ON THE NUMERICAL SOLUTION OF FRACTIONAL PARTIAL DIFFERENTIAL EQUATIONS: AN INVESTIGATION OF SPECTRAL METHODS

A dissertation submitted to the University of KwaZulu-Natal

For the degree of Master of Science

In the College of Agriculture, Engineering and Science

By

Roger M Martins

School of Mathematics, Statistics and Computer Science

February 16, 2017

Abstract

Fractional partial differential equations are generalisations of classical partial differential equations, which relax the requirement that the derivatives are of integer order. A computational cost inherent to fractional derivatives is their non-local nature, and so they naturally benefit from the global approximation functions that characterise spectral methods. Using Jacobi polynomials integrated under Gaussian quadrature, several spectral collocation schemes are developed and tested on a variety of partial differential equations, fractional in both the time and space dimensions. The methods are tested on problems of varying degree, dimension, and linearity, as well as problems with derivative boundary conditions. Numerical results are compared to those obtained with both similar and dissimilar methods investigated in prior literature, where it is found that the methods implemented in this project compare generally favourably, and occasionally present the best known approximation.

Declaration

The work described in this dissertation was carried out under the supervision of Prof. P. Sibanda, School of Mathematics, Statistics and Computer Science, University of KwaZulu-Natal, Pietermaritzburg, from February 2016 to December 2016. No portion of the work referred to in this dissertation has been submitted in support of an application for another degree or qualification of this or any other university or institution of learning. The dissertation is my original work except where due reference and credit is given.

Student signature:...

Supervisor signature:...

Date:...

Acknowledgements

I am thankful to my supervisor, Prof. Precious Sibanda, for his support of my ideas, for his securing of the financial aid that made my project possible, and for his trust in allowing me to approach my research in my own way. I am grateful for the assistance provided by Prof. Sandile Motsa, Sicelo Goqo, and the rest of the Numerical Analysis sessions attendees that allowed me to understand spectral methods to the point that I could investigate those presented in this project.

I am grateful for the bursaries and scholarships awarded to me by the NRF and UKZN, without which I would have been unable to support myself during the year of research.

To my parents; I cannot thank you enough in one lifetime. Without your unwavering faith in me, your willingness to make continual sacrifices so that I might indulge my academic aspirations, I would never have been able to achieve what I have in my academic career. I thank you for being my best friends, and for putting up all of with my idiosyncracies and quirks. I only wish that I might use what you have given me to repay you in the time that we have left.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Fractional calculus generalises the classical integral and differential calculus by allowing arbitrary orders of integration and differentiation [1]. Fractional partial differential equations (FPDEs), then, are generalisations of their classical integral counterparts [2]. By allowing differentials of arbitrary order, FPDEs are frequently able to better model real phenomena than strictly integer ordered partial differential equations (PDEs) [1], and so their application to physics [3, 4, 5, 6, 7], finance [8, 9, 10, 11, 12], and fluid dynamics [13, 14, 15, 16, 17] has been investigated with ever increasing vigour. Given the non-local nature of fractional differential equations [18, 19, 20], their numerical solutions are typically computationally complex, as evidenced by the growing number of bespoke methodologies being developed. It is convenient then that we are also seeing rapid development in the subdiscipline of spectral methods; a class of numerical techniques that involve expansion of functions as sums of non-local basis functions, weighted by sets of coefficients that minimize the residual error between the function and its expansion [21]. The global nature of these methods makes them inherently well suited to the solution of FPDEs, and the primary aim of this project is to contribute to this growing, but as yet infantile, area of research.

Almost as old as classical calculus, having first been discussed in a correspondence between Gottfried Wilhelm Liebniz and Guillaume de L'Hospital in 1695, wherein Leibniz claimed prophetically that the fractional derivative was "an apparent paradox, from which one day useful consequences will be drawn" [22]. The paradox to which Leibniz ostensibly refers is that of the alternate definitions of the fractional derivative itself. In 1832, Joseph Liouville began the first major study of fractional calculus, that eventually led to the Riemann-Liouville factional integral in 1847 [22], which is still one of the most frequently used definitions. In 1867 and the following year, Anton Karl Grünwald and Aleksey Vasilievich Letnikov respectively derived what is now known as the Grünwald-Letnikov derivative, which under certain conditions is equivalent to the Riemann-Liouville definition [22]. Since then, many alternate definitions have been developed, but perhaps the most important was in 1967, with the advent of the Caputo derivative, for its ability to admit conventional boundary conditions [1, 22]. For the numerous subtleties that separate these various definitions, what they have in common is that they interpolate the integer derivatives obtained through classical calculus [1], and in this way are perhaps less alien than their unfamiliarity might suggest.

Before one can motivate why they may wish to use one definition or another, however, they should first inform their usage of fractional derivatives by relevant modelling conditions. Following the development of his own fractional derivative definition [23], Michele Caputo used derivatives of fractional order to model viscoelasticity of geological strata [24, 25]. Further influential work in this realm was done by Bagley and Torvik, starting in 1983, when they further developed the theoretical basis of fractional derivatives in viscoelasticity with an empirical model that accurately portrayed the mechanical properties of the transition regions of viscoelastic materials [26]. In 1994, appealing to the fractional Hausdorff dimensionality of fractal spaces, Metzler, Glöckle and Nonnenmacher derived uniquely determined FPDEs to model anomalous diffusions [27]. Anomalous diffusions, being diffusions with non-linear relationships between mean squared displacement and time, have been known for nearly a century [28], with applications to transport problems in disordered media [5, 29], long-range correlations [30], continuous time random walks [19] and fractional Brownian motion as explored famously by Mandlebrot [31]. More recently, however, there have been a number of studies demontrating the importance of fractional calculus in modelling these phenomena. In 1997, Compte and Metzler found that the Cattaneo equation, which describes diffusions with finite propagation velocity, was generalised to a fractional diffusion equation. In 2001, Baeumer, Meerschaert, Benson and Wheatcraft derived a fractional diffusion model for contaminant flow [14], which contributed to seminal work by Meerschaert, Tadjeran and their collaborators, where they made early forays into the numerical solution of FPDEs for anomalous fluid flow problems [2, 18, 32]. Towards the turn of the millennium, many other applications of FPDEs to problems in anomalous diffusion were explored, and one can consult the review by Metzler and Klafter for a summary of these studies [28]. While it was well-known that path integrals over Brownian trajectories produced the Schrödinger equation, Laskin found, in the early 2000s, that path integrals over Lévy trajectories produced the generalised fractional Schrödinger equation [33, 34, 35]. More contemporary models benefitting from FPDEs include Jacob's generalisation of a Fitzhugh-Nagumo equation to include fractional orders of time, resulting in a physically motivated and effective binarization process for images [36], and Angstmann, Henry and McGann appealed to the underlying stochastic processes to develop a fractional SIR model [19].

