Fractional Differential Equations and Numerical Methods

A J Landy

June 22, 2009

Dissertation submitted to the University of Chester for the degree of Master of Sciences (Mathematics) in part fulfilment of award.

Four Module Dissertation.

Abstract

The increasing use of Fractional Calculus demands more accurate and efficient methods for the numerical solution of fractional differential equations. We introduce the concepts of Fractional Calculus and give the definitions of fractional integrals and derivatives in the Riemann-Liouville and Caputo forms. We explore three existing Numerical Methods of solution of Fractional Differential Equations.

1. Diethelm's Backward Difference Form (BDF) method.
2. Lubich's Convolution Quadrature method.
3. Luchko and Diethelm's Operational Calculus (using the Mittag-Leffler function) method.

We present useful recursive expressions we developed to compute the Taylor Series coefficients in the Operational Calculus method. These expressions are used in the calculation of the convolution and starting weights.
We compare their accuracy and performance of the numerical methods, and conclude that the more complex methods produce the more accurate results.

Contents

List of Figures

List of Tables

Acknowledgments

Firstly I would like to thank the University of Chester for enabling me to study mathematics again, many years after I graduated in 1968.

I would also like to thank the lecturers of the taught modules of this degree for their skill, diligence and patience in the face of my demonstrable ignorance.

Turning to this dissertation, my thanks go to Professor Kai Diethelm of the Technical University of Braunschweig, who kindly supplied a copy of one of his papers, and offered suggestions on other sources of information related to fractional calculus.

In the same manner, I extend my thanks to Professor Rudolf Gorenflo, who passed on my request for one of his joint papers with Dr. Yuri Luchko onto Dr. Luchko.

My thanks go to Dr. Yuri Luchko, who in addition to sending me his joint paper with Professor Gorenflo, also kindly sent me some additional papers that he thought would be useful to me.

Finally I would like to thank Professor Neville Ford of the University of Chester for his continual help and support during the writing of this dissertation; his encouraging comments revitalised me during the days when I wondered whether this dissertation would ever be finished.

Chapter 1

Introduction and Preamble

Aim

Fractional Calculus is the evaluation of derivatives and integrals, where the order of the derivative is not an integer. Consequently we must find methods of solving differential and integral equations where the terms are non-integer derivatives or integrals or both. Nowadays fractional calculus includes irrational and complex orders, whereas originally fractional calculus considered only orders in the real line. We shall concentrate in this dissertation on fractional differential equations where the order is a rational number.

In this dissertation we shall :-

- Review some of the existing methods of solving fractional differential equations,
- Develop equations of existing numerical methods for their solution,
- Find solutions to the equations of these numerical methods using the SciLab package,
- Compare the efficiency and accuracy of these numerical methods,
- Analyse and comment on our results.

1 Fractional Calculus - a quick summary

R. Gorenflo and F. Mainardi ([15], page 1) describe *Fractional Calculus* as

> ... the field of mathematical analysis which deals with the investigation and applications of integrals and derivatives of arbitrary order.

Gorenflo and Mainardi ([15], page 2) regard Fractional Calculus as both an old yet a novel topic.
Old because the existence of non-integer orders was recognised at the onset of calculus, and novel due to the increasing interest in the topic over the last 30 years.

Serious interest in fractional calculus took place in the 19th century. The basic principles were established, and by the early years of the new century interest had become so widespread that it could no longer be considered merely an interesting paradox.

During the 20th century more research and investigation took place until by the 1960s mathematicians (and others) started looking towards fractional calculus as a means of solving practical problems.

In June 1974 the first conference on Fractional Calculus was organized by B. Ross and held at the University of New Haven.

With the arrival of significant computing power in the 1980s, much of the work involved in the solution of fractional calculus equations could be delegated to these machines. More people turned to fractional calculus seeking solutions to their practical problems which up to then had been either intractable or impossible.

Fractional Calculus is still considered a minority topic despite its increasing use and its application in fields other than mathematics.

A search of the Amazon.com website in September 2007 identified only 27 books which contained the words “Fractional Calculus” in their titles.

2 Growth of interest in Fractional Calculus

The following bar chart shows the yearly total of papers published in journals or presented at conferences between 1975 and 2006. It is not a comprehensive survey of all papers, more a "snapshot" obtained on 3rd August 2007. The source of the data is Web of Knowledge [17], and the papers were found by doing a search for the words "fractional" and "calculus". No checks have been made for duplication of any items. The papers identified were found in all disciplines; engineering, finance, physics, biology and others as well as mathematics.

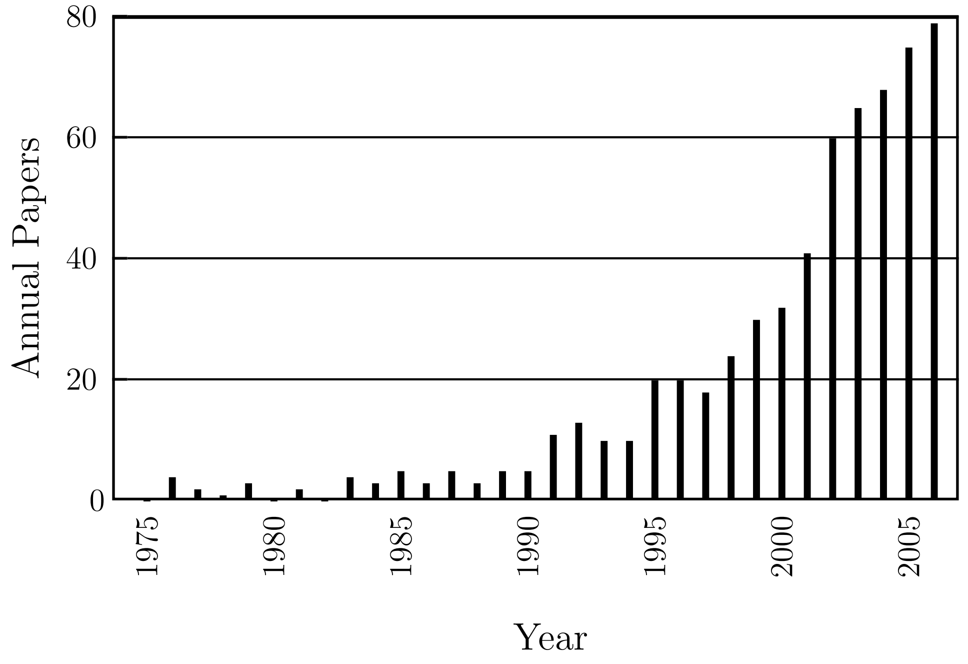

Figure 1.1: Annual Totals

As the annual totals are small in the years 1975 to 1990, the following bar chart shows the running total of papers from 1975. This gives a clearer indication of the growth of the study of fractional calculus.
The data for the bar charts are in Appendix A.

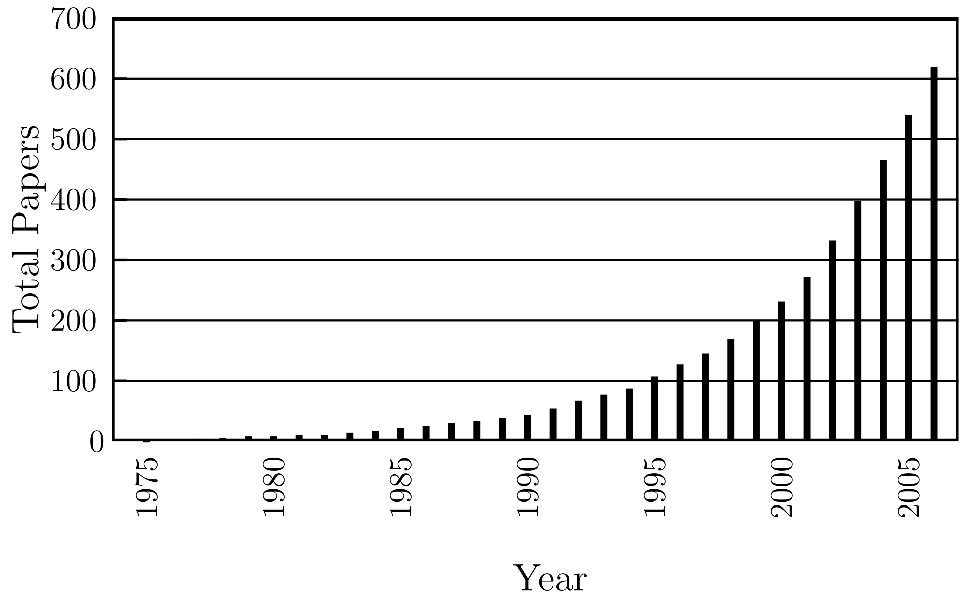

Figure 1.2: Running Annual Totals

In fact the number of published papers is greater than I have identified. J. Choi [3] in 1997 referred to Miller & Ross [30] who stated in 1993 that about 400 papers had been published since 1975.
In comparison, my running total for 1993 is 79 papers.

3 Organization of this document

In Chapter 2 we describe the history and development of Fractional Calculus, and we specify the notation used in this document.
Chapter 3 reviews the tools and methods involved in solving fractional differential equations.
Chapter 4 looks at how we implement our solutions for a fractional differential equation using the existing numerical methods.
Chapter 5 brings together the results, and includes an analysis of them.
In Chapter 6 we summarise the dissertation, comment on the results, and give our conclusions.

Chapter 2

History and Development

Aim

We discuss the history of Fractional Calculus and briefly explore its development. We look at the initial definitions of fractional calculus operators, and how they evolve into more useful statements.

1 Origin and History

After G.W. Leibniz created the notation $\mathrm{d}^n y/\mathrm{d}x^n$, the French mathematician G.F.A. de l'Hôpital enquired "what would be the result if $n = 1/2$?". Leibniz replied that it was a paradox that one day would lead to useful consequences [21]. Thus B. Ross [32] in his article in *Mathematics Magazine* describes the origin of fractional calculus. The history of fractional calculus we give here is based mainly on his article.

In the 18th century L. Euler (1730) realised that his Gamma function has a role in the evaluation of the derivative of the power function for non-integer order (Gorenflo & Mainardi [15]).

According to Gorenflo & Mainardi [15], from the 19th century to the middle of the 20th century mathematicians from P.S. Laplace to W. Feller made important contributions.

From now on the term 'fractional' (a misnomer) will be used to indicate an arbitrary order when used with calculus, derivatives and integrals.

Laplace in 1812 defined a fractional derivative in terms of an integral.

S.F. Lacroix [19] defined the first expression of a fractional derivative in 1819. For $y = x^m$, with m a positive integer, he expressed its nth derivative in terms of the Legendre symbol for Euler's Gamma function. Hence we have

$$\frac{d^n y}{dx^n} = \frac{m!}{(m-n)!}x^{m-n} = \frac{\Gamma(m+1)}{\Gamma(m-n+1)}x^{m-n}.$$

He then set m=1 and n=1/2 thus giving the derivative of order 1/2 of the function x:

$$\frac{d^{1/2} y}{dx^{1/2}} = \frac{\Gamma(2)}{\Gamma(3/2)}x^{1/2}.$$

By expanding $\Gamma(2)$ and $\Gamma(3/2)$, the equation is then solely in terms of $\Gamma(1/2)$. Using Euler's reflection theorem $\Gamma(x)\Gamma(1-x) = \pi/\sin(x)$ where $x < 1$, and setting $x = 1/2$ determines $\Gamma(1/2)$ and thus we have Lacroix's result:

$$\frac{d^{1/2} y}{dx^{1/2}} = \frac{2}{\sqrt{\pi}}\sqrt{x}.$$

J.B.J. Fourier (1822) also commented on derivatives of arbitrary order. But none of these mathematicians gave any application of fractional calculus. It was not until N.H. Abel (1823) [1] considered the tautochrone problem that there was the first practical use of fractional calculus. The tautochrone problem is to determine the shape of a frictionless wire in a vertical plane such that the time for a bead to slide to the lowest point of the wire is independent of the start point. Abel deduced a fractional integral equation for the tautochrone problem. He converted it to a fractional differential equation, then manipulated it so that the fractional differential operator was on a constant. By using the known result of the fractional derivative of a constant with Lacroix's method he found the curve of the tautochrone.

Abel's solution stimulated Joseph Liouville to embark on the first major study of Fractional Calculus. He presented several papers from 1832 onwards ([23] is the first), and gave a definition of a fractional derivative based on an infinite series. A disadvantage of this is that the order of the fractional derivative can only have values for which the series converges. Eventually Liouville came up with another definition by considering a definite integral related to Euler's gamma integral.

$$\int_0^\infty u^{a-1}\mathrm{e}^{-xu}\mathrm{d}u = \frac{1}{x^a}\int_0^\infty t^{a-1}\mathrm{e}^{-t}\mathrm{d}t = \frac{\Gamma(a)}{x^a}$$

Taking the first and last expressions Liouville formed the equation

$$x^{-a} = \frac{1}{\Gamma(a)} \int_0^\infty u^{a-1} \mathrm{e}^{-t} \mathrm{d}t.$$

Liouville now took the νth derivative of both sides of this equation, assumed that $\mathrm{d}^\nu(\mathrm{e}^{ax})/\mathrm{d}x^\nu = a^\nu \mathrm{e}^{ax}$,where ν is a real number greater than zero, and derived his second definition:

$$\frac{\mathrm{d}^\nu}{\mathrm{d}x^\nu} x^{-a} = \frac{(-1)^\nu \Gamma(a+\nu)}{\Gamma(a)} x^{-(a+\nu)}.$$

In another of his papers Liouville attempted to solve an arbitrary order differential equation. This came by analogy with the solution of an ordinary differential equation $\mathrm{d}^n y/\mathrm{d}x^n = 0$, namely $y = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}$. Liouville then argued that the arbitrary order equation $\mathrm{d}^\nu y/\mathrm{d}x^\nu = 0$ should have a corresponding series solution. Liouville ignored the trivial case $x = 0$ which gives a constant as the solution. By analogy the fractional order derivative should be zero. This led to a contradiction. Later W. Center (1850) demonstrated, using the Lacroix method, the remarkable fact that the fractional derivative of a constant is not zero.

The situation was not resolved until the end of the 19th century when mathematicians were able to agree on a robust definition of a fractional derivative as part of the general theory of fractional operators.

A reconciliation of Lacroix's classic-oriented approach with Liouville's methods took place with these new definitions.

In 1847 G.F.B. Riemann as a student formed a different theory of fractional operators. He developed an expression for a fractional integral by a Taylor series generalization, but did not release it. Nevertheless it was published in 1876 after his death. The theory is described by Davis in [5].

Due to the efforts of these early mathematicians, the mathematicians of the late 19th and 20th centuries were able to develop constructive definitions of both the fractional derivative and the fractional integral, and to find methods for the solution of various types of equations.

It is to these developments that we now turn our attention.

2 Development

Before we can continue with the development of approaches for the fractional derivative, we must consider the various definitions of the fractional integral that were established in the latter half of the 19th century.

Riemann & Liouville continuous perspective

An approach, based on A.L. Cauchy's formula for a repeated integral, is stated by Gorenflo & Mainardi([15], page 4) and Podlubny([31], Sections 2.1 & 2.3.1).

For a sufficiently well behaved function $\phi(x)$ where x is a real number in the closed interval $[a, b]$, $(-\infty \leq a < b \leq +\infty)$, the n-fold integral is:

$$J^n_{a+}\phi(x) := \int_a^x \int_a^{x_{n-1}} \ldots \int_a^{x_1} \phi(x_0)\mathrm{d}x_0 \ldots \mathrm{d}x_{n-1}, \qquad \text{where } n \text{ is a positive integer.}$$

Following the argument of Podlubny, we can then express the repeated integral as:

$$J^n_{a+}\phi(x) = \frac{1}{(n-1)!} \int_a^x (x-\xi)^{n-1}\phi(\xi)\mathrm{d}\xi, \qquad \text{where } x > a.$$

The n-fold integral can be extended to positive real values. Noting that $(n-1)! = \Gamma(n)$ and replacing the positive integer n with the positive real number α, where α is arbitrary, then we have the definition of the fractional integral of order α:

$$J^\alpha_{a+}\phi(x) := \frac{1}{\Gamma(\alpha)} \int_a^x (x-\xi)^{\alpha-1}\phi(\xi)\mathrm{d}\xi, \qquad \text{where } x > a.$$

A corresponding form of the fractional integral is:

$$J^\alpha_{b-}\phi(x) := \frac{1}{\Gamma(\alpha)} \int_x^b (\xi-x)^{\alpha-1}\phi(\xi)\mathrm{d}\xi, \qquad \text{where } x < b.$$

Gorenflo & Mainardi ([15], page 4) refer to these fractional integrals J^α_{a+} and J^α_{b-} as *progressive* and *regressive* respectively. Other writers call them *left-side* and *right-side* integrals.

The integral deduced by Riemann was over the set of non-negative real numbers, that is $a = 0$ and $b = \infty$. That determined by Liouville comprised the entire set of real numbers, $a = -\infty$ and $b = \infty$.

Due to the independent formulation of the fractional integral by these two mathematicians, the general term for the above definitions is the *Riemann-Liouville fractional integral.*

At this point we must not forget that it was Abel in 1823 who formed the first fractional integral equation, and who undoubtedly has some claim on nomenclature. Accordingly the first of these, with $a = 0$ and $b = \infty$, is sometimes known as the *Abel-Riemann fractional integral.* Over the years other combinations of values assigned to a and b have led to a number of double-barrelled descriptions.

Thus we can now define the Riemann-Liouville fractional integral.

Definition: *The Riemann-Liouville fractional integral of order α is denoted by the expression*:

$$J_a^\alpha \phi(x) := \frac{1}{\Gamma(\alpha)} \int\limits_a^x (\xi - x)^{\alpha-1} \phi(\xi) \mathrm{d}\xi$$

where x and a are real numbers; $x > a$; and α is a positive real number.

In his article, Ross [32] calls the limits of integration a and x the terminals of integration. Podlubny [31] also uses this expression in his book.

The pioneers of fractional calculus were looking for a general theory of fractional operators. They were unable to do so because they were unable to determine the operations of fractional calculus in the complex region. H. Laurent (1884) eventually came to their rescue when his work in the complex plane revealed a valid solution [20].

H.T. Davis [5], in his massive opus on linear operators, invented the notation ${}_aD_x^{-\alpha} f(x)$ to denote the integration of arbitrary order α of the function $f(x)$; also ${}_aD_x^{\alpha} f(x)$ to denote the differentiation of arbitrary order α. This notation is only used in the requirements stated below. We will address the issues of notation at the end of this chapter, as there appears to be no standard usage of symbols representing integration and differentiation.

In his article in *Mathematics Magazine*, Ross [32] stated the requirements that the early mathematicians were looking for.

1. If $f(x)$ is an analytic function, the fractional derivative ${}_aD_x^{\alpha} f(x)$ must be an analytic function both of the variable x and of the order α.

2. The operation ${}_aD_x^\alpha f(x)$ must produce the same result as ordinary differentiation when α is a positive integer (${}_aD_x^n f(x) = f^{(n)}(x)$ for $n \geq 0$) and the same result as ordinary n-fold integration when α is a negative integer. Moreover, ${}_aD_x^{-n} f(x)$ must vanish together with all its $n-1$ derivatives at $x = a$, the lower terminal of integration.

