts to swapping the order of fractional integration and integer differentiation when compared to the Riemann-Liouville fractional derivative.

We would now like to develop similar a relationship between ${}_a^C\mathcal{D}^\alpha$ and I_a^α as was done for ${}_a\mathcal{D}^\alpha$ but to do that we will first give a relationship between the Riemann-Liouville fractional derivative and the Caputo fractional derivative.

Lemma 10. *For an admissable function f we have that*

$$\left({}_a^C\mathcal{D}^\alpha f\right)(t) = \left({}_a\mathcal{D}^\alpha f\right)(t) - \sum_{k=0}^{n-1} \frac{t^{k-\alpha}}{\Gamma(k-\alpha+1)} f^{(k)}(a). \tag{108}$$

Proof. TODO □

Lemma 11. *For an admissable function f we have that*

$$\left({}_a^C\mathcal{D}^\alpha I_a^\alpha f\right)(t) = f(t) - \sum_{k=0}^{n-1} \frac{t^{k-\alpha}}{\Gamma(k-\alpha+1)} \left[\frac{d^k}{dt^k} I_a^\alpha f(t)\right]_{t=a} \tag{109}$$

The proof is immediate from lemmas 4 and 10. A similar but more complicated result exists for $I_a^\alpha \, {}_a^C\mathcal{D}^\alpha$ by using the result of lemma 5 but we will not need it for future results so we omit it. What will prove useful are results for the Laplace and Fourier transforms of the Caputo fractional derivative of a fucntion.

Lemma 12. *The Laplace transform of the Caputo derivative of an admissable function f is given by*

$$\mathcal{L}\left\{\left({}_0^C\mathcal{D}_t^\alpha f\right)\right\} = s^{\alpha-n}\left[s^n\mathcal{L}\{f\} - \sum_{k=0}^{n-1} s^{n-k-1}\left(\frac{d^k f}{dt^k}\right)(0)\right]. \tag{110}$$

Proof. See that

$$\mathcal{L}\left\{\left({}_0^C\mathcal{D}_t^\alpha f\right)\right\} = \mathcal{L}\left\{\left(I_0^{n-\alpha}\frac{d^n f}{dt^n}\right)\right\} \tag{111}$$

$$= \frac{1}{\Gamma(n-\alpha)}\mathcal{L}\left\{\int_0^t (t-u)^{n-\alpha-1}\frac{d^n f}{dt^n}du\right\} \tag{112}$$

$$\tag{113}$$

which is the Laplace transform of a convolution so

$$\Gamma(n-\alpha)\mathcal{L}\left\{\int_0^t (t-u)^{n-\alpha-1}\frac{d^n f}{dt^n}du\right\} = \mathcal{L}\left\{t^{n-\alpha-1}\right\}\mathcal{L}\left\{\frac{d^n f}{dt^n}\right\} \tag{114}$$

$$= \frac{1}{n-\alpha}\left(s^{-(n-\alpha)}\Gamma(n-\alpha)\right) \tag{115}$$

$$\times\left(s^n\mathcal{L}\{f\} - \sum_{k=0}^{n-1} s^{n-k-1}\left(\frac{d^k f}{dt^k}\right)(0)\right) \tag{116}$$

$$= s^{\alpha-n}\left[s^n\mathcal{L}\{f\} - \sum_{k=0}^{n-1} s^{n-k-1}\left(\frac{d^k f}{dt^k}\right)(0)\right]. \tag{117}$$

□

Lemma 13. *For an admissable function f the Fourier transform of the Riemann-Liouville fractional derivative with $a = -\infty$ and of order α is given by*

$$\mathcal{F}\left\{\left({}_{-\infty}\mathcal{D}^{\alpha}f\right)(t)\right\} = (-i\omega)^{\alpha}\mathcal{F}\left\{f(t)\right\}. \tag{118}$$

Proof. This proof follows in much the same way as the proof of lemma 8. Firstly we have that

$$\left({}^{C}_{-\infty}\mathcal{D}^{\alpha}f\right)(t) = I^{n-\alpha}_{-\infty}\frac{d^n}{dt^n}f(t) \tag{119}$$

and so by using the result of lemma 3 we have that

$$\mathcal{F}\left\{\left({}_{-\infty}\mathcal{D}^{\alpha}f\right)(t)\right\} = \mathcal{F}\left\{\left[I^{n-\alpha}_{-\infty}\frac{d^n}{dt^n}f(t)\right]\right\} \tag{120}$$

$$= (-i\omega)^{\alpha-n}\mathcal{F}\left\{\frac{d^n}{dt^n}f(t)\right\} \tag{121}$$

and then by using elementry Fourier transform results we get

$$(-i\omega)^{\alpha-n}\mathcal{F}\left\{I^{n-\alpha}_{-\infty}f(t)\right\} = (-i\omega)^{\alpha-n}(-i\omega)^{n}\mathcal{F}\left\{f(t)\right\} \tag{122}$$

$$= (-i\omega)^{\alpha}\mathcal{F}\left\{f(t)\right\} \tag{123}$$

□

A carefull reader might notice that the results of lemmas 8 and 13 are the same. This coincidence is actually hiding a more remarkable result which we formalise in the following lemma.

Lemma 14. *For an admissible function f such that $\lim_{t\longrightarrow-\infty} f^{(k)}(t) = 0$ for all $0 \leq k \leq n-1$ with n defined from α in the usual way we have that*

$$\left({}^{C}_{-\infty}\mathcal{D}^{\alpha}f\right)(t) = \left({}_{-\infty}\mathcal{D}^{\alpha}f\right)(t). \tag{124}$$

Proof. By using the result of lemma 10 we have that

$${}^{C}_{-\infty}\mathcal{D}^{\alpha}f(t) = {}_{-\infty}\mathcal{D}^{\alpha}f(t) - \sum_{k=0}^{n-1}\frac{t^{k-\alpha}}{\Gamma(k-\alpha+1)}\lim_{a\longrightarrow-\infty} f^{(k)}(a) \tag{125}$$

$$= {}_{-\infty}\mathcal{D}^{\alpha}f(t). \tag{126}$$

□

The restriction on f is not as limiting as one might imagine because for the Fourier transform to have nice properties such as being an automorphism one would want f to come from some sort of Schwartz space and the membership criteria for a Schwartz space are considerably more restrictive than the conditions given in the above lemma. In fact in defining precisely which functions are *admissable* one ends up having to place similar conditions on f. This is not a rabbit hole that we wish to explore further, however, and we refer the interested reader to [16] for a more thorough discussion of the matter.

3 Special Functions

The field of fractional calculus is intimately linked with the field of special functions. To express the fractional derivative of many functions one has to consider special functions. Just like the field of *ordinary* calculus has collections of functions which have a number of desirable properties like invariance (exponential) and *periodic invariance* (sine, cosine, hyperboid sine, hyperbolic cosine), fractional calculus has functions which have a very similar role.

We will look in detail at two functions, the Mittag-Leffler function and the Wright function. It may seem that just looking at two functions will be limiting, but this is not the case. Both of these functions are generalisations of more *ordinary* functions and they are general enough to serve considercompbz332979re8able utility. That being said the field of fractional calculus is surounded by a virtual zoo of special functions including the Meijer G-function, Fox-H function, the generalised hypergeometric functions, MacRobert E-function and the Mainardi function. This list could be made arbitrarily large by simply reading more and more litrature on the subject, but these functions give us a way of formalising remarkable relationships.

3.1 Mittag-Leffler Function

Just like the exponential function has an invariance property under differentiation, we would like a function that has a similar property for fractional differentiation. With that in mind we define the Mittag-Leffler function.

Definition 4. *Define the two parameter Mittag-Leffler function as*

$$E_{\alpha,\beta}(z) = \sum_{k=0}^{\infty} \frac{z^k}{\Gamma(\alpha k + \beta)}. \tag{127}$$

The reason we defined it as the *two paramter* Mittag-Leffler function is because some authors use the phrase Mittag-Leffler function to refer to a one parameter form, $E_\alpha(z) = E_{\alpha,1}(z)$. The two parameter version of this function is in common use and some more general results can be shown about the two paramter form. For the rest of our disucssions we will use the phrase Mittag-Leffler function to refer to the two paramter version.

The first thing that any reader should notice about the Mittag-Leffer function is that it is an immediate generalisation of the exponential function, with $E_{1,1}(z) = \exp(z)$.

It should be clear to see that at least for $\alpha, \beta \in \mathbb{R}$ this series uniformly converges on compact subsets in much that same way that that the series for the exponential function converges. This fact is important as it will allow us to interchange sums and integrals in several of the results which follow.

Interestingly a considerable number of other functions can be expressed in terms of Mittag-Leffler functions. By setting $\alpha = 0, \beta = 1$ we can easily see that we have a geometric series and so when $|z| < 1$ we have

$$E_{0,1}(z) = \frac{1}{z}. \tag{128}$$

Setting $\alpha = 2, \beta = 1$ we get

$$E_{2,1}(z) = \sum_{k=0}^{\infty} \frac{z^k}{\Gamma(2k+1)} \tag{129}$$

$$= \sum_{k=0}^{\infty} \frac{z^k}{(2k)!} \tag{130}$$

Now also notice that Taylor expansion of $\cosh(\sqrt{z})$ about 0 is

$$\cosh(\sqrt{z}) = \sum_{k=0}^{\infty} \frac{z^k}{(2k)!} \tag{131}$$

and so $E_{2,1}(z) = \cosh(\sqrt{z})$.

There are considerably more relationships. We refer the interested reader to [14] or [16] for a more extensive listing of these relationships.

We will now calculate the Laplace transform of some special cases of the Mittag-Leffler function. These will prove useful for latter results.

Lemma 15. *The Laplace transform of $E_{\alpha,1}(z)$ is given by*

$$\mathcal{L}\left\{E_{\alpha,1}(z)\right\} = \frac{s^{\alpha-1}}{s^\alpha - \gamma}. \tag{132}$$

Proof. We have that

$$\mathcal{L}\left\{E_{\alpha,1}(\gamma z^\alpha)\right\} = \int_0^\infty e^{-st} \sum_{k=0}^{\infty} \frac{(\gamma t^\alpha)^k}{\Gamma(\alpha k+1)} dt \tag{133}$$

$$= \sum_{k=0}^{\infty} \int_0^\infty \frac{e^{-st}(\gamma t^\alpha)^k}{\Gamma(\alpha k+1)} dt \tag{134}$$

$$\circledast = \sum_{k=0}^{\infty} \frac{\gamma^k}{\Gamma(\alpha k+1)} \int_0^\infty e^{-st} t^{\alpha k} dt. \tag{135}$$

With a change of variables $x = st$ we get that

$$\circledast = \sum_{k=0}^{\infty} \frac{\gamma^k s^{-(\alpha k+1)}}{\Gamma(\alpha k+1)} \int_0^\infty e^{-x} x^\alpha k dx \tag{136}$$

$$= \sum_{k=0}^{\infty} \frac{\gamma^k s^{-(\alpha k+1)\Gamma(\alpha k+1)}}{\Gamma(\alpha k+1)} \tag{137}$$

$$= \sum_{k=0}^{\infty} \gamma^k s^{-(\alpha k+1)} \tag{138}$$

$$= \frac{s^{\alpha-1}}{s^\alpha - \gamma}. \tag{139}$$

So we have that

$$\mathcal{L}\left\{E_{\alpha,1}(\gamma t^\alpha)\right\} = \frac{s^{\alpha-1}}{s^\alpha - \gamma} \tag{140}$$

□

The follopwing lemma is essentially a generalisation of lemma 15 but it is important as we will make use of it in future results.

Lemma 16. *The Laplace transform of $t^{\alpha m+\gamma-1}E^{(m)}_{\alpha,\gamma}(t)$ is given by*

$$\mathcal{L}\left\{t^{\alpha m+\gamma-1}E^{(m)}_{\alpha,\gamma}(\beta t^{\alpha})\right\} = \frac{m!s^{\alpha-\gamma}}{(s^{\alpha}-\beta)^{m+1}} \tag{141}$$

Proof. Firstly see that

$$E^{(m)}_{\alpha,\gamma}(t) = \sum_{k=m}^{\infty} \frac{\frac{k!}{(k-m)!}t^{k-m}}{\gamma(\alpha k+\gamma)} \tag{142}$$

$$= \sum_{k=0}^{\infty} \frac{(k+m)!t^{k}}{k!\Gamma(\alpha k+\gamma)} \tag{143}$$

so we have that

$$E^{(m)}_{\alpha,\gamma}(\beta t^{\alpha}) = \sum_{k=0}^{\infty} \frac{(k+m)!t^{\alpha k}\beta^{k}}{k!\Gamma(\alpha(k+m)+\gamma)}. \tag{144}$$

We can then write that

$$\mathcal{L}\left\{t^{\alpha m+\gamma-1}E^{(m)}_{\alpha,\gamma}(t)\right\} = \int_0^{\infty} t^{\alpha m+\gamma-1}\sum_{k=0}^{\infty} \frac{(k+m)!t^{\alpha k}\beta^{k}}{k!\Gamma(\alpha(k+m)+\gamma)} \tag{145}$$

$$= \sum_{k=0}^{\infty} \frac{\beta^{k}(k+m)!}{\Gamma(\alpha(k+m)+\gamma)k!}\underbrace{\int_0^{\infty} e^{-st}t^{\alpha(k+m)+\gamma-1}dt}_{\circledast}. \tag{146}$$

Considering just $\circledast$ and performing the substitution $x = st$ we get that

$$\circledast = s^{-\alpha(k+m)-\gamma}\int_0^{\infty} e^{-x}x^{\alpha(k+m)+\gamma-1}dx \tag{147}$$

$$= s^{-\alpha(k+m)-\gamma}\Gamma(\alpha(k+m)+\gamma) \tag{148}$$

and so

$$\mathcal{L}\left\{t^{\alpha m+\gamma-1}E^{(m)}_{\alpha,\gamma}(t)\right\} = s^{-\alpha m-\gamma}\sum_{k=0}^{\infty}\left(\frac{\beta}{s^{\alpha}}\right)^{k}\frac{(k+m)!}{k!}. \tag{149}$$

Now by the derivative rule for geometric series we get

$$\sum_{k=0}^{\infty}\left(\frac{\beta}{s^{\alpha}}\right)^{k}\frac{(k+m)!}{k!} = \frac{m!}{(1-\frac{\beta}{s^{\alpha}})^{m+1}} \tag{150}$$

$$= \frac{s^{\alpha(m+1)}m!}{(s^{\alpha}-\beta)^{m+1}} \tag{151}$$

and so

$$\mathcal{L}\left\{t^{\alpha m+\gamma-1}E^{(m)}_{\alpha,\gamma}(t)\right\} = \frac{m!s^{\alpha-\gamma}}{(s^{\alpha}-\beta)^{m+1}}. \tag{152}$$

□

It should be easy to see that

$$\frac{d}{dz}E_{\alpha,\beta}(z) = \sum_{k=0}^{\infty} \frac{(k+1)z^k}{\Gamma(\alpha(k+1)+\beta)} \tag{153}$$

and further that

$$\frac{d^n}{dz^n}E_{\alpha,\beta}(z) = \sum_{k=0}^{\infty} \frac{z^k}{\Gamma(\alpha(k+n)+\beta)} \prod_{j=1}^{n}(k+j). \tag{154}$$

So the Mittag-Leffler function retains a *similar* structure under differentiation. Obviously if $\alpha = 1$ and $\beta = 1$ this just reduces to the invariance of the exponential function under the differentiation map. It turns out that there are a whole collection of near-invariance and invariance properties of the Mittag-Leffler function. For a very general result we have the following lemma from [14].