Given the typical complexity associated with fractional calculus, it is unsurprising that the development of their numerical solution has seen the advent of numerous related but dissimilar numerical methods. Among the earliest seminal contributions to numerical solutions in fractional calculus was from Bagley and Torvik, also in 1983, complementing their aforementioned investigations into fractional viscoelasticity models [26], where they solved fractional models of stress-strain relationships of viscoelastic materials with Finite Element Analysis [37]. Around the same time, Lubich developed several quadrature schemes for fractional Volterra and Abel integrals [38, 39], while Baker and Derakhshan investigated Fast Fourier Transform solutions to the same integrals [40]. In 1994, before the publication of his profoundly influential book [1], Podlubny used a discretisation of the Grünwald-Letnikov definition to solve a number of fractional ordinary differential equations [41, 42, 43]. In 1996, using a polynomial spline collocation method, Blank investigated a number of differential equations involving the ubiquitous Mittag-Leffler function as a component of their solutions [20]. Diethelm has made significant contributions to the numerical solution of fractional differential equations; in 1996, he proposed a method involving Hadamard regularisation of the Riemann-Liouville integral, allowing the solution of fractional differential equations by quadrature [44]. He later collaborated with a number of other researchers in this domain, most notably Ford, exploring such topics as the existence, uniqueness and stability properties of numerical solutions for fractional differential equations [45], and the development of a predictor-corrector method for solving fractional initial value problems with Caputo derivatives [46].

The development of numerical solutions for fractional partial differential equations specifically has seen the bulk of its interest in more recent years. One of the earliest contributors to the numerical solution of FPDEs came from Lynch, Carrera, del-Castillo-Negrete, Ferreira-Mejias and Hicks in 2003, where they used a discretisation method derived by Oldham and Spanier [22] to solve an anomalous diffusion equation [47]. Investigating FPDEs that are fractional in the spatial dimension, they take advantage of a second-order finite difference approximation of the fractional derivative. In addition to the usual truncation error associated with any method involving finite differences, their representation of the second-order discrete derivative requires knowledge of function values beyond the boundaries [47], and thus involved assumptions of symmetry that make their method potentially less useful for other problems.

Meerschaert and Tadjeran's aforementioned contributions to the numerical solution of FPDEs are among the most influential. In 2004, they detailed the derivation and analysis of finite difference methods for one-dimensional fractional diffusion equations with variable coefficients [18]. Using a truncated Grünwald-Letnikov derivative, evaluated at shifted grid points, Meerschaert and Tadjeran were able to derive an unconditionally stable implicit Euler method [18]. This method produces a super-diagonal coefficient matrix for local grid points, as one might expect with a finite difference method, but adds to this a lower-triangular matrix for the non-local fractional derivative [18], which, in this researcher's view, diminishes much of the computational advantage conferred by finite difference methods. In 2006, Meerschaert, Scheffler and Tadjeran then extended this method to work for two-dimensional FPDEs [32]. Benefitting again from a shifted, truncated Grünwald-Letnikov derivative, they develop an Alternating Direction Implicit (ADI) method to treat the two spatial dimensions of their anomalous diffusion equations, which they prove to be unconditionally stable, consistent and convergent [32]. As functional as this method is, it is this researcher's contention that it suffers from the same issues as the aforementioned applications of finite differences, and so other methods will be attempted in this project, with a demonstration of their superiority, by comparison of a particular numerical example solved by both methods, presented in Chapter 7. Also in 2006, Meerschaert and Tadjeran made an important contribution by extending their method to address two-sided FPDEs [2]; if one considers the non-local memory effects inherent to FPDEs, they might imagine a one-sided FPDE to have memory of the past, and by analogy, a two-sided FPDE additionally involves memories of the future. As before, their method involved a shifted, truncated Grünwald-Letnikov derivative, with which they derived a consistent, unconditionally stable implicit Euler scheme, and a consistent but conditionally stable explicit Euler scheme [2].

The history of using spectral methods to solve fractional partial differential equations is comparatively short, however, interest has grown dramatically in recent years. Significant contributions to this particular area have been made by Doha, Bhrawy, Ezz-Eldien and their collaborators. In 2011, Doha, Bhrawy, and Ezz-Eldien represented Caputo derivatives by shifted Chebyshev polynomials, from which they were able to derive operational matrices of the fractional derivatives to be used in a spectral tau method [48]. While this early application was not on FPDEs specifically, the results were extended by Bhrawy, Zaky and Machado to solve a two-sided time-fractional telegraph partial differential equation in 2016 [49]. In 2012, Doha, Bhrawy and Ezz-Eldien expanded this approach to the more general Jacobi polynomials, but were at this point still only considering fractional differential equations, again solving them with the spectral tau method [50]. In 2013, a slightly different approach was taken by Bhrawy and Baleanu, where they represented Caputo derivatives as Legendre polynomials, integrated under Gauss-Lobatto quadrature, to be subsequently evaluated by collocation [51]. Bhrawy then used a similar approach in 2014 to extend this Legendre-Gauss-Lobatto method into FPDEs with two spatial dimensions [52]. Given the versatility and power of these relatively early methods, they will make an ideal starting point for the investigation of this project, and are thus discussed in more detail in Chapter 4 and Chapter 5. In 2015, Bhrawy extended the aforementioned Jacobi operational matrix method [50] to two-dimensional time-fractional diffusion equations [53], and this extension will be a fundamental result upon which significant progress will be made in this project, presented in Chapter 6 and Chapter 7. A primary departure point of this project will be to adapt the method to deal with a far wider variety of equations, of varying dimensionality, degree and linearity, and with varying types of boundary conditions, towards the ends of developing a more general tool for the solution of FPDEs other than the commonly investigated diffusion equations.

A variety of other methods have been investigated for the solution of fractional partial differential equations, but they are outside the scope of this project. Semi-analytical methods, such as Adomian decomposition [54, 55], homotopy analysis [56], homotopy pertubation [57], variational iteration method [58], and generalised differential transform method [59] have been investigated successfully, but while they may have exceptional accuracy qualities, their limited applicability makes them inappropriate for comparison.

This dissertation is structured as follows; in Chapter 2, we introduce some useful fundamentals of fractional calculus, and discuss some of the technical details that will be relevant to their numerical solution. In Chapter 3, we will discuss the orthogonal functions and quadratures, and their role in the spectral collocation methods that will be used to numerically solve the FPDEs that will be considered in this thesis. In Chapter 4, we discuss the first class of FPDE of interest - one-dimensional space-fractional advection diffusion equations - and the bespoke method developed for their solution - a shifted Legendre-Gauss-Lobatto collocation solution. In Chapter 5, we extend this method to function in two spatial dimensions, investigating the solution of two-dimensional space-fractional diffusion equations. In Chapter 6, we consider a more general method, able to solve one-dimensional space- and time-fractional equations, and investigate a number of equations that fall under this class. In Chapter 7, we extend this method into two dimensions, allowing the solution of an even broader class of equations, and demonstrate its effectiveness with a number of distinct examples. The project is concluded in Chapter 8, after which the References and Appendices are presented.

Chapter 2

Fundamentals of Fractional Calculus

With the context and value of fractional calculus discussed previously, this chapter will consider key theory and results that will feature in the development of the numerical methods under investigation. For more detailed coverage of the topics addressed here, the reader is encouraged to consult the extensive works of Podlubny [1], and Oldham and Spanier [22].