3. The fractionals must be linear.

4. The operation of order 0 must leave the function unchanged: ${}_aD_x^0 f = f$.

5. The law of exponents must hold for for integration of arbitrary order: ${}_aD_x^{-\alpha}({}_aD_x^{-\beta} f) = {}_aD_x^{-\alpha-\beta} f$

In 1884 H. Laurent [20] created a definition of the fractional integral in his paper. He advanced the themes of what is now considered the foundation of fractional calculus. The proof is long and will not be given here, but we state a summary of it. Taking Cauchy's integral formula in the complex plane for analytic functions, he generalized it to work for arbitrary values. Generalizing the Cauchy formula produces a branch point instead of a pole. By using a point on the real axis, and a branch cut along the real axis, Laurent was able to create a loop (Laurent's loop) which formed a contour. He was then able to integrate around the contour by using the upper and lower edges of the branch cut with a circle around the designated point. The Cauchy integral was then expressed as the sum of 3 integrals, which he manipulated to produce a single integral. Letting the radius of the circle tend to zero, the branch cut is formed, and the Laurent integral is produced. The Laurent integral, identical to the Riemann-Liouville definition given above, using Davis's notation is:

$$ {}_aD_x^{-\alpha} f(x) = \frac{1}{\Gamma(\alpha)} \int_a^x (x-\tau)^{\alpha-1} f(\tau) \mathrm{d}\tau, \qquad \text{where } \alpha > 0 $$

It might be thought to obtain the fractional derivative by replacing α with $-\alpha$, but this leads to a divergent integral. In addition, Gorenflo & Mainardi([15], page 20) argue that the derivative and integral operators are not inverse, even for integer orders. The way to obtain a definition of the fractional derivative ${}_aD_x^\alpha f(x)$ is to integrate enough times so that the integrable function is m-times differentiable, where $m - 1 < \alpha \leq m$.

The Riemann-Liouville fractional derivative of order α is defined as the left inverse (but not the right inverse) of the Riemann-Liouville fractional integral.

$$D^\alpha J^\alpha = \mathbf{I} \qquad \text{where } \mathbf{I} \text{ is the identity operator and } \alpha > 0$$

If we consider just the integral and differential operators then $D^m J^m = \mathbf{I}$, where m is a positive integer and $m-1 < \alpha \leq m$ then

$$D^m J^m = D^m(J^{m-\alpha} J^\alpha) = (D^m J^{m-\alpha})J^\alpha.$$

So by comparison with the fractional derivative definition:

$$D^\alpha := D^m J^{m-\alpha}$$

Definition: *The Riemann-Liouville fractional derivative of order α is denoted by the expression*:

$${}_aD_x^\alpha f(x) := \begin{cases} \dfrac{1}{\Gamma(m-\alpha)} \dfrac{\mathrm{d}^m}{\mathrm{d}x^m} \displaystyle\int_a^x (x-\tau)^{m-\alpha-1} f(\tau)\,\mathrm{d}\tau & \text{where } m-1 < \alpha < m \\ \dfrac{\mathrm{d}^m}{\mathrm{d}x^m} f(x) & \text{where } \alpha = m \end{cases}$$

where x and a are real numbers; $x > a$; α is a positive real number and m is a positive integer.

The Riemann-Liouville approach described refers to the behaviour of continuous functions.

The next part examines the fractional derivative from a discrete aspect.

Grünwald and Letnikov discrete perspective

A.K. Grünwald (1867) [16] and A.V. Letnikov (1868) [22] independently used a method of backward differences to develop a discrete view of the fractional derivative.
Definition: *The Grünwald-Letnikov fractional derivative of order α is*:

$$D_+^\alpha f(x) = \lim_{h\to 0} \frac{\Delta_h^\alpha f(x)}{h^\alpha} = \lim_{h\to 0} \frac{1}{h^\alpha} \sum_{k=0}^{n} (-1)^k \frac{\Gamma(n+1)}{\Gamma(k+1)\Gamma(n-k+1)} f(x-kh), \qquad a < x < b,$$

where $\Delta_h^\alpha f(x)$ is the backward difference operator.

We recall from L. Debnath's paper ([6], page 3438) the definition of the nth order derivative

$$D^n f(x) = \lim_{h\to 0} \frac{1}{h^n} \Delta_h^n f(x)$$

where the backward difference operator $\Delta_h^n f(x)$ is defined as

$$\Delta_h^n f(x) := \sum_{k=0}^{n} (-1)^k \binom{n}{k} f(x-kh).$$

To see where this definition came from, consider the backward finite difference formed by the step length $h > 0$. Using $T^h = f(x-h)$ as the translation by a step, the backward finite difference of order n is defined as

$$\Delta_h^n f(x) := (T^0 - T^h)^n f(x).$$

Expanding $(T^0 - T^h)^n$ and substituting for T^h then

$$\Delta_h^n f(x) := \sum_{k=0}^{n} (-1)^k \binom{n}{k} f(x-kh).$$

Now the binomial term with n and k can be expressed as

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{\Gamma(n+1)}{\Gamma(k+1)\Gamma(n-k+1)}$$

$$\Delta_h^n f(x) := \sum_{k=0}^{n} (-1)^k \frac{\Gamma(n+1)}{\Gamma(k+1)\Gamma(n-k+1)} f(x-kh)$$

Debnath replaced n by α to generalize the expression and then the Grünwald-Letnikov fractional derivative is

$$D_+^\alpha f(x) = \lim_{h\to 0} \frac{1}{h^\alpha} \sum_{k=0}^{[(x-a)/h]} (-1)^k \frac{\Gamma(\alpha+1)}{\Gamma(k+1)\Gamma(\alpha-k+1)} f(x-kh)$$

within the limits $a < x < b$, and where $[(x-a)/h]$ is the integer part.

Podlubny([31], page 52 et seq.) has taken this definition and by extended mathematics shown that (not using his style)

$$D_+^\alpha f(x) = \sum_{k=0}^{m-1} \frac{f^{(k)}(a)\,(t-a)^{-\alpha+k}}{\Gamma(k-\alpha+1)} + \frac{1}{\Gamma(m-\alpha)} \int\limits_a^x (t-\tau)^{m-1-\alpha} f^{(m)}(\tau) \mathrm{d}\tau$$

for $m-1 < \alpha < m$.

Podlubny([31], Sections 2.2.1 & 2.2.2) has also shown that the Grünwald and Letnikov viewpoint can be used to derive the definition of the fractional integral. In fact he unifies both the derivative and the integral representation in a single expression.

Now Podlubny([31], Section 2.3) has given a connection between the Riemann-Liouville fractional derivative and the Grünwald-Letnikov fractional derivative under certain conditions.

Taking the case for $m-1 < \alpha < m$ for the Riemann-Liouville derivative from equation (2.79) of Podlubny's book (again not using his style)

$${}_aD_x^\alpha f(x) := \frac{1}{\Gamma(m-\alpha)} \frac{\mathrm{d}^m}{\mathrm{d}x^m} \int\limits_a^x (x-\tau)^{m-\alpha-1} f(\tau)\,\mathrm{d}\tau$$

Note: Equation (2.79) in his book, and the first line of the equation at the bottom of the page, have the same printing error. The factor $1/\Gamma(-p+m+1)$ is missing before the differential operator for both equations.

Performing an integration by parts m times, and applying the derivative operator

$${}_aD_x^\alpha f(x) = \sum_0^{m-1} \frac{f^{(k)}(a)(x-a)^{-\alpha+k}}{\Gamma(-\alpha+k+1)} + \frac{1}{\Gamma(m-\alpha)} \int\limits_a^x (x-\tau)^{m-1-\alpha} f^{(m)}(\tau) \mathrm{d}\tau \qquad (2.1)$$

The right-hand side of this equation is the Grünwald-Letnikov fractional derivative definition.

What are the conditions that these two definitions are equivalent?

1. The functions $f(x)$ exist and are continuous in $[a, x]$.

2. The functions $f(x)$ are m-times continuously differentiable in $[a, x]$.

3. The m derivatives of $f(x)$ exist and are continuous in $[a, x]$.

This equivalence means that we can treat a discrete problem for fractional derivatives as a continuous problem, and solve it using the definition of the Riemann-Liouville fractional derivative.
Practical applications of fractional calculus can use this equivalence, for example, to monitor and model physical processes. It is usual to sample a physical parameter in industrial or scientific processes by taking a reading at a fixed sample rate. These readings are usually read by a computer, or more correctly by the computer obtaining the reading from a dedicated electronic interface which maintains the current value of the parameter. As the parameter value is read at a fixed rate this is discrete sampling (for example once a second). By the reasoning above this discrete sampling method can be represented by a continuous one, which is easier to evaluate, under the assumptions detailed above. Using this method the parameter value can be monitored and appropriate actions taken providing the parameter is sampled at a sufficiently fast rate. If the process is being modelled then the values of the physical parameters can be used to determine the behaviour of the process and choices made to determine the future behaviour of the physical process.

Physical processes can only be monitored if their start conditions or initial values are known. These can be either deduced from the model, or from the first readings taken by the computer, or empirically from some known behaviour of the parameter. Fractional differential equations based on the Riemann-Liouville derivative suffer from a weakness in terms of initial values. To solve an initial value problem for Riemann-Liouville fractional differential equation we must know the initial value of each separate fractional derivative.

Caputo derived another form of the fractional derivative to represent the continuous viewpoint of Riemann-Liouville. In this form he showed that it is possible to represent the initial value problem in terms of integer order derivatives for the initial values.

Gorenflo and Mainardi [14] define the Caputo fractional derivative $D_*^\alpha f(x) := J^{m-\alpha} D^m f(x)$ for $m - 1 < \alpha \leq m$. Writing this out fully, we can define the Caputo fractional derivative of order α.

Definition: *The Caputo fractional derivative of order α is*:

$$D_*^\alpha f(x) := \begin{cases} \dfrac{1}{\Gamma(m-\alpha)} \displaystyle\int_0^x (x-\tau)^{m-\alpha-1} f^{(m)}(\tau)\,\mathrm{d}\tau & \text{where } m-1<\alpha<m \\ \dfrac{\mathrm{d}^m}{\mathrm{d}x^m} f(x) & \text{where } \alpha = m. \end{cases}$$

Luchko & Gorenflo have shown a relationship between the Caputo derivative and the Riemann-Liouville derivative.

Now it can be seen that the above definition (for $m-1<\alpha\le m$) is also in the equation (2.1) for Reimann-Liouville derivative, as the second term.
Thus we can now state:

$${}_aD_x^\alpha f(x) = D_*^\alpha f(x) + \sum_0^{m-1} \frac{f^{(k)}(a)(x-a)^{-\alpha+k}}{\Gamma(-\alpha+k+1)}.$$

As we have already stated that the Riemann-Liouville and Grünwald-Letnikov definitions are equivalent (under certain conditions), we can now state

$$D_+^\alpha f(x) = {}_aD_x^\alpha f(x) = D_*^\alpha f(x) + \sum_0^{m-1} \frac{f^{(k)}(a)(x-a)^{-\alpha+k}}{\Gamma(-\alpha+k+1)}.$$

If we now choose the initial condition $f(a)$ is a constant, then the terms containing $f^{(m)}(a)$ disappear and all three definitions are the same.

Thus we can solve an initial value problem practically using the Caputo form of the derivative, express it under certain conditions with Riemann-Liouville form, and finally use the Grünwald-Letnikov form to find solutions where there is no exact solution.

To solve fractional differential equations various tools and methods are used, and it is to these we now give our attention in the next chapter.

Before turning to these, we need to agree on the notation to be used throughout the rest of this document.

3 Notation

As we have seen in this chapter, and from reading other papers and books, there is no common agreement on the symbology used in fractional calculus. Some authors use subscripts and superscripts, both before and after an operator symbol to give a unique means of identification, which can lead to unwieldy symbol structures. We give in the table below the symbols and their definitions that we will us from now on, irrespective of any use we have made up to this point. Some of these are taken from Diethelm's 'book' [9, Appendix A]. We intend to follow Gorenflo and Mainardi's practice [14, p. 232] of not using the $D^{-\alpha}$ symbol to indicate fractional integration.

Δ_h^α	The backward difference operator
D^n	n^{th} order derivative; $n \in \mathbb{N}$
D_{a*}^α	Caputo fractional differential operator; Order $\alpha \in \mathbb{R}_+$; Lower limit of a
D_{a+}^α	Grünwald-Letnikov fractional differential operator; Order $\alpha \in \mathbb{R}_+$; Lower limit of a
D_a^α	Riemann-Liouville fractional differential operator; Order $\alpha \in \mathbb{R}_+$; Lower limit of a
J_a^α	Riemann-Liouville fractional integral operator; Order $\alpha \in \mathbb{R}_+\backslash\mathbb{N}$; Lower limit of a

Chapter 3

Tools and Methods

Aim

The tools and methods used to solve fractional differential equations are examined.

1 Tools

We have already seen how the Gamma function is used in the definition of fractional integrals and derivatives. In this section we will see how other functions are used.

1.1 Beta Function

The Beta Function, derived from the Beta integral with the upper limit set as 1, is defined by the expression:

$$B(p,q) = \int_0^1 u^{p-1}(1-u)^{q-1}\mathrm{d}u.$$

By the convolution property of the Laplace Transform we can show that:

$$B(p,q) = \frac{\Gamma(p)\Gamma(q)}{\Gamma(p+q)}.$$

The function is obviously symmetric in p and q.
This expression is often more convenient in use than the combination of Gamma functions. In addition the Beta function is used to determine fractional integrals without having to directly evaluate the integral.

Consider the expression below as a convolution integral

$$G(x) = \int_0^x u^p (x-u)^q \,\mathrm{d}u.$$

By taking the Laplace transform and using the convolution property, and then taking the inverse Laplace transform we obtain

$$G(x) = \frac{\Gamma(p+1)\Gamma(q+1)}{\Gamma(p+q+2)} x^{p+q+1}.$$

1.2 Mittag-Leffler Function

The Mittag-Leffler function is a generalization of the exponential function e^z. It takes 2 forms; a one parameter form defined by

$$E_\alpha(z) = \sum_{k=0}^{\infty} \frac{z^k}{\Gamma(\alpha k + 1)}$$

and a two parameter form defined by

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

We know that e^z is important in the solution of integer order differential equations. By extension the Mittag-Leffler function takes a similar role in fractional calculus. In particular the second form is very useful in fractional calculus in the various transform methods and also in numerical methods. The derivative of the Mittag-Leffler function has been computed by Gorenflo, Loutchko and Luchko [33]. In addition Diethelm and Ford [11] state the jth derivative of the Mittag-Leffler function as

$$E^{(j)}_{\alpha,\beta}(z) = \sum_{k=0}^{\infty} \frac{(k+j)! z^k}{k! \Gamma(\alpha k + \alpha j + \beta)}.$$

1.3 Laplace Transform

The Laplace transform of a function $f(t)$, $(t > 0)$, is defined as

$$\mathcal{L}\left\{f(t)\right\} = F(s) = \int_0^\infty e^{-st} f(t) dt$$

where s can be either real or complex.
In this document we shall only consider real values of s.
The Laplace transform of $f(t)$ exists if the integral converges for some value of s.
Correspondingly $f(t)$ is called the inverse Laplace transform of $F(s)$ which we denote as

$$f(t) = \mathcal{L}^{-1}\left\{F(s)\right\}.$$

We can recognise the convolution property of the Laplace transform as

$$f(t) * g(t) = \int_0^t f(u)g(t-u)du = \mathcal{L}^{-1}\left\{F(s)G(s)\right\} = \mathcal{L}^{-1}\left\{F(s)\right\}\mathcal{L}^{-1}\left\{G(s)\right\}$$

where $\mathcal{L}^{-1}\left\{F(s)\right\} = f(t)$ and $\mathcal{L}^{-1}\left\{G(s)\right\} = g(t)$. Consequently we can also write

$$\mathcal{L}\left\{f(t) * g(t)\right\} = F(s)G(s).$$

Regarding the n^{th} order derivative of a function $f(t)$, where n is an integer, there is the well known expression

$$\mathcal{L}\left\{f^{(n)}(t)\right\} = s^nF(s) - \sum_{k=0}^{n-1} s^{n-k-1}f^{(k)}(0) = s^nF(s) - \sum_{k=0}^{n-1} s^k f^{(n-k-1)}(0).$$

All of these definitions and properties are useful in the evaluation of our fractional integrals.

Podlubny has shown [31, Section 2.8.2] that the Laplace transform of the Riemann-Liouville derivative is

$$\mathcal{L}\left\{D_0^\alpha f(x)\right\} = s^\alpha F(s) - \sum_{k=0}^{n-1} s^k \left[D_0^{\alpha-k-1}f(x)\right]_{t=0}.$$

Additionally, he has also shown that the Laplace transform of the Grünwald-Letnikov derivative is

$$\mathcal{L}\left\{D_{0+}^\alpha f(x)\right\} = s^\alpha F(s).$$

Also that the Laplace transform of the Caputo derivative is

$$\mathcal{L}\left\{D_{0*}^\alpha f(x)\right\} = s^\alpha F(s) - \sum_{k=0}^{n-1} s^{\alpha-k-1}f^{(k)}(0).$$

1.4 Fourier Transform

The Fourier transform of a function $f(t)$ is defined by

$$\mathcal{F}\left\{f(t)\right\} = F_e(w) = \int_{-\infty}^{\infty} e^{-iwt} f(t)dt,$$

where $f(t)$ is a continuous function integrable in the range $(\infty, -\infty)$.

We also have the inverse Fourier transform defined by

$$\mathcal{F}^{-1}\left\{F_e(w)\right\} = f(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} e^{iwt} F_e(w)dw.$$

In a similar manner to the Laplace transform we can specify the convolution property of two functions $f(t)$ and $g(t)$ as

$$f(t) * g(t) = \int_{-\infty}^{\infty} f(t-u)g(t)du.$$

Taking the Fourier transform of the above yields

$$\mathcal{F}\left\{f(t) * g(t)\right\} = \mathcal{F}\left\{f(t)\right\}\mathcal{F}\left\{g(t)\right\}.$$

Podlubny [31, p. 110] states a useful property of the Fourier transform of the derivatives $f^{(n)}(t)$. If the derivatives $f(t),\ f^{(1)}(t),\ \ldots,\ f^{(n-1)}(t)$ vanish as $t \to \pm\infty$, then the Fourier transform of the n^{th} derivative of $f(t)$ is

$$\mathcal{F}\left\{f^{(n)}(t)\right\} = \left(-i\omega\right)^n F_e(\omega),$$

where $F_e(w)$ is the Fourier transform of $f(t)$.

The Fourier transform of the Riemann-Liouville integal with a lower terminal of $-\infty$ is given by Podlubny [31, p. 111] as

$$\mathcal{F}\left\{J^{\alpha}_{-\infty} f(x)\right\} = (-i\omega)^{-\alpha} F_e(w)$$

where $F_e(w)$ is the Fourier transform of $f(x)$.

As $f^{(m)}(x)$ vanish as $x \to -\infty$ then this expression also gives the Fourier transform of the Grünwald-Letnikov integral.

For the Fourier transform of the Caputo fractional derivative with a lower terminal of $-\infty$, Podlubny [31, p. 111] gives it as

$$\mathcal{F}\left\{D^{\alpha}_{-\infty *}f(x)\right\} = (-i\omega)^{\alpha}F_e(w)$$

where $F_e(w)$ is the Fourier transform of $f(x)$.

As $f^{(m)}(x)$ vanish as $x \to -\infty$ then this expression also gives the Fourier transform of the Riemann-Liouville and Grünwald-Letnikov derivatives.

1.5 Green's Function

Green's function has many uses in the solution of integer order differential equations and integral equations. In particular it is used in the solution of initial value or boundary condition problems.

2 Non-Numerical Methods of Solution

2.1 Laplace Transform method

Podlubny [31, Chapter 4] introduces the Laplace transform method in his book. Using the expressions from Section 1.3 of this chapter for the Riemann-Liouville derivative it is possible to solve fractional differential equations with constant coefficients.

Examples from Podlubny's book [31, p.138-140] of ordinary linear fractional differential equations which can be solved using the Laplace transform method are given below.

1. An example with a Riemann-Liouville derivative and an initial value.

$$D_0^{\frac{1}{2}}f(t) + af(t) = 0, \quad t > 0 \quad \left[D^{-\frac{1}{2}}f(t)\right]_{t=0} = C$$

   After applying the Laplace transform, gathering terms, and using the inverse Laplace transform we have the exact solution

$$f(t) = Ct^{-\frac{1}{2}}E_{\frac{1}{2},\frac{1}{2}}(-a\sqrt{t}).$$

2. This non-homogeneous example contains two Riemann-Liouville derivatives.

$$D_0^{Q-1}f(t) + D_0^{q-1}f(t) = h(t) \quad \text{where0 < q < Q < 1.}$$

By the Laplace transform method the solution is

$$f(t) = CG(t) + \int_0^t G(t-\tau)h(\tau)\mathrm{d}\tau,$$

where $C = \left[D_0^{Q-1}f(t) + D_0^{q-1}f(t)\right]_{t=0}$ and $G(t) = t^{Q-1}E_{Q-q}(-t^{Q-q})$.