Lemma 17. *We have that the fractional derivative of* $z^{\alpha k+\beta-1}E^{(k)}_{\alpha,\beta}(\lambda z^\alpha)$ *is given by*

$${}_0\mathcal{D}^\gamma z^{\alpha k+\beta-1}E^{(k)}_{\alpha,\beta}(\lambda z^\alpha) = z^{\alpha k+\beta-\gamma-1}E^{(k)}_{\alpha,\beta-\gamma}(\lambda z^\alpha) \tag{155}$$

Proof. We could prove this result by using lemma 9 and equation (154) but that turns out to be very convoluted. A simpler method is to get this result in Laplace space. □

4 Analytical Results for Ordinary Fractional Differential Equations

In this section we establish some analytical results for ordinary fractional differential equations. These results allow us to solve some simple initial value problems and establish a theoretical framework from which results in other sections can draw.

4.1 Solution to a linear intial value problem via Laplace transforms

We aim to get a solution to the following fractional differential equation (in terms of Caputo derivatives)

$$\left({}^{C}\mathcal{D}_0^{\alpha} y\right)(t) = \beta y(t) \tag{156}$$

along with the initial conditions

$$y^{(k)}(0) = \begin{cases} 1 & k = 0 \\ 0 & 1 \leq k \leq \lfloor \alpha \rfloor - 1 \end{cases} \tag{157}$$

It turns out that this initial value problem has the solution $y(t) = E_{\alpha}\left(\beta t^{\alpha}\right)$. Where E_{α} is the one parameter Mittag-Lefler function.

This solution can be arrived at by a Laplace transform method that is very similar to how this might be done in the integer order case.

We now have sufficient tools to attack the original problem, that is finding a solution to (156), (157).

Lemma 18. *The initial value problem defined in* (156) *and* (157)*, restated here for completeness*

$$\left({}^{C}\mathcal{D}_0^{\alpha} y\right)(t) = \beta y(t) \tag{158}$$

along with the initial conditions

$$y^{(k)}(0) = \begin{cases} 1 & k = 0 \\ 0 & 1 \leq k \leq \lfloor \alpha \rfloor - 1 \end{cases} \tag{159}$$

has solution $y(t) = E_{\alpha}\left(\beta t^{\alpha}\right)$.

Proof. Taking the Laplace transform of both sides of (156) yields

$$\mathcal{L}\left\{\left({}_0^C\mathcal{D}_t^\alpha y\right)\right\} = \beta\mathcal{L}\{y\} \tag{160}$$

$$s^{-(n+\alpha)}\left[s^n\mathcal{L}\{y\} - \sum_{k=0}^{n-1} s^{n-k-1}y^{(k)}(0)\right] = \beta\mathcal{L}\{y\} \tag{161}$$

by the result of lemma **??**. Then taking into account (157) we get

$$s^{-(n+\alpha)}\left[s^n\mathcal{L}\{y\} - s^{n-1}\right] = \beta\mathcal{L}\{y\} \tag{162}$$

and so

$$\mathcal{L}\{y\} = \frac{s^{\alpha-1}}{s^\alpha - \beta}. \tag{163}$$

By using the result of lemma 15 we have that

$$y(t) = E_\alpha(\beta t^\alpha) \tag{164}$$

□

4.2 Solution to a multi-order initial value problem via Laplace transforms

This section follows the technique outlined in [14].

We wish to consider the following differential equation

$$\left({}_0\mathcal{D}_\Lambda^y()\,t\right) + \left({}_0\mathcal{D}_\lambda^y()\,t\right) = f(t) \tag{165}$$

where $0 < \lambda < \Lambda < 1$.

Firstly note that this differential equation is in terms of Riemann-Liouville derivatives. If we were to specify initial conditions we would be compelled to specify them in terms of fractional derivatives, so we leave them unspecified here to see the solution in general.

Lemma 19. *The initial value problem, 165, restated here for completeness,*

$$\left({}_0\mathcal{D}_\Lambda^y()\,t\right) + \left({}_0\mathcal{D}_\lambda^y()\,t\right) = f(t) \tag{166}$$

has solution, given by

$$y(t) = Cg(t) + \int_0^t g(t-\tau)f(\tau)d\tau \tag{167}$$

where

$$C = \left({}_0\mathcal{D}_{\Lambda-1}^y()\right)0) + \left({}_0\mathcal{D}_{\lambda-1}^y()\right)0) \tag{168}$$

$$g(t) = t^{\Lambda-1}E_{\Lambda-\lambda,\Lambda}(-t^{\Lambda-\lambda}). \tag{169}$$

Proof. Taking the Laplace transform of both sides of 165 and using the result of lemma 7 we get that

$$\mathcal{L}\left\{\left({}_0\mathcal{D}^y_\Lambda()\,t\right)\right\}+\mathcal{L}\left\{\left({}_0\mathcal{D}^y_\lambda()\,t\right)\right\}=\mathcal{L}\left\{f(t)\right\} \tag{170}$$

$$s^\Lambda Y(s)+s^\lambda Y(s)-\left({}_0\mathcal{D}^y_{\Lambda-1}(\right)0)-\left({}_0\mathcal{D}^y_{\lambda-1}(\right)0)=F(s). \tag{171}$$

Note that

$$C=\left({}_0\mathcal{D}^y_{\Lambda-1}(\right)0)+\left({}_0\mathcal{D}^y_{\lambda-1}(\right)0)$$

is a constant so we write

$$Y(s)=\frac{C+F(s)}{s^\Lambda+s^\lambda} \tag{172}$$

$$=(C+F(s))\,\frac{s^{-\lambda}}{s^{\Lambda-\lambda}+1}. \tag{173}$$

Let

$$G(s)=\frac{s^{-\lambda}}{s^{\Lambda-\lambda}+1}$$

and by using lemma 16 with $\alpha=\Lambda-\lambda$ and $\gamma=\Lambda$ we get that

$$g(s)=t^{\Lambda-1}E_{\Lambda-\lambda,\Lambda}(-t^{\Lambda-\lambda})$$

where

$$\mathcal{L}\left\{g(t)\right\}=G(s)$$

.

Then using the Laplace convolution theorem we get that

$$y(t)=Cg(t)+\int_0^t g(t-\tau)f(\tau)d\tau \tag{174}$$

where

$$C=\left({}_0\mathcal{D}^y_{\Lambda-1}(\right)0)+\left({}_0\mathcal{D}^y_{\lambda-1}(\right)0) \tag{175}$$

$$g(t)=t^{\Lambda-1}E_{\Lambda-\lambda,\Lambda}(-t^{\Lambda-\lambda}). \tag{176}$$

□

4.3 Existence and uniquness of solutions to initial value problems

After looking at the solution to a couple of fractional differential equations we wish to consider the existence an uniqueness of solutions to a class fractional differential equations. This generalizes a result and technique of Tisdell [17] but a similar result for Miller-Ross sequential fractional differential equations can be found in [14].

Theorem 1 (Uniqueness)**.** *Consider the following intial value problem,*

$$\sum_{j=1}^{N} \beta_j \left({}_0^C\mathcal{D}_t^{\alpha_j} x \right)(t) = f(t, x(t)) \tag{177}$$

$$x(0) = A_0, x_1(0) = A_1, \ldots, x^{n_N}(0) = A_{n_N} \tag{178}$$

where $\alpha_1 > \alpha_2 > \ldots > \alpha_N$, $n_j = \lceil \alpha_j \rceil - 1$ *and* $\beta_1 = 1$.

Define

$$S := \{(t, p) \in \mathbb{R}^2 : t \in [0, a], p \in \mathbb{R}\}$$

Let $f : S \longrightarrow \mathbb{R}$ *be continuous. If there is a positive constant* L *such that*

$$|f(t, u) - f(t, v)| \leq L|u - v|, \text{ for all } (t, u), (t, v) \in S$$

and the constants $\{\alpha_j\}_{j=1}^N$, $\{\beta_j\}_{j=1}^N$ *are such that*

$$\sum_{j=2}^{N} |\beta_j| \, a^{\alpha_1 - \alpha_j} < 1$$

then the intial value problem defined in 177 and 178 has a unique solution.

To prove this we will need several lemmas.

Lemma 20. *The IVP defined in* (177), (178) *is equivalent to the integral equation*

$$x(t) = \sum_{k=1}^{n_1} \frac{A_k t^k}{k!} + \frac{1}{\beta_1} \Big(\frac{1}{\Gamma(\alpha_1)} \int_0^t (t - s)^{\alpha_1 - 1} f(s, x(s)) ds \tag{179}$$

$$- \sum_{j=2}^{N} \beta_j \frac{1}{\Gamma(\alpha_1 - \alpha_j)} \int_0^t (t - s)^{\alpha_1 - \alpha_j - 1} \left(x(s) - \sum_{k=1}^{n_j} \frac{A_k s^k}{k!} \right) ds \Big) \tag{180}$$

Proof. Apply (I_0^α) to both sides of (177) and recognize that

$$\left(I_0^\alpha \left({}_0^C\mathcal{D}_t^\alpha x \right) \right)(t) = x(t) + \sum_{k=0}^{n} \frac{x^{(k)}(0) t^k}{k!}$$

where $n = \lceil \alpha \rceil - 1$. □

Lemma 21.

$$\left(I_0^\xi E_\alpha(\gamma t^\alpha) \right) \leq t^\xi E_\alpha(\gamma t^\alpha) \tag{181}$$

Proof. See that

$$\left(I_0^{\xi} E_\alpha(\gamma t^\alpha)\right) = \frac{1}{\Gamma(\xi)} \int_0^t E_\alpha(\gamma s^\alpha)(t-s)^{\xi-1} ds \tag{182}$$

$$= \frac{1}{\Gamma(\xi)} \int_0^t \sum_{k=0}^{\infty} \frac{\gamma^k s^{\alpha k}}{\Gamma(\alpha k+1)} (t-s)^{\xi-1} ds \tag{183}$$

$$= \frac{1}{\Gamma(\xi)} \sum_{k=0}^{\infty} \frac{\gamma^k}{\Gamma(\alpha k+1)} \underbrace{\int_0^t s^{\alpha k}(t-s)^{\xi-1} ds}_{\circledast}. \tag{184}$$

Letting $\tau = \frac{s}{t}$ we have that

$$\circledast = \int_0^1 (t\tau)^{\alpha k}(t-t\tau)^{\xi-1} t d\tau \tag{185}$$

$$= t^{\alpha k+\xi} \int_0^1 (\tau)^{\alpha k}(1-1\tau)^{\xi-1} d\tau \tag{186}$$

$$= t^{\alpha k+\xi} B(\alpha k+1, \xi) \tag{187}$$

$$= t^{\alpha k+\xi} \frac{\Gamma(\alpha k+1)\Gamma(\xi)}{\Gamma(\alpha k+\xi+1)}. \tag{188}$$

This means that

$$\left(I_0^{\xi} E_\alpha(\gamma t^\alpha)\right) = \sum_{k=0}^{\infty} \frac{\gamma^k t^{\alpha k+\xi}}{\Gamma(\alpha k+\xi+1)} \tag{189}$$

$$= t^{\xi} \sum_{k=0}^{\infty} \frac{\gamma^k t^{\alpha k}}{\Gamma(\alpha k+\xi+1)} \tag{190}$$

$$\leq t^{\xi} \sum_{k=0}^{\infty} \frac{\gamma^k t^{\alpha k}}{\Gamma(\alpha k+1)} \tag{191}$$

$$= t^{\xi} E_\alpha(\gamma t^\alpha). \tag{192}$$

□

Lemma 22.

$$(I_0^{\alpha} E_\alpha(\gamma t^\alpha)) = \frac{1}{\gamma}\left(E_\alpha(\gamma t^\alpha) - 1\right) \tag{193}$$

Proof. See that

$$(I_0^{\alpha} E_\alpha(\gamma t^\alpha)) = \frac{1}{\Gamma(\alpha)} \int_0^t E_\alpha(\gamma s^\alpha)(t-s)^{\alpha-1} ds \tag{194}$$

$$= \frac{1}{\Gamma(\alpha)} \sum_{k=0}^{\infty} \frac{\gamma^k}{\Gamma(\alpha k+1)} \underbrace{\int_0^t s^{\alpha k}(t-s)^{\alpha} ds}_{\circledast}. \tag{195}$$