2.1 Unification of Differentiation and Integration

We will begin our discussion of fractional calculus with a requisite treatment of the unification of the classically separate n^{th} order derivatives and n-fold integrals. First, we suppose a continuous function $y = f(t)$, and recall the typical backward-difference definition of the derivative, with h being the change in t:

$$f'(t) = \frac{df}{dt} = \lim_{h \to 0} \frac{f(t) - f(t-h)}{h}. \tag{2.1}$$

It can be shown by induction that further application generates the n^{th} order derivative [1],

$$f^{(n)}(t) = \frac{d^{(n)}f}{dt^{(n)}} = \lim_{h \to 0} \frac{1}{h^n} \sum_{r=0}^{n} (-1)^r \binom{n}{r} f(t - rh), \tag{2.2}$$

where $\binom{n}{r}$ is the usual binomial expansion. Setting aside the limit, we consider

$p \in \mathbb{Z}$ where $0 \leq p \leq n$ and define

$$f_h^{(p)}(t) = \frac{1}{h^p} \sum_{r=0}^{n} (-1)^r \binom{p}{r} f(t - rh), \tag{2.3}$$

where we clearly have

$$\lim_{h \to 0} f_h^{(p)}(t) = f^{(p)}(t). \tag{2.4}$$

Now, towards the ends of representing integrals, we consider orders of $-p < 0$. Importantly, such values do not work with the usual binomial expansion, so we rely on the more general definition:

$$\binom{n}{r} = \frac{\Gamma(n+1)}{\Gamma(r+1)\Gamma(n-r+1)} = \frac{(n)_r}{\Gamma(r+1)}, \tag{2.5}$$

where $\Gamma(x)$ represents the familiar Gamma function of argument x, and

$$(n)_r = \frac{\Gamma(n+1)}{\Gamma(n-r+1)}, \tag{2.6}$$

denotes the falling factorial. This generalisation produces the same results for positive integers, but accepts any complex argument, although for now we consider only $n \in \mathbb{Z}$. We note here the slight divergence in expression (but not meaning) from Podlubny [1], in anticipation of the usefulness of the Gamma function and falling factorial in subsequent applications. Considering our generalised binomial expansion, we observe that (2.3) now holds for all $p \leq n$.

Now, in anticipation of creating an operation that performs as integration does, we require limits. Our upper limit naturally comes from our function argument, t, so we introduce real constant a as a lower limit. Then, setting

$$h = \frac{t-a}{n} \implies n = \frac{t-a}{h}, \tag{2.7}$$

such that $n \to \infty$ as $h \to 0$, we define a new operator [1]:

$${}_a\mathbf{D}_t^{-p} f(t) = \lim_{h \to 0} f_h^{(-p)}(t). \tag{2.8}$$

As an example, let us consider when $p = 1$, and appeal to the limit definition of an integral:

$$ {}_a\mathbf{D}_t^{-1} f(t) = \lim_{h\to 0} f_h^{(-1)}(t) = \lim_{h\to 0} h \sum_{r=0}^{n} f(t-rh) = \int_0^{t-a} f(t-z)dz = \int_a^t f(\tau)d\tau, \tag{2.9} $$

where $\tau = t - a$. Evidently, for $p = 1$, the operator ${}_a\mathbf{D}_t^{-1} f(t)$ returns the desired integral. In fact, it can be shown by induction that we have the following relation in generality [1]:

$$ {}_a\mathbf{D}_t^{-p} f(t) = \lim_{h\to 0} h^p \sum_{r=0}^{n} (-1)^r \binom{p}{r} f(t-rh) = \frac{1}{\Gamma(p)} \int_a^t (t-\tau)^{p-1} f(\tau)d\tau, \tag{2.10} $$

which is shown to be a representation of a p-fold integral [1]. This provides us with a general expression:

$$ {}_a\mathbf{D}_t^{p} f(t) = \lim_{h\to 0} \frac{1}{h^p} \sum_{r=0}^{n} (-1)^r \binom{p}{r} f(t-rh), \tag{2.11} $$

which, by (2.4), respresents a derivative of order p for $p \geq 0$, and, by (2.11), represents a p-fold integral for $p < 0$.

2.2 Differentiation and Integration of Arbitrary Order

It may be unsurprising, considering the admission of complex numbers for our particular choice of a generalised binomial expansion, that (2.11) is in fact our most general expression for fractional integrals and derivatives, should we simpy allow p to be a real number [1, 22, 60], and this is referred to as the *Grünwald-Letnikov* definition, for $p \in \mathbb{R}$ and $f(t)$ continuous;

$$ {}^{GL}_{a}\mathbf{D}_t^{p} f(t) = \lim_{h\to 0} h^{-p} \sum_{r=0}^{n} (-1)^r \binom{p}{r} f(t-rh), \tag{2.12} $$

While ignoring the limit and instead setting a sufficiently small h does allow this definition to often act as a suitable discrete approximation [61, 60], in analytical applications, however, the limit is not easily obtained, making manipulation difficult [1, 60].

To allay this complication, we must sacrifice some generality in our assumptions, by assuming additionally that $f(t)$ must now also have at least derivatives $f^{(k)}(t)$, for $k = 1, 2, ..., m + 1$ being continuous in $[a, t]$, where m is an integer such that $m < p \leq m + 1$. While strictly less general, this extra assumption tends to be satisfied in most applications [1, 22]. After much manipulation, we obtain

$$ {}^{GL}_{a}\mathbf{D}^{p}_{t} f(t) = \sum_{k=0}^{m} \frac{f^{(k)}(a)(t-a)^{-p+k}}{\Gamma(-p+k+1)} + \frac{1}{\Gamma(-p+m+1)} \int_a^t (t-\tau)^{m-p} f^{(m+1)}(\tau) d\tau, \tag{2.13} $$

and are hence free from computing the problematic limit. This expression, however, is also obtained by the repeated integration by parts and differentiation of another well-known expression [1]:

$$ {}^{RL}_{a}\mathbf{D}^{p}_{t} f(t) = \left(\frac{d}{dt}\right)^{m+1} \int_a^t (t-\tau)^{m-p} f(\tau) d\tau, \qquad (m < p \leq m+1). \tag{2.14} $$

This expression is perhaps the most widely known of the fractional derivatives, and it is the *Riemann-Liouville* definition. It will be convenient to express this definition in terms of its integral compoment, for $p > 0$ [1, 60],

$$ {}^{RL}_{a}\mathbf{I}^{p}_{t} f(t) = \frac{1}{\Gamma(p)} \int_a^t (t-\tau)^{p-1} f(\tau) d\tau \tag{2.15} $$

which leads us to

$$ {}^{RL}_{a}\mathbf{D}^{p}_{t} f(t) = \left(\frac{d}{dt}\right)^{k} \left({}^{RL}_{a}\mathbf{I}^{k-p}_{t} f(t)\right) = \frac{1}{\Gamma(k-p)} \left(\frac{d}{dt}\right)^{k} \int_a^t (t-\tau)^{k-p-1} f(\tau) d\tau, \tag{2.16} $$

for $k - 1 \leq p < k$.