3. A non-homogeneous example with non-zero initial values.

$$D_0^{\alpha}f(t) - \lambda f(t) = h(t)$$

where $\left[D_0^{\alpha-k}f(t)\right]_{t=0} = b_k, \quad (k = 1, 2, \ldots, n)$, and $n - 1 < \alpha < n$.
Using the Laplace transform method we have the solution

$$f(t) = \sum_{k=1}^{n} b_k t^{\alpha-k}E_{\alpha,\alpha-k+1}(\lambda t^{\alpha}) + \int_0^t (t-\tau)^{\alpha-1}E_{\alpha,\alpha}(\lambda(t-\tau)^{\alpha})h(t)\mathrm{d}\tau.$$

3 Numerical Methods

3.1 Diethelm's backward difference method

Weilbeer [34, Lemma 5.1.5, p. 115] in his thesis of 2005 states an approximation for the Caputo derivative.

$$D_{0*}^{\alpha}y(x) \approx \frac{x^{-\alpha}}{\Gamma(-\alpha)}Q_j[g]$$

where $g(u) = y(x-xu) - T_{m-1}[y;0](x-xu)$ and $T_{m-1}[y;0](x-xu)$ is the Taylor polynomial $((m-1) < \alpha \leq m)$.
This whole procedure was previously described in Diethelm's 'book' [9, p. 101] in 2003, and for this reason the overall method employed is called the Diethelm fractional backward difference method.
Diethelm develops his method by means of dividing the interval of integration using a mesh, and for each subinterval defined by the mesh, constructs a polynomial of degree d. In this

document we will only consider the value $d = 1$, that is a piecewise linear polynomial. The Caputo derivative is defined by Diethelm [9, p. 38] as

$$D_{a*}^{\alpha} y(x) = D_a^{\alpha} \left[y(x) - \Gamma_{m-1} \left[y; a \right] (x) \right].$$

where

$$\Gamma_{m-1} \left[y; a \right] (t) = \sum_{k=0}^{m-1} \frac{y^{(k)} (a)}{k!} (t-a)^k$$

and $(m-1)$ is the degree of the Taylor polynomial, such that $(m-1) < \alpha \leq m$.
Using two lemmas of Diethelm [9, Lemmas 7.1 and 7.2,p. 95] we have

$$D_{a*}^{\alpha} y(x) = \frac{1}{\Gamma(-\alpha)} \int_a^x (x-t)^{-\alpha-1} \left(y(t) - \Gamma_{m-1} \left[y; a \right] (t) \right) \mathrm{d}t.$$

We can now assume the lower limit $a = 0$ without loss of generality. Making the transformation $t = x - xu$ we then have

$$D_{0*}^{\alpha} f(x) = \frac{x^{-\alpha}}{\Gamma(-\alpha)} \int_0^1 u^{-\alpha-1} g(u) \mathrm{d}u$$

where $g(u) = f(x - xu) - \Gamma_{m-1}[f; a](x - xu)$.
We have now reduced our interval of integration to [0,1] and can construct a simpler mesh on this interval. Taking the function $g(u)$ we can approximate it by a function $\hat{g}(u)$ and a d^{th} degree polynomial on the mesh subinterval by defining nodal points within each subinterval of the form $x_{\nu-1} + \mu(x_\nu - x_{\nu-1})/d$, where $\mu = 0, 1, ..., d$. For the case $d = 0$ we only have the node $x_{\nu-1}$. The case $d = 1$ gives 2 nodal points which form a double trapezium inside the subinterval, and is a piecewise linear interpolation.
The required approximation is thus

$$Q_j[g] := \int_0^1 u^{-\alpha-1} \hat{g}(u) \mathrm{d}u$$

The evaluation of each $Q_j[g]$ is required to arrive at our complete approximation within the range $[0, 1]$.
We will only consider the piecewise linear approximation, that is $d = 1$, for the remaining chapters of the dissertation.
As we have $d = 1$ then there are only 2 nodes for each subinterval and we must compute $Q_j[g]$ using this condition. It can be seen that the function $u^{-\alpha-1}$ acts as a weight on the

function $\hat{g}(u)$. For the 2 node case there are 2 weights, each attached to the node either side of the centre point of the subinterval. The lower node for the k^{th} subinterval can be represented by $(k-1)/j$, the centre point by k/j, and the upper node by $(k+1)/j$ where $k = 0, 1, ..., j$. The subintervals can be considered independent for the evaluation of the weights, so the weights of other subintervals do not affect the calculation of the weights of the desired subinterval.

Thus the approximation can be represented by a quadrature formula of a product trapezoidal form

$$Q_j[g] = \sum_{k=0}^{j} \omega_{kj} g(k/j)$$

The weights are determined by evaluating the above integral for with $\hat{g}(u)$ replaced by a function that fairly represents the weight distribution within the subinterval. A fair representation would be to attach a coefficient of 1 at the centre point of the subinterval, and 0 at each end of the subinterval.
To that end, we consider the function $\phi_{kj}(x)$ defined as

$$\phi_{kj}(x) = \begin{cases} jx-(k-1) & \text{for } x \in [(k-1)/j, k/j] \\ k+1-jx & \text{for } x \in [k/j, (k+1)/j] \\ 0 & \text{elsewhere} \end{cases} .$$

This function gives a fair representation as the centre point k/j has a value of 1, and the nodal end points have values of zero.
We need to consider the 4 cases:-

1. The 1^{st} subinterval, when there is no lower node.
2. The general case of the k^{th} subinterval.
3. The last subinterval, when there is no upper node.
4. The case when there is only 1 subinterval, that is $k = j = 1$.

Substituting the values of ϕ_{kj} into the integral in place of $\hat{g}(u)$, and evaluating the integrals is a fairly tedious exercise, with the following result.

The weights are calculated for a piecewise linear interpolation (d=1) by

$$\alpha(\alpha-1)j^{-\alpha}\omega_{kj} = \begin{cases} -1 & \text{for } k=0 \\ \alpha & \text{for } k=j=1 \\ 2k^{1-\alpha}-(k-1)^{1-\alpha}-(k+1)^{1-\alpha} & \text{for } 1 \leq k \leq j-1 \\ (\alpha-1)k^{-\alpha}-(k-1)^{1-\alpha}+k^{1-\alpha} & \text{for } k=j \text{ and } j \geq 2. \end{cases} \tag{3.1}$$

We can now calculate the weights for each subinterval using the above expressions, determine the value of $Q_j[g]$, and thus determine the approximated value of the Caputo derivative.
The above formula is also used by Diethelm [8, Lemma 2.1, p. 3] for the Riemann-Liouville derivative.
Now we are able to consider the fractional differential equation

$$D^{\alpha}_{0*}y(x) = f(x, y(x)). \tag{3.2}$$

Firstly our range of values of α is restricted. Diethelm [9, p. 98] states that α is restrained by the inequality $\alpha < m \leq (d+1)$. As we have $d=1$, and excluding $\alpha=2$ and $\alpha=1$, then we are forced to have $0 < \alpha \leq 2$.
According to Diethelm [9, p. 103,(7.4b)] the initial conditions are

$$\begin{cases} y(0)=y_0 & & \text{if } 0<\alpha<1, \\ y(0)=y_0 & y'(0)=y'_0 & \text{if } 1<\alpha<2 \end{cases}$$

Now we substitute the expression

$$D^{\alpha}_{0*}y(x) = D^{\alpha}_{0}y(x) - D^{\alpha}_{0}\Gamma_{m-1}\left[y;0\right]$$

in the equation and obtain

$$D^{\alpha}_{0}\left[y(x)\right] - D^{\alpha}_{0}\left[\Gamma_{m-1}\left[y;a\right]\right] = f(x,y(x)).$$

The fractional derivative of a Taylor polynomial has already been derived by Diethelm [9, p. 31]. We can now impose our interval method described above on the equation as N intervals of a range $[0,T]$ so that each interval has a width $h=T/N$. Each interval is nominated by $t_j := jh$ where $j=0,1,...,N$. Setting $x=t_j$ and using Diethelms expressions for $D^{\alpha}_{0}[y(x)]$

and $D_0^\alpha \Gamma_{m-1}[y;0]$ we have

$$\begin{aligned} f(t_j, y(t_j) &= D_{0*}^\alpha y(t_j) = D_0^\alpha y(t_j) - D_0^\alpha \Gamma_{m-1}[y;0](t_j) \\ &= \frac{t_j^{-\alpha}}{\Gamma(-\alpha)} \int_0^1 u^{-\alpha-1} y(t_j - t_j u)\, \mathrm{d}u - D_0^\alpha \Gamma_{m-1}[y;0](t_j) \\ &= \frac{t_j^{-\alpha}}{\Gamma(-\alpha)} \int_0^1 u^{-\alpha-1} y(t_j - t_j u)\, \mathrm{d}u - \frac{y_0 t_j^{-\alpha}}{\Gamma(1-\alpha)} - \frac{y_0' t_j^{1-\alpha}}{\Gamma(2-\alpha)} \end{aligned}$$

taking account of $0 < \alpha < m \leq 2$ and the initial conditions.
Setting $y_0' := 0$ allows us to treat both cases of $\alpha < 1$ and $\alpha < 2$. Now we replace the integral with the quadrature expression Q_j, and introduce the quadrature error R_j. So with the notation $g_j(u) := y(t_j - t_j u)$ and $u = k/j$ we have

$$\frac{t_j^{-\alpha}}{\Gamma(-\alpha)} \left(\sum_{k=0}^{j} \alpha_{kj} g_j(k/j) + R_j[g_j] \right) - \frac{y_0 t_j^{-\alpha}}{\Gamma(1-\alpha)} - \frac{y_0' t_j^{1-\alpha}}{\Gamma(2-\alpha)} = f(t_j y(t_j)).$$

There are items in this equation which have unknown values. Firstly $g_j(k/j)$ is unknown except for the case $j = k$. Additionally the quadrature error R_j is unknown. The value of $y(t_j)$ is unknown, so the right hand side of the equation cannot be evaluated. Following the practice of Diethelm, we ignore the quadrature error. Since $y(t_j)$ is unknown, we assume that at least we have approximate values y_ν for $y(t_\nu)$ where $\nu = 0, 1, ..., j-1$. Only $y(t_j)$ now remains unknown, and since we have used y_ν as a replacement for $g_j(k/j)$, the equation then depends on previously determined approximate values, and so yields an approximate solution, which we will designate y_j. Rewriting the equation in terms of the approximate values yields

$$\frac{t_j^{-\alpha}}{\Gamma(-\alpha)} \sum_{k=0}^{j} \alpha_{kj} y_{j-k} - \frac{y_0 t_j^{-\alpha}}{\Gamma(1-\alpha)} - \frac{y_0' t_j^{1-\alpha}}{\Gamma(2-\alpha)} = f(t_j, y_j)$$

Now we have a backward difference relationship for y_j if $f(t_j, y_j)$ is a sensible function. From the initial conditions y_1 can be calculated; indeed if $y_0' = 0$ we can use the formula for $1 < \alpha < 2$ and $0 < \alpha < 1$. Having determined y_1 and knowing y_0 we can successively find $y_2, y_3, ..., y_N$.
For the purposes of evaluation it is useful to substitute $t_j = jh$ and to rewrite the equation as

$$\sum_{k=0}^{j} \beta_{kj} y_{j-k} = j^{-\alpha}(1-\alpha) y_0 + h j^{1-\alpha} y_0' + h^\alpha \Gamma(2-\alpha) f(t_j, y_j)$$

where

$$\beta_{kj} = -\alpha(1-\alpha)j^{-\alpha}\alpha_{kj}$$

is seen to be the expression on the left hand side in (3.1).
If we now take $f(x, y)$ as a simple form

$$f(x, y) = -\lambda y + q(x)$$

with $\lambda > 0$ and $q(x)$ a continuous function then we can evaluate the approximations of $y(x)$ directly. Making the appropriate substitutions for x and $f(x, y)$ then we have

$$\sum_{k=0}^{j} \beta_{kj} y_{j-k} = j^{-\alpha}(1-\alpha)y_0 + hj^{1-\alpha}y_0' + h^\alpha\Gamma(2-\alpha)q(t_j) - h^\alpha\Gamma(2-\alpha)\lambda y_j.$$

Collecting the terms for y_j together and equating to the remaining terms then

$$(h^\alpha\Gamma(2-\alpha)\lambda + \beta_{0j})\, y_j = j^{-\alpha}(1-\alpha)y_0 + hj^{1-\alpha}y_0' + h^\alpha\Gamma(2-\alpha)q(t_j) - \sum_{k=1}^{j}\beta_{kj}y_{j-k} \quad (3.3)$$

The factor applying to y_j is positive as β_{0j} is non-zero from (3.1) for $k = 0$, together with $\lambda > 0$ and $\Gamma(2-\alpha) > 0$. Hence our equation has a unique solution y_j.

3.2 Lubich's convolution quadrature method

Before embarking on this method, we need to return to examining the numerical solution of ordinary differential equations, that is to say equations whose order is non-fractional.
If for the equation

$$D^n y(x) = f(x, y(x)), \quad (3.4)$$

where $n \in \mathbb{N}$, we have the initial conditions $D^k y(0) = b_k$ $(k = 0, ..., n-1)$, with b_k as a constant, then we have an inital value problem.
By repeated integration of (3.4), using the initial conditions and the Cauchy formula for a repeated integral, we can express the solution of (3.4) as the solution $y(x)$ of the integral equation

$$y(x) = \sum_{k=0}^{n-1} \frac{x^k}{k!} D^k y(0) + \frac{1}{(n-1)!}\int_0^x (x-t)^{n-1} f(t, y(t))\mathrm{d}t.$$

We are now interested in seeking a numerical solution to the differential equation with the stated initial conditions. Practically we look to find a solution in an interval $[0, X]$ where $X > 0$ is a fixed value. The solution is not obtained for every value x in the interval, instead we specify a set of x, namely x_m where $m = 0, ..., N$. The set forms a series of nodes on the interval $[0, X]$, and we assume that the nodes are spaced evenly with a stepsize $h = X/N$, with $x_0 = 0$ and $x_N = X$. For each node with value x_m we aspire to find an approximated value of y, which will denote as y_m, so that y_m is an approximation of $y(x_m)$. By the same analogy we specify $f(x_m, y_m)$ as f_m. Thus we have replaced our continuous representation of the solution to the differential equation with a discretized numerical form. These discretized methods have the general title of linear multistep methods.

For a first order differential equation, i.e. $n = 1$ in (3.4), we can generally define a linear multistep method of s steps as

$$\sum_{k=0}^{s} \alpha_k y_{m-k} = h \sum_{k=0}^{s} \beta_k f_{m-k}. \tag{3.5}$$

where α_k, β_k are real constants for $k = 0, ..., s$, and there exist characteristic polynomials ρ and σ such that

$$\rho(\tau) \quad = \quad \sum_{k=0}^{s} \alpha_k \tau^{s-k}. \tag{3.6}$$

and

$$\sigma(\tau) \quad = \quad \sum_{k=0}^{s} \beta_k \tau^{s-k}. \tag{3.7}$$

The characteristic polynomials determine the nature of the linear multistep method, and so any linear multistep method is referred to as of type (ρ, σ) of s steps.
We can determine a particular y_m by extracting the term for $k = 0$ and dividing throughout by α_0.

$$y_m = \frac{1}{\alpha_0} \left[-\sum_{k=1}^{s} \alpha_k y_{m-k} + h \sum_{k=0}^{s} \beta_k f_{m-k} \right]$$

We will now look at a linear multistep method known as the Adams-Bashforth method.

Consider a first order differential equation of the form $Dy(x) = f(x, y(x))$ which has the solution

$$y(x) = \int_0^x f(t, y(t)) \mathrm{d}t.$$

We can discretize this for values of $y(x)$ and obtain these values for y

$$y_m = y_{m-1} + \int_{t_{m-1}}^{t_m} f(x,y)\mathrm{d}t.$$

The integral involving $f(x,y)$ is now approximated as

$$\int_{t_{m-1}}^{t_m} f(x,y)\mathrm{d}t \approx \int_{t_{m-1}}^{t_m} P(t)\mathrm{d}t.$$

where $P(t)$ is an approximating polynomial that uses an interpolating polynomial of the Lagrange form. Using the Lagrange form of an interpolating polynomial we can define

$$P(t) = \sum_{j=0}^{s} f_{m-j} \frac{(-1)^j}{j!(s-j)!} \prod_{\substack{i=0\\ i\neq j}}^{s} \frac{t - t_{m-i}}{t_{m+1-i} - t_{m-i}}.$$

Now $t_{m+1-i} - t_{m-i}$ equals our stepsize h so we can write the integral as

$$\int_{t_{m-1}}^{t_m} P(t)\mathrm{d}t = \int_{t_{m-1}}^{t_m} \sum_{j=0}^{s} f_{m-j} \frac{(-1)^j}{h^{s-1} j!(s-j)!} \prod_{\substack{i=0\\ i\neq j}}^{s} (t - t_{m-i})\mathrm{d}t.$$

The summation factor is independent of t, so the integral operator can be put inside the expression. The product term can be changed to a simpler form by making the substitution $t = t_{m-1} + uh$, which changes the limits of integration to 0 and 1 respectively.

$$\int_{t_{m-1}}^{t_m} P(t)\mathrm{d}t = h \sum_{j=0}^{s} f_{m-j} \frac{(-1)^j}{j!(s-j)!} \int_0^1 \prod_{\substack{i=0\\ i\neq j}}^{s} (u+i)\mathrm{d}u.$$

Substituting this expression for the integral of $P(t)$ in the above equation we have our expression for the approximated value of y

$$y_m = y_{m-1} + h \sum_{j=0}^{s} f_{m-j} \frac{(-1)^j}{j!(s-j)!} \int_0^1 \prod_{\substack{i=0\\ i\neq j}}^{s} (u+i)\mathrm{d}u.$$

We can now write our formula for approximated values as

$$y_m = y_{m-1} + h \sum_{j=0}^{s} \beta_j f_{m-j}$$

$$\text{where} \quad \beta_j = \frac{(-1)^j}{j!(s-j)!} \int_0^1 \prod_{\substack{i=0 \\ i \neq j}}^{s} (u+i) \mathrm{d}u.$$

By comparison with the general expression of the linear multistep method we can see that we have $\alpha_0 = 1$ and $\alpha_1 = -1$ together with the expression for β_j. We observe that $\alpha_k = 0$ for $k \geq 2$. The corresponding characteristic polynomials are thus

$$\rho(\tau) = \tau^s - \tau^{s-1} \quad \text{and} \quad \sigma(\tau) = \sum_{k=0}^{s} \beta_k \tau^k.$$

Generally for the linear multistep method we can now see that we need to set α_k arbitrarily according to the scheme we wish to use, and then determine the values β_k for the particular method adopted. We can then deduce the characteristic polynomials ρ and σ.

Another type of linear multistep method is the backward difference form (BDF). Instead of integrating an approximating polynomial as with $f(x, y)$ as above, we consider an approximating polynomial $P(x)$ to give an approximated value for y and differentiate it to equal $f(x_m, y_m)$. Taking a first order differential equation of the form $Dy(x) = f(x, y(x))$, we approximate its solution $y(x)$ such that $P(x_{m-i}) \approx y(x_{m-i})$ for $i = 0, ..., s$.