Letting $\tau = \frac{s}{t}$ we have that

$$\circledast = \int_0^1 (t\tau)^{\alpha k}(t - t\tau)^{\alpha-1} t d\tau \tag{196}$$

$$= t^{\alpha(k+1)} \int_0^1 \tau^{\alpha k}(1-\tau)^{\alpha-1} d\tau \tag{197}$$

$$= t^{\alpha(k+1)} B(\alpha k + 1, \alpha) \tag{198}$$

$$= t^{\alpha(k+1)} \frac{\Gamma(\alpha k + 1)\Gamma(\alpha)}{\Gamma(\alpha(k+1)+1)}. \tag{199}$$

This then means that

$$(I_0^\alpha E_\alpha(\gamma t^\alpha)) = \sum_{k=0}^{\infty} \frac{\gamma^k t^{\alpha(k+1)}}{\Gamma(\alpha(k+1)+1)} \tag{200}$$

$$= \frac{1}{\gamma} \sum_{k=1}^{\infty} \frac{\gamma^k t^{\alpha k}}{\Gamma(\alpha k + 1)} \tag{201}$$

$$= \frac{1}{\gamma} \left(\sum_{k=0}^{\infty} \frac{\gamma^k t^{\alpha k}}{\Gamma(\alpha k + 1)} - 1 \right) \tag{202}$$

$$= \frac{1}{\gamma} \left(E_\alpha(\gamma t^\alpha) - 1 \right). \tag{203}$$

□

Proof of theorem 1. To arrive at this we only have to prove that the map

$$[Fx](t) := \sum_{k=1}^{n_1} \frac{A_k t^k}{k!} + \frac{1}{\beta_1} \Big(\frac{1}{\Gamma(\alpha_1)} \int_0^t (t-s)^{\alpha_1 - 1} f(s, x(s)) ds \tag{204}$$

$$- \sum_{j=2}^{N} \frac{\beta_j}{\Gamma(\alpha_1 - \alpha_j)} \int_0^t (t-s)^{\alpha_1 - \alpha_j - 1} \left(x(s) - \sum_{k=1}^{n_j} \frac{A_k s^k}{k!} \right) ds \Big) \tag{205}$$

is contractive in the metric space $\left(C[0,a], d_\gamma^{\alpha_1}\right)$ where

$$d_\gamma^{\alpha_1}(x,y) = \max_{t \in [0,a]} \frac{|x(t) - y(t)|}{E_{\alpha_1}(\gamma t^{\alpha_1})}.$$

To see this note that

$$d_\gamma^{\alpha_1}(Fx, Fy) = \max_{t \in [0,a]} \frac{1}{E_{\alpha_1}(\gamma t^{\alpha_1})} \left| \frac{1}{\beta_1} \right| \Big| \frac{1}{\Gamma(\alpha_1)} \int_0^t (t-s)^{\alpha_1 - 1} (f(s, x(s)) - f(s, y(s)) ds \tag{206}$$

$$- \sum_{j=2}^{N} \frac{\beta_j}{\Gamma(\alpha_1 - \alpha_j)} \int_0^t (t-s)^{\alpha_1 - \alpha_j - 1} (x(s) - y(s)) ds \Big| \tag{207}$$

$$\leq \max_{t \in [0,a]} \frac{1}{E_{\alpha_1}(\gamma t^{\alpha_1}) |\beta_1|} \Big(\frac{1}{\Gamma(\alpha_1)} \int_0^t (t-s)^{\alpha_1 - 1} |f(s, x(s)) - f(s, y(s))| ds \tag{208}$$

$$+ \sum_{j=2}^{N} \frac{|\beta_j|}{\Gamma(\alpha_1 - \alpha_j)} \int_0^t (t-s)^{\alpha_1 - \alpha_j - 1} |x(s) - y(s))| ds \Big). \tag{209}$$

By exploiting the Lipshitz condition we can further write that

$$d_\gamma^{\alpha_1}(Fx,Fy) \leq \max_{t\in[0,a]} \frac{1}{E_{\alpha_1}(\gamma t^{\alpha_1})|\beta_1|}\Big(\frac{L}{\Gamma(\alpha_1)}\int_0^t (t-s)^{\alpha_1-1}|x(s)-y(s)|ds \tag{210}$$

$$+\sum_{j=2}^{N}\frac{|\beta_j|}{\Gamma(\alpha_1-\alpha_j)}\int_0^t (t-s)^{\alpha_1-\alpha_j-1}|x(s)-y(s))|ds\Big) \tag{211}$$

$$=\max_{t\in[0,a]} \frac{1}{E_{\alpha_1}(\gamma t^{\alpha_1})|\beta_1|}\Big(\frac{L}{\Gamma(\alpha_1)}\int_0^t (t-s)^{\alpha_1-1}\frac{|x(s)-y(s)|}{E_{\alpha_1}(\gamma s^{\alpha_1})}E_{\alpha_1}(\gamma s^{\alpha_1})ds \tag{212}$$

$$+\sum_{j=2}^{N}\frac{|\beta_j|}{\Gamma(\alpha_1-\alpha_j)}\int_0^t (t-s)^{\alpha_1-\alpha_j-1}\frac{|x(s)-y(s))|}{E_{\alpha_1}(\gamma s^{\alpha_1})}E_{\alpha_1}(\gamma s^{\alpha_1})ds\Big) \tag{213}$$

$$\leq d_\gamma^{\alpha_1}(x,y)\max_{t\in[0,a]} \frac{1}{E_{\alpha_1}(\gamma t^{\alpha_1})|\beta_1|}\Big(\frac{L}{\Gamma(\alpha_1)}\int_0^t (t-s)^{\alpha_1-1}E_{\alpha_1}(\gamma s^{\alpha_1})ds \tag{214}$$

$$+\sum_{j=2}^{N}\frac{|\beta_j|}{\Gamma(\alpha_1-\alpha_j)}\int_0^t (t-s)^{\alpha_1-\alpha_j-1}E_{\alpha_1}(\gamma s^{\alpha_1})ds\Big) \tag{215}$$

$$=d_\gamma^{\alpha_1}(x,y)\max_{t\in[0,a]} \frac{1}{E_{\alpha_1}(\gamma t^{\alpha_1})|\beta_1|}\Big(L\left(I_0^{\alpha_1}E_{\alpha_1}(\gamma t^{\alpha_1})\right. \tag{216}$$

$$\left.+\sum_{j=2}^{N}|\beta_j|\left(I_0^{\alpha_1-\alpha_j}E_{\alpha_1}(\gamma t^{\alpha_1})\right)\right). \tag{217}$$

$$\tag{218}$$

We can now use the results of lemmas 21 and 22 to write

$$d_\gamma^{\alpha_1}(Fx,Fy) \leq d_\gamma^{\alpha_1}(x,y)\max_{t\in[0,a]} \frac{1}{E_{\alpha_1}(\gamma t^{\alpha_1})|\beta_1|}\Big(\frac{L}{\gamma}\left(E_{\alpha_1}(\gamma t^{\alpha_1})-1\right) \tag{219}$$

$$+\sum_{j=2}^{N}|\beta_j|t^{\alpha_1-\alpha_j}E_{\alpha_1}(\gamma t^{\alpha_1})\Big) \tag{220}$$

$$=d_\gamma^{\alpha_1}(x,y)\max_{t\in[0,a]} \frac{1}{|\beta_1|}\Big(\frac{L}{\gamma}\left(1-\frac{1}{E_{\alpha_1}(\gamma t^{\alpha_1})}\right)+\sum_{j=2}^{N}|\beta_j|t^{\alpha_1-\alpha_j}\Big) \tag{221}$$

$$\tag{222}$$

and finally we get that

$$d_\gamma^{\alpha_1}(Fx,Fy) \leq d_\gamma^{\alpha_1}(x,y)\frac{1}{|\beta_1|}\left(\frac{L}{\gamma}+\sum_{j=2}^{N}|\beta_j|a^{\alpha_1-\alpha_j}\right). \tag{223}$$

By choosing γ sufficiently large we get that

$$\frac{1}{|\beta_1|}\left(\frac{L}{\gamma}+\sum_{j=2}^{N}|\beta_j|a^{\alpha_1-\alpha_j}\right)<1$$

and so F is a contractive mapping and thus the IVP defined in (177), (178) has a unique solution on $[0,a]$. □

Note that although existence is resolved (by virtue of the solutions given above) for the differential equations in (156, 157) and 165, this guarentees uniqueness on some closed interval starting at 0 for both cases.

Its also important to note that this result can be extended to differential equations involving Riemann-Liouville derivatives, by virtue of the correspondence between the Caputo derivative and the Riemann-Liouville derivative [14].

5 Numerical Methods for Fractional Differential Equations

In this section we describe numerical methods which can be used for solving fractional differential equations. We begin by looking at two ways for approximating a fractional derivative and then look at a specific scheme outlined by Diethelm [6].

5.1 Approximations of Fractional Derivatives and Integrals

We first recall that the Grünwald-Letnikov derivative of a function f is defined by

$$\left({}_a^{GL}\mathcal{D}_t^\alpha f\right)(t) = \lim_{h\longrightarrow 0} \frac{\left({}_a\Delta_h^\alpha f\right)(t)}{h^\alpha} \qquad\qquad ({}_a\Delta_h^\alpha f)(t) = \sum_{j=0}^{\lfloor \frac{t-a}{h} \rfloor} (-1)^j \binom{\alpha}{j} f(t-jh) \tag{224}$$

For a very large class of functions this coincides with the Riemann-Liouville fractional derivative [14] which we again recall here

$$({}_a\mathcal{D}_t^\alpha f)(t) = \left(\frac{d}{dt}\right)^{\lfloor\alpha\rfloor+1} \int_a^t (t-\tau)^{\lfloor\alpha\rfloor-\alpha} f(\tau)d\tau. \tag{225}$$

These two definitions suggest two general methods for approximating a fractional derivative.

5.1.1 Grünwald-Letnikov Approximation

In much the same was as we might use a finite difference method to approximate an integer order derivative we apply a similar approach here. By choosing some small, but non-zero value of h we can approximate the fractional derivative of a function by

$$\widehat{\left({}_a^{GL}\mathcal{D}_t^\alpha f\right)}(t) = \frac{1}{h^\alpha}\sum_{j=0}^{n}(-1)^j\binom{\alpha}{j}f(t-jh). \tag{226}$$

It can be shown [14] that this yields a first order approximation of the Riemann-Liouville fractional derivative.

5.1.2 Quadrature Approximation

A completely different approach is to use a quadrature rule to evaluate the integral from the Riemann-Liouville definition. If one is simply evaluating a derivative this method is not ideal because it also requires the approximation of an integer order derivative. If, however, we wish to calculate a fractional integral, such as might be required in a numerical FDE solver, then the quadrature method may be a more practical method.

There is no *set way* to do the quadrature approximation and it will all depend on the scheme being used.

In the following sections we will examine a general initial value problem and an Adams Moulton Bashforth scheme which is based off a quadrature method. [6]

5.1.3 An Initial Value Problem

Lets consider the initial value problem

$$\left({}_0^C\mathcal{D}_x^\alpha y\right) = f(x,y) \tag{227}$$

$$y^{(k)}(0) = y_0^{(k)} \tag{228}$$

for $0 \leq k < \lfloor \alpha \rfloor$. Note that this initial value problem is stated in terms of Caputo derivatives. The motivation for such initial value problems was discussed in [INSERT CC REF].

One of the most important things to note about this equation is that for non-integer values of α it is non-local, in the sense that the value of the solution at $y(x_0+h)$ depends not only on $y(x_0)$ but also on $y(x), x \in [0, x_0]$ [5]. This is in contrast to ordinary differential equations and this fact is what makes fractional differential equations considerably more complex to solve. Even multi-step methods which can be used to solve ordinary differential equations, only rely on some *fixed* number previous time steps. As a solution to a fractional differential equation progresses it relies on more and more previous time steps. As one might suspect this fundamentally increases the computational complexity of the schemes used to solve fractional differential equations as compared with the schemes used to solve traditional ordinary differential equations. [1]

5.1.4 The Adams Moulton Bashforth Scheme

We briefly outline a method explained and analyzed in detail in [8]. As shown in section 4.3 the initial value problem (227), (228) is equivalent to the Volterra equation.

$$y(x) = \sum_{k=0}^{\lceil \alpha \rceil - 1} y_0^{(k)} \frac{x^k}{k!} + \frac{1}{\Gamma(\alpha)} \int_0^x (x-t)^{\alpha-1} f(t, y(t)) dt \tag{229}$$

In order to approximate the solution to this integral equation over the interval $[0, T]$, we select a number of grid points N so that $h = T/N$ and $x_k = hk$ where $k \in \{0, 1, ..., N\}$.

[1] It is possible to reduce these computations to look at some large but fixed number of previous grid- points but this results in a speed vs. accuracy tradeoff which will be discussed in section **??**

We apply the approximation

$$\int_0^{x_{k+1}} (x_{k+1}-t)^{\alpha-1} g(t)dt \approx \int_0^{x_{k+1}} (x_{k+1}-t)^{\alpha-1} \tilde{g}_{k+1}(t)dt \tag{230}$$

where $g(t) = f(t, y(t))$ and $\tilde{g}_{k+1}(t)$ is the piecewise linear approximation of $g(t)$ with nodes at the grid-points x_k. As outlined in [8] we can approximate the integral in (229) as

$$\int_0^{x_{k+1}} (x_{k+1}-t)^{\alpha-1} \tilde{g}_{k+1}(t)dt = \sum_{j=0}^{k+1} a_{j,k+1} g(x_j) \tag{231}$$

where

$$a_{j,k+1} = \frac{h^\alpha}{\alpha(\alpha+1)} \times \begin{cases} (k^{\alpha+1} - (k-\alpha)(k+1)^\alpha) & \text{if } j = 0 \\ ((k-j+2)^{\alpha+1} + (k-j)^{\alpha+1} - 2(k-j+1)^{\alpha+1}) & \text{if } 1 \le j \le k \\ 1 & \text{if } j = k+1. \end{cases} \tag{232}$$

By separating the final term in the sum we can write

$$y_{k+1} = \sum_{j=0}^{\lceil\alpha\rceil-1} y_0^{(j)} \frac{x_{k+1}^j}{j!} + \frac{1}{\Gamma(\alpha)} \left(\sum_{j=0}^{k} a_{j,k+1} f(x_j, y_j) + a_{k+1,k+1} f(x_{k+1}, y_{k+1}^P) \right) \tag{233}$$

where y_{k+1}^P is a *predicted* value for y_{k+1} which must be calculated due to the potential non-linearity of f [8].