2.3 Alternate Formulations of the Fractional Derivative

One caveat in the use of the Riemann-Liouville fractional derivative is that at the lower terminals of $t = a$, we have the initial or boundary conditions producing equations in terms of the fractional derivatives, which while solvable, provide little physical meaning [1]. To address this, Caputo [1, 23] developed a fractional integral that reduces to integral derivatives at the boundaries, and is thus more readily applicable to physical problems. His definition is thus

$$ {}_{a}^{C}\mathbf{D}_{t}^{\alpha} f(t) = \frac{1}{\Gamma(\alpha - n)} \int_{a}^{t} \frac{f^{(n)}(\tau) d\tau}{(t-\tau)^{\alpha+1-n}}, \qquad (n-1 \leq \alpha < n), \tag{2.17} $$

and this definition will serve us in subsequent exercises.

Another note must be made to acknowledge the difference of left and right fractional derivatives. Thus far, we have considered only derivatives with a lower boundary $a \leq t$, however, it is also possible to consider fractional derivatives with moving lower terminal t, and fixed upper boundary b. The fractional derivative with a lower terminal a, with which we are familiar, is the left fractional derivative, given in (2.16), while the fractional derivative with upper terminal b is the right derivative, and its corresponding Riemann-Liouville definition is [1]

$$ {}_{t}^{RL}\mathbf{D}_{b}^{p} f(t) = \frac{1}{\Gamma(k-p)} \left(\frac{-d}{dt} \right)^{k} \int_{t}^{b} (t-\tau)^{k-p-1} f(\tau) d\tau, \tag{2.18} $$

with similar distinctions being available under the other definitions. If one views $f(t)$ as a function that evolves with time, they might view that the non-local left derivative at state t is dependent on the history following state a, while the right derivative will be at a state t dependent on future events up until time b [1]. It must be acknowledged that in the spatial dimensions, however, it is reasonable that the derivatives can be non-local in both directions, and so the use of right- and two-sided equations is often desirable. While there is clearly value in a treatment of both left- and right-fractional derivatives, the relative popularity of left-sided time-fractional differential equations over space-fractional equations in prior literature, and the existence of sufficient influential research addressing only left-sided space-fractional derivatives, we contend to consider only left-sided equations to be within the scope of this project.

2.4 Fractional Derivative of $(t-a)^\beta$

The power function $(t-a)^\beta$ is a particular example for which the fractional derivative will prove useful, given the prominence of polynomials in the spectral methods to follow. Setting $f(t) = (t-a)^\beta$ in (2.15), for $-p < 0$ and $\beta > -1$, and then substituting $\tau = a + z(t-a)$, we obtain in the fractional integral [1]

$$\begin{aligned} {}_a\mathbf{D}_t^p (t-a)^\beta &= \frac{1}{\Gamma(-p)} \int_a^t (t-\tau)^{-p-1} (\tau - a)^\beta d\tau \\ &= \frac{1}{\Gamma(-p)} (t-a)^{\beta-p} \int_0^1 z^\beta (1-z)^{-p-1} dz \\ &= \frac{1}{\Gamma(-p)} (t-a)^{\beta-p} \mathbf{B}(-p, \beta+1) \\ &= \frac{\Gamma(\beta+1)}{\Gamma(\beta-p+1)} (t-a)^{\beta-p}, \end{aligned} \tag{2.19}$$

where $\mathbf{B}(x,y)$ is the Beta function,

$$\mathbf{B}(x,y) = \int_0^1 \tau^{x-1} (1-\tau)^{y-1} d\tau = \frac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)}. \tag{2.20}$$

Now, for fractional derivatives where $m < p \leq m+1$, we set $f(t) = (t-a)^\beta$ in (2.13), and note that we require $\beta > m$ for the convergence of the integral [1]. We thus obtain

$${}_a\mathbf{D}_t^p (t-a)^\beta = \frac{1}{\Gamma(-p+m+1)} \int_a^t (t-\tau)^{m-p} \frac{d^{m+1}(\tau-a)^\beta}{d\tau^{m+1}} d\tau. \tag{2.21}$$

Noting the integer calculus result that

$$\frac{d^{m+1}(\tau-a)^\beta}{d\tau^{m+1}} = \beta(\beta-1)...(\beta-m)(\tau-a)^{\beta-m-1} = \frac{\Gamma(\beta+1)}{\beta-m}(\tau-a)^{\beta-m-1}, \quad (2.22)$$

and then substituting $\tau = a + z(t-a)$, we obtain

$$\begin{aligned} {}_a\mathbf{D}_t^p(t-a)^\beta &= \frac{\Gamma(\beta+1)}{\Gamma(\beta-m)\Gamma(-p+m+1)}\int_a^t (t-\tau)^{m-p}(\tau-a)^{\beta-m-1}d\tau \\ &= \frac{\Gamma(\beta+1)\mathbf{B}(-p+m+1,\beta-m)}{\Gamma(\beta-m)\Gamma(-p+m+1)}(\tau-a)^{\beta-p} \\ &= \frac{\Gamma(\beta+1)}{\Gamma(\beta-p+1)}(t-a)^{\beta-p}, \end{aligned} \quad (2.23)$$

which is conveniently the same result as (2.19), and thus we have an explicit formula [1, 22],

$${}_a\mathbf{D}_t^p(t-a)^\beta = \frac{\Gamma(\beta+1)}{\Gamma(\beta-p+1)}(t-a)^{\beta-p}, \quad (2.24)$$

for $p < 0$ and $\beta > -1$ or $m < p \leq m+1$ and $\beta > m$. We note that this expression holds under both the Rieman-Liouville and Caputo definitions of the fractional derivative [1, 51]. Now, considering only integer values of β, as are relevant to the polynomials we will be analysing, we recall from (2.14) that the classical integer derivative term will result in zero for any $\beta < m+1$, hence we have

$${}_a\mathbf{D}_t^p(t-a)^\beta = \begin{cases} 0 & \text{if } \beta < m+1 \\ \frac{\Gamma(\beta+1)}{\Gamma(\beta-p+1)}(t-a)^{\beta-p}, & \text{if } \beta \geq m+1 \end{cases} \quad (2.25)$$

2.5 Concluding Remarks

In conclusion, this chapter has provided us with the requisite definitions and expressions to represent fractional integrals and derivatives in ways amenable to numerical approximation. By unifying the notions of n^{th} order derivatives and n-fold integrals, we have a consistent representation that is extendable to fractional orders. We have discussed the Grünwald-Letnikov, Riemann-Liouville and Caputo definitions, allowing us to treat a variety of fractional differential equations. We have considered the cannonical example of the fractional derivative of $(t-a)^{\beta}$, which will be directly useful to us in subsequent examples.

Chapter 3

Spectral Methods

Spectral methods are a class of methods within the broader variations of the Method of Weighted Residuals (MWR) [21]. MWRs are defined by their trial and test functions, also commonly referred to as expansion and weight functions, respectively [21]. The trial functions are the basis functions by which the solution is represented as a truncated series expansion, and the test functions ensure that the differential equation is represented as closely as possible by the truncated series expansion of trial functions, by minimising the residual, being the difference between the exact and approximate solutions [21].