For a 1-step BDF method (BDF1) we have an approximating polynomial $P(x)$ so that (writing $y_m = y(x_m)$),

$$y(x) \approx P(x) = y_m + (x - x_m) \frac{y_m - y_{m-1}}{x_m - x_{m-1}}.$$

Differentiating then

$$Dy(x) = f(x, y) \approx P^{'}(x) = \frac{y_m - y_{m-1}}{x_m - x_{m-1}}.$$

We then approximate $P^{'}(x)$ to $f(x, y)$ at $x = x_m$ by setting $P^{'}(x_m) = f(x_m, y_m)$. Taking $x_m - x_{m-1}$ as the stepsize h then we can write

$$f(x_m, y_m) = \frac{y_m - y_{m-1}}{h}$$

which on rearranging is

$$y_m = y_{m-1} + h f(x_m, y_m).$$

Comparing this with equation (3.5) we can see that it matches for the BDF1 method ($s = 1$) if we set $\alpha_0 = 1$, $\alpha_1 = -1$, $\beta_0 = 1$ and β_1 to zero.
For BDF1 the characteristic polynomials are then $\rho(\tau) = \tau - 1$ and $\sigma(\tau) = \tau$.
We recognise the above expression for y_m as the usual backward Euler form.

Now, following the argument of Dattani [4], we are able to generalise the BDF method for s steps by considering the backward difference operator

$$\nabla y_m = y_m - y_{m-1}.$$

This is the first backward difference so we should more properly write

$$\nabla^1 y_m = y_m - y_{m-1}.$$

We can also state $\nabla^0 y_m = y_m$ and then $\nabla^1 y_m = \nabla^0 y_m - \nabla^0 y_{m-1}$.

The second backward difference is defined as $\nabla^2 y_m = \nabla^1 y_m - \nabla^1 y_{m-1}$, and we can then define the general backward difference operator as

$$\nabla^i y_m = \nabla^{i-1} y_m - \nabla^{i-1} y_{m-1}.$$

Now we can build a s-step BDF with the following statement of the approximating polynomial:

$$\begin{aligned} y(x) \approx P(x) = y_m + \frac{1}{h}(x - x_m)\nabla y_m + \frac{1}{2h^2}(x - x_m)(x - x_{m-1})\nabla^2 y_m + \cdots \\ \cdots + \frac{1}{s! h^s}(x - x_m)\ldots(x - x_{m-s+1})\nabla^s y_m. \end{aligned}$$

Differentiating the above representation, setting $x = x_m$, and observing that $P^{'}(x_m) = f(x_m, y_m)$ then

$$\sum_{i=1}^{s} \frac{1}{i}\nabla^i y_m = h f(x_m, y_m). \tag{3.8}$$

Looking back at the general expression for a linear multistep method (3.5) the expression we have obtained is found directly if $\beta_k = 1$ for $k = 0$ and $\beta_k = 0$ for $k \geq 1$. The values for α_k can be found by applying the difference operator and summing the terms.

As we have already seen, the solution $y(x)$ of the differential equation (3.4) is seen to be the solution of an integral equation. We consider a first order differential equation

$$Dy(x) = f(x, y(x)) \tag{3.9}$$

that has the solution

$$y(x) = \int_0^x f(t, y(t))\mathrm{d}t. \tag{3.10}$$

Lubich [24, Lemma 2.1] states that the application of a convolution quadrature with a linear multistep method for the differential equation results in the expression

$$y_m = h\sum_{j=0}^{m} w_{m-j} f(x_m, y(x_m)) + h\sum_{j=0}^{s} w_{m,j} f(x_m, y(x_m))$$

with a positive stepsize h, where w_{m-j} are the convolution weights and $w_{m,j}$ are the starting weights. The starting weights originate from correction terms added to the approximated integral because of the behaviour of the function near zero. This leads us to the following expression which approximates the convolution integral of the differential equation as

$$_hJ f(x, y(x) = h\sum_{j=0}^{m} w_{m-j} f(x_m, y(x_m)) + h\sum_{j=0}^{s} w_{m,j} f(x_m, y(x_m))$$

where $_hJ$ is the discretization of the integral with positive stepsize h.
Lubich [24, Lemma 2.1] states the convolution weights w_{m-j} are given by the coefficients of the generating function

$$w(\tau) = \frac{\sigma(1/\tau)}{\rho(1/\tau)}$$

where σ and ρ are the characteristic polynomials of the linear multistep method.
For example, the generating function for the BDF1 method outlined above is

$$w(\tau) = \frac{\sigma\left(1/\tau\right)}{\rho\left(1/\tau\right)} = \frac{1/\tau}{(1/\tau) - 1} = \frac{1}{1-\tau} = (1-\tau)^{-1}.$$

As a more complicated example let us consider BDF2 where we then have $s = 2$. Forming the backward differences using (3.8) and collecting terms gives

$$\frac{3}{2}y_m - 2y_{m-1} + \frac{1}{2}y_{m-2} = hf(x_m, y_m).$$

The characteristic polynomials are, from expressions (3.6) and (3.7),

$$\rho(\tau) = \frac{3}{2}\tau^2 - 2\tau + \frac{1}{2} \quad \text{and} \quad \sigma(\tau) = \tau^2.$$

The generating function for BDF2 is then

$$w(\tau) = \frac{\sigma\left(1/\tau\right)}{\rho\left(1/\tau\right)} = \frac{1/\tau^2}{\frac{3}{2}\left(\frac{1}{\tau}\right)^2 - 2\left(\frac{1}{\tau}\right) + \frac{1}{2}} = \frac{1}{\frac{3}{2} - 2\tau + \frac{1}{2}\tau^2} = \left(\frac{3}{2} - 2\tau + \frac{1}{2}\tau^2\right)^{-1}.$$

Thus the generating function is solely a function of the inverse of the characteristic polynomial $\rho(\tau)$ expression. By examination of the expansion of each ∇ term in in Dattani's expression (3.8), we can see that the coefficients of the BDF formula (3.8) are formed by the series

$$\delta(\tau) = \sum_{k=1}^{s} \frac{1}{k}(1-\tau)^k.$$

The general expression for the generating function for the BDF s-step method as applied to the differential equation (3.9) is then

$$w(\tau) = \frac{1}{\sum\limits_{k=1}^{s} \frac{1}{k}(1-\tau)^k} = \left(\sum_{k=1}^{s} \frac{1}{k}(1-\tau)^k\right)^{-1}.$$

In a later paper on convolution quadrature [27], Lubich considered the approximation of the convolution integral

$$\int_0^x f(t)g(x-t)\,\mathrm{d}t \quad \text{where}\, x \geq 0$$

by a discrete convolution representation

$$\sum_{0 \leq jh \leq x} W_j(h)g(x-jh).$$

where h is the positive stepsize and $W_j(h)$ are the convolution quadrature weights. He then stated that the convolution weights $W_j(h)$ are the coefficients of the the power series

$$F(\delta(\tau)/h) = \sum_{j=0}^{\infty} W_j(h)\tau^j$$

where F is the Laplace transform of the function f and $\delta(\tau)$ is the generating function

derived from the characteristic polynomials ρ and σ.
If we now apply this convolution quadrature to the BDF linear multistep method for the differential equation (3.9) which has the solution (3.10) we can see that from the convolution integral representation $f(t) \equiv 1$ and $g(t) \equiv f(t, y(t))$. To find the convolution quadrature weights, take the Laplace transform F of f, and then we have

$$\sum_{j=0}^{\infty} W_j(h)\tau^j = F\left(\delta(\tau)/h\right) = \frac{1}{\delta(\tau)/h} = h\,\delta(\tau)^{-1} \quad \text{as}\, \mathcal{L}\left(1\right) = 1/s.$$

If we now let w_j be the coefficients of $\delta(\tau)^{-1}$ then $W_j(h) = hw_j$.
This convolution quadrature of the differential equation for the m^{th} interval of the BDF linear multistep method then leads, as we have seen above from Lubich [24, Lemma 2.1], to the approximated solution

$$y_m = h\sum_{j=0}^{m} w_{m-j} f(x_m, y(x_m)) + h\sum_{j=0}^{s} w_{m,j} f(x_m, y(x_m))$$

where the convolution weights w_{m-j} are the coefficients of $\delta(\tau)^{-1}$ and $w_{m,j}$ are the starting weights.
Lubich then adapted these classical linear multistep methods with his convolution quadrature method for the numerical approximation of fractional integrals. Lubich evolved his discretized operational calculus as a discretized fractional calculus in his eponymous paper [26]. Weilbeer has shown [34, Theorem 4.2.3] that the fractional differential equation of the Caputo type can be replaced by an Abel-Volterra equation

$$y(x) = \Gamma_{n-1}[y;0](x) + \frac{1}{\Gamma(\alpha)}\int_0^x (x-t)^{\alpha-1} f(t, y(t))\, \mathrm{d}t.$$

Diethelm [9, p. 38] defines the Caputo derivative as

$$D^{\alpha}_{a*}y(x) = D^{\alpha}_{a}\left[y(x) - \Gamma_{m-1}\left[y;a\right](x)\right]$$

where

$$\Gamma_{m-1}\left[y;a\right](t) = \sum_{k=0}^{m-1} \frac{y^{(k)}\left(a\right)}{k!}\left(t-a\right)^k$$

and $(m-1)$ is the degree of the Taylor polynomial, such that $(m-1) < \alpha \leq m$.
We now follow the ideas of Lubich [25] [26], as stated by Weilbeer [34] and Diethelm, Ford,

Ford and Weilbeer [18] to approximate a fractional convolution integral by a fractional convolution quadrature. Lubich [26] defines an approximation to the integral equation

$$y(x) = J_a^\alpha[f(x, y(x))] = \frac{1}{\Gamma(\alpha)} \int_0^x (x-t)^{\alpha-1} f(t, y(t))\,\mathrm{d}t,$$

which is equivalent to the fractional differential equation $D^\alpha y(x) = f(x, y(x))$, as

$$J_a^\alpha[f(x_m, y(x_m))] = h^\alpha \sum_{j=0}^{m} w_{m-j} f(x_m, y(x_m)) + h^\alpha \sum_{j=0}^{s} w_{m,j} f(x_m, y(x_m)).$$

In this equation w_{m-j} are the convolution weights, and $w_{m,j}$ are the starting weights.
We can see the origin of this expression by considering the integral above as a convolution integral and then applying the convolution quadrature method of Lubich.
Apply the Riemann-Liouville integral operator with the lower terminal $a = 0$ to the Caputo derivative definition above, we have

$$J_0^\alpha\left(D_{*0}^\alpha y(x)\right) = J_0^\alpha\left(D_0^\alpha y(x)\right) - J_0^\alpha\left(D_0^\alpha T_{m-1}[y; 0](x)\right)$$

Now $J_0^\alpha D_0^\alpha = \mathrm{I}$ and $D_{*0}^\alpha y(x) = f(x, y(x))$ so

$$J_0^\alpha\left(f(x, y(x))\right) = y(x) - T_{m-1}[y; 0](x).$$

Using Lubich's approximation of the convolution integral $J_0^\alpha[f(x_m, y(x_m))]$, setting $y_m = y(x_m)$ as the approximated y value for $x = x_m$, and collecting terms the approximated solution to the fractional differential equation is

$$y_m = h^\alpha \sum_{j=0}^{m} w_{m-j} f(x_m, y_m) + h^\alpha \sum_{j=0}^{s} w_{m,j} f(x_m, y_m) + T_{m-1}[y; 0](x_m).$$

We now need to find the convolution and starting weights.
Taking F as the Laplace transform of $(x-t)^{\alpha-1}/\Gamma(\alpha)$ we have

$$F\left(\frac{(x-t)^{\alpha-1}}{\Gamma(\alpha)}\right) = \frac{1}{s^\alpha}.$$

By Lubich's method we then have

$$F\left(\delta(\tau)/h\right) = \frac{1}{(\delta(\tau)/h)^{\alpha}} = h^{\alpha}\delta(\tau)^{-\alpha}.$$

Thus extending our reasoning as applied to the first order differential equation we can define the convolution weights for the fractional order differential equation as

$$w^{\alpha}(\tau) = \delta(\tau)^{-\alpha} = \left(\sum_{k=1}^{p}\frac{1}{k}(1-\tau)^{k}\right)^{-\alpha}$$

where $\delta(\tau)$ is the generating function of the characteristic polynomials ρ and σ.
Applying the same principles of automatic differentiation to $f(x) = [g(x)]^{-\alpha}$, as Weilbeer [34, Theorem 5.3.1] and Diethelm, Ford, Ford and Weilbeer [18] applied to $f(x) = [g(x)]^{\alpha}$, the following formula for the convolution weights is obtained.

$$w_m = \frac{1}{mu_0}\sum_{j=0}^{m-1}[-\alpha(m-j)-j]w_j u_{m-j}$$

u_n are the coefficients of the generating function found from the characteristic polynomials ρ and σ for the classical linear multistep method.
The convolution weight for the m^{th} node is thus found in terms of the convolution weights of the previous $(m-1)$ nodes. Ultimately we need to determine the value w_0. We do this by setting $\tau = 0$ in the expression for $w^{\alpha}(\tau)$, and taking the resulting value as w_0.
The starting weights are found by solving the linear system of equations

$$\sum_{j=0}^{s} w_{m,j} j^{\gamma} = \frac{\Gamma(1+\gamma)}{\Gamma(1+\gamma+\alpha)} m^{\gamma+\alpha} - \sum_{j=1}^{m} w_{m-j} j^{\gamma}$$

where $\gamma \in \mathcal{A}$ and $\mathcal{A} = \{k + l\alpha\}$ with $k, j \in N$ and $s = \text{Card}\mathcal{A} - 1$. For each node within the interval a set of $(s+1)$ starting weights must be calculated.
Weilbeer [34, Theorem 5.1.9] derived this equation from the expression of Lubich [26, 4.2]. The formula above for the approximated solution of the differential equation is perfectly adequate for the evaluation of y_m. Unfortunately there is a penalty in terms of computation because the function $f(x_m, y_m)$ is calculated for all the previous nodes of the interval in order to compute the current nodal value y_m. This penalty will be reflected in the computing time of the program for the numerical solution.

We checked the validity of the computation formula above by a Scilab program in Appendix F, however we have not included the results for the execution times for the range of interval sizes.
The actual solution values computed will of course be the same as the computation formula adopted below.
We can avoid the computation of $f(x_m, y_m)$ for each y_m by considering a better formula developed as follows.
Restating Diethelms's definition of the Caputo differential operator with the lower terminal $a = 0$ we have

$$D_{0*}^{\alpha} y(x) = D_0^{\alpha} \left[y(x) - \Gamma_{m-1} \left[y; 0 \right] (x) \right]$$

and by rearranging the terms then

$$D_0^{\alpha} \left[y(x) \right] = D_{0*}^{\alpha} y(x) + D_0^{\alpha} \Gamma_{m-1} \left[y; 0 \right] (x)$$

so that

$$D_0^{\alpha} \left[y(x) \right] = f(x, y(x)) + D_0^{\alpha} \Gamma_{m-1} \left[y; 0 \right] (x).$$

The Riemann-Liouville differential operator is defined as $D^{\alpha} = D^m J^{m-\alpha}$ (in terms of operators only). Now $D^{\alpha} = D^m J^{m-\alpha} = D^m J^m J^{-\alpha} = J^{-\alpha}$ as $D^m J^m = \mathrm{I}$. Hence we now have

$$J^{-\alpha} \left[y(x) \right] = f(x, y(x)) + D_0^{\alpha} \Gamma_{m-1} \left[y; 0 \right] (x)$$

where $\alpha > 0$. We can approximate $J^{-\alpha} \left[y(x) \right]$ by using Lubich's fractional quadrature formula with $-\alpha$ replacing $+\alpha$.
This means the same formulas and equations for convolution and starting weights can be used as above with $-\alpha$ instead of $+\alpha$.
Replacing $J^{-\alpha} \left[y(x) \right]$ with the fractional quadrature formula then

$$h^{-\alpha} \sum_{j=0}^{m} w_{m-j} y(x_j) + h^{-\alpha} \sum_{j=0}^{s} w_{m,j} y(x_j) = f(x_m, y(x_m)) + D_0^{\alpha} \Gamma_{m-1} \left[y; 0 \right] (x_m).$$

Extracting the $j = m$ term from the first summand, multiplying throughout by h^{α}, collecting terms and dividing by w_0 gives an expression for the approximated value y_m as

$$y_m = \frac{1}{w_0} \left[h^{\alpha} f(x_m, y(x_m)) - \sum_{j=0}^{m-1} w_{m-j} y(x_j) - \sum_{j=0}^{s} w_{m,j} y(x_j) + h^{\alpha} D_0^{\alpha} \Gamma_{m-1} \left[y; 0 \right] (x_m) \right]$$

where $\alpha > 0$. We note that this differs from Weilbeer's formula [34, Theorem 5.1.10] by a factor $1/w_0$.
We now have a more workable expression to calculate y_m as we are now using the values y_m calculated for the previous nodes of the interval.

Having determined both the the convolution weights and starting weights we are now able to calculate y_m for each node. However, due to the summation term with the starting weights, each computation of the first s values of y_m contain the first y_n values ($n = 1, ...s$), and thus form a set of linear equations. Hence we solve a set of linear equations for the values y_1 to y_s and for the remaining values of y_m we can use a step-by-step method using the above formula to arrive at our final value as the solution of the equation.

3.3 Luchko and Diethelm's operational calculus method

Diethelm and Luchko's paper [12] uses the formulas of an analytical solution developed by Luchko and Gorenflo [29] with a Green's function representation, combines them with the convolution quadrature and discretized operational calculus of Lubich [27], [28], [26] to produce an algorithm to solve the following equation

$$D_{0*}^{\alpha} y(x) - \sum_{i=1}^{\nu} \lambda_i \left(D_{0*}^{\alpha_i} y(x)\right) = f(x) \tag{3.11}$$

where D_{0*}^{α} is the Caputo fractional differential operator; $\lambda_i \in \mathbb{R}$; $0 < x \leq T$ with $T < \infty$; $\alpha > \alpha_1 > \ldots > \alpha_\nu \geq 0$; $m_i - 1 < \alpha_i \leq m_i$ with $m_i \in \mathbb{N}_0$.
The equation has the initial values $y^{(k)}(0) = c_k$ where $c_k \in \mathbb{R}$, $(k = 0, 1, \ldots, m-1)$ and $m - 1 < \alpha \leq m$ $(m \in \mathbb{N})$.
We need to begin by examining the use of fractional Green's function in the solution of fractional differential equations. Podlubny devotes a whole chapter of his book [31, Chapter 5] to the fractional Green's function. He states the fractional Green's function for the equation

$$a_n D^{\beta_n} y(x) + a_{n-1} D^{\beta_{n-1}} y(x) + \ldots + a_1 D^{\beta_1} y(x) + a_0 D^{\beta_0} y(x) = f(x)$$

as

$$G_n(x) = \frac{1}{a_n}\sum_{m=0}^{\infty}\frac{(-1)^m}{m!}\sum_{\substack{k_0+k_1+\ldots+k_{n-2}=m\\ k_0\geq 0;\ldots;k_{n-2}\geq 0}}(m;k_0,k_1,\ldots,k_{n-2})$$
$$\times\prod_{i=0}^{n-2}\left(\frac{a_1}{a_n}\right)^{k_i}x^{(\beta_n-\beta_{n-1})m+\beta_n+\sum_{j=0}^{n-2}(\beta_{n-1}-\beta_j)k_j-1}$$
$$\times E^{(m)}_{\beta_n-\beta_{n-1},\beta_n+\sum_{j=0}^{n-2}(\beta_{n-1}-\beta_j)k_j}\left(-\frac{a_{n-1}}{a_n}x^{\beta_n-\beta_{n-1}}\right)$$

where $(m;k_0,k_1,\ldots,k_{n-2})$ is the multinomial coefficient. $E^{(j)}_{\alpha,\beta}$ is the jth derivative of the Mittag-Leffler function as defined in Section 1.2 of this chapter. Diethelm and Ford have expressed in their paper [11] the above equation and its fractional Green's function in a slightly different form, and state the multinomial coefficient as $(m;k_0,k_1,\ldots,k_{n-2}) = m!/\prod_{i=0}^{n-2}(k_i!)$.