This predicted value is calculated by taking a rectangle approximation to the integral in (229), to get

$$\int_0^{x_{k+1}} (x_{k+1}-t)^{\alpha-1} g(t)dt \approx \sum_{j=0}^{k} b_{j,k+1} g(x_j) \tag{234}$$

where

$$b_{j,k+1} = \frac{h^\alpha}{\alpha} \left((k+1-j)^\alpha - (k-j)^\alpha \right). \tag{235}$$

and thus a predicted value of y_{k+1} can be calculated by

$$y_{k+1}^P = \sum_{j=0}^{\lceil\alpha\rceil-1} \frac{x_{k+1}^j}{j!} y_0^{(j)} + \frac{1}{\Gamma(\alpha)} \sum_{j=0}^{k} b_{j,k+1} f(x_j, y_j). \tag{236}$$

This outlines a fractional Adams Moulton Bashforth scheme. Of particular note are the sums which arise in (231) and (234). These sums do not arise in the integer order Adams Moulton Bashforth method. They arise as a result of the non-local nature of the fractional derivative [8]. These sums represent a significant computational cost as the number of terms grow linearly as the solution progress. This means in the fractional case the Adams Moulton Bashforth method has computational complexity $O(N^2)$ [6]. Section 5.1.5 will outline a task based parallel approach and section 5.1.7 will outline a massively parallel approach to solving (227), (228) with NVIDIA's CUDA.

5.1.5 Parallelizing: A Task Based Approach

Diethelm's paper [6] outlines a parallel method of numerically approximating the solution to (227), (228) by using a thread based parallel implementation of the Adams Moulton Bashforth method outlined in 5.1.4. We base our approach in this section broadly on those of [6] but reformulate the scheme in terms of tasks instead of threads. This has a number of distinct benefits including scalability, especially from an implementation standpoint and clarity.

After we have setup the grid-points $x_0, \ldots, x_{N-1}$ we create an array of solution values $y_0, \ldots, y_{N-1}$ which are to be populated with the calculated solution. From the initial conditions we can immediately calculate y_0. We then break up the vector (array) $\mathbf{y}$ into blocks of size p. Suppose that $p = 2$ for example. Then the first block would contain y_1 and y_2. Each of the p variables in each block can be calculated almost entirely in parallel. There is some dependency between values in each block (i.e. y_2 depends on y_1) but this can be done after the bulk of the parallel computations have been completed.

To illustrate this idea we consider figure 6. We have broken the vector (array) $\mathbf{y}$ up into $K = \lceil \frac{N}{p} \rceil$ blocks of p variables.

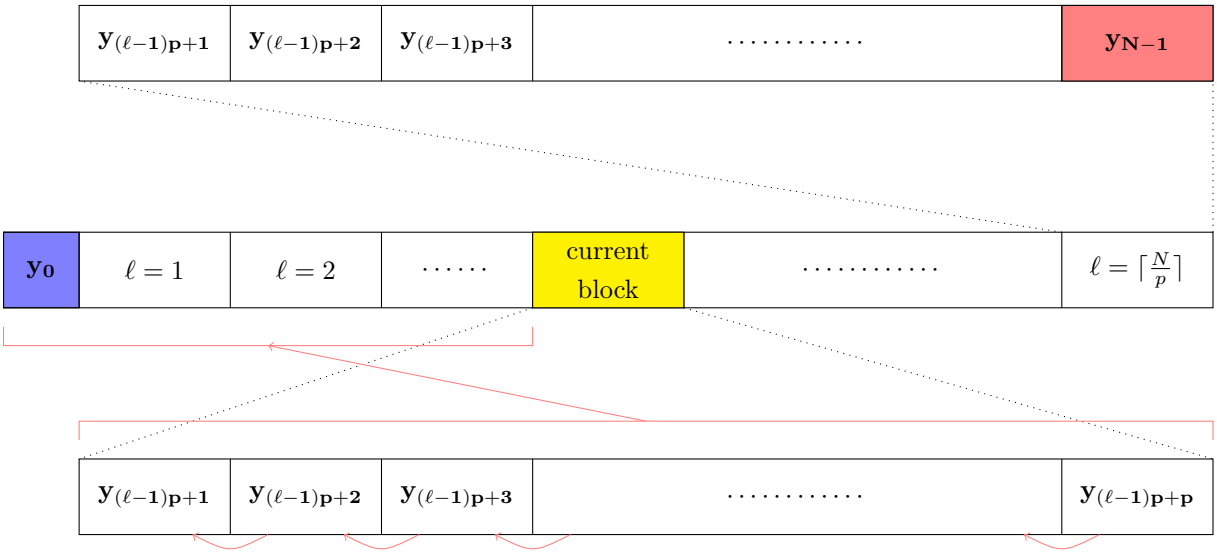


Figure 6: Computation diagram for the values in the vector (array) $\mathbf{y}$.

The center row shows how $\mathbf{y}$ is broken up into blocks, and the bottom row is a detailed view of the contents of the *current block* being computed.

The red lines indicate dependency, in that $\mathbf{y}_{(\ell-1)p+1}$ depends on all the y_j values calculated in the previous blocks. $\mathbf{y}_{(\ell-1)p+2}$ depends on all the y_j values calculated in previous blocks *and* on $\mathbf{y}_{(\ell-1)p+1}$.

The idea is that in each block we can do all of the computations which only depend on previous blocks in parallel and then perform the calculations which are dependent on other values within the same block in sequence. After all the values in a particular block are calculated we move on to the next block and repeat the process.

An exploded view of the last block is also provided to emphasize the fact that this block might not contain p variables if p does not divide $N - 1$. This fact causes us no bother as it is easy to handle in code.

As in [6] we rewrite the equations (233) and (236) as

$$y_{j+1}^P = I_{j+1} + h^\alpha H_{j,\ell}^P + h^\alpha L_{j,\ell}^P \tag{237}$$

and

$$y_{j+1} = I_{j+1} + h^\alpha H_{j,\ell} + h^\alpha L_{j,\ell} \tag{238}$$

where

$$I_{j+1} := \sum_{k=0}^{\lceil\alpha\rceil-1} \frac{x_{j+1}^k}{k!} y_0^{(k)} \tag{239}$$

$$H_{j,\ell}^P := \sum_{k=0}^{(\ell-1)p} b_{j-k} f(x_k, y_k) \tag{240}$$

$$L_{j,\ell}^P := \sum_{k=(\ell-1)p+1}^{j} b_{j-k} f(x_k, y_k) \tag{241}$$

$$H_{j,\ell} := c_j f(x_0, y_0) + \sum_{k=1}^{(\ell-1)p} a_{j-k} f(x_k, y_k) \tag{242}$$

$$L_{j,\ell} := \sum_{k=(\ell-1)p+1}^{j} aj - kf(x_k, y_k) + \frac{f(x_{j+1}, y_{j+1}^P)}{\Gamma(\alpha+2.)} \tag{243}$$

In each block the values of H, I and H^P can be calculated in parallel for each y_j in the block.Then there is a somewhat more complex dependency between these sums.

To best explain how the processes proceeds in each block and what dependencies there are we will consider a specific example where there are two values to be calculated per block (i.e. $p = 2$). Consider figure 7.

Each box in the figure represents a task. A task can execute when all the tasks with arrows pointing to it have completed. The little red number represents the index of the variable *within* the block which is being calculated. All the task names are reasonably self explanatory except for perhaps S^P and S. These take the values calculated in the other sums and add them together to get y^P and y respectively. These are very small tasks which take very little time to execute.

The red arrow illustrate the interdependency of variables within a block. The second variable cannot have the sum L^P calculate until the first variable is calculated. This means that in effect each of the L tasks have to execute in series but these sums are relatively small so the time taken is quite short. Each block is then executed in sequence as each block depends on the blocks before it. See Appendix A for a C# implementation of this.

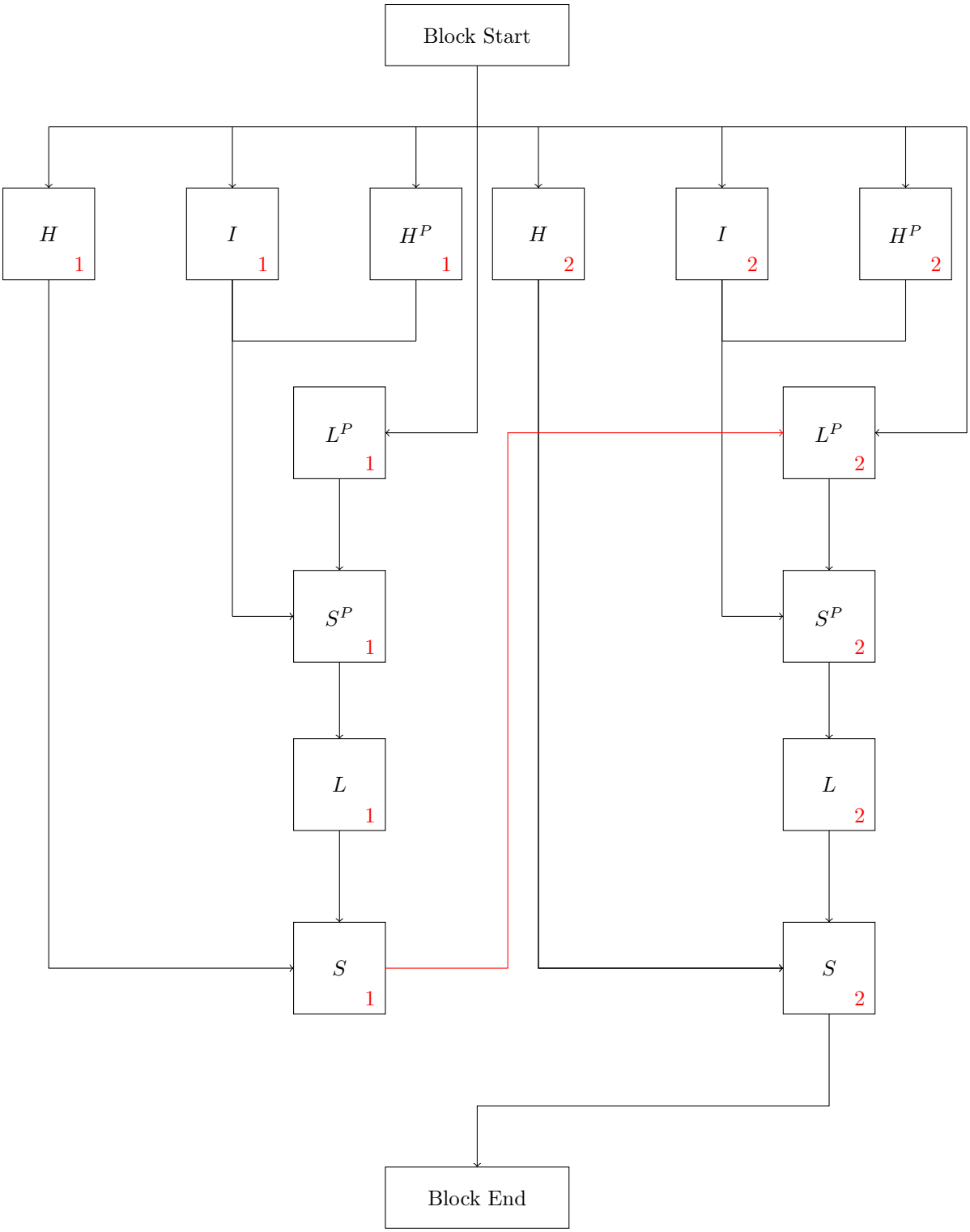


Figure 7: Task flow diagram for each block in the case when $p = 2$

5.1.6 Analysis of the Performance of the Scheme in C# and MONO

We wish to analyze the performance of this scheme both from the a theoretical standpoint and from at practical standpoint.

For the theoretical analysis of the scheme we introduce Amdahl's Law or Amdahl's principle.

Principle 1 (Amdahl)**.** *Let P be the proportion of a computation which can be parallelized in a given progress. Then the speedup given by Q cores is*

$$S(Q) = \frac{1}{(1-P) + \frac{P}{Q}} \tag{244}$$

This law was first introduced in a 1967 paper by computer architect Amdahl. We introduce a new notion of Amdahl efficiency to characterize the how *good* a parallel algorithm is in comparison with it serial counterpart in the limit of large input.

To formalize this notion we introduce the following definition.

Definition 5 (Amdahl Efficiency)**.** *Let $S(N, Q)$ represent the parallel speedup for a problem with input size N and executed with Q cores. Then we define the Amdahl efficiency of parallel scheme as*

$$E(Q) = \lim_{N \longrightarrow \infty} \frac{S(N,Q)}{Q}. \tag{245}$$

Further we say that a parallel scheme is Amdahl efficient with respect to its serial counterpart if $E(p) = 1$.

Proposition 1. *The parallel Adams Moulton Bashforth scheme is Amdahl efficient with respect to its serial counterpart.*

Proof. We present the paralellizable fraction of computation in each block of variables in the following table. The order column represents the number of *multiply-add* operations per task.