A key distinction between spectral methods, and other MWRs, such as the Finite Difference or Finite Element methods, is that the trial functions for spectral methods are infinitely differentiable global functions [21]. While the Finite Difference method will define approximations of the derivative terms with respect to a small neighbourhood of local function values, or the Finite Element method will divide the domain into small subsections, each with their own trial function, spectral methods will typically rely on global eigenfunctions of singular Sturm-Liouville problems [21]. Aside from the ostensible appropriateness of approximating global fractional derivatives with global functions, other benefits typical of spectral methods tend to be greater accuracy for a given resolution, or similarly, they require a lower resolution for a given accuracy [62], with a meaningful difference made by the absence of phase error, common to other methods [63]. The disadvantages of spectral methods are their difficulty in dealing with irregular domains, and their requirement of some smootheness of the problem data [21, 62].

After a suitable trial function has been chosen, the choice of test function is what distinguishes between the specfic subtypes of spectral method, with the three most prominent being the Galerkin, tau and collocation methods [21]. The Galerkin method employs test functions the same as the trial functions, and approximates the differential equation by forcing the integral of the product of the residuals and test functions to be zero [21]. The tau method is similar to the Galerkin method, however, the test functions are not required to satisfy the boundary conditions, for which an additional set of equations is used [21]. The Galerkin and tau methods, however, are outside the scope of this project, and so all further discussion will be focused on the collocation methods. In the collocation approach, also referred to as the pseudospectral method [63], the test functions are translated Dirac delta functions centered at specific collocation points, chosen with respect to the trial functions, at which the differential equation must be solved exactly [21].

There are many details that define the distinct varieties of collocation methods, and those that are relevant to the development of the methods of this research will be detailed throughout this chapter, starting with the orthogonal polynomials at the center of the approximation.

3.1 Orthogonal Polynomials

Many numerical methods are based upon the expansion of a function in terms of an infinite sequence of orthogonal polynomials, with the most familiar example being the approximation of periodic functions by expansions of Fourier series [21]. However, even non-periodic functions, should they be sufficiently smooth, can be approximated by finite expansions of eigenfunctions of suitable Sturm-Liouville eigenvalue problems of the form

$$-\Big(p(x)u'(x)\Big) + q(x)u(x) = \lambda w(x)u(x), \qquad -1 \leq x \leq 1, \tag{3.1}$$

where $p(x)$ is continuously differentiable, strictly positive, and continuous at the end points, $q(x)$ is continuous, bounded and non-negative, weight function $w(x)$ is continuous, non-negative and integrable over the domain, and $u(x)$ has appropriately defined boundary conditions [21]. Should these conditions provide a singular Sturm-Liouville problem, the approximation will be of spectral accuracy, and it can be shown that it is uniquely the classes of Jacobi polynomials that arise as eigenfunctions to singular Sturm-Liouville problems [21].

Now, considering orthogonal eigenfunctions ϕ_n of our Sturm-Liouville problem:

$$\int_{-1}^{1} \phi_k(x)\phi_m(x)w(x)dx = 0, \qquad \text{whenever } m \neq k, \tag{3.2}$$

and some function $u(x)$ which can be represented by a series expansion of these eigenfunctions, such that

$$u(x) = \sum_{n}^{\infty} \hat{u}_n \phi_n(x), \qquad -1 \leq x \leq 1, \tag{3.3}$$

we have that the constant coefficients $\hat{u}_n$ are found by the normalised inner product [21]:

$$\hat{u}_n = \frac{\langle u(x), \phi_n \rangle}{||\phi_n||^2} = \frac{1}{||\phi_n||^2} \int_{-1}^{1} \phi(x)u(x)w(x)dx, \tag{3.4}$$

where we have L^2-norm

$$||\phi_n||^2 = \int_{-1}^{1} |\phi_n(x)|^2 w(x)dx. \tag{3.5}$$

By the Weierstress theorem, this system is complete, so if we consider the truncated series, for integer $N > 0$,

$$u_N = \sum_{n}^{N} \hat{u}_n \phi_n(x), \tag{3.6}$$

which by (3.2) is the orthogonal projection of u onto $\mathbb{P}_N$ - the space of all polynomials of degree $\leq N$ - we have that [21]

$$\lim_{N\to\infty} ||u - u_N|| = 0, \tag{3.7}$$

so we have that, for sufficiently large N, our expansion of orthogonal polynomials approximates our function when represented as a Sturm-Liouville eigenvalue problem.

3.2 Gaussian Quadrature

The next significant step in the spectral approximation procedure is to compute the integral in (3.4). To this end, we rely on one of a number of Gaussian quadratures, with the three most commonly used quadratures discussed below.

3.2.1 Gauss Integration

If we let $x_0, ..., x_N$ be the roots of the orthogonal polynomial of degree $N + 1$, denoted p_{N+1}, and let quadrature weights, also known as Christoffel numbers, $\omega_0, ..., \omega_N$ be the solution to the linear system

$$\sum_{j=0}^{N} (x_j)^n \omega_j = \int_{-1}^{1} x^n w(x)dx, \qquad 0 \leq n \leq N, \tag{3.8}$$

then $\omega_j > 0$ for $j = 0, ..., N$, and

$$\sum_{j=0}^{N} p(x_j)\omega_j = \int_{-1}^{1} p(x)w(x)dx, \qquad \forall\, p \in \mathbb{P}_{2N+1}. \tag{3.9}$$

While Gauss integration benefits from the highest available degree of polynomial [64], at $2N + 1$, it is unsuitable for boundary value problems, as the collocation points are all in the interior of the domain $(-1, 1)$, thus neglecting to provide solutions at these key gridpoints [21, 64].

3.2.2 Gauss-Radau Integration

We begin our discussion of the Gauss-Radau formula by considering the polynomial [21]

$$q(x) = p_{N+1}(x) + ap_N(x), \tag{3.10}$$

where

$$a = -\frac{p_{N+1}(-1)}{p_N(-1)}, \tag{3.11}$$

so that $q(-1) = 0$. We thus let $x_0, ..., x_N$ be the $N+1$ roots of (3.10), with $x_0 = -1$ by (3.11), and let weights $\omega_0, ..., \omega_N$ be the solution to the linear system [21]

$$\sum_{j=0}^{N} (x_j)^n \omega_j = \int_{-1}^{1} x^n w(x) dx, \qquad 0 \leq n \leq N, \tag{3.12}$$

then $\omega_j > 0$ for $j = 0, ..., N$, and

$$\sum_{j=0}^{N} p(x_j)\omega_j = \int_{-1}^{1} p(x)w(x)dx, \qquad \forall\, p \in \mathbb{P}_{2N}. \tag{3.13}$$

While we have lost a degree in our polynomial space, we now have a collocation point at the -1, which makes Gauss-Radau integration suitable for problems with lower boundary or initial conditions [64].

3.2.3 Gauss-Lobatto Integration

We obtain the Gauss-Lobatto formula in a similar fashion, this time considering polynomial [21]

$$q(x) = p_{N+1}(x) + ap_N(x) + bp_{N-1}(x), \tag{3.14}$$

where a and b are chosen so that $q(-1) = q(1) = 0$. We thus let $x_0, ..., x_N$ be the $N + 1$ roots of (3.14), with $x_0 = -1$ and $x_N = 1$, and let weights $\omega_0, ..., \omega_N$ be the solution to the linear system [21]

$$\sum_{j=0}^{N} (x_j)^n \omega_j = \int_{-1}^{1} x^n w(x) dx, \qquad 0 \leq n \leq N, \tag{3.15}$$

then $\omega_j > 0$ for $j = 0, ..., N$, and

$$\sum_{j=0}^{N} p(x_j)\omega_j = \int_{-1}^{1} p(x)w(x)dx, \qquad \forall\, p \in \mathbb{P}_{2N-1}. \tag{3.16}$$

We have now lost two degrees in our polynomial space, however, we have collocation points at both ends of our domain, allowing us to treat fully specified boundary conditions [64].