The paper of Luchko and Gorenflo [29, Theorem 4.1] has the proof of the solution of fractional differential equation considered by Diethelm and Luchko. In theorem 1 of their paper Diethelm and Luchko take $f(x) = x^{\gamma-1}\hat{f}(x)$, and use the solution of Luchko and Gorenflo given by the formula

$$y(x) = y(x)_{\sim f} + \sum_{k=0}^{m-1}c_k u_k(x).$$

Here

$$y(x)_{\sim f} = \int_0^x \tilde{E}(\alpha;x)f(x-t)\mathrm{d}t \tag{3.12}$$

is a solution of (3.11) with zero initial conditions, and the system of functions

$$u_k(x) = \frac{x^k}{k!} + \sum_{i=l_k+1}^{\nu}\lambda_i\tilde{E}(k+1+\alpha-\alpha_i;x), \quad \text{where } k=0,\ldots,m-1,$$

satisfies the initial conditions $u_k^{(l)} = \delta_{kl},\ k,l=0,\ldots,m-1$ (δ_{kl} is the Kronecker symbol). We can see that the function

$$\tilde{E}(\beta;x) = x^{\beta-1}E_{(\alpha-\alpha_1,\ldots,\alpha-\alpha_\nu),\beta}(\lambda_1 x^{\alpha-\alpha_1},\ldots,\lambda_\nu x^{\alpha-\alpha_\nu}), \quad \text{where } \beta>0,$$

is a Green's function and can be stated in terms of the multivariate Mittag-Leffler function

$$E_{(\alpha_1,\ldots,\alpha_\nu),\beta}(z_1,\ldots,z_\nu) := \sum_{k=0}^{\infty} \sum_{\substack{l_0+l_1+\ldots+l_\nu=k \\ l_1\geq 0;\ldots;l_\nu\geq 0}} \frac{k!}{l_1! \ \times\cdot\times \ l_\nu!} \frac{\prod_{i=1}^{\nu} z_i^{l_i}}{\Gamma\left(\beta+\sum_{i=1}^{\nu}\alpha_i l_i\right)}.$$

Diethelm and Luchko's method revolves around the specification of a new algorithm for the evaluation of the multivariate Mittag-Leffler function. They use the method of Lubich to obtain an inverse Laplace transform of $\tilde{E}(\beta; x)$, together with his convolution quadrature method in [27]. This involves applying the linear multistep methods described in Section 3.2, which are applied by Lubich via convolution quadrature and discretized operational calculus. Diethelm and Luchko thus produce a numerical solution of the fractional differential equation by a new algorithm which numerically approximates the multivariate Mittag-Leffler function using the linear multistep scheme of Lubich. They then compute some numerical examples, and with them compare their new method with the BDF method of Diethelm [8] and the PECE method (Predict,Evaluate,Correct,Evaluate) method of Diethelm and Freed [7].

The method firstly requires a Laplace transform of $\tilde{E}(\beta; x)$ given by Podlubny [31] in combination with Luchko and Gorenflo [29, Theorem 4.1] to obtain the formula

$$F(\beta; s) = \frac{s^{\alpha-\beta}}{s^{\alpha} - \sum_{i=1}^{\nu} \lambda_i s^{\mu_i}} \tag{3.13}$$

Lubich's convolution quadrature method states that for the convolution integral

$$\int_0^x f(t)g(x-t)\,\mathrm{d}t \quad \text{where}\, x \geq 0$$

the convolution weights $W_j(h)$ are given by the coefficients of the the power series

$$F(\delta(\tau)/h) = \sum_{j=0}^{\infty} W_j(h)\tau^j$$

where F is the Laplace transform of the function f and $\delta(\tau)$ is the generating function of the linear multistep method.

In our case $F(\delta(\tau)/h) \equiv F(\beta; \delta(\tau)/h)$, where F is the Laplace transform of $\tilde{E}(\beta; x)$, so we can write

$$F(\beta;\delta(\tau)/h) = \sum_{j=0}^{\infty} W_j(h)\tau^j$$

where F is the Laplace transform of the function $\tilde{E}(\beta;x)$ and $\delta(\tau)$ is the generating function of the linear multistep method.
The solution of the fractional differential equation is given by (3.12).
We can now approximate the convolution integral in equation (3.12) in the same manner as Lubich's convolution quadrature method. Thus we can express the approximated solution $y(x_m)_{\sim f}$ at the m^{th} node of the interval of operation $[0, X]$ as

$$y(x_m)_{\sim f} = \sum_{j=0}^{m} W_j(h) f(x_{m-j})$$

where $m = 1, \ldots, N$ and N is the number of nodes of the operating interval.
For easier evaluation we now replace j with $m - j$ with the resulting equation

$$y(x_m)_{\sim f} = \sum_{j=0}^{m} W_{m-j}(h) f(x_j)$$

We can then find the convolution weights W_{m-j} by using (3.13) with β replaced by α.
For the full implementation of Lubich's convolution quadrature method we need to add some correction terms to compensate for the behaviour of the function near zero, giving

$$y(x_m)_{\sim f} = \sum_{j=0}^{m} W_{m-j} f(x_j) + \sum_{j=j_0}^{q-1} W_{m,j} f(x_j) \tag{3.14}$$

where $W_{m,j}$ are the correction or starting weights, and $j_0 = 1$ if $0 < \gamma < 1$ or $j_0 = 0$ if $\gamma \geq 1$.
The parameter q is determined by the relation $q - 1 < p - \gamma \leq q$.
The starting weights are found by the same method as in Section 3.2, that is taking the above equation with a power function $f(x_j) = (x_j)^\eta$. Adopting the same technique as in Section 3.2 we set $\eta = k + \gamma - 1$ where $k = 0, 1, \ldots, q - 1$ giving the equation

$$y(x_m)_{\sim x^{k+\gamma-1}} = \sum_{j=0}^{m} W_{m-j}(x_j)^{k+\gamma-1} + \sum_{j=j_0}^{q-1} W_{m,j}(x_j)^{k+\gamma-1}.$$

Rearranging the terms we then have

$$\sum_{j=j_0}^{q-1} W_{m,j}(x_j)^{k+\gamma-1} = y(x_m)_{\sim x^{k+\gamma-1}} - \sum_{j=0}^{m} W_{m-j}(x_j)^{k+\gamma-1}.$$

Now Diethelm and Luchko [12] state that Luchko and Gorenflo showed the solution of (3.11) for the power function $f(x) = x^\eta$ is

$$y(x)_{\sim x^\eta} = \Gamma(\eta+1)\tilde{E}(\alpha+\eta+1; x).$$

If we let $\eta = k+\gamma-1$ then we can set the approximated value $y(x)_{\sim x^\eta}$ as

$$y(x_m)_{\sim x^{k+\gamma-1}} = \Gamma(k+\gamma)\tilde{E}(k+\alpha+\gamma; x_m).$$

Substituting this into the starting weights equation gives us

$$\sum_{j=j_0}^{q-1} W_{m,j}(x_j)^{k+\gamma-1} = \Gamma(k+\gamma)\tilde{E}(k+\alpha+\gamma; x_m) - \sum_{j=0}^{m} W_{m-j}(x_j)^{k+\gamma-1}. \tag{3.15}$$

This equation can be solved to find the starting weights $W_{m,j}$ at each interval node, providing we can evaluate $\tilde{E}(\beta; x)$.
The Laplace transform of $\tilde{E}(\beta; x)$ is $F(\beta; s)$, so the inverse Laplace transform of $F(\beta; s)$ is the Mittag-Leffler type function.
Lubich's paper [27, Section 4] states that the inverse Laplace transform of $F(s)$ can be approximated by W_n/h, where W_n are the coefficients of the convolution quadrature associated with $F(s)$. Thus we can approximate $\tilde{E}(k+\alpha+\gamma; x_m)$ for the m^{th} node by calculating the weight for $\beta = k+\alpha+\gamma$ at the m^{th} node and dividing by the stepsize h.

Hence we can calculate the approximated value at each node of the interval using (3.14), and thus eventually find the final solution to the fractional differential equation.

Chapter 4

Equation and Solution

Aim

In this chapter we will examine an example fractional differential equation and apply the methods considered in Chapter 3. We will consider an equation with an exact solution, find the solution, and use the numerical methods to find various approximate solutions.

1 Equation example

The equation we consider is derived from (3.2) in Chapter 3, namely

$$D_{0*}^{\alpha} y(x) = f(x, y(x))$$

with $f(x, y)$ as the form $f(x, y) = -\lambda y + q(x)$. We consider the equation from Diethelm and Ford's paper [10] which we obtain by setting $\alpha = 0.5$ in the equation, and choosing $\lambda = 1$ so that

$$q(x) = x^2 + \frac{2}{\Gamma(2.5)} x^{1.5}.$$

Thus our equation for solution is

$$D_{0*}^{0.5} y(x) = -y(x) + x^2 + \frac{2}{\Gamma(2.5)} x^{1.5}$$

with the initial conditions $y(0) = 0$ and $y^{'}(0) = 0$. Diethelm and Ford's paper [10] indicates that this equation has an exact solution $y(x) = x^2$. We can show this by the use of Podlubny's Laplace transform method from Chapter 3. Applying the Laplace transform operator then

$$\mathcal{L}\left\{D_{0*}^{0.5}y(x)\right\} = -\mathcal{L}\left\{y(x)\right\} + \mathcal{L}\left\{x^2\right\} + \mathcal{L}\left\{\frac{2}{\Gamma(2.5)}x^{1.5}\right\}.$$

We have already shown the expression of Podlubny for the Laplace transform of the Caputo derivative in Section 1.3 of Chapter 3, so we are now able to evaluate the above transformation, taking into account the initial conditions.

$$s^{0.5}Y(s) = -Y(s) + \frac{\Gamma(3)}{s^3} + \frac{2}{\Gamma(2.5)} \cdot \frac{\Gamma(2.5)}{s^{2.5}}$$

$Y(s)$ is the Laplace transform of $y(x)$. Gathering terms together, cancelling factors and setting $\Gamma(3) = 2$ we have

$$\left(s^{0.5}+1\right)Y(s) = \frac{2}{s^3} + \frac{2}{s^{2.5}} = \frac{2}{s^3}\left(1+s^{0.5}\right)$$

Cancelling the factor $s^{0.5}+1$ leaves a simple expression, and taking the inverse Laplace transform, we then have our solution $y(x) = x^2$.

1.1 Solution with Diethelm's BDF method

We will now obtain numerically approximate the solution to the equation above by adopting Diethelm's BDF method.
For the equation

$$D_{0*}^{\alpha}y(x) = f(x, y(x))$$

we have seen from (3.2) that Diethelm's BDF method gives the solution to the equation as

$$\left(h^{\alpha}\Gamma(2-\alpha)\lambda + \beta_{0j}\right)y_j = j^{-\alpha}(1-\alpha)y_0 + hj^{1-\alpha}y_0' + h^{\alpha}\Gamma(2-\alpha)q(t_j) - \sum_{k=1}^{j}\beta_{kj}y_{j-k},$$

having taken $f(x,y)$ as the form $f(x,y) = -\lambda y + q(x)$.
We now take as the initial conditions y_0 and $y_0' = 0$, thus giving the equation

$$\left(h^{\alpha}\Gamma(2-\alpha)\lambda + \beta_{0j}\right)y_j = h^{\alpha}\Gamma(2-\alpha)q(t_j) - \sum_{k=1}^{j}\beta_{kj}y_{j-k}. \tag{4.1}$$

We then have our expression for y_j as

$$y_j = \frac{1}{h^\alpha \Gamma(2-\alpha)\lambda + \beta_{0j}} \left(h^\alpha \Gamma(2-\alpha) q(t_j) - \sum_{k=1}^{j} \beta_{kj} y_{j-k} \right)$$

where β_{kj} is evaluated by the formula $\beta_{kj} = -\alpha(1-\alpha)j^{-\alpha}\alpha_{kj}$ and the various expressions for interval weights given by (3.1).

The particular equation we are interested has $\alpha = 0.5$ and $\lambda = 1$, so setting these values we have our final expression for evaluation

$$y_j = \frac{1}{h^{0.5}\Gamma(1.5) + \beta_{0j}} \left(h^{0.5}\Gamma(1.5) q(t_j) - \sum_{k=1}^{j} \beta_{kj} y_{j-k} \right) \tag{4.2}$$

where $q(t_j)$ is taken from the formula above.
The solution is computed by the Scilab program detailed in Appendix E, and the results are given in Chapter 5.

1.2 Solution with Lubich's convolution quadrature method

We solve the same equation using Lubich's convolution quadrature method, which from Chapter 3 states the solution as

$$y_m = h^\alpha f(x_m, y_m) - \sum_{j=0}^{m-1} w_{m-j} y(x_j) - \sum_{j=0}^{s} w_{m,j} y(x_j) + h^\alpha D^\alpha \Gamma_{n-1}[y;0](x_m).$$

In our treatment of Diethelm's BDF method we stated that the fractional derivative of the Taylor polynomial is expressed in terms of y_0 and y'_0. We have taken as our initial conditions $y_0 = 0$ and $y'_0 = 0$ so we are able to set this term to zero.
The form of $f(x, y)$ we have adopted in this equation is $f(x, y) = -\lambda y + q(x)$, so we can say

$$y_m = h^\alpha(-\lambda y_m + q(x_m)) - \sum_{j=0}^{m-1} w_{m-j} y(x_j) - \sum_{j=0}^{s} w_{m,j} y(x_j).$$

Before we can calculate y_m for each node in the interval, we must determine the convolution weights and the starting weights for each node.
The convolution weights are found for the m^{th} node by the simple formula which we derived

in Chapter 3

$$w_m = \frac{1}{mu_0}\sum_{j=0}^{m-1}[-\alpha(m-j)-j]w_j u_{m-j}.$$

As we also stated stated in Chapter 3 we can use the same formula as the $f(x_m)$ version of the method, merely replacing α with $-\alpha$ solely for the computation of the convolution and starting weights.
The starting weights for the m^{th} node are more difficult to calculate.
We use the formula from Chapter 3 for the computation of starting weights.
At each node we need to find the solution $w_{m,j}$ to the set of equations

$$\sum_{j=0}^{s} w_{m,j} j^{\gamma} = \frac{\Gamma(1+\gamma)}{\Gamma(1+\gamma+\alpha)} m^{\gamma+\alpha} - \sum_{j=1}^{m} w_{m-j} j^{\gamma}$$

where $\gamma \in \mathcal{A}$ and $\mathcal{A} = \{k + l\alpha\}$ with $k, j \in N$ and $s = \text{Card}\mathcal{A} - 1$.
The same formula for the $f(x_m)$ version of the method is used, replacing α with $-\alpha$.
For example with BDF2 and $\alpha = 0.5$ then $\gamma \in \mathcal{A} = \{0, 0.5, 1\}$ and thus $s = 2$.
As there are as many values of γ as there are equations we can now write

$$\sum_{j=0}^{s} w_{m,j} j^{\gamma_j} = \frac{\Gamma(1+\gamma_j)}{\Gamma(1+\gamma_j+\alpha)} m^{\gamma_j+\alpha} - \sum_{j=1}^{m} w_{m-j} j^{\gamma_j}.$$

with $\gamma_j = \mathcal{A}\{j\}$ $(j = 0, ..., s)$, solve this system of equations for $w_{m,j}$, and obtain the starting weights for each node in the interval.
Now collecting terms in the expression for y_m we have

$$(1 + h^{\alpha}\lambda)y_m = h^{\alpha}q(x_m) - \sum_{j=0}^{m-1} w_{m-j}y(x_j) - \sum_{j=0}^{s} w_{m,j}y(x_j).$$

The equation we are dealing with has $\alpha = 0.5$ and $\lambda = 1$ so we have

$$(1 + h^{0.5})y_m = h^{0.5}q(x_m) - \sum_{j=0}^{m-1} w_{m-j}y(x_j) - \sum_{j=0}^{s} w_{m,j}y(x_j). \tag{4.3}$$

We calculate the first s values y_m $(m = 1, ..., s)$ which all appear together in the system of equations defined by (4.3). As we know the stepsize h from the number of steps specified for the interval, and setting $y(x_j) = y_j$ on the RHS so that they are designated as approximated values, the first values y_m $(m = 1, ..., s)$ can be determined by solving the set of equations. Now the remaining values of y_m $(m = s + 1, ..., N)$ are found step-by-step using the approximated previous values of y_m. The solution is computed by the Scilab program detailed in Appendix G, and the results are given in Chapter 5.

1.3 Solution with Luchko and Diethelm's operational calculus method

We are seeking the solution to the equation

$$D^{\alpha}_{0*}y(x) = f(x, y(x))$$

where $f(x, y) = -\lambda y + q(x)$.
Comparing this with Diethelm and Luchko's general expression (3.11) we have $\nu = 1$, $\alpha_i = 0$ and $f(x) = q(x)$.
Thus we have our equation for solution, keeping Diethelm and Luchko's notation, as

$$D^{\alpha}_{0*}y(x) - \lambda y(x) = f(x) \tag{4.4}$$

where $\alpha = 0.5$, $\lambda = -1$ and $f(x) = q(x)$. The initial conditions are $y(0) = 0$ and $y^{'}(0) = 0$.
We have shown in Chapter 3 that Diethelm and Luchko's operational calculus method gives the solution as

$$y(x_m)_{\sim f} = \sum_{j=0}^{m} W_{m-j} f(x_j) + \sum_{j=j_0}^{q-1} W_{n,j} f(x_j). \tag{4.5}$$

The convolution weights W_{m-j} and the starting weights $W_{n,j}$ must be calculated to find the value of $y(x_m)_{\sim f}$.
The convolution weights are found using the Laplace transform formula $F(\beta; s)$ given by (3.13) with $\beta = \alpha$. This gives the simplified expression for the Laplace transform of the Mittag-Leffler type function, including the format of the equation above, as

$$F(\beta; s) = \frac{1}{s^{\alpha} - \lambda}.$$

The actual convolution weights are found by setting $s = \delta(\tau)/h$, where $\delta(\tau)$ is the generating function of the linear multistep method, and then determining the Taylor coefficients of the related power series.
In Chapter 3 we referred to the principles of automatic differentiation when calculating the convolution weights for Lubich's convolution quadrature method. These principles allow recursive formulas to be obtained for Taylor series coefficients. We can use these expressions to calculate the convolution weights.
Firstly, we need to state some basic definitions. These can be found in the papers of Armstrong [2] and Blomquist, Hofshuster, and Krämer [13].
The Taylor expansion of an analytic function $f(x)$ around the value x_0 is

$$f(x) = \sum_{k=0}^{\infty} \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k.$$

We define the Taylor coefficient as

$$(f)_k \equiv \frac{f^{(k)}(x_0)}{k!}.$$

For the constant function $f(x) = c$ we have $(c)_0 = c$, $(c)_k = 0$, for $k \geq 1$. Consider the two analytic functions $u(x)$ and $v(x)$.
We then have the following expressions which we can use to derive the formula for the convolution weights.

$$(u \pm v)_k = (u)_k \pm (v)_k$$

$$(u \cdot v)_k = \sum_{j=0}^{k} (u)_j (v)_{k-j} = \sum_{j=0}^{k} (u)_{k-j} (v)_j$$

$$(u^\alpha)_k = \frac{1}{k(u)_0} \sum_{j=0}^{k-1} [\alpha(k-j) - j](u^\alpha)_j (u)_{k-j} \tag{4.6}$$

Taking our formula for $F(\beta : s)$ with $s = u(x)/h$, we need to find an expression for the k^{th} Taylor coefficient of

$$\frac{1}{(u/h)^\alpha - \lambda} = \frac{h^\alpha}{u^\alpha - \lambda h^\alpha}$$

where we set $u \equiv u(x)$ for easier algebraic manipulation.
Using the product property $(u \cdot v)_k$ with

$$u = \frac{u^\alpha - \lambda h^\alpha}{h^\alpha} \quad \text{and} \quad v = \frac{h^\alpha}{u^\alpha - \lambda h^\alpha}$$

we then have

$$\left(\frac{u^\alpha - \lambda h^\alpha}{h^\alpha} \cdot \frac{h^\alpha}{u^\alpha - \lambda h^\alpha}\right)_k = \sum_{j=0}^{k} \left(\frac{u^\alpha - \lambda h^\alpha}{h^\alpha}\right)_j \left(\frac{h^\alpha}{u^\alpha - \lambda h^\alpha}\right)_{k-j}.$$