Task	Computational Complexity	Number of Variables per Block	Total Complexity
H	$O(\ell p)$	p	$O(\ell p^2)$
H_p	$O(\ell p)$	p	$O(\ell p^2)$
I	$O(\alpha)$	p	$O(p)$
L	$O(p)$	p	$O(p^2)$
L_p	$O(p)$	p	$O(p^2)$

In each block H, H_p and I can be computed in parallel across all variables in the block. For the purposes of simplicity we assume that the tasks L and L_p must all be computed in serial across all variables in a block. This means that in the ℓ-th block the non-paralellizable fraction of computation is

$$\overline{P}_{block}(\ell) \sim \frac{2p^2}{2p^2(\ell+1)} \sim \frac{1}{\ell} \tag{246}$$

The amount of computation in the ℓ-th block is $O(\ell p^2)$ and there are $K \approx \frac{N}{p}$ blocks in the total computation. This means that the total amount of computation is

$$\sum_{\ell=1}^{K} \ell p^2 \sim \frac{K^2 p^2}{2} \tag{247}$$

and by taking a weighted average sum, the total fraction of non-paralellizable computation is

$$\overline{P}_{total} \sim \sum_{\ell=1}^{K} \frac{2\ell p^2}{K^2 \ell p^2} = \frac{2}{K} = \frac{2p}{N}. \tag{248}$$

Noticing the fact that the computation in the task I is insignificant compared to H and H_p we would suppose using $Q = 2p$ cores would be in some sense optimal. Applying Amdahl's principle with the result in 248 and with $Q = 2p$ we get

$$S(N, Q) = S(N, 2p) \tag{249}$$

$$\sim \frac{1}{(\frac{2p}{N}) + \frac{1 - \frac{2p}{N}}{2p}} \tag{250}$$

$$\tag{251}$$

then letting $N \longrightarrow \infty$ we get

$$E(p) = \lim_{N \longrightarrow \infty} \left(\frac{1}{(\frac{2p}{N}) + \frac{1 - \frac{2p}{N}}{2p}} \right) / p = 1. \tag{252}$$

□

What this result means is that if we have $2p$ free cores this parallel method should take $\frac{1}{2p}$ the amount of time to run compared to it's serial counterpart given a sufficiently large number of grid points, N. Due to the fact that this a task based scheme this is also dependent on an efficient scheduler.

We now compare this with practical experimentation. This method was developed in C# with MONO (see Appendix A for C# code) and run on a machine with an Intel ®Core ™ i7-4930K CPU @ 3.40GHz. This particular CPU has 6 physical cores with 12 logical cores provided by hyper-threading.

We solved the following initial value problem

$$\left({}_0^C \mathcal{D}_x^{\frac{1}{2}} y \right)(x) = -y(x) \qquad\qquad y(0) = 1 \tag{253}$$

with $N = 2000$ through to $N = 10000$ time steps over the interval $[0, 10]$ with p ranging from 1 though to 12. Each solution was computed 5 times and the average runtime was computed, yielding the *performance surface* shown in figure 8. [2]

[2]See Appendix B for a full table of runtimes.

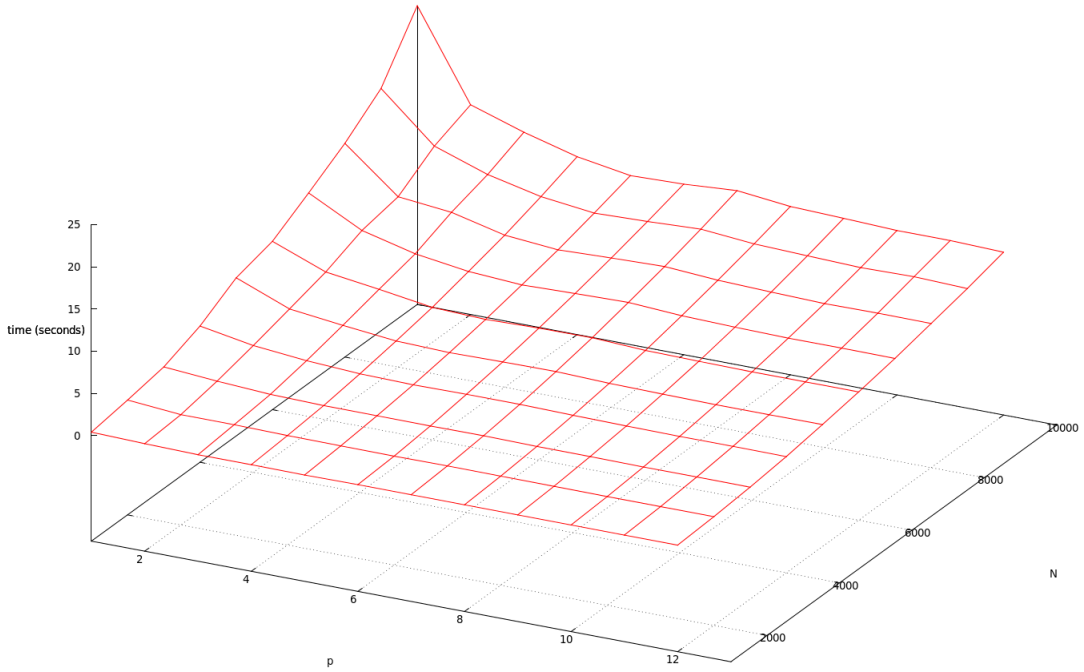


Figure 8: Average runtime of the scheme for different values of p and N.

Figure 8 clearly shows the $O(N^2)$ complexity of the algorithm, especially in the case where $p = 1$. We present some of the results from the $N = 10000$ computations in a figure 5.1.6 to illustrate the speedup.

p	Runtime (s)	Speedup (wrt p = 1)
1	22.89	1
2	12.35	1.85
3	10.28	2.22
4	8.58	2.66
5	7.53	3.04
6	7.70	2.97

From a theoretical perspective we would expect speedups of 2, 3 and 4 for the cases $p = 2, 3, 4$ respectively, however, this is not achieved. There are a number of possible reasons for this. Firstly the MONO task scheduler (based of the .Net scheduler) is not performance optimized [4].

There is the possibility that we haven't chosen a large enough value of N to get the full relative speedup. This is supported by the fact that for the $N = 8000$ case the speedup from $p = 1$ to $p = 2$ was 1.53 (less than that in the $N = 10000$ case).

Also for cases where $p > 3$ we have 6 computationally heavy tasks (H and H_p) being completed per block and are therefore reaching the limit of the number of physical cores on the system. Obviously we cannot expect an improvement once p gets too large as the number of tasks to be executed in parallel exceeds the number of available cores. We see the full effect of this in the $p = 6$ case where the runtime was actually longer than that for $p = 5$.

5.1.7 Parallelizing: A CUDA Approach

One of the main benefits of the scheme outlined in section 5.1.5, especially noted in it's form presented by Diethelm in [6] is the ability for this scheme to be implemented across multiple machines with a small amount of message passing between executing threads (or in this case tasks). The main reason for wanting to minimize message passing is that it can significantly impact performance to the point where most of the time is actually spent communicating updates between execution nodes. This is given significant consideration in [6]. In this section we abandon these measures in favor of massive performance gains when executing in a massively parallel manner on a GPU.

Modern graphics cards have the advantage of being able to efficiently perform massively parallel computations, especially calculations involving floating point arithmetic [10]. For example the NVIDIA GeForce GTX 780 has 2304 cores clocked at 863MHz. We use one of these graphics cards in an effort to dramatically speed up the fractional Adams Moulton Bashforth Method using NVIDIA's CUDA tool-chain.

Instead of using blocks of variables as was done in [6] and reimplemented by us in section 5.1.5 we return to the original specification of the scheme in section 5.1.4 and seek out other opportunities of parallelism.

The essential problem with the scheme is that the number of terms in the sums defined in (236) and (233) grows linearly with each time step taken. It is clear that the actual terms in each sum are independent and so breaking the sum up into smaller parts to be calculated and then summed together is a reasonable approach. This is so long as communication between cores doing different parts of the sum is sufficiently fast and global memory access is sufficiently fast. In the case of a GPU both of these criteria are satisfied and so this approach is reasonable.[3]

We aim to use as much of the GPU as possible to perform the calculations and so unlike the scheme outlined in section 5.1.5 we will automatically scale the number of threads to use as many cores as possible. We define one *performance tuning* parameter $opsPerThread$ which define the number of *multiply-add* operations per thread. Figure 9 shows a *performance surface* showing runtime against number of time steps N and $opsPerThread$ [4]. The actual initial value problem being solved is exactly the same as in section 5.1.5.

[3] It is easy to see that this scheme would be Amdahl efficient in the sense of definition 5.

[4] See appendix B for a full table of runtimes

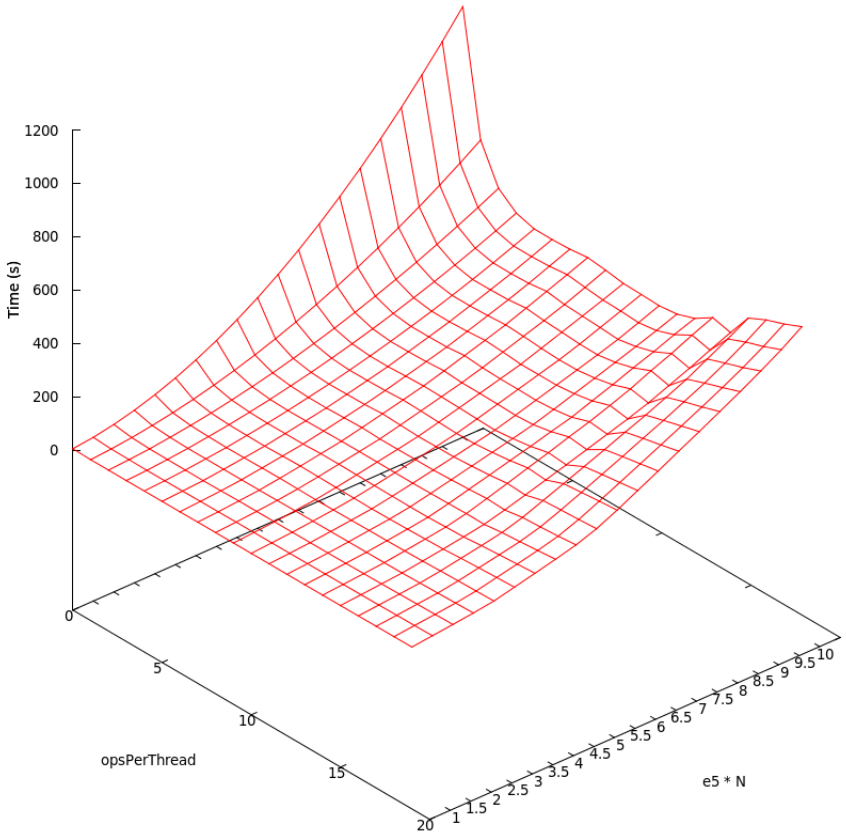


Figure 9: Runtime for the scheme for various values of opsPerThread and N.

Firstly we note the size of the problems being solved here. The last point on the N axis is 10^6 time-steps, 100 times larger than was computed in the CPU based C# approach. Given the quadratic computational complexity of this algorithm that means this CUDA implementation is solving problems which would take roughly 2.5 days to compute using the previous parallel scheme, in less than 6 minutes.

As can be seen from this chart, as N increases it makes sense to perform more operations per thread. This arises from the fact that there is an overhead in dealing with each thread and this overhead becomes significant as N gets large. This effect is also exacerbated by the fact that for small values of N the overhead of managing more threads on the GPU is offset by the increased parallelism that a smaller number of operations per thread affords. Once, however, every core is being utilized it makes sense to start to increase the number of operations per thread as N is increased.

The speedup from running these computation on CUDA is enormous. It is arguable that any advantage achieved through reducing the number of messages passed between multiple threads and nodes by using the scheme outlined in [6] and in section 5.1.5, is completely outweighed by the sheer speed of computation on a GPU.

6 Fractional Diffusion

In this last section we would like to take a look at fractional diffusion. This is one of the main areas where fractional calculus is getting heavily used today. Modeling over the last two decades has shown that it is extremely important for explaining anomylous diffusion in beiological contexts. It also appears to play a role in diffusion in porus media and has extensions into physics and chemistry. It is perhaps now, and in the field of fractional diffusion, that fractional calculus is finally being pulled out of the cuboard of pure mathematics and Liebniz' fabeled *ueseful consequences* are finally being drawn from this field.

Fractional diffusion is particularly interesting in that it is not just a phenomenological model which appears to fit with real world data better simply by virtue of its reduced parsemony. It can actually be explained by some deeper and very justifiable physical principles. In order to give a good account of these phsycial ideas and in some sense *derive* fractional diffusion we will start with ordinary diffusion and then work our way to fractional diffusion by considering continuous time random walks.

We will examine diffusion models with just one spacial dimension. The generalisation to multiple spacial dimeansions is straight forward and only serves to add unnecessary notational complexity.

6.1 Standard Diffusion

6.1.1 Derivation

Starting with a discrete time, discrete step (unbiased) random walk we can arrive at a *continium* approximation the random walk as follows.

If $P(x,t)$ is the probability of being at location x at time t we have that

$$P(x,t) = \frac{1}{2}P(x-\Delta x, t-\Delta t) + \frac{1}{2}P(x+\Delta x, t-\Delta t) \tag{254}$$

where Δx is the spacial step size and Δt is the time step size. Now taking Taylor expansions we get that

$$\begin{aligned} P(x+\Delta x, t-\Delta t) = P(x,t) + \Delta x\frac{\partial P}{\partial x} - \Delta t\frac{\partial P}{\partial t} + \frac{(\Delta x)^2}{2}\frac{\partial^2 P}{\partial x^2} + \frac{(\Delta t)^2}{2}\frac{\partial^2 P}{\partial t^2} \\ - \Delta t\Delta x\frac{\partial^2 P}{\partial x \partial t} + O((\Delta t)^3) + O((\Delta x)^3) \end{aligned} \tag{255}$$

and

$$\begin{aligned} P(x-\Delta x, t-\Delta t) = P(x,t) - \Delta x\frac{\partial P}{\partial x} - \Delta t\frac{\partial P}{\partial t} + \frac{(\Delta x)^2}{2}\frac{\partial^2 P}{\partial x^2} + \frac{(\Delta t)^2}{2}\frac{\partial^2 P}{\partial t^2} \\ + \Delta t\Delta x\frac{\partial^2 P}{\partial x \partial t} + O((\Delta t)^3) + O((\Delta x)^3) \end{aligned} \tag{256}$$

and so by substituting (255) and (256) into (254) we get that

$$\frac{(\Delta x)^2}{2}\frac{\partial^2 P}{\partial x^2} = (\Delta t)\frac{\partial P}{\partial t} + O((\Delta t)^2) + O((\Delta x)^3). \tag{257}$$

Now sending Δx and Δt to zero *at the same rate* we get that

$$\frac{\partial P}{\partial t} = D\frac{\partial^2 P}{\partial x^2}$$

where

$$D = \lim_{\Delta x \longrightarrow 0, \Delta t \longrightarrow 0} \frac{(\Delta x)^2}{2\Delta t}. \tag{258}$$

Note that (258) looks like the standard diffusion equation except it is in terms of probabilities instead of concentrations. By taking an ensamble of points we can reason that there is a direct correspondence between probabilities and the concentrations.