3.3 Jacobi Polynomials

While we have discussed at some length how to expand our functions as series of orthogonal polynomials, and the quadratures we will use to evaluate the expansion coefficients, we have yet to discuss what type of polynomials we will consider. As mentioned, it is specifically the class of Jacobi polynomials that arise as eigenfunctions of the Sturm-Liouville eigenvalue problem, and so we will consider the general Jacobi polynomial, and two of its most commonly used variants.

3.3.1 Generalised Jacobi Polynomial

Jacobi polynomials are solutions of (3.1) with

$$p(x) = (1 - x)^{1+\alpha}(1 + x)^{1+\beta}, \qquad \alpha, \beta > -1, \tag{3.17}$$

where $q(x) = 0$, and $w(x) = (1-x)^\alpha(1+x)^\beta$. We denote the polynomials $P_n^{(\alpha,\beta)}(x)$, and normalise $P_n^{(\alpha,\beta)}(1) = \binom{n+\alpha}{n}$, which provides [65]

$$P_n^{(\alpha,\beta)}(x) = \sum_{j=0}^{n} \binom{n+\alpha}{n-j}\binom{n+\beta}{j}\left(\frac{x-1}{2}\right)^j\left(\frac{x+1}{2}\right)^{n-j}. \tag{3.18}$$

We obtain the Jacobi polynomials by the following recursive relationship:

$$P_0^{(\alpha,\beta)}(x) = 1, \tag{3.19}$$

$$P_1^{(\alpha,\beta)}(x) = (1+\alpha)x, \tag{3.20}$$

$$a_{1,n}P_{n+1}^{(\alpha,\beta)}(x) = a_{2,n}(x)P_n^{(\alpha,\beta)}(x) - a_{3,n}P_{n-1}^{(\alpha,\beta)}(x), \tag{3.21}$$

where

$$a_{1,n} = 2(n+1)(n+\alpha+\beta+1)(2n+\alpha+\beta), \tag{3.22}$$

$$a_{2,n}(x) = (2n+\alpha+\beta+1)(\alpha^2-\beta^2) + x\frac{\Gamma(2n+\alpha+\beta+3)}{\Gamma(2n+\alpha+\beta)}, \tag{3.23}$$

$$a_{3,n} = 2(n+\alpha)(n+\beta)(2n+\alpha+\beta+2). \tag{3.24}$$

Thus, for the Jacobi series

$$u(x) = \sum_{n=0}^{\infty} \hat{u}_n P_n^{(\alpha,\beta)}(x), \tag{3.25}$$

we obtain coefficients

$$\hat{u}_n = \frac{(2n+\alpha+\beta+1)n!\,\Gamma(n+\alpha+\beta+1)}{2^{\alpha+\beta+1}\Gamma(n+\alpha+1)\Gamma(n+\beta+1)}\int_{-1}^{1} u(x)P_n^{(\alpha,\beta)}(x)(1-x)^\alpha(1+x)^\beta dx. \tag{3.26}$$

As an example, we consider $\alpha = \beta = 1$, which provides the polynomials in Figure 3.1

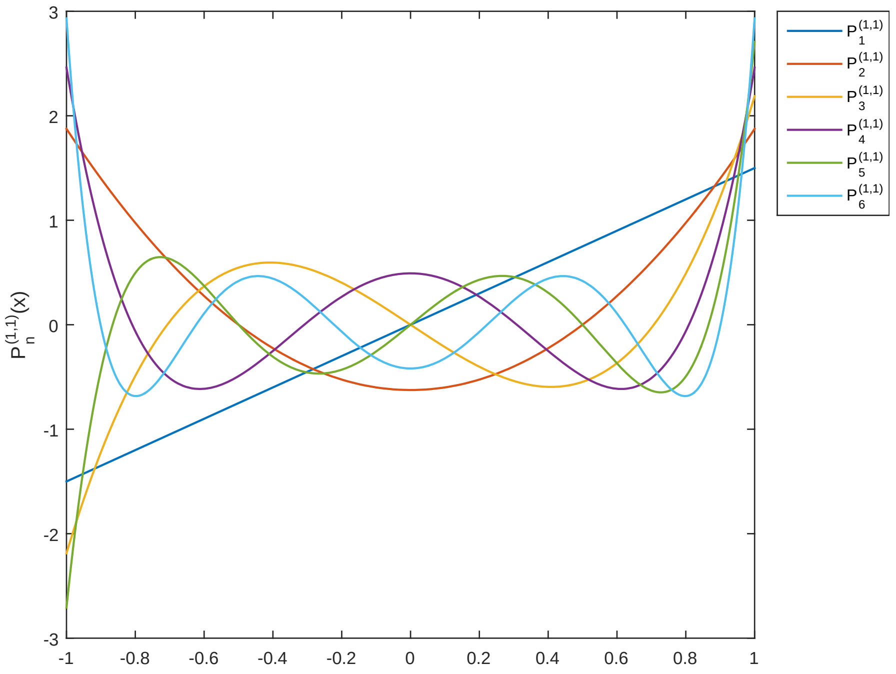


Figure 3.1. Jacobi polynomials of degrees 1 through 6 with $\alpha = 1$ and $\beta = 1$

The first derivative of the Jacobi polynomial can be expressed explicitly as [65]

$$\frac{dP_n^{(\alpha,\beta)}(x)}{dx} = \frac{n+\alpha+\beta+1}{2} P_{n-1}^{(\alpha+1,\beta+1)}(x), \tag{3.27}$$

and further derivation leads to the general formula

$$\frac{d^m P_n^{(\alpha,\beta)}(x)}{dx^m} = \frac{\Gamma(\alpha+\beta+n+1+m)}{2^m\Gamma(\alpha+\beta+n+1)} P_{n-m}^{(\alpha+m,\beta+m)}(x), \tag{3.28}$$

We will see further use of Jacobi polynomials in the subsequent solutions of FPDEs.