Extracting $j = 0$ term, and noting that the left hand term of the equation is $(1)_k = 0$ for $k \geq 1$ then

$$0 = \sum_{j=1}^{k} \left(\frac{u^\alpha - \lambda h^\alpha}{h^\alpha}\right)_j \left(\frac{h^\alpha}{u^\alpha - \lambda h^\alpha}\right)_{k-j} + \left(\frac{u^\alpha - \lambda h^\alpha}{h^\alpha}\right)_0 \left(\frac{h^\alpha}{u^\alpha - \lambda h^\alpha}\right)_k.$$

Rearranging terms and dividing we then have

$$\left(\frac{h^\alpha}{u^\alpha - \lambda h^\alpha}\right)_k = -\left(\frac{h^\alpha}{u^\alpha - \lambda h^\alpha}\right)_0 \sum_{j=1}^{k} \left(\frac{u^\alpha - \lambda h^\alpha}{h^\alpha}\right)_j \left(\frac{h^\alpha}{u^\alpha - \lambda h^\alpha}\right)_{k-j}$$

which on separating the first factor of the summand gives

$$\left(\frac{h^\alpha}{u^\alpha - \lambda h^\alpha}\right)_k = -\left(\frac{1}{u^\alpha - \lambda h^\alpha}\right)_0 \sum_{j=1}^{k} (u^\alpha)_j \left(\frac{h^\alpha}{u^\alpha - \lambda h^\alpha}\right)_{k-j}$$

as $(\lambda)_j = 0$ for $j \geq 1$.
By definition the term on the left hand side is W_k and the similar factor on the right hand side is W_{k-j}.
We now remember that $u \equiv \delta(\tau)$, the generating function of the linear multistep method. Thus for the m^{th} node of the interval we have an expression for the convolution weights with u as the generating function of the linear multistep method.

$$W_m = -\left(\frac{1}{u^\alpha - \lambda h^\alpha}\right)_0 \sum_{j=1}^{m} (u^\alpha)_j W_{m-j} = -\frac{1}{((u)_0)^\alpha - \lambda h^\alpha} \sum_{j=1}^{m} (u^\alpha)_j W_{m-j}$$

Each convolution weight for a node is found in terms of the convolution weights of the previous nodes of the interval and the Taylor coefficient of the power function $(u^\alpha)_j$. The formula (4.6) given above enables us to find the Taylor coefficient of the power function $(u^\alpha)_j$ at each of the previous nodes. The initial convolution weight W_0 is found from the expression

$$W_0 = \left(\frac{h^\alpha}{u^\alpha - \lambda h^\alpha}\right)_0 = \left(\frac{h^\alpha}{((u)_0)^\alpha - \lambda h^\alpha}\right).$$

The starting weights are found by solving equation (3.15), which we repeat here.

$$\sum_{j=j_0}^{q-1} W_{n,j}(x_j)^{k+\gamma-1} = \Gamma(k+\gamma)\tilde{E}(k+\alpha+\gamma;x_m) - \sum_{j=0}^{m} W_{m-j}(x_j)^{k+\gamma-1} \tag{4.7}$$

We stated in Chapter 3 that $\tilde{E}(k+\alpha+\gamma;x_m)$ can be approximated by using a convolution quadrature to find a weight. This weight is the Taylor coefficient of the Laplace transform $F(\beta;s)$ with $\beta = k+\alpha+\gamma$. Dividing this weight by the stepsize h then gives us the approximation of the inverse Laplace transform of $F(\beta;s)$. As $\tilde{E}(\beta;s)$ is the inverse Laplace transform of $F(\beta;s)$ we then have the desired approximated value.
The complete expression for $F(\beta;s)$, taking into account our equation, is

$$F(\beta;s) = \frac{s^{\alpha-\beta}}{s^\alpha - \lambda}.$$

So to approximate $\tilde{E}(k+\alpha+\gamma;x_m)$ we need to find an expression for the k^{th} Taylor coefficient of

$$\frac{(u/h)^{\alpha-\beta}}{(u/h)^\alpha - \lambda} = \frac{h^\beta (u)^{\alpha-\beta}}{u^\alpha - \lambda h^\alpha}$$

where $s = u(x)/h$ and $u \equiv u(x)$ as before.
We now use the product property $(u \cdot v)_k$ with

$$u = \frac{h^\beta u^{\alpha-\beta}}{u^\alpha - \lambda h^\alpha} \quad \text{and} \quad v = u^\alpha - \lambda h^\alpha$$

to give us the expression

$$\left(\frac{h^\beta u^{\alpha-\beta}}{u^\alpha - \lambda h^\alpha} \cdot (u^\alpha - \lambda h^\alpha)\right)_k = \sum_{j=0}^{k} (u^\alpha - \lambda h^\alpha)_j \left(\frac{h^\beta u^{\alpha-\beta}}{u^\alpha - \lambda h^\alpha}\right)_{k-j}.$$

Extracting the $j = 0$ term we then have

$$\left(h^\beta u^{\alpha-\beta}\right)_k = \sum_{j=1}^{k} (u^\alpha - \lambda h^\alpha)_j \left(\frac{h^\beta u^{\alpha-\beta}}{u^\alpha - \lambda h^\alpha}\right)_{k-j} + (u^\alpha - \lambda h^\alpha)_0 \left(\frac{h^\beta u^{\alpha-\beta}}{u^\alpha - \lambda h^\alpha}\right)_k .$$

Rearranging the terms and dividing throughout by $(u^\alpha - \lambda h^\alpha)_0$

$$\left(\frac{h^\beta u^{\alpha-\beta}}{u^\alpha - \lambda h^\alpha}\right)_k = \frac{1}{(u^\alpha - \lambda h^\alpha)_0} \left[\left(h^\beta u^{\alpha-\beta}\right)_k - \sum_{j=1}^{k} (u^\alpha - \lambda h^\alpha)_j \left(\frac{h^\beta u^{\alpha-\beta}}{u^\alpha - \lambda h^\alpha}\right)_{k-j}\right].$$

Now h^β is a constant and as $(\lambda h^\alpha)_j = 0$ for $j \geq 1$ we then have

$$\left(\frac{h^\beta u^{\alpha-\beta}}{u^\alpha - \lambda h^\alpha}\right)_k = \frac{1}{((u)_0)^\alpha - \lambda h^\alpha} \left[h^\beta \left(u^{\alpha-\beta}\right)_k - \sum_{j=1}^{k} (u^\alpha)_j \left(\frac{h^\beta u^{\alpha-\beta}}{u^\alpha - \lambda h^\alpha}\right)_{k-j}\right].$$

We can use this expression to calculate the value of the Laplace transform $F(\beta; s)$ for the m^{th} node with the appropriate substitution of $\beta = k + \alpha + \gamma$.
Thus we can approximate $\tilde{E}(k + \alpha + \gamma; x_m)$ for the m^{th} node by taking this value and dividing by h.
The starting weights for each node can be found by solving the system of equations (4.7).
We can now evaluate the solution of the differential equation (4.4) at each node using the formula (4.5) with the convolution and starting weight values and the value of the function $f(x_j)$ at each node.
The solution is computed by the Scilab program detailed in Appendix H, and the results are given in Chapter 5.

Chapter 5

Results and Analysis

Aim

Here we present the results obtained for the solutions of the equation by the different methods described in Chapter 3 and implemented by the numerical methods of Chapter 4.

1 Diethelms BDF method - Results for equation 4.2

The equation for our numerical solution is

$$D_{0*}^{0.5}y(x) = -y(x) + x^2 + \frac{2}{\Gamma(2.5)}x^{1.5}$$

with the initial conditions $y(0) = 0$ and $y^{'}(0) = 0$, and we evaluate the expression y_m for the each interval in turn of the required number of intervals (10, 20, 30 etc.) by means of equation (4.2).
The results are obtained for a range of interval sizes from 10 to 200. As we are seeking a solution for the value $x = 1$ this corresponds to a range of stepsize h of 0.1 to 0.005.
The results that are calculated for each interval size are given in the table in Appendix B, and shown in graphical form as Figure 5.1. The execution times for each set of intervals are shown as Figure 5.2, and are also included in Appendix B.
We can see from the graph that the results are stable, consistent, convergent and asymptotic.

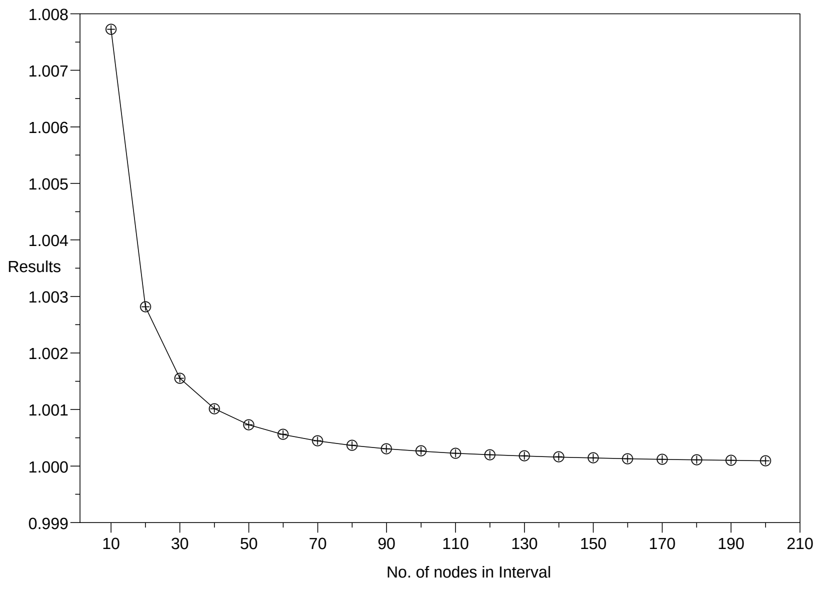

Figure 5.1: Diethelm's BDF method - results for x=1

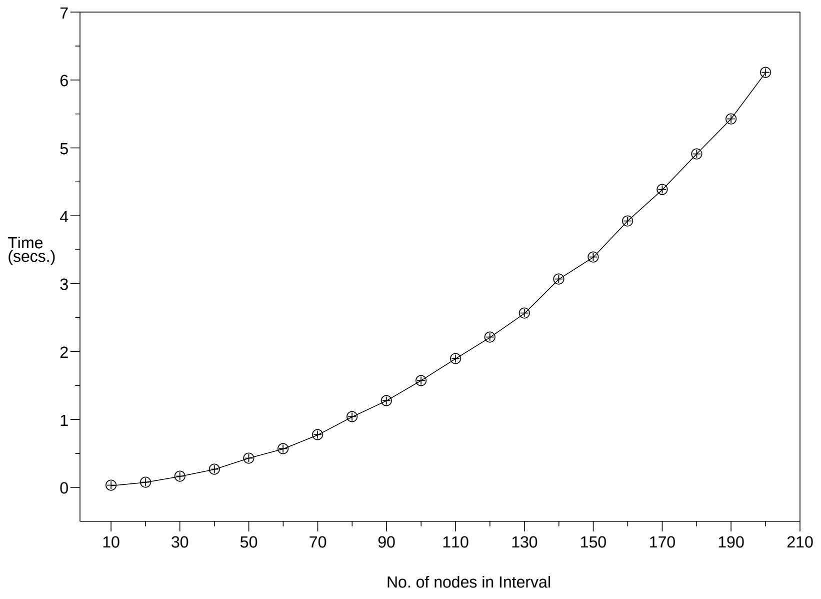

Figure 5.2: Diethelm's BDF method - execution times for x=1

2 Lubich's CQ method - Results for equation 4.3

We are solving the same equation as in the previous section, namely

$$D_{0*}^{0.5}y(x) = -y(x) + x^2 + \frac{2}{\Gamma(2.5)}x^{1.5}.$$

with the initial conditions $y(0) = 0$ and $y^{'}(0) = 0$.
We evaluate the expression y_m for each interval in turn of the required number of intervals (10, 20, 30 etc.) by means of equation (4.3).
The results are obtained for a range of interval sizes from 10 to 200. As we are seeking a solution for the value $x = 1$ this corresponds to a range of stepsize h of 0.1 to 0.005.
The results are calculated for various BDF methods from BDF1 to BDF6, and are shown in Appendix C as approximation errors compared with the exact solution at $x = 1$.
The graphical form of the numerical results is shown in Figure 5.3 for BDFp, where p is the order of the BDF method and p=1,2,3,4,5,6.
The execution times for each set of intervals are shown as Figure 5.2, and are also included in Appendix B.
The results for BDF6 at interval size 10 are not computed, as the value s to calculate the first s values is greater than the interval size.
We can see from both the actual data and the graphical representations that the numerical approximations converge quickly for increasing interval size for the BDF2 to BDF6 methods. Graphically we are unable to separate the methods at the higher interval sizes. For the method BDF1 there is a perceptive difference between the numerically approximated values, even at the higher interval sizes. The BDF1 method still exhibits convergent and asymptotic behaviour.
The execution times for the different BDF methods show consistent results for increasing interval size, as would be expected from the larger number of calculations, particularly for starting weights.

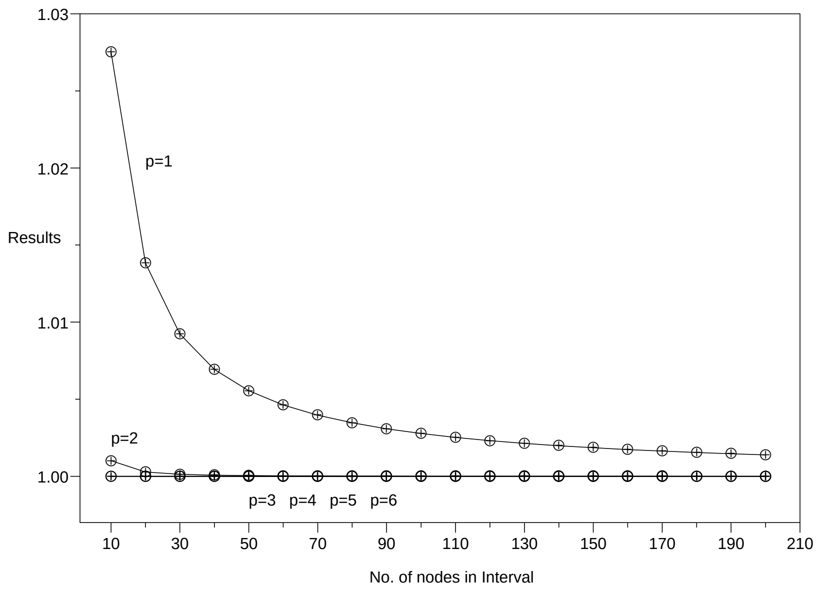

Figure 5.3: Lubich's convolution quadrature method - results for x=1

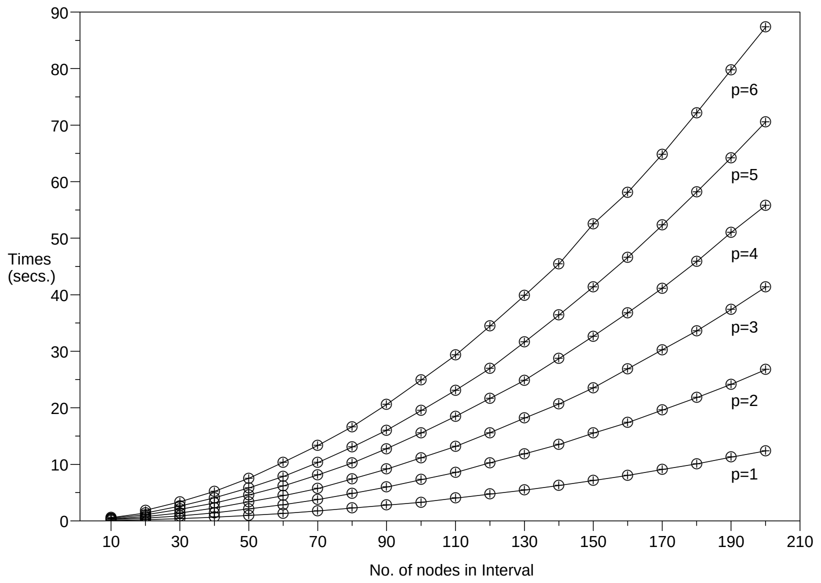

Figure 5.4: Lubich's Convolution quadrature method - execution times for x=1

3 Luchko and Diethelm's OC method - Results for equation 4.4

The equation we are solving is the same equation as in the previous two sections, namely

$$D_{0*}^{0.5}y(x) = -y(x) + x^2 + \frac{2}{\Gamma(2.5)}x^{1.5}.$$

with the initial conditions $y(0) = 0$ and $y^{'}(0) = 0$.
We evaluate the expression y_m for each interval in turn of the required number of intervals by means of equation (4.4).
The results are obtained for a range of interval sizes from 10 to 100. As we are seeking a solution for the value $x = 1$ this corresponds to a range of stepsize h of 0.1 to 0.01.
We restrict our selection of interval sizes because of the long execution time of the program. For the same reason, we do not include the execution times of the program for the range of intervals.
We show the results for the BDF methods BDF2 and BDF3.
For BDF2 we have the following table of results for the interval sizes 10, 20, 40, and 100 corresponding to stepsizes of 0.1, 0.05, 0.025 and 0.01 respectively.
The notation $1.23(-12)$ means 1.23×10^{-12}.

Intervals	Stepsize	Error
10	0.10	1.44(-3)
20	0.05	3.57(-4)
40	0.025	8.91(-5)
100	0.01	1.42(-5)

Table 5.1: Luchko and Diethelm's OC method - BDF2 - errors for x=1

For BDF3 we only have the results for the interval sizes 10, 20 and 40, again due to the long execution time of the program.

Intervals	Stepsize	Error
10	0.10	-3.22(-4)
20	0.05	-4.35(-5)
40	0.025	-6.2 (-6)

Table 5.2: Luchko and Diethelm's OC method - BDF3 - errors for x=1

Chapter 6

Summary and Conclusion

Aim

We summarize the dissertation and comment on the results obtained for the different numerical methods.
Finally, we give our conclusions.

1 Summary

The early chapters describe the evolution of Fractional Calculus, and the establishment of basic definitions for fractional integral and differential operators.
Next we describe the three numerical methods to be investigated in the dissertation.

1. Diethelm's Backwards Difference Forms method.

2. Lubich's Convolution Quadrature method.

3. Luchko and Diethelm's Operational Calculus method.

We develop formulas to express the solutions of these numerical solutions.
We write programs to calculate the solutions for each numerical method, and we discover and tabulate the results.
Finally we compare the results in terms of accuracy and efficiency.

Commentary on results.

Diethelm's BDF method produced results that were consistent and convergent for decreasing stepsize (increasing number of intervals) over a wide range of stepsizes. The results became more accurate as the stepsize decreased. The method exhibited a performance that showed a smooth relationship with the stepsize. From the graph (Figure 5.2) this relationship is observed to be non-linear.

Lubich's Convolution Quadrature method produced results whose accuracy increased with decreasing stepsize. We note that the results for the linear multistep methods BDF2 and above were significantly more accurate than the BDF1 method. Each implementation of the different BDF methods exhibited a smooth non-linear relationship of performance with stepsize. We note that for each particular stepsize the relative spacing of the execution times is reasonably constant. The greater complexity of the higher BDF methods gave results that were more accurate.

Luchko and Diethelm's Operational Calculus method produced results that were of the same order of accuracy as Lubich's Convolution Quadrature method. We produce results only for the BDF2 and BDF3 linear multistep methods over a small set of stepsizes, due to the long execution time of the Scilab program. This point is elaborated upon in the Conclusion section of this chapter. The results demonstrate an increasing accuracy with decreasing stepsize.

2 Conclusion

Diethelm's BDF method is the most efficient method in that it has the least execution time, but it is the least accurate. It also has the restriction that it can only be used to solve simple fractional differential equations whose order is less than two. However due to the relative simplicity of the method, it is the easiest Scilab program to write.

Lubich's convolution quadrature method is more accurate than the BDF method of Diethelm. However it is not as efficient as Diethelm's BDF method, with the Scilab program taking longer to execute. The overall execution time is acceptable for reasonable values of the stepsize. It does cope with a wider range of order of a fractional differential equation than Diethelm's BDF method, providing of course that there exists a solution to that fractional differential equation.