6.1.2 Characterising Results

We will breifly go through some results which help characterise how solutions to standard diffusion behave as these same ideas will be used in characterising the fractional differential equation.

6.1.3 Analytic Solution Method

There are actually relatively few situations where we can solve the diffusion equation in terms of named functions. Often these are solved by application of a Fourier transform and involve the error function which we define here for completeness.

Definition 6 (Error Function)**.** *We define the error function as*

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt \tag{259}$$

and the compelementry error function as

$$\text{erfc}(x) = \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-t^2} dt. \tag{260}$$

It is rather elementry to show that

$$\int_0^\infty e^{-t^2} dt = \frac{2}{\sqrt{\pi}} \tag{261}$$

and hence we have $\text{erfc}(x) = 1 - \text{erf}(x)$.

6.2 Continuous Time Random Walks

In the section above *derived* the standard diffusion equation from a random walk. What was important about this derivation was that we took a discrete time / discrete space model, (descritized by Δt and Δx respectively) turned it into a continuous model by taking Δx and Δt to zero at the same rate.

In this section we explore another method for dealing with random walks. One where the step size is distributed according to some probability distribution and the waiting time between steps is distributed according to some other distribution.

We will need to deal with two main densities in this process, the *arrival density* and the *being density*.

We define $a_n(x, t|x_0, t_0)$ as the conditional probability density which describes the probability that after n steps the random walk discussed above which started at x_0 at time t_0 arrives at x at time t.

Like in the case of standard diffusion we can give a recurrence relationship which begins the characterise this density. We can say that

$$a_{n+1}(x, t|x_0, t_0) = \int_{-\infty}^{\infty} \int_{t_0}^{t} \Psi(x - u, t - \tau) a_n(u, \tau|x_0, t_0) d\tau du \tag{262}$$

where $\Psi(x - u, t - \tau)$ is the probability density which describes the probability that we take a step of size $x - u$ after waiting for a time $t - \tau$.

The probability distribution Ψ characterises both the waiting time distribution and the step size distribution. We will assume that the waiting times and step sizes are independent of each other and hence we can *factorize* this distribution as $\Psi(x - u, t - \tau) = \lambda(x - u)\phi(t - \tau)$ where λ the step size probability density function and ϕ is the waiting time probability density function.

We can write down the condition probability density for the random walk arriving at x at time t after *any* number of steps from the starting position x_0 at t_0 as

$$a(x, t|x_0, t_0) = \sum_{n=0}^{\infty} a_n(x, t|x_0, t_0) \tag{263}$$

by summing over the all the possible numbers of steps taken to arrive at x.

By performing this same sum over (262) we get that

$$\sum_{n=1}^{\infty} a_{n+1}(x, t|x_0, t_0) = \sum_{n=1}^{\infty} \int_{-\infty}^{\infty} \int_{t_0}^{t} \Psi(x - u, t - \tau) a_n(u, \tau|x_0, t_0) d\tau du \tag{264}$$

$$a(x, t|x_0, t_0) - \delta_{x_0}(x)\delta_{t_0}(t) = \int_{-\infty}^{\infty} \int_{t_0}^{t} \sum_{n=0}^{\infty} \Psi(x - u, t - \tau) a_n(u, \tau|x_0, t_0) d\tau du \tag{265}$$

$$= \int_{-\infty}^{\infty} \int_{t_0}^{t} \Psi(x - u, t - \tau) a(u, \tau|x_0, t_0) d\tau du. \tag{266}$$

Notice that in (265) we have subtracted $\delta_{x_0}(x)\delta_{t_0}(t)$ from the left hand side. This is done to ballance the sums otherwise we would be missing a term on the left hand side. This can be interpreted as an initial condition on the random walk which says that at $x = x_0, t = t_0$ we have $a(x, t|x_0, t_0) = 1$ which is an obvious requirement as it would not make sense otherwise, given the definition of a.

We now turn our attention to the *being density* which we define as the probability density which describes that probability that at a time t the random walk is at x. Note that this is different to the arrival density which describes the probability of *arriving at* x at time t.

Using the definition of ϕ above it is clear to see that the survival probability density Φ which describes the probability of *not* making a step at time t is

$$\Phi(t) = 1 - \int_0^t \phi(\tau)d\tau \tag{267}$$

From this it makes sense to define the being density b by

$$b(x,t|x_0,t_0) = \int_{t_0}^t a(x,t-\tau)|x_0,t_0)\Phi(\tau)d\tau \tag{268}$$

$$= \int_{t_0}^t a(x,\tau|x_0,t_0)\Phi(t-\tau)d\tau. \tag{269}$$

Without loss of generality we can set $t_0 = 0$ because we can always shift our problem in time. Doing this has the advantage that in the following arguments we can easily invoke the Laplace convolution theorem which usually requires the base of the integral to be 0.

Summarising results we have that

$$a(x,t|x_0,0) = \int_{-\infty}^{\infty}\int_0^t \Psi(x-u,t-\tau)x(u,\tau|x_0,0)d\tau du + \delta_{x_0}(x)\delta(t) \tag{270}$$

and

$$b(x,t|x_0,0) = \int_0^t a(x,\tau|x_0,0)\Phi(t-\tau)d\tau. \tag{271}$$

Now denoting Laplace transforms by writing $\hat{f}(s) = \mathcal{L}\{f(t)\}$ and by taking a Laplace transform of (270) we get

$$\hat{a}(x,s|x_0,0) = \int_{-\infty}^{\infty} \hat{\Psi}(u,s)\hat{a}(x-u,s|x_0,0)du + \delta_{x_0}(x). \tag{272}$$

This follows from the Laplace convolution theorem and the Laplace transform of the δ function. By also taking the Laplace transform of (271) and using the Laplace convolution theorem we get

$$\hat{b}(x,s|x_0,0) = \hat{a}(x,s|x_0,0)\hat{\Phi}(s). \tag{273}$$

By subsituting (272) into (273) we get

$$\hat{b}(x,s|x_0,0) = \underbrace{\int_{-\infty}^{\infty} \hat{\Psi}(u,s)\hat{\Phi}(s)\hat{a}(x-u,s|x_0,0)du}_{\circledast} + \hat{\Phi}(s)\delta_{x_0}x \tag{274}$$

and if we put (273) in $\circledast$ we finally get

$$\hat{b}(x,s|x_0,0) = \int_{-\infty}^{\infty} \hat{\Psi}(u,s)\hat{b}(x-u,s|x_0,0)du + \hat{\Phi}(s)\delta_{x_0}(x). \tag{275}$$

It is worth of note that we can take the inverse Laplace transform to get what is known as the continuous time random walk master equation

$$b(x,t|x_0,0) = \int_{-\infty}^{\infty}\int_0^t \Psi(x-u,t-\tau)b(u,\tau|x_0,0)dtdu + \Phi(t)\delta_{x_0}(x) \tag{276}$$

however we will try to decouple (275) futher in what follows by applying a Fourier transform.

Firstly we will appeal to the same reasoning used in the last part of section 6.1 to replace the probability density b with the concentration c. The point of doing this is that it allows us to talk about an ensamble of points with an initial distribution $c(x_0,0)$, instead of just the starting location of just one point.

This means that (273) can be rewritten as

$$\hat{c}(x,s) = \int_{-\infty}^{\infty} \hat{\Psi}(u,s)\hat{c}(x-u,s)du + \hat{\Phi}(s)c(x,0). \tag{277}$$

Now note that using (267) and the Laplace transform of an integral we can write

$$\hat{\Phi}(s) = \frac{1}{s} - \frac{\phi(s)}{s}. \tag{278}$$

Now using the factorisation of Ψ mentioned above we can rewrite (275) as

$$\hat{c}(x,s) = \int_{-\infty}^{\infty} \hat{\phi}(s)\lambda(u)\hat{b}(x-u,s)du + \left(\frac{1}{s} - \frac{\phi(s)}{s}\right)c(x,0). \tag{279}$$

Now denoting the Fourier transform of a function f by $F(\omega) = \mathcal{F}\{f(x)\}$ and by invoking the Fourier convolution theorem we can write

$$\hat{C}(\omega,s) = \mathcal{F}\left\{\hat{\phi}(s)\int_{-\infty}^{\infty}\lambda(u)\hat{c}(x-u,s)du + \left(\frac{1}{s} - \frac{\phi(s)}{s}\right)c(x,0)\right\} \tag{280}$$

$$= \hat{\phi}(s)\Lambda(\omega)\hat{C}(\omega,s) + \left(\frac{1}{s} - \frac{\hat{\phi}(s)}{s}\right)C(\omega,0). \tag{281}$$

This is essentially as far as we will go without specifying the distributions λ or ϕ. Different specifications for λ and ϕ will result in different forms of diffusion.

6.2.1 Standard Diffusion

In this section we will show that in the context of the model in the above section we can arive at standard diffusion just like in section 6.1 by saying the step length is *normally* distributed and the waiting time between jumps is *exponentially* distributed.

If we set

$$\lambda(x) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{x^2}{2\sigma^2}\right) \tag{282}$$

and

$$\phi(t) = \frac{1}{\alpha} \exp\left(-\frac{t}{\alpha}\right) \tag{283}$$

which is to say that the step size $\Delta x \sim \mathcal{N}(0, \sigma^2)$ and the time step $\Delta t \sim \text{Exp}(\frac{1}{\alpha})$.

In this case we have that

$$\Lambda(\omega) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}\sigma^2\omega^2} \tag{284}$$

$$= \frac{1}{\sqrt{2\pi}} - \frac{\sigma^2\omega^2}{2\sqrt{2\pi}} + O(\omega^4) \tag{285}$$

and

$$\hat{\phi}(s) = \frac{1}{1+\alpha s} \tag{286}$$

$$= 1 - \alpha s + O(s^2) \tag{287}$$

By putting (285) and (287) into (281) we get that

$$\hat{C}(\omega, s) = \frac{1}{\sqrt{2\pi}}(1-\alpha s)\left(1 - \frac{\sigma^2\omega^2}{2}\right)\hat{C}(\omega, s) + \alpha C(\omega, 0) \tag{288}$$

and so

$$s\hat{C}(\omega, s) = \frac{1}{\sqrt{2\pi}}(s-\alpha s^2)\left(1 - \frac{\sigma^2\omega^2}{2}\right)\hat{C}(\omega, s) + \alpha C(\omega, 0) \tag{289}$$