3.3.2 Legendre Polynomials

Recall again (3.1), only now we make substitutions $p(x) = 1 - x^2$, $q(x) = 0$ and $w(x) = 1$. This means that Legendre polynomials $L_n(x)$ are Jacobi polynomials with parameters $\alpha = \beta = 0$, that is,

$$L_n(x) = P_n^{(0,0)}(x). \tag{3.29}$$

This provides us with the Sturm-Liouville eigenvalue problem

$$\Big((1-x^2)L_n'(x)\Big)' + n(n+1)L_n(x) = 0, \tag{3.30}$$

where $L_n(x)$, $n = 0, 1, ...$ are the obtained eigenfunctions. Then, normalising for $L_n(1) = 1$, we obtain

$$L_n(x) = \frac{1}{2^n}\sum_{j=0}^{\lfloor \frac{n}{2} \rfloor}(-1)^j \binom{n}{j}\binom{2n-2j}{n}x^{n-2j}, \tag{3.31}$$

where $\lfloor x \rfloor$ denotes the floor function of x. This provides the following recursive relationship:

$$L_{n+1}(x) = \frac{2n+1}{n+1}xL_n(x) - \frac{n}{n+1}L_{n-1}(x), \tag{3.32}$$

where $L_0(x) = 1$ and $L_1(x) = x$. This recurrence provides the useful form [66]

$$L_n(x) = \sum_{j=0}^{n}(-1)^j \binom{n}{j}^2 \left(\frac{1+x}{2}\right)^{n-j}\left(\frac{1-x}{2}\right)^j. \tag{3.33}$$

Thus, for the Legendre series

$$u(x) = \sum_{n=0}^{\infty}\hat{u}_n L_n(x), \tag{3.34}$$

we have coefficients

$$\hat{u}_n = \left(n + \frac{1}{2}\right)\int_{-1}^{1} u(x)L_n(x)dx. \tag{3.35}$$

We observe polynomials of orders 1 through 6 in Figure 3.2:

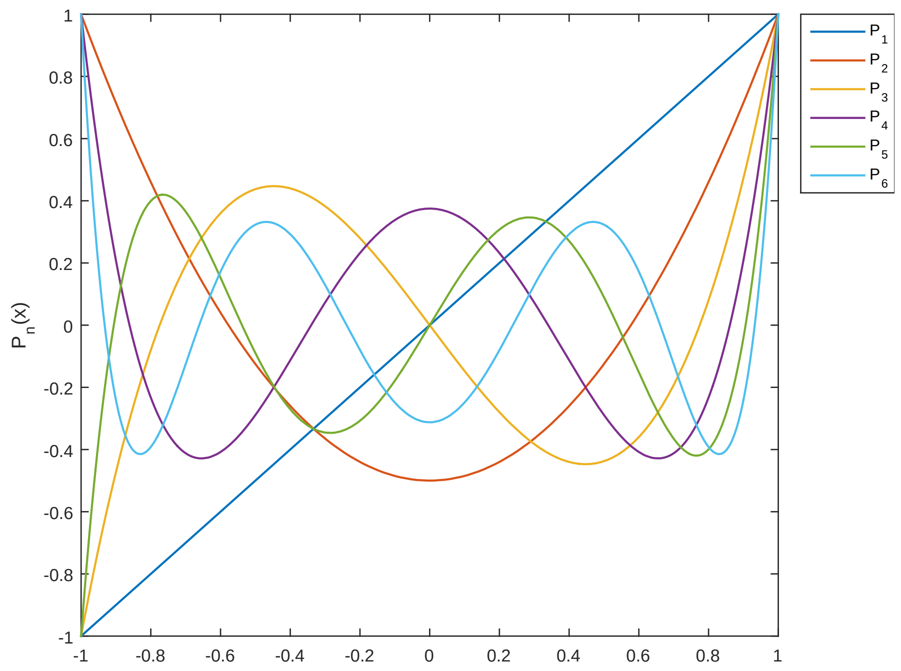


Figure 3.2. Legendre polynomials of degrees 1 through 6

We obtain an explicit general expression for Legendre polynomials by substituting $\alpha = \beta = 0$ into (3.28),

$$\frac{d^m L_n(x)}{dx^m} = \frac{\Gamma(n+1+m)}{2^m \Gamma(n+1)} P_{n-m}^{(m,m)}(x). \tag{3.36}$$

We will rely heavily on Legendre polynomials in subsequent problems, under a variety of quadratures.

3.3.3 Chebyshev Polynomials

Once again recall (3.1), this time making substitutions $p(x) = \sqrt{1-x^2}$, $q(x) = 0$ and $w(x) = \frac{1}{\sqrt{1-x^2}}$. This means that Chebyshev polynomials $T_n(x)$ are related to Jacobi polynomials by

$$T_n(x) = \frac{2^{2n}(n!)^2}{(2n)!} P_n^{(-\frac{1}{2},-\frac{1}{2})}(x). \tag{3.37}$$

This obtains a Sturm-Liouville eigenvalue problem of

$$\left(\sqrt{1-x^2}T_n'(x)\right)' + \frac{n^2}{\sqrt{1-x^2}}T_n(x) = 0, \tag{3.38}$$

for which $T_n(x)$, $n = 0, 1, ...$ are the eigenfunctions. Then, normalising for $T_n(1) = 1$, we obtain

$$T_n(x) = \cos(n\theta), \qquad \theta = \arccos(x). \tag{3.39}$$

This can be expanded in the power series

$$T_n(x) = \frac{n}{2}\sum_{j=0}^{\lfloor \frac{n}{2} \rfloor}(-1)^n \frac{(n-j-1)!}{j!\,(n-2j)!}(2x)^{n-2j}, \tag{3.40}$$

and we have the recursive relation

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x), \tag{3.41}$$

with $T_0(x) = 1$ and $T_1(x) = x$. Thus, for the Chebyshev series

$$u(x) = \sum_{n=0}^{\infty} \hat{u}_n T_n(x), \tag{3.42}$$

we obtain coefficients

$$\hat{u}_n = \frac{2}{\pi c_n}\int_{-1}^{1} u(x)T_n(x)w(x)dx, \tag{3.43}$$

where

$$c_n = \begin{cases} 2 & \text{if } n = 0 \\ 1 & \text{if } n \geq 1 \end{cases}$$

As an example, we present in Figure 3.3 the polynomials of degree 1 through 6,

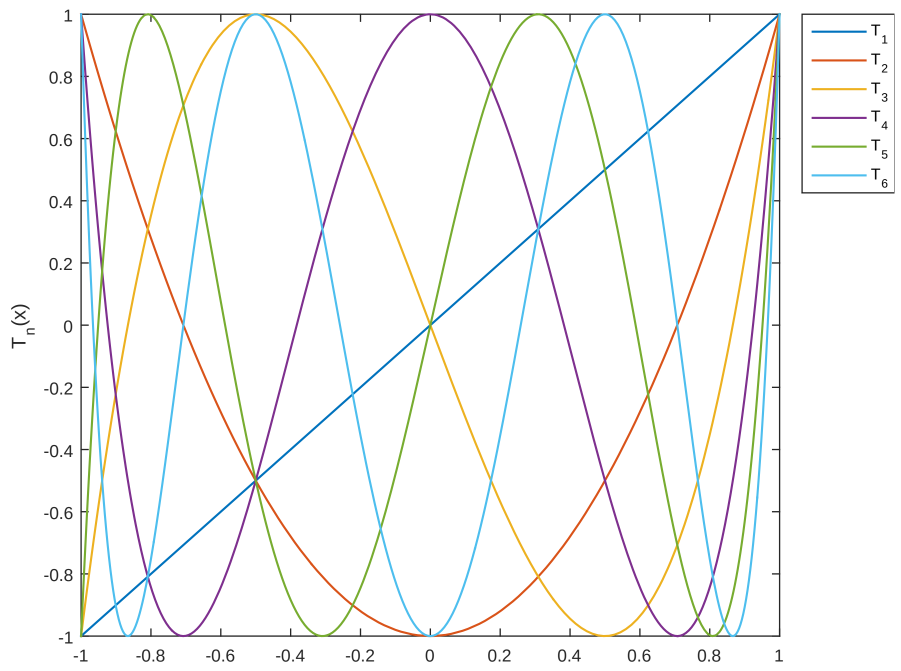


Figure 3.3. Chebyshev polynomials of degrees 1 through 6

Chebyshev polynomials are perhaps the most popular variety of Jacobi polynomial, likely due to their generally superior rate of convergence [67], and so one will benefit from their consideration in problems to follow.