Luchko and Diethelm's method is of the same order of accuracy as Lubich's convolution quadrature method for the reduced set of stepsizes implemented. However our implementation of the method is very inefficient as the execution time of the Scilab program was very long. The great advantage of Luchko and Diethelm's method is its application to more general types of fractional differential equations, containing several terms of fractional derivatives of different order. This outweighs its more complex approach in that a wide range of fractional differential equations can be solved numerically.

We feel we need to comment on our implementation of this method, as we believe there is scope for improvement in the performance of the Scilab program. The program is written in its primitive form in order to get the Operational Calculus method working, and we believe it requires a serious re-examination of its structure. The program would benefit being rewritten to eliminate repeated calculations, and possibly moving some calculations outside embedded loops. In addition the functions that calculate the recursive Taylor Series coefficients, **mitt2tc** and **powtc**, can be rewritten to eliminate duplicated calculations. The calculations of starting weights involve calling the function **mitt2tc**, and it may be possible to rewrite this calculation to pre-calculate the Taylor Series coefficients, so that they are not repeatedly evaluated for each node of the working interval. The reason these modifications have not been implemented has been the deadline to finish the dissertation, coupled with the programmer's maxim no. 1 that I have learnt over many years in writing computer programs; "Never change a working program".

Appendices

A Totals of published papers 1975 - 2006

Year	Annual Total	Running Total
1975	0	0
1976	4	4
1977	2	6
1978	1	7
1979	3	10
1980	0	10
1981	2	12
1982	0	12
1983	4	16
1984	3	19
1985	5	24
1986	3	27
1987	5	32
1988	3	35
1989	5	40
1990	5	45
1991	11	56
1992	13	69
1993	10	79
1994	10	89
1995	20	109
1996	20	129
1997	18	147
1998	24	171
1999	30	201
2000	32	233
2001	41	274
2002	60	334
2003	65	399
2004	68	467
2005	75	542
2006	79	621

B Diethelms BDF method - results for equation 4.2

The results are for x=1 in equation (4.2).

The times are in seconds.

Intervals	Stepsize	Value	Times
10	0.10	1.0077248	0.023
20	0.05	1.0028153	0.074
30	$0.0333\dot{3}$	1.001552	0.161
40	0.025	1.0010155	0.265
50	0.02	1.0007302	0.429
60	$0.01\dot{6}$	1.0005575	0.566
70	0.01428...	1.0004436	0.770
80	0.0125	1.0003639	1.037
90	$0.0111\dot{1}$	1.0003055	1.277
100	0.01	1.0002613	1.573
110	$0.009\dot{0}\dot{9}$	1.0002268	1.894
120	$0.0083\dot{3}$	1.0001993	2.208
130	0.00769...	1.0001769	2.560
140	0.00714	1.0001584	3.067
150	$0.0066\dot{6}$	1.000143	3.390
160	0.00625	1.0001299	3.923
170	0.00588	1.0001187	4.382
180	$0.0055\dot{5}$	1.000109	4.911
190	0.00526...	1.0001005	5.426
200	0.005	1.0000931	6.109

C Lubich's CQ method - results for equation 4.3

Error values - BDFp - x=1							
Intervals	Stepsize	p=1	p=2	p=3	p=4	p=5	p=6
10	0.10	2.75(-2)	1.01(-3)	0.00(-16)	-3.33(-16)	-1.66(-15)	-
20	0.05	1.38(-2)	2.79(-4)	0.00(-16)	2.22(-16)	2.66(-15)	3.53(-08)
30	$0.0333\dot{3}$	9.23(-3)	1.29(-4)	-3.33(-16)	0.00(-16)	-5.55(-16)	1.08(-08)
40	0.025	6.93(-3)	7.48(-5)	0.00(-16)	0.00(-16)	1.11(-15)	4.66(-09)
50	0.02	5.54(-3)	4.87(-5)	-1.11(-16)	2.22(-16)	6.66(-16)	2.43(-09)
60	$0.01\dot{6}$	4.62(-3)	3.43(-5)	2.22(-16)	6.66(-16)	2.22(-16)	1.44(-09)
70	0.01428.	3.96(-3)	2.55(-5)	0.00(-16)	4.44(-16)	0.00(-16)	9.31(-10)
80	0.0125	3.47(-3)	1.96(-5)	-4.44(-16)	-3.33(-16)	-2.22(-16)	6.40(-10)
90	$0.0111\dot{1}$	3.08(-3)	1.56(-5)	-4.44(-16)	-4.44(-16)	-2.22(-16)	4.61(-10)
100	0.01	2.77(-3)	1.27(-5)	-4.44(-16)	0.00(-16)	2.22(-16)	3.44(-10)
110	$0.009\dot{0}\dot{9}$	2.52(-3)	1.06(-5)	-1.11(-16)	-5.55(-16)	-1.11(-16)	2.65(-10)
120	$0.0083\dot{3}$	2.31(-3)	8.90(-6)	-6.66(-16)	0.00(-16)	-5.55(-16)	2.09(-10)
130	0.00769.	2.13(-3)	7.60(-6)	2.22(-16)	0.00(-16)	6.66(-16)	1.68(-10)
140	0.00714	1.98(-3)	6.60(-6)	-6.66(-16)	-5.55(-16)	-4.44(-16)	1.37(-10)
150	$0.0066\dot{6}$	1.85(-3)	5.80(-6)	4.44(-16)	6.66(-16)	4.44(-16)	1.14(-10)
160	0.00625	1.73(-3)	5.10(-6)	0.00(-16)	4.44(-16)	-1.11(-16)	9.59(-11)
170	0.00588	1.63(-3)	4.50(-6)	6.66(-16)	2.22(-16)	2.22(-16)	8.13(-11)
180	$0.0055\dot{5}$	1.54(-3)	4.00(-6)	0.00(-16)	0.00(-16)	6.66(-16)	6.97(-11)
190	0.00526.	1.46(-3)	3.60(-6)	2.22(-16)	2.22(-16)	2.22(-16)	5.95(-11)
200	0.005	1.38(-3)	3.30(-6)	-4.44(-16)	-1.11(-16)	-5.55(-16)	5.13(-11)

The notation $1.23(-12)$ means 1.23×10^{-12}.

D Lubich's CQ method - times for equation 4.3

Timing values (secs.) - BDFp - x=1							
Intervals	Stepsize	p=1	p=2	p=3	p=4	p=5	p=6
10	0.10	0.08	0.25	0.30	0.44	0.56	-
20	0.05	0.20	0.48	0.75	1.06	1.37	1.86
30	$0.0333\dot{3}$	0.38	0.88	1.39	2.03	2.59	3.36
40	0.025	0.64	1.41	2.26	3.13	4.07	5.21
50	0.02	0.96	2.12	3.37	4.59	5.83	7.53
60	$0.01\dot{6}$	1.31	2.86	4.43	6.17	7.84	10.36
70	0.01428.	1.76	3.75	5.70	8.11	10.29	13.33
80	0.0125	2.26	4.83	7.40	10.20	13.04	16.65
90	$0.0111\dot{1}$	2.78	5.95	9.14	12.75	16.01	20.57
100	0.01	3.29	7.26	11.15	15.55	19.50	24.91
110	$0.009\dot{0}\dot{9}$	4.00	8.56	13.16	18.48	23.08	29.30
120	$0.0083\dot{3}$	4.73	10.25	15.58	21.65	26.95	34.44
130	0.00769.	5.43	11.86	18.18	24.84	31.60	39.86
140	0.00714	6.25	13.49	20.66	28.68	36.39	45.43
150	$0.0066\dot{6}$	7.13	15.50	23.52	32.64	41.37	52.51
160	0.00625	8.03	17.38	26.88	36.79	46.57	58.07
170	0.00588	9.08	19.58	30.25	41.15	52.26	64.83
180	$0.0055\dot{5}$	10.08	21.80	33.59	45.84	58.16	72.17
190	0.00526.	11.25	24.13	37.35	50.99	64.20	79.76
200	0.005	12.37	26.75	41.35	55.79	70.56	87.32

E Diethelms BDF method - Scilab program

This program is for equation (4.2)

```
// John Landy
//
// Date: January 2009
//
// Dissertation: Fractional Differential Equations and Numerical Solutions
//
// Program written for SciLab package
//
// Abstract:
// Evaluate numerical solutions for Fractional Differential Equations
// Equation (4.2)
//
// Diethelm backward difference method
//
// Number of interval sizes
M=20;
//Int=linspace(0.0, M, M+1); // range of interval sizes
// Set up interval sizes (10,20,...) and
// initialise times for interval sizes
for i=1:M
Int(i)=i*10;
T(i)=0;
end; // i
// Order of equation
alpha=0.5;
// Variable value to evaluate
V=1;       // Value
// Accumulate timings for loop of 20 to smooth out variations in cpu time
//for m=1:20
for m=1:1
// Find solution for each interval size
```

```
for i=1:M;
N=Int(i);                      // number of subintervals
q=linspace(0.0, V, N+1);  // range of q
tj=linspace(0.0, V, N+1); // range of t_j
X=linspace(0.0, V, N+1);  // range of interval
Yj=linspace(0.0, V, N+1); // range of answer
betakj=linspace(0.0, V, N+1); // range of beta_kj
// Multistep method
h=V/N;    // Calculate stepsize
// Initial conditions
Yj(1)=0; // y(0)=0
// Set up common term used in calculation
term=(h^(alpha))*gamma(2-alpha);
//
// Set up tj values for each node
for k=0:N
tj(k+1)=(k)*h;
end; //k
for k=0:N
q(k+1)=((tj(k+1))^2)+(2/gamma(2.5))*(tj(k+1)^1.5);
end; //k
// Calculate betakj weights for the N+1 nodes (N intervals)
betakj(1)=1; //k=0
betasum=1;
for k=1:(N-1);
betakj(k+1)=((k-1)^(1-alpha)+(k+1)^(1-alpha)-2*(k)^(1-alpha));
end; //k
k=N;
betakj(N+1)=(k-1)^(1-alpha)-(k)^(1-alpha)-(alpha-1)*(k)^(-alpha); //k=N
beta0j=1; //k=0
// Evaluate each Yj in turn for N intervals
timer(); // start timer
for j=1:N;
// Calculate betasum with previous Yj values
```

```
betasum=0;
for k=1:j;
betasum=betasum+betakj(k+1)*Yj(j-k+1);
end; //k
Num=term*q(j+1)-betasum;
Den=term+beta0j;
Yj(j+1)=Num/Den; // calculate Yj for interval
end; //j
Yint(i)=Yj(j+1); // save final Y value for the interval size
T(i)=T(i)+(timer()); // stop timer, update time for interval size
Yerr(i)=Yint(i)-V*V; // calculate error for q(x) as stated
end; //i
end; //m

// Open file for interval sizes data
u=file('open','c:\chester college\thesis\programs\de01\dintsize.txt'...
,'unknown',"unformatted");
writb(u,Int);
file('close',u);

// Open file for interval values data
u=file('open','c:\chester college\thesis\programs\de01\dintvals.txt'...
,'unknown',"unformatted");
writb(u,Yint);
file('close',u);

// Open file for interval errors data
u=file('open','c:\chester college\thesis\programs\de01\dinterrs.txt'...
,'unknown',"unformatted");
writb(u,Yerr);
file('close',u);

// Open file for interval times data
u=file('open','c:\chester college\thesis\programs\de01\dinttime.txt'...
,'unknown',"unformatted");
writb(u,T);
file('close',u);

disp("finished")
```

F Lubich's CQ method - Scilab program

This program is for equation (4.3).
This program calculates the approximated values using $f(x_m)$ at previous nodes.

```
// John Landy
//
// Date: February 2009
//
// Dissertation: Fractional Differential Equations and Numerical Solutions
//
// Program written for SciLab package
//
// Abstract:
// Evaluate numerical solutions for Fractional Differential Equations
// Equation (4.3)
//
// Lubich convolution quadrature method BDFp - function method
//

// Order of BDF
p=2;

// Number of interval sizes
M=20;
// Set up interval sizes (10,20,...) and
// initialise times for interval sizes
for i=1:M
Int(i)=i*10;
T(i)=0;
end; // i

if p==6
// Set up interval sizes (20,...) and
// initialise times for interval sizes
```

```
for i=2:M
Int(i-1)=i*10;
T(i-1)=0;
end; // i
M=M-1;
else
// Set up interval sizes (10,20,...) and
// initialise times for interval sizes
for i=1:M
Int(i)=i*10;
T(i)=0;
end; // i
end; // p=6

// Order of equation - less than one
alpha=0.5;
// Variables
//V=10;       // Value
V=1;       // Value

// Calculate gamma values as a set for starting weights using p and alpha
i=1;
if p<>1
j=0;
k=0;
gam=0;
for r=1:(p-1)
while gam<r
g(i)=k+j*alpha;
gam=k+(j+1)*alpha;
j=j+1;
i=i+1;
end; // gam<r
k=k+1;
```

```
j=0;
end; // r
end; // if p<>1
g(i)=p-1;

// Accumulate timings for loop of 20 to smooth out variations in cpu time
//for m=1:20
for m=1:1
// Find solution for each interval size
//for i=1:M
//for i=1:6
for i=1:1 // temp
//N=10;
N=20;
//N=Int(i);                      // number of subintervals
q=linspace(0.0, V, N+1);  // range of q
tj=linspace(0.0, V, N+1); // range of t_j
X=linspace(0.0, V, N+1);  // range of interval
Ym=linspace(0.0, V, N+1); // range of answer
// Multistep method
h=V/N;    // Calculate stepsize
// Initial conditions
Ym(1)=0; // y(0)=0

// Set up x=tj values for (N+1) nodes
timer(); // start timer
for k=0:N
tj(k+1)=(k)*h;
end; //k
// Calculate q(x) for equation at each node
for k=0:N
q(k+1)=((tj(k+1))^2)+(2/gamma(3-alpha))*(tj(k+1)^(2-alpha));
end; //k
```

```
T(i)=T(i)+(timer()); // stop timer, update time for interval size

// Calculate aggregated Taylor polynomial coefficients
x=poly(0,"x");
a=0;
for k=1:p
a=a+((1-x)^k)/k;
end; //end k
un=coeff(a);

alpha=+0.5;

timer(); // start timer
// Calculate convolution weights for each interval node
cw(1)=1/(un(1)^alpha); // Interval node 0 - general
// Interval nodes 1 to p
for k=1:p
cw(k+1)=0;
for j=0:(k-1)
cw(k+1)=cw(k+1)+(-alpha*(k-j)-j)*cw(j+1)*un(k-j+1);
end; // j
cw(k+1)=cw(k+1)/(k*un(1)); // original
//cw(k+1)=cw(k+1)/(k*((un(1)^(alpha))));
end; // k
// Interval nodes p+1 to N
for k=(p+1):N
cw(k+1)=0;
for j=(k-p):(k-1)
cw(k+1)=cw(k+1)+(-alpha*(k-j)-j)*cw(j+1)*un(k-j+1);
end; // j
cw(k+1)=cw(k+1)/(k*un(1));
end; // k
T(i)=T(i)+(timer()); // stop timer, update time for interval size
```

```
alpha=+0.5;

// Calculate starting weights for each interval
// Number of starting weights (s+1) at each interval node
s=size(g,'r')-1;
// Starting weights for all interval nodes
sw=zeros(N,s+1);
// Starting weights for current interval node
w=zeros(1,s+1);
// Calculate (s+1) starting weights for each interval
// Calculate (s+1) terms for the convolution weights
//                 (for starting weights) and store in array
// Calculate (s+1) gamma terms for the interval node
// Set up the calculation arrays for each interval
rA=zeros(s+1,1);
RR=zeros(s+1,1);
timer(); // start timer
// Calculate starting weights in turn for N interval nodes
for k=1:N
// Calculate (s+1) starting weights for each interval node
rB=zeros(s+1,1); // zero the convolution weight terms sum for the interval
for ja=1:(s+1)
for j=1:k // original
rB(ja,1)=rB(ja,1)+cw(k-j+1)*j^(g(ja)); // calc convolution weight terms
end; //j
for jb=1:(s+1)
lB(ja,jb)=(jb-1)^(g(ja)); // set up starting weight gamma power terms
//lB(ja,jb)=(jb)^(g(ja)); // set up starting weight gamma power terms
end; // jb
// set up interval gamma terms
rA(ja,1)=gamma(1+g(ja))*k^(g(ja)+alpha)/gamma(1+g(ja)+alpha);
RR(ja,1)=rA(ja,1)-rB(ja,1); // calculate RHS of equation
end; //ja
w=lB\RR; // calculate starting weights for the current interval node
```

```
// Save starting weights for current interval node
for j=1:(s+1)
sw(k,j)=w(j);
end; // j
end; // k

T(i)=T(i)+(timer()); // stop timer, update time for interval size

alpha=+0.5;

//******** COMMENT OUT

//if p<>1
//// The s expressions for Yn (n=1,...,s) contain the unknowns Y1 to Ys
//// Thus each Ym for a node cannot be determined on a step by step basis
//// A linear system of s equations with s unknowns is solved instead
//A=zeros(s,s);
//B=zeros(s,1);
//y=zeros(s,1);
//timer(); // start timer
//// For all equations in the linear system
//for u=0:(s-1)
//// Calculate  coefficients of Yn (n=1,...,s) for LHS of equation
//for v=0:(s-1)
//if v==u then
//   A(u+1,v+1)=cw(1)-sw(u+1,v+2)+h^(-alpha);
//elseif v<u then
//   A(u+1,v+1)=cw(u-v+1)-sw(u+1,v+2);
//else
//   A(u+1,v+1)=-sw(u+1,v+2);
//end; // if
//end; // v
//// Calculate RHS terms
//qsum=0;
```

```
//for v=0:(u+1)
//qsum=qsum+cw(u-v+2)*q(v+1);
//end; // v
//B(u+1,1)=(cw(u+2)-sw(u+1,1))*Ym(1)-qsum;
//end; // u
////Calculate Y1 to Ys by solving Ay+B=0
//[y,x]=linsolve(A,B);
//// Save the Y1 to Ys values in Ym for the s nodes
//for k=1:s
//Ym(k+1)=y(k);
//end; // k
//end; // if p<>1

//*****************

alpha=+0.5;

// Evaluate each Ym in turn for remaining interval nodes
//for k=(s+1):N // orig
for k=1:N
// Calculate convolution weight terms total for interval node
cwsum=0;
for j=0:k-1
cwsum=cwsum+cw(k-j+1)*(q(j+1)-Ym(j+1));
end; //j
// Calculate starting weight terms total for interval node
swsum=0;
for j=0:s
swsum=swsum+sw(k,j+1)*(q(j+1)-Ym(j+1)); // original
end; //j
Ym(k+1)=(h^alpha)*(cw(1)*q(k+1)+cwsum+swsum)/(1+cw(1)*h^alpha); // orig
//Ym(k+1)=(h^alpha)*(cw(1)*q(k+1)+cwsum)/(1+cw(1)*h^alpha);
end; //k
```

```
Yint(i)=Ym(k+1); // save final value for interval
T(i)=T(i)+(timer()); // stop timer, update time for interval size

alpha=+0.5;

end; // i interval size
end; // m smooth cpu timing loop

// Open file for interval sizes data
u=file('open','c:\chester college\thesis\programs\de20\dintsize.txt'...
,'unknown',"unformatted");
writb(u,Int);
file('close',u);

// Open file for interval values data
u=file('open','c:\chester college\thesis\programs\de20\dintvals.txt'...
,'unknown',"unformatted");
writb(u,Yint);
file('close',u);

// Open file for interval times data
u=file('open','c:\chester college\thesis\programs\de20\dinttime.txt'...
,'unknown',"unformatted");
writb(u,T);
file('close',u);

disp("finished")
```