6.2.2 Fractional Diffusion

6.2.3 Characterising Results for Fractional Diffusion

6.2.4 Solution Method for Fractional Diffusion

7 Appendix A: Adams-Moulton-Bashforth FDE Solver Code

7.1 C# Implementation of AMB FDE Solver

7.1.1 Program.cs

```
using System;
using System.Threading;
using System.Threading.Tasks;
using System.Collections.Generic;
using System.IO;
using System.Text;

namespace FDE_Solver
{
	class MainClass
	{
		/// <summary>
		/// This is the delegate use for the right hand side of the
		/// differential equation. It define the signature that the
		/// forcing function / right hand side must satisfy.
		/// </summary>
		public delegate double ForcingFunction (double x, double y);

		/// <summary>
		/// Main entry point of the program.
		/// </summary>
		/// <param name="args">The command-line arguments.</param>
		public static void Main (string[] args)
		{
		    double[] y = Compute (0.5, new double[] { 1 }, 10, 10000, 12, new
		        ForcingFunction (ff));
		}
		/// <summary>
		/// This is the RHS of the differential equation.
		/// For the purposes of demonstation this code is setup to
		/// solve D^{1/2}y = -y
		/// </summary>
		public static double ff(double x, double y)
		{
			return -y;
		}

		/// <summary>
		/// This calculates the a coefficient described in the K. Diethelm paper
		/// referenced in the body of thesis.
		/// </summary>
		public static double a(int mu, double alpha)
		{
			return (Math.Pow(mu + 2, alpha + 1) - 2 * Math.Pow(mu + 1, alpha + 1) +
			    Math.Pow(mu, alpha + 1))/SpecialFunctions.Gamma(alpha + 2);
		}

		/// <summary>
```

```
/// This calculates the c coefficient described in the K. Diethelm paper
/// referenced in the body of thesis.
/// </summary>
public static double c(int mu, double alpha)
{
       return (Math.Pow(mu, alpha+1) - (mu - alpha)*Math.Pow(mu + 1, alpha)) /
           SpecialFunctions.Gamma(alpha + 2);
}

/// <summary>
/// This calculates the b coefficient described in the K. Diethelm paper
/// referenced in the body of thesis.
/// </summary>
public static double b(int mu, double alpha)
{
       return (Math.Pow (mu + 1, alpha) - Math.Pow (mu, alpha)) /
           SpecialFunctions.Gamma (alpha + 1);
}

/// <summary>
/// This calculates the sum I_{j+1} described in the K. Diethelm paper
/// referenced in the body of thesis.
/// </summary>
public static double I_1(int j, double alpha, double[] y_0_diffs, double[] x)
{
       double value = 0;
       for (int k = 0; k <= Math.Ceiling (alpha) - 1; k++) {
              value += Math.Pow (x [j + 1], k) / SpecialFunctions.Factorial (k) *
                  y_0_diffs [k];
       }
       return value;
}

/// <summary>
/// This calculates the sum H^p_{j,\ell} described in the K. Diethelm paper
/// referenced in the body of thesis.
/// </summary>
public static double H_p(int j, int ell, int p, double[] x, double[] y, double
    alpha, ForcingFunction f)
{
       double value = 0;
       for (int k = 0; k <= (ell - 1) * p; k++) {
              value += b (j - k, alpha) * f (x [k], y [k]);
       }
       return value;
}
/// <summary>
/// This calculates the sum L^p_{j,\ell} described in the K. Diethelm paper
/// referenced in the body of thesis.
/// </summary>
public static double L_p(int j, int ell, int p, double[] x, double[] y, double
    alpha, ForcingFunction f)
{
       double value = 0;
       for (int k = (ell - 1) * p + 1; k <= j; k++) {
              value += b (j - k, alpha) * f (x [k], y [k]);
```

```
        }
        return value;
}

/// <summary>
/// This calculates the sum H^_{j,\ell} described in the K. Diethelm paper
/// referenced in the body of thesis.
/// </summary>
public static double H(int j, int ell, int p, double[] x, double[] y, double alpha,
    ForcingFunction f)
{
        double value = 0;
        value += c (j, alpha) + f (x [0], y [0]);
        for (int k = 1; k <= (ell - 1) * p; k++) {
                value += a (j - k, alpha) * f (x [k], y [k]);
        }
        return value;
}
/// <summary>
/// This calculates the sum L_{j,\ell} described in the K. Diethelm paper
/// referenced in the body of thesis.
/// </summary>
public static double L(int j, int ell, int p, double[] x, double[] y, double alpha,
    ForcingFunction f, double y_p_1)
{
        double value = 0;
        for (int k = (ell - 1)*p + 1; k <= j; k++) {
                value += a(j-k, alpha) * f(x[k], y[k]);
        }
        value += f(x[j+1], y_p_1) / SpecialFunctions.Gamma(alpha + 2);
        return value;
}
/// <summary>
/// This does the actual parallel computation for the method.
/// This is done by setting up a series of tasks with a carefully defined
/// continuation / dependency structure which ensures that computations which
/// can run in parallel are allowed to, and ones which are dependent on other
/// computations run in the right order. For a full description of the dependency
    structure
/// see the body of the thesis.
/// </summary>
/// <param name="alpha">The order of differentiation.</param>
/// <param name="y_0_diffs">An array containing the initial conditions in order of
    increasing
/// differentiation order.</param>
/// <param name="T">The last time to compute to.</param>
/// <param name="N">The number of time steps to use.</param>
/// <param name="p">Task granularity. This essentially defined the maximum level or
    concurrency.</param>
/// <param name="f">The right hand side of the differential equation.</param>
public static double[] Compute(double alpha, double[] y_0_diffs, double T, int N,
    int p, ForcingFunction f)
{
        double[] x = new double[N];
        double[] y = new double[N];
        double[] y_p = new double[N];
```

```
                //Drops in the 0th order initial condition.
                y [0] = y_0_diffs [0];
                //Calculates the time step.
                double h = T / N;
                //Sets up all the x values.
                for (int i = 0; i < N; i++)
                {
                        x [i] = h * i;
                }
                //Compute each block
                for (int ell = 1; ell <= Math.Ceiling ((double)N / (double)p); ell++) {
                        Task<double> taskSum_p = null;
                        Task<double> taskSum = null;
                        //Compute each variable in each block.
                        for (int i = 0; i < p && ((ell - 1) * p) + i < N - 1; i++) {
                                //Calculate the j (index) for this variable.
                                int j = ((ell - 1) * p) + i;
                                //Setup the task dependency structure and set each task
                                    running.
                                Task<double> taskI = Task.Factory.StartNew (() => I_1 (j,
                                    alpha, y_0_diffs, x));

                                Task<double> taskH_p = Task.Factory.StartNew (() => H_p (j,
                                    ell, p, x, y, alpha, f));
                                Task<double> taskH = Task.Factory.StartNew (() => H (j, ell,
                                    p, x, y, alpha, f));
                                Task<double> taskL_p = null;
                                if (taskSum != null) {
                                        taskL_p = taskSum.ContinueWith ((t) => L_p (j, ell, p,
                                            x, y, alpha, f));
                                } else {
                                        taskL_p = Task.Factory.StartNew (() => L_p (j, ell, p,
                                            x, y, alpha, f));
                                }
                                taskSum_p = Task.Factory.ContinueWhenAll(new [] { taskL_p,
                                    taskH_p, taskI }, (ts) => y_p[j + 1] = taskI.Result +
                                    Math.Pow(h, alpha) * ( taskH_p.Result + taskL_p.Result ) );
                                Task<double> taskL = taskSum_p.ContinueWith ((t) => L (j, ell,
                                    p, x, y, alpha, f, y_p [j + 1]));
                                taskSum = Task.Factory.ContinueWhenAll(new [] { taskH, taskL,
                                    taskI }, (ts) => y[j+1] = taskI.Result + Math.Pow(h,
                                    alpha) * (taskH.Result + taskL.Result ));
                        }
                        // Wait for the block to complete.
                        if (taskSum != null) {
                                taskSum.Wait ();
                        }
                }
                //Return the solution.
                return y;
        }


    }
}
```

7.1.2 SpecialFunctions.cs

```
using System;

namespace FDE_Solver
{
	/// <summary>
	/// Provides special functions that are not available
	/// in System.Math
	/// </summary>
	public class SpecialFunctions
	{
		/// <summary>
		/// Gamma the specified z.
		/// This uses the Lanczos approximation and is only valid
		/// for positive real values of z. This code is essentially
		/// a translation of a python implementation available
		/// at http://en.wikipedia.org/wiki/Lanczos_approximation
		/// on 22nd July 2014
		/// </summary>
		/// <param name="z">The z value.</param>
		public static double Gamma(double z)
		{
			double g = 7;
			double[] p = new double[] { 0.99999999999980993, 676.5203681218851,
			    -1259.1392167224028,
                                                                771.32342877765313,
                                                                    -176.61502916214059,
                                                                    12.507343278686905,
				-0.13857109526572012, 9.9843695780195716e-6, 1.5056327351493116e-7 };
			if (z < 0.5) {
				return Math.PI / (Math.Sin (Math.PI * z) * Gamma (1 - z));
			} else {
				z -= 1;
				double x = p [0];
				for (int i = 1; i < g + 2; i++)
				{
					x += p [i] / (z + i);
				}
				double t = z + g + 0.5;
				return Math.Sqrt (2 * Math.PI) * Math.Pow (t, z + 0.5) * Math.Exp
				    (-t) * x;
			}
		}
		/// <summary>
		/// Calculates the factorial of k.
		/// One could use the gamma function above but it does have slight inaccuracies
		/// so the factorial function has also been provided which returns an integer.
		/// </summary>
		/// <param name="k">The value to take the factorial of.</param>
		public static int Factorial(int k)
		{
			int value = 1;
			for (int i = 1; i <= k; i++) {
				value *= i;
			}
```

```
                    return value;
                }
        }
}
```

7.2 CUDA C/C++ Implementation of AMB FDE Solver

7.2.1 FDE_Solver.cu

```
#include <iostream>
#include <ctime>

// This is the size of the memory that we will have in each thread block
// Currently thread blocks can be at most 1024 in size so we allocate double
// that for good measure.
#define SHARED_MEMORY_SIZE 2048

using namespace std;

// The RHS of the differential equation
__device__ float f(float x, float y) {
    return -y;
}

// This uses the Lanczos approximation and is only valid
// for positive real values of z. This code is essentially
// a translation of a python implementation available
// at http://en.wikipedia.org/wiki/Lanczos_approximation
// on 22nd July 2014
__device__ float Gamma(float z) {
    float g = 7;
    float p [] = { 0.99999999999980993, 676.5203681218851, -1259.1392167224028,
                                771.32342877765313, -176.61502916214059, 12.507343278686905,
                                -0.13857109526572012, 9.9843695780195716e-6, 1.5056327351493116e-7
                                    };
        float result = 0;
        if (z < 0.5) {
            result = M_PI / (sin(M_PI * z) * Gamma(1-z));
        } else {
            z -= 1;
            float x = p[0];
            for (int i = 1; i < g + 2; i++) {
                x += p[i] / (z +i);
            }
            float t = z + g + 0.5;
            result = sqrt(2 * M_PI) * pow(t, z + 0.5f) * exp(-t) * x;
        }
        return result;
}

// Calculates the factorial of an integer
__device__ int Factorial(int k) {
```

```
    int value = 1;
    for (int i = 1; i <= k; i++) {
        value *= i;
    }
    return value;
}

// This calculates the a coefficient described in the K. Diethelm paper
// referenced in the body of thesis.
__device__ float a(int mu, float alpha) {
    return (pow((float)mu + 2, alpha + 1) - 2 * pow((float)mu + 1, alpha + 1) + pow((float)mu,
        alpha + 1))/Gamma(alpha + 2);
}

// This calculates the b coefficient described in the K. Diethelm paper
// referenced in the body of thesis.
__device__ float b(int mu, float alpha) {
    return (pow ((float)mu + 1, alpha) - pow ((float)mu, alpha)) / Gamma (alpha + 1);
}

// This calculates the c coefficient described in the K. Diethelm paper
// referenced in the body of thesis.
__device__ float c(int mu, float alpha) {
    return (pow((float)mu, alpha+1) - (mu - alpha)*pow((float)mu + 1, alpha)) / Gamma(alpha + 2);
}

// This calculates the sum R in parallel on the GPU. The sum R is discussed in the body of the
// thesis.
__global__ void calcR(int j, float* x, float* y, float alpha, int opsPerThread, float* R) {
    __shared__ float temp[SHARED_MEMORY_SIZE];
    int i                       = (blockDim.x * blockIdx.x + threadIdx.x) * opsPerThread;
    float sum             = 0;
        for (int l = 0; l < opsPerThread && i + l <= j; ++l) {
               sum              += b(j - (i + l), alpha) * f(x[i+l], y[i+l]);
        }
        temp[threadIdx.x] = sum;
        __syncthreads();
    if ( 0 == threadIdx.x ) {
        sum           = 0;
        for (int k    = 0; k < blockDim.x; ++k) {
            sum       += temp[k];
        }
        atomicAdd( &R[j], sum );
    }
}


// This calculates the sum S in parallel on the GPU. The sum R is discussed in the body of the
// thesis.
__global__ void calcS(int j, float* x, float* y, float alpha, int opsPerThread, float* S) {
    __shared__ float temp[SHARED_MEMORY_SIZE];
    int i                       = (blockDim.x * blockIdx.x + threadIdx.x) * opsPerThread;
    float sum             = 0;
        for (int l = 0; l < opsPerThread && i + l <= j; ++l) {
               sum              += a(j - (i + l), alpha) * f(x[i+l], y[i+l]);
        }
```

```
        temp[threadIdx.x] = sum;
        __syncthreads();
    if ( 0 == threadIdx.x ) {
        sum            = 0;
        for (int k     = 0; k < blockDim.x; ++k) {
            sum        += temp[k];
        }
        atomicAdd( &S[j], sum );
    }
}

// This is used to initialize the very first value in the solution array
// from the initial conditions.
__global__ void setY0(float* y, float y0) {
    y[0] = y0;
}

// This is used to initialize the x array with the correct grid points.
// This will execute in parallel
__global__ void initialize_x(float* x, float h, int N, int opsPerThread) {
    int j = (blockIdx.x * blockDim.x + threadIdx.x) * opsPerThread;
    for (int i = 0; i < opsPerThread && i + j < N; i++) {
        x[j + i] = (j + i) * h;
    }
}

// This calculate the predictor value from the sums already calculated.
// This does not run in parallel and should be called with <<< 1, 1 >>>
__global__ void calcYp(int j, float* I, float* R, float* Yp, float h, float alpha) {
    Yp[j+1] = I[j+1] + pow(h, alpha) * R[j];
}

// This calculate the y value from the sums already calculated.
// This does not run in parallel and should be called with <<< 1, 1 >>>
__global__ void calcY(int j, float* I, float* S, float* Yp, float* y, float* x, float h, float
    alpha) {
    y[j+1] = I[j+1] + pow(h, alpha) * ( c(j, alpha) * f(x[0], y[0]) + S[j] + f(x[j+1], Yp[j+1]) /
        Gamma(alpha + 2) );
}


// This calculates the I sum discussed in the body of the thesis. The entirety of
// the I sums can be precalculated and so that is what this does. This does run
// in parallel with as many threads as there are steps.
__global__ void initialize_I(float* I, float* x, float* y_0_diffs, float alpha, int N) {
        float sum = 0;
        int j = blockIdx.x * blockDim.x + threadIdx.x;
    if (j < N) {
        for (int k = 0; k <= (ceil(alpha) - 1); k++) {
                    sum += pow(x[j], (float)k) / Factorial(k) * y_0_diffs[k];
            }
            I[j] = sum;
    }
}

// This sets up the actual computations on the graphics card and launches them.
```

```
// N                 - Number of steps
// T                 - Maximum value of x
// y_0_diffs   - The initial conditions
// y                 - The vector into which the solution should be copied. (This needs to be
//                           malloced out by the caller of this function.
// opsPerThread - Number of operations per thread to actually perform.
void compute(int N, float alpha, float T, float* y_0_diffs, float* y, int opsPerThread){
       float*              dev_x                       = NULL;
       float*              dev_y                       = NULL;
       float*              dev_R                       = NULL;
       float*              dev_S                       = NULL;
       float*              dev_y_0_diffs        = NULL;
       float*              dev_y_p                     = NULL;
       float*              dev_I                       = NULL;
       cudaDeviceProp* props                   = NULL;
       float               h                                 = 0;
       int                 size                        = 0;
       int                 size_ics                    = 0;
       int                 blocks                      = 0;
       float               blocksf                     = 0;
       int                 threads                     = 0;
       float               threadsf                    = 0;


       // The amount of space needed for the main arrays
       size                = sizeof(float) * N;

       // The amount of space needed for the initial conditions
       size_ics            = sizeof(float) * ceil(alpha);

       // Allocate memory on device for computations
       cudaMalloc( (void**)&dev_x, size );
       cudaMalloc( (void**)&dev_y, size );
       cudaMalloc( (void**)&dev_R, size );
       cudaMalloc( (void**)&dev_S, size );
       cudaMalloc( (void**)&dev_I, size );
       cudaMalloc( (void**)&dev_y_p, size );
       cudaMalloc( (void**)&dev_y_0_diffs, size_ics );

       // Set the device memory up so that it is otherwise set to 0
       cudaMemset( dev_y, 0, size );
       cudaMemset( dev_y_p, 0, size );
       cudaMemset( dev_R, 0, size );
       cudaMemset( dev_S, 0, size );

       // Move the 0th order initial condition across.
       setY0<<< 1, 1 >>>( dev_y, y_0_diffs[0] );

       // Move the initial conditions across.
       cudaMemcpy( dev_y_0_diffs, y_0_diffs, size_ics, cudaMemcpyHostToDevice );

       // Query device capabilities.
       props               = (cudaDeviceProp*) malloc( sizeof(cudaDeviceProp) );
       cudaGetDeviceProperties( props, 0 ); // Assume that we will always use device 1.

       // initialize the x vector
```

```
        h                           = T / N;
        blocksf             = (float) N / (float) (props->maxThreadsPerBlock * opsPerThread);
        blocks              = ceil(blocksf);
        threads             = props->maxThreadsPerBlock;
        if (1 == blocks) {
                threadsf    = (float) N / (float)opsPerThread;
                threads     = ceil(threadsf);
        }
        initialize_x<<< blocks, threads >>>(dev_x, h, N, opsPerThread);

        // initialize the I vector
        blocksf             = (float) N / (float) props->maxThreadsPerBlock;
        blocks              = ceil(blocksf);
        if (1 == blocks) {
                threads     = N;
        } else {
                threads     = props->maxThreadsPerBlock;
        }
        initialize_I<<< blocks, threads >>>(dev_I, dev_x, dev_y_0_diffs, alpha, N);

        // Perform the actual calculations.
        for (int j = 0; j < N; ++j) {
                blocksf             = (float) ( j + 1 )/ (float) (props->maxThreadsPerBlock *
                    opsPerThread);
                blocks              = ceil(blocksf);
                if (1 == blocks) {
                        threadsf    = (float) j / (float)opsPerThread;
                        threads     = ceil(threadsf) + 1;
                } else {
                        threads = props->maxThreadsPerBlock;
                }
                calcR<<< blocks, threads >>>(j, dev_x, dev_y, alpha, opsPerThread, dev_R);
                calcYp<<< 1, 1 >>>(j, dev_I, dev_R, dev_y_p, h, alpha);
                calcS<<< blocks, threads >>>(j, dev_x, dev_y, alpha, opsPerThread, dev_S);
                calcY<<< 1, 1 >>>(j, dev_I, dev_S, dev_y_p, dev_y, dev_x, h, alpha);
        }

        // Copy y vector back onto host memory
        cudaMemcpy( y, dev_R, size, cudaMemcpyDeviceToHost );

        // Free up all resources that were used.
        cudaFree( dev_x );
        cudaFree( dev_y );
        cudaFree( dev_R );
        cudaFree( dev_S );
        cudaFree( dev_I );
        cudaFree( dev_y_0_diffs );
        cudaFree( dev_y_p );

        // Free up all space used on the host.
        free( props );
}

int main( void ) {
        // This code will benchmark the performance of this solver with different
        // paramters for the number of steps and number of operations per thread.
```

```
        float      y_0_diffs[]              = { 1 };
        int        N                               = 10000;
        int        T                               = 10;
        clock_t    start             = 0;
        clock_t    end               = 0;
        double     duration          = 0;
        for (int i = 0; i <= 50; i+=5) {
               // Need to make sure that the correct ammount of memory is available.
               int        size                     = N * i * sizeof( float );
               float*     y                    = (float*) malloc( size );
           for (int j = 1; j <= 20; ++j) {
               start          = clock();
               compute(N * i, 0.5f, T, y_0_diffs, y, j );
               end            = clock();
               duration       = (end - start) / (double) CLOCKS_PER_SEC;
               cout << duration << "\t" << flush;
           }
           free(y);
           cout << "\n";
        }
    return 0;
}
```