G Lubich's CQ method - Scilab program

This program is for equation (4.3).
This program calculates the approximated values using $y(x_m)$ at previous nodes.

```
// John Landy
//
// Date: February 2009
//
// Dissertation: Fractional Differential Equations and Numerical Solutions
//
// Program written for SciLab package
//
// Abstract:
// Evaluate numerical solutions for Fractional Differential Equations
// Equation (4.3)
//
// Lubich convolution quadrature method BDFp - approximated y method
//

lines(0);

// Order of BDF
p=2;

// Number of interval sizes
M=20;

if p==6
// Set up interval sizes (20,...) for p=6 and
// initialise times for interval sizes
for i=2:M
Int(i-1)=i*10;
T(i-1)=0;
end; // i
```

```
M=M-1;
else
// Set up interval sizes (10,20,...) for p=1 to 5 and
// initialise times for interval size
for i=1:M
Int(i)=i*10;
T(i)=0;
end; // i
end; // if p==6

// Order of equation - less than one
alphaorig=0.5;
// Variables
V=1;       // Value

// Set alpha positive
alpha=+alphaorig;
//
// Calculate gamma values as a set using p and alpha
// for later starting weight caclulation
i=1;
if p<>1
j=0;
k=0;
gam=0;
for r=1:(p-1)
while gam<r
g(i)=k+j*alpha;
gam=k+(j+1)*alpha;
j=j+1;
i=i+1;
end; // gam<r
k=k+1;
j=0;
```

```
end; // r
end; // if p<>1
g(i)=p-1;

// Find solution for each interval size
//for i=1:M
for i=1:1
//N=Int(i);                      // number of subintervals
N=10;
q=linspace(0.0, V, N+1);  // range of q
tj=linspace(0.0, V, N+1); // range of t_j
X=linspace(0.0, V, N+1);  // range of interval
Ym=linspace(0.0, V, N+1); // range of answer
// Multistep method
h=V/N;    // Calculate stepsize
// Initial conditions
Ym(1)=0; // y(0)=0

// Set up x=tj values for (N+1) nodes
timer(); // start timer
for k=0:N
tj(k+1)=(k)*h;
end; //k
// Calculate q(x) for equation at each node - alpha positive
for k=0:N
q(k+1)=((tj(k+1))^2)+(2/gamma(3-alpha))*(tj(k+1)^(2-alpha));
end; //k

T(i)=T(i)+(timer()); // stop timer, update time for interval size

// Calculate aggregated Taylor polynomial coefficients
x=poly(0,"x");
a=0;
for k=1:p
```

```
a=a+((1-x)^k)/k;
end; //end k
un=coeff(a);

// Set alpha negative for convolution weight and
// starting weight calculation
alpha=-alphaorig;

timer(); // start timer
// Calculate convolution weights for each interval node
cw(1)=1/(un(1)^alpha); // Interval node 0 - general
// Interval nodes 1 to p
for k=1:p
cw(k+1)=0;
for j=0:(k-1)
cw(k+1)=cw(k+1)+(-alpha*(k-j)-j)*cw(j+1)*un(k-j+1);
end; // j
cw(k+1)=cw(k+1)/(k*un(1)); // original
end; // k
// Interval nodes p+1 to N
for k=(p+1):N
cw(k+1)=0;
for j=(k-p):(k-1)
cw(k+1)=cw(k+1)+(-alpha*(k-j)-j)*cw(j+1)*un(k-j+1);
end; // j
cw(k+1)=cw(k+1)/(k*un(1)); // original
end; // k
T(i)=T(i)+(timer()); // stop timer, update time for interval size

// Keep alpha negative for starting weights calculation
//
// Calculate starting weights for each interval
// Number of starting weights (s+1) at each interval node
s=size(g,'r')-1;
```

```
// Starting weights for all interval nodes
sw=zeros(N,s+1);
// Starting weights for current interval node
w=zeros(1,s+1);
// Calculate (s+1) starting weights for each interval
// Calculate (s+1) terms for the convolution weights
//                 (for starting weights) and store in array
// Calculate (s+1) gamma terms for the interval node
// Set up the calculation arrays for each interval
rA=zeros(s+1,1);
RR=zeros(s+1,1);
timer(); // start timer
// Calculate starting weights in turn for N interval nodes
for k=1:N
// Calculate (s+1) starting weights for each interval node
rB=zeros(s+1,1); // zero the convolution weight terms sum for the interval
for ja=1:(s+1)
for j=1:k
//for j=0:k
rB(ja,1)=rB(ja,1)+cw(k-j+1)*j^(g(ja)); // calc convolution weight terms
end; //j
for jb=1:(s+1)
lB(ja,jb)=(jb-1)^(g(ja)); // set up starting weight gamma power terms
end; // jb
 // set up interval gamma terms
rA(ja,1)=gamma(1+g(ja))*k^(g(ja)+alpha)/gamma(1+g(ja)+alpha);
RR(ja,1)=rA(ja,1)-rB(ja,1); // calculate RHS of equation
end; //ja
w=lB\RR; // calculate starting weights for the current interval node

// Save starting weights for current interval node
for j=1:(s+1)
sw(k,j)=w(j);
end; // j
```

```
end; // k

T(i)=T(i)+(timer()); // stop timer, update time for interval size

// Set alpha back to positive for Ym calculation at all nodes of interval
alpha=+alphaorig;

// Y1 to Ys values not calculated for p=1
//
if p<>1
// The s expressions for Yn (n=1,...,s) contain the unknowns Y1 to Ys
// Thus each Ym for a node cannot be determined on a step by step basis
// A linear system of s equations with s unknowns is solved instead
A=zeros(s,s);
B=zeros(s,1);
y=zeros(s,1);
timer(); // start timer
// For all equations in the linear system
for u=0:(s-1)
// Calculate  coefficients of Yn (n=1,...,s) for LHS of equation
for v=0:(s-1)
if v==u then
   A(u+1,v+1)=sw(u+1,v+2)+cw(1)+h^(alpha);
elseif v<u then
   A(u+1,v+1)=cw(u-v+1)+sw(u+1,v+2); // original
else
   A(u+1,v+1)=sw(u+1,v+2); // original
end; // if
end; // v
// Calculate RHS terms
B(u+1,1)=(cw(u+2)+sw(u+1,1))*Ym(1)-h^(alpha)*q(u+2); // original
end; // u
//Calculate Y1 to Ys by solving Ay+B=0
[y,x]=linsolve(A,B);
```

```
// Save the Y1 to Ys values in Ym for the s nodes
for k=1:s
Ym(k+1)=y(k);
end; // k
T(i)=T(i)+(timer()); // stop timer, update time for interval size
end; // if p<>1

// Keep alpha positive for remaining Ym calculations
//
timer(); // start timer
// Evaluate each Ym in turn for remaining interval nodes
for k=(s+1):N
// Calculate convolution weight terms total for interval node
cwsum=0;
for j=0:k-1
cwsum=cwsum+cw(k-j+1)*Ym(j+1);
end; //j
// Calculate starting weight terms total for interval node
swsum=0;
for j=0:s
swsum=swsum+sw(k,j+1)*Ym(j+1);
end; //j
Ym(k+1)=(h^(alpha)*q(k+1)-cwsum-swsum)/(cw(1)+h^(alpha));
end; //k

Yint(i)=Ym(k+1); // save final value for interval
T(i)=T(i)+(timer()); // stop timer, update time for interval size
Yerr(i)=Yint(i)-V*V; // calculate error for y(x)

end; // i interval size

// Open file for interval sizes data
u=file('open','c:\chester college\thesis\programs\de02\dintsize.txt'...
,'unknown',"unformatted");
```

```
writb(u,Int);
file('close',u);

// Open file for interval values data
u=file('open','c:\chester college\thesis\programs\de02\dintvals.txt'...
,'unknown',"unformatted");
writb(u,Yint);
file('close',u);

// Open file for interval times data
u=file('open','c:\chester college\thesis\programs\de02\dinttime.txt'...
,'unknown',"unformatted");
writb(u,T);
file('close',u);

// Open file for interval errors data
u=file('open','c:\chester college\thesis\programs\de02\dinterrs.txt'...
,'unknown',"unformatted");
writb(u,Yerr);
file('close',u);

lines(72,28);

disp("finished")
```

H Luchko/Diethelm's OC method - Scilab program

This program is for equation (4.4).

```
// John Landy
//
// Date: March 2009
//
// Dissertation: Fractional Differential Equations and Numerical Solutions
//
// Program written for SciLab package
//
// Abstract:
// Evaluate numerical solutions for Fractional Differential Equations
// Equation (4.4)
//
// Luchko & Diethelm operational calculus method
//

lines(0);

function tp=taylor(pol,n);
for i=1:n;
fd(1,i)=pol;
pol=derivat(pol);
end;
t=0;
for i=n:-1:1;
tp=horner(fd(1,i),0)/gamma(i)+t
t=tp*x;
end;
endfunction;

function mw=mitt2tc(n,a,b,f);
// Calculate n Taylor coefficients for
// Mittag-Leffler type function E(b;t)
// n - number of coefficients
// a - first Mittag-Leffler parameter
// b - second Mittag-Leffler parameter
// f - exit flag
//     0 - return final TC calculated value
//     1 - return array of TC calculated values
// Return the final TC calculated
w(1)=((un(1)^(a-b))*(h^b))/((un(1)^a)-lambda*(h^a)); // First coefficient
// Taylor coefficients 1 to n
for k=1:n
w(k+1)=0;
for j=1:k
w(k+1)=w(k+1)+powtc(j,a)*w(k-j+1); // method 2
end; // j
w(k+1)=(powtc(k,a-b)*(h^b)-w(k+1))/((un(1)^a)-lambda*(h^a)); // method 2
end; // k
if f==0 then
mw=w(k+1);
else
for k=1:(n+1)
mw(k)=w(k);
end; // k
end; // else
endfunction;

function pw=powtc(n,a);
// Calculate n Taylor coefficients for function u^(a)
// n - number of coefficients
// a - exponent of u
// Return the final TC calculated
w(1)=(un(1)^a); // First coefficient - orig
if n<=p then q=n; else q=p; end;
// Taylor coefficients 1 to p OR 1 to n if n<=p
```

```
for k=1:q
w(k+1)=0;
for j=0:(k-1)
w(k+1)=w(k+1)+(a*(k-j)-j)*w(j+1)*un(k-j+1);
end; // j
w(k+1)=w(k+1)/(k*un(1));
end; // k
// Taylor coefficients p+1 to n
if n>p then
for k=(p+1):n
w(k+1)=0;
for j=(k-p):(k-1)
w(k+1)=w(k+1)+(a*(k-j)-j)*w(j+1)*un(k-j+1);
end; // j
w(k+1)=w(k+1)/(k*un(1));
end; // k
end;
pw=w(k+1);
endfunction;

disp("start")

// Order of BDF
p=3;

// Order of equation - less than one
alpha=0.5;
lambda=-1;

// Variables
V=1;        // Value
j0=0;
gam=0.5;
if gam<1 then j0=1;
```

```
// Calculate gamma values as a set for starting weights using p and j0
//if p<>1
for i=0:(p-1)
g(i+1)=i+j0+gam-1;
end; // i
//end; // if p<>1

// Calculate aggregated Taylor polynomial coefficients
x=poly(0,"x");
a=0;
for k=1:p
a=a+((1-x)^k)/k;
end; //end k
un=coeff(a);

// Find solution for interval size
N=10;                     // number of subintervals
q=linspace(0.0, V, N+1);  // range of q
tj=linspace(0.0, V, N+1); // range of t_j
X=linspace(0.0, V, N+1);  // range of interval
Ym=linspace(0.0, V, N+1); // range of answer

// Multistep method
h=V/N;    // Calculate stepsize

// Initial conditions
Ym(1)=0; // y(0)=0

// Set up x=tj values for (N+1) nodes
for k=0:N
tj(k+1)=(k)*h;
end; //k
// Calculate f(x) for equation at each node
```

```
for k=1:N
f(k+1)=(tj(k+1))^(gam-1)*((tj(k+1))^(2-(gam-1))+...
(2/gamma(3-alpha))*(tj(k+1)^(2-(gam-1)-alpha))); // original
end; //k

disp("cw")

// Calculate convolution weights of the solution to the FDE
// for each interval node
// using a formula for the Taylor coefficients of u^alpha
cw(1)=(h^alpha)/((un(1)^alpha)-lambda*(h^alpha)); // Interval node 0
// Interval nodes 1 to N
for k=1:N
disp(k)
cw(k+1)=0;
for j=1:k
cw(k+1)=cw(k+1)+powtc(j,alpha)*cw(k-j+1);
end; // j
cw(k+1)=-cw(k+1)/(un(1)^alpha-lambda*(h^alpha));
end; // k

disp("sw")

// Calculate starting weights for each interval
// Number of starting weights (s+1) at each interval node
//s=size(g,'r')-1;
s=size(g,'r')-2;
// Starting weights for all interval nodes
sw=zeros(N,s+1);
// Starting weights for current interval node
w=zeros(1,s+1);
// Calculate (s+1) starting weights for each interval
// Calculate (s+1) terms for the convolution weights
//                 (for starting weights) and store in array
// Calculate (s+1) gamma terms for the interval node
// Set up the calculation arrays for each interval
RR=zeros(s+1,1);
// Calculate starting weights in turn for N interval nodes
for k=1:N
disp(k)
// Calculate (s+1) starting weights for each interval node
rA=zeros(s+1,1);
rB=zeros(s+1,1); // zero the convolution weight terms sum for the interval
for ja=1:(s+1)
for j=1:k
rB(ja,1)=rB(ja,1)+cw(k-j+1)*((j)^(g(ja))); // calc convolution wt terms
end; //j
rB(ja,1)=(h^(g(ja)))*rB(ja);
//for jb=1:(s+1)
for jb=j0:(s+1)
lB(ja,jb)=h^(g(ja))*((jb)^(g(ja))); // starting weight gamma power terms
end; // jb
// Taylor coefficient of Mittag-Leffler type function
mw=mitt2tc(k,alpha,g(ja)+alpha+1,0);
// Approximation of inverse Laplace transform of F(beta;s)
rA(ja,1)=mw*gamma(g(ja)+1)/h;
RR(ja,1)=rA(ja,1)-rB(ja,1); // calculate RHS of equation
end; //ja
w=lB\RR; // calculate starting weights for the current interval node
// Save starting weights for current interval node
for j=1:(s+1)
sw(k,j)=w(j);
end; // j
end; // k

disp("Y")

// Evaluate each Ym in turn for interval nodes m=1 to N
```

```
for k=1:N
disp(k)
// Calculate convolution weight terms total for interval node
cwsum=0;
//for j=0:k-1
for j=1:k
cwsum=cwsum+cw(k-j+1)*f(j+1);
end; //j
// Calculate starting weight terms total for interval node
swsum=0;
for j=0:s
swsum=swsum+sw(k,j+1)*f(j+1);
end; //j
Ym(k+1)=cwsum+swsum;
end; //k

disp("finished")

lines(72,28);
```

Bibliography

[1] Niels H. Abel. Solution de quelques problèmes à l'aide d'intégrals définies, Ouevres Complètes. Christiania (Grondahl), 1881.

[2] J. Armstrong. *Solving Ordinary Differential Equations with Recursive Taylor Series*, October 2005.

[3] Junesang Choi. A formula related to fractional calculus. *Comm. Korean Math. Soc.*, 12(No. 2):pp. 457–466, 1997.

[4] Nikesh S. Dattani. *Linear Multistep Methods for Ordinary Differential Equations.* University of Waterloo, Waterloo, Ontario, Canada, October 2008.

[5] H.T. Davis. *The Theory of Linear Operators.* The Principia Press Inc., Bloomington, Indiana, 1936.

[6] Lokenath Debnath. Recent applications of fractional calculus to science and engineering. In *International Journal of Mathematics and Mathematical Sciences*, volume 54, pages 3413–3442. Hindawi Publishing Corp., 2003. Based on 2 lectures at the 2002 IEEE Conference on Decision and Control.

[7] K. Diethelm and A.D. Freed. On the solution of nonlinear fractional-order differential equations used in the modeling of viscoplasticity. In H. Voss F. Keil, W. Mackens and J. Werther, editors, *Scientific Computing in Chemical Engineering II: Computational Fluid Dynamics, Reaction Engineering, and Molecular Properties*, pages 217–224. Springer Verlag, Heidelberg, 1999.

[8] Kai Diethelm. An algorithm for the numerical solution of differential equations of fractional order. *Elec. Trans. on Numerical Analysis*, 5:pp. 1–6, 1997.

[9] Kai Diethelm. *Fractional Differential Equations - Theory and Numerical Treatment.* Technical University of Braunschweig, February 2003.

[10] Kai Diethelm and Neville J. Ford. Analysis of fractional differential equations. *J. Math. Anal. Appl.*, 265(2):229–248, 2002.

[11] Kai Diethelm and Neville J. Ford. Multi-order fractional differential equations and their numerical solution. *Appl. Math. Comput.*, 154:621–640, 2004.

[12] Kai Diethelm and Yuri Luchko. *Numerical Solution of Linear Multi-Term Initial Value Problems of Fractional Order*, February 2005.

[13] W. Krämer F. Blomquist, W. Hofshuster. *Real and Complex Taylor Arithmetic in C-XSC.* Bergische Universität Wuppertal, 2005.

[14] Rudolf Gorenflo and Francesco Mainardi. Fractional calculus:integral and differential equations of fractional order. In *Fractals and Fractional Calculus in Continuum Mechanics*, pages 223–276. Springer Verlag, 1997.

[15] Rudolf Gorenflo and Francesco Mainardi. *Essentials of Fractional Calculus.* MaPhySto Center, January 2000.

[16] A. K. Grünwald. Über "begrenzte" Derivation und deren Anwendung. *Z. angew. Math. und Phys.*, 12:441–480, 1867.

[17] http://isiwebofknowledge.com. *ISI Web of Knowledge.*

[18] Neville J. Ford Kai Diethelm, Judith M. Ford and Marc Weilbeer. Pitfalls in fast numerical solvers for fractional differential equations. *J. Comput. Appl.*, 186(2):482–503, 2006.

[19] S. F. Lacroix. *Traité du Calcul Différentiel et du Calcul Intégral*, volume 3, sec. ed., pp. 409-410. Paris(Courcier), 1819.

[20] H. Laurent. Sur le calcul des dérivées à indices quelconques. *Nouvelles Ann. Math*, 3(3):pp. 240–252, 1884.

[21] G.W. Leibniz. Letter from Hanover, Germany to G.F.A. de l'Hôpital, September 30 1695, in Math. Schriften 1849, reprinted in 1962, Hildesheim, Germany (Olms Verlag), 2, pp 301-302.

[22] A. V. Letnikov. Theory of differentiation with an arbitrary index. *Mat. Sb.*, 3:1–66, 1868. In Russian.

[23] Joseph Liouville. Mémoire sur le théorème des complèmentaires. *J. Reine Angew, Math.*, 11:1–19, 1834.

[24] C. Lubich. On the stability of linear multistep methods for Volterra convolution equations. *IMA J. Math. Anal.*, 3(4):439–465, 1983.

[25] C. Lubich. Fractional linear multistep methods for Abel-Volterra equations of the second kind. *Math. Comp.*, 45(172):463–469, 1985.

[26] C. Lubich. Discretized fractional calculus. *SIAM J. Math. Anal.*, 17(3):704–719, 1986.

[27] C. Lubich. Convolution quadrature and discretized operational calculus. I. *Numer. Math.*, 52:129–145, 1988.

[28] C. Lubich. Convolution quadrature and discretized operational calculus. II. *Numer. Math.*, 52:413–425, 1988.

[29] Yuri Luchko and Rudolf Gorenflo. An operational method for solving fractional differential equations with the Caputo derivatives. *Acta Vietnamica*, 24:207–233, 1999.

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

[31] Igor Podlubny. *Fractional Differential Equations.* Mathematics in Science and Engineering: Volume 198. Academic Press, 1999.

[32] Bertram Ross. Fractional calculus. *Mathematics Magazine*, 50(No. 3):pp. 115–122, May 1977.

[33] Joulia Loutchko Rudolf Gorenflo and Yuri Luchko. *Computation of the Mittag-Leffler Function $E_{\alpha,\beta}(z)$ and its Derivative.*

[34] Marc Weilbeer. *Efficient Numerical Methods for Fractional Differential Equations and their Analytical Background.* Technical University of Braunschweig, 2005.