8 Appendix B

8.1 Runtimes for C# implementation described in section 5.1.4

N \ p	1	2	3	4	5	5	7	8	9	10	11	12
1000	0.4070868	0.2362202	0.1191832	0.1293034	0.0938632	0.1450464	0.1427508	0.1377472	0.133987	0.125388	0.1216968	0.1200818
2000	1.0937832	0.558851	0.556602	0.46559	0.430715	0.403747	0.4425184	0.4152856	0.4051934	0.3774052	0.3794404	0.3809334
3000	1.9052022	1.4103498	1.0329988	0.895754	0.883083	0.8648174	0.8972584	0.820875	0.813499	0.7662662	0.7725402	0.7604456
4000	3.6316786	2.5111496	1.9091866	1.5481538	1.4340788	1.4993042	1.5020828	1.3928906	1.3221538	1.27441	1.2869578	1.263289
5000	6.2298358	3.7344554	2.9771892	2.3199244	2.1266484	2.220891	2.2675012	2.068062	1.980582	1.9098458	1.921069	1.9492734
6000	7.4398042	4.9997078	4.135724	3.1844926	2.9543306	3.1121518	3.1542058	2.8458522	2.7509526	2.6820768	2.6935614	2.6925528
7000	10.055285	6.8022498	5.2268638	4.3256554	3.9238724	4.0993682	4.2330766	3.845559	3.6778138	3.5685162	3.5791386	3.5445644
8000	12.8123234	7.6822148	7.023647	5.5118882	4.962013	5.1901768	5.3650102	4.9373976	4.7257116	4.549266	4.6082242	4.571809
9000	16.2015754	10.5661146	8.3860596	6.959551	6.2022426	6.3978686	6.6804982	6.1248574	5.9008362	5.6928986	5.8067794	5.600407
10000	22.8870988	12.3470092	10.2837944	8.575945	7.5265192	7.7042254	8.1302114	7.4418008	7.2273678	6.9739174	6.9767076	6.7921516

5

8.2 Runtimes for CUDA implementation described in section 5.1.7

N \ opsPerThread	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
50,000	4.87108	3.32297	3.37962	3.46946	3.64845	3.87464	4.13553	4.43697	4.57786	4.71845	4.85645	4.99309	5.12561	5.2565	5.38823	5.58507	5.87955	6.1915	6.5155	6.85466
100,000	13.7332	9.2564	8.29588	7.51481	7.89867	8.36102	8.77676	9.19788	9.52294	9.85837	10.1958	10.5321	10.8622	11.1902	11.5218	11.9984	12.7003	13.4505	14.2386	15.069
150,000	27.9077	17.7804	14.7961	14.2085	13.8015	13.4937	14.4337	15.507	15.7728	15.9378	16.0039	16.1438	16.6539	17.1985	17.7416	18.5025	19.609	20.7922	22.0386	23.3795
200,000	47.0168	28.939	23.8399	21.6257	21.6106	21.677	21.8518	22.1316	22.628	23.2031	23.6663	24.039	24.3042	24.5237	24.7081	25.3161	26.8756	28.4815	30.2146	32.0608
250,000	71.0519	42.6913	34.5362	31.5671	30.0385	30.939	31.7693	32.404	31.6023	30.755	31.4151	32.2341	32.9353	33.7153	34.6383	35.7562	38.0052	39.8973	41.2286	42.9506
300,000	100.033	59.0803	46.8046	42.238	41.1102	40.6073	42.502	44.0988	43.1857	42.2975	41.477	40.7721	41.6668	43.0575	44.9286	46.8037	51.3096	54.6343	57.8365	60.6987
350,000	133.784	77.9304	61.372	55.2624	53.0512	53.5426	53.5646	56.3827	56.1513	55.4469	54.2999	53.8824	53.206	53.5035	56.1716	58.2979	64.7001	69.3878	74.0164	78.8297
400,000	172.509	99.4371	77.6313	69.2178	66.592	67.1919	68.302	69.2099	69.3106	69.2914	68.8276	68.4649	69.3828	70.6512	72.3131	71.232	79.6719	84.8636	90.7919	97.2622
450,000	215.96	123.333	95.4081	85.2786	81.7588	81.2605	83.8495	86.0194	82.8356	83.161	83.6226	83.8407	86.6134	90.099	94.5162	91.1048	102.314	105.757	110.677	116.999
500,000	264.41	149.922	115.271	102.267	97.4631	98.4406	99.3867	103.704	100.777	97.5023	98.3143	99.3144	103.798	109.351	117.381	113.081	131.434	135.395	139.066	143.586
550,000	317.531	178.85	136.995	121.337	115.664	116.319	117.559	121.546	119.587	116.542	113.796	115.009	121.197	129.498	141.149	135.374	160.621	167.084	173.385	179.086
600,000	375.574	210.454	160.125	141.451	134.63	134.439	137.721	139.959	138.566	136.459	133.901	131.22	138.704	148.854	164.37	158.017	190.778	200.624	209.304	218.444
650,000	438.333	244.445	185.481	163.521	155.259	155.915	157.971	162.504	157.517	156.625	155.25	152.742	157.018	169.943	189.083	181.18	220.261	233.223	245.357	257.474
700,000	506.486	281.27	212.58	186.674	177.392	177.773	178.82	185.737	179.677	176.794	176.563	175.212	179.988	189.953	212.321	204.599	251.396	267.425	281.956	297.777
750,000	578.928	320.209	241.267	211.658	200.474	200.262	203.419	208.98	204.186	197.723	198.007	197.835	204.805	217.196	239.112	228.087	281.324	300.706	318.765	337.085
800,000	656.638	362.042	271.795	237.863	225.392	225.654	228.412	233.173	229.111	223.378	219.488	220.567	229.216	243.557	267.325	253.588	314.309	336.544	356.23	378.126
850,000	738.779	406.038	304.418	265.773	251.271	251.736	253.47	261.266	253.93	249.597	244.374	243.305	254.169	272.859	301.262	284.64	348.23	372.77	394.878	418.79
900,000	826.216	452.979	338.473	295.148	279.101	278.412	281.179	290.1	279.968	275.995	272.18	267.148	279.3	299.379	331.805	315.486	389.705	413.51	436.016	462.42
950,000	918.002	501.929	374.475	326.098	308.433	308.072	311.171	318.883	310.039	302.488	300.025	295.68	305.017	329.237	366.771	349.544	430.481	458.217	480.702	508.34
1,000,000	1015.2	553.967	412.362	358.399	338.406	337.998	340.872	348.722	340.862	330.045	328.021	325.068	332.821	355.923	398.319	381.349	475.024	507.402	530.949	560.252

6

[5]The times listed are in seconds and are calculated as the average over 5 runs.
[6]The times listed are in seconds and are for one run.

References

[1] S. Abbas, M. Benchohra, and N'Guérékata G.M. *Topics in Fractional Differential Equations*. Springer, 2012.

[2] M. Caputo. Linear models of dissipation whose q is almost frequency independent-ii. *Geophysical Journal of the Royal Astronomical Society*, 13:529–539, November 1967.

[3] M. Caputo and Mainardi F. Linear models of dissipation in anelastic solids. *La Rivista del Nuovo Cimento*, 1:161–198, 1971.

[4] Microsoft Copropration. Task schedulers. `http://msdn.microsoft.com/en-us/library/dd997402%28v=vs.110%29.aspx`. Accessed: 2014-08-01.

[5] K. Diethelm. *The Analysis of Fractional Differential Equations*. Springer, 2010.

[6] K. Diethelm. An efficient parallel algorithm for the numerical solution of fractional differential equations. *Fractional Calculus and Applied Analysis*, 14:475–490, 2011.

[7] K. Diethelm and N.J. Ford. Analysis of fractional differential equations. *Journal of Mathematical Analysis and Applications*, 265:229–248, 2002.

[8] K. Diethelm, N.J. Ford, and A.D. Freed. Detailed error analysis for a fractional adams method. *Numerical Algorithms*, pages 31–52, 2004.

[9] R. Gorenflo and F. Mainardi. *Fractals and Fractional Calculus in Continuum Mechanics*. CISM Courses and Lectures. Springer Verlag, 1997.

[10] M. Harris. Mapping computational concepts to gpus. In M. Pharr, R. Fernando, and T. Sweeney, editors, *GPU Gems 2*. Addison-Wesley Professional, 2005.

[11] N. Heymans and I. Podlubny. Physical interpretation of initial conditions for fractional differential equations with riemann-liouville fractional derivatives. *Rheologica Acta*, 45(5):765–771, 2005.

[12] K.S. Miller and Ross B. *An Introduction to the Fractional Calculus and Fractional Differential Equations*. Wiley, 1993.

[13] V.A. Narayanan and Prabhu K.M.M. The fractional fourier transform: theory, implementation and error analysis. *Microprocessors and Microsystems*, 27:511–521, 2003.

[14] I. Podlubny. *Fractional Differential Equations*. Academic Press, 1999.

[15] R. K. Raina and C. L. Koul. On weyl fractional calculus. *Proceedings of the American Mathematical Society*, 73:188–192, Februrary 1979.

[16] S.G. Samko, A.A. Kilbas, and O.I. Marichev. *Fractional Integrals and Derivatives*. Breach Science Publishers, 1993.

[17] C.C. Tisdell. On the application of sequential and fixed-point methods to fractional differential equations of arbitrary order. *Journal of Integral Equations*, 24, 2012.