While Chebyshev polynomials are frequently used, different applications will benefit from different Jacobi polynomials [67], and indeed, there are cases where any ultraspherical polynomials, these being Jacobi polynomial where $\alpha = \beta$ [65], will have identical convergence characteristics. For this research, however, we will defer to the choices made in prior literature, for both convenience and consistent comparison.

3.4 Shifted Polynomials

We should not presume that all problems to be solved with spectral methods will fit conveniently between -1 and 1, and so we will discuss what happens when we move to the positive half-domain. To change our domain from $[-1, 1]$ to $[0, R]$, we make the transform

$$p_{R,n}(x) = p_n\left(\frac{2x}{R} - 1\right). \tag{3.44}$$

3.4.1 Shifted Jacobi Polynomials

Recalling (3.18), we obtain the following analytical expression for shifted Jacobi polynomials:

$$\begin{aligned} P_{R,n}(x)^{(\alpha,\beta)} = P_n^{(\alpha,\beta)}\left(\frac{2x}{R} - 1\right) &= \sum_{j=0}^{n} \binom{n+\alpha}{n-j}\binom{n+\beta}{j}\left(\frac{x}{R} - 1\right)^j\left(\frac{x}{R}\right)^{n-j} \\ &= \sum_{j=0}^{n} \frac{\Gamma(n+\alpha+1)\Gamma(n+\beta+1)}{\Gamma(\alpha+j+1)\Gamma(n+\beta+1-j)(n-j)!\,j!}\left(\frac{x}{R} - 1\right)^j\left(\frac{x}{R}\right)^{n-j}, \end{aligned} \tag{3.45}$$

and note that it can be alternately expressed as [65]

$$P_{R,n}(x)^{(\alpha,\beta)} = \sum_{j=0}^{n} (-1)^{n+j} \frac{\Gamma(n+\alpha+1)\Gamma(n+j+\alpha+\beta+1)}{\Gamma(\alpha+j+1)\Gamma(n+\alpha+\beta+1)(n-j)!\,j!\,R^j} x^j. \tag{3.46}$$

The shifted weight function is $w_R^{(\alpha,\beta)} = x^\alpha(R-x)^\beta$, and we have the orthogonality of the polynomials by [50]

$$\int_0^R P_{R,k}(x)^{(\alpha,\beta)} P_{R,m}(x)^{(\alpha,\beta)} w_R^{(\alpha,\beta)} dx = h_m \tag{3.47}$$

where

$$h_m = \begin{cases} \frac{R^{\alpha+\beta+1}\Gamma(m+\alpha+1)\Gamma(m+\beta+1)}{(2m+\alpha+\beta+1)m!\Gamma(m+\alpha+\beta+1)} & \text{if } m = k \\ 0 & \text{if } m \neq k. \end{cases}$$

Thus, we obtain the shifted Jacobi series

$$u(x) = \sum_{n=0}^{\infty} \hat{u}_n P_{R,n}(x)^{(\alpha,\beta)}(x), \tag{3.48}$$

with coefficients

$$\hat{u}_n = \frac{(2n+\alpha+\beta+1)n!\,\Gamma(n+\alpha+\beta+1)}{R^{\alpha+\beta+1}\Gamma(n+\alpha+1)\Gamma(n+\beta+1)} \int_0^R u(x) P_{R,n}^{(\alpha,\beta)}(x) x^{\alpha}(R-x)^{\beta} dx. \tag{3.49}$$

3.4.2 Shifted Legendre Polynomials

In the case of Legendre polynomials, we recall (3.33), and make the transform from $[-1,1]$ to $[0,1]$ [66],

$$\begin{aligned} L_{1,n}(x) = L_n(2x-1) &= \sum_{j=0}^{n} (-1)^j \binom{n}{j}^2 (x)^{n-j} (1-x)^j \\ &= \sum_{j=0}^{n} (-1)^n \binom{n}{j} \binom{n+j}{j} (-x)^j \\ &= \sum_{j=0}^{n} (-1)^{n+j} \frac{(n+j)!\, x^j}{(n-j)!\,(j!\,)^2}, \end{aligned} \tag{3.50}$$

which is then expanded to domain $[0,R]$ [68],

$$L_{R,n}(x) = \sum_{j=0}^{n} (-1)^{n+j} \frac{(n+j)!\, x^j}{(n-j)!\,(j!\,)^2 R^j}. \tag{3.51}$$

The shifted Legendre weight function is still $w_R = 1$, and we have the orthogonality of the polynomials by [68]

$$\int_0^R L_{R,k}(x) L_{R,m}(x) dx = h_m, \tag{3.52}$$

where [69]

$$h_m = \begin{cases} \frac{R}{2m+1} & \text{if } m = k \\ 0 & \text{if } m \neq k. \end{cases}$$

We thus obtain the shifted Legendre series

$$u(x) = \sum_{n=0}^{\infty} \hat{u}_n L_{R,n}(x), \tag{3.53}$$

with coefficients

$$\hat{u}_n = \frac{2n+1}{R} \int_0^R u(x) L_{R,n}(x) dx. \tag{3.54}$$

3.5 Concluding Remarks

We have discussed the fundamentals of orthogonal polynomials, how they are obtained as the eigenfunctions of Sturm-Liouville eigenvalue problems, and their value in approximating functions. We considered three variaties of Gaussian quadrature, including traditional Gauss integration, Guass-Radau and Gauss-Lobatto, how these quadratures are obtained, and how they benefit in the numerical solution of different types of problems. We then discussed Jacobi polynomials, being the unique class of polynomial arising from a singular Sturm-Liouville problem, and how Legendre and Chebyshev polynomials are variants of the general Jacobi polynomial. We went on to discuss shifted polynomials, and how they are obtained from the standard formulae.

The subsequent chapters will present a number of worked cases of spectral methods applied to particular FPDEs. Much theory has been covered so far, which will inform the methods explained henceforth, where techniques specific to the problems under investigation will be developed and implemented.

Chapter 4

Space-Fractional Advection Diffusion Equations

This chapter will focus on the development and implementation of our first spectral collocation method for solving a class of FPDEs, and is based on the work of Bhrawy and Baleanu [51]. We consider space-fractional PDEs with coefficients dependent on both space and time [51]. The spatial derivatives, both integer-ordered and fractional, will be solved spectrally, by expanding them as a series of shifted Legendre polynomials, integrated under Gauss-Lobatto quadrature, producing a system of ordinary differential equations in time, which will be solved using the explicit Runge-Kutta 4.

For this exercise, we will consider equations of the form

$$\frac{\partial u(x,t)}{\partial t} + a(x,t)\frac{\partial u(x,t)}{\partial x} - b(x,t)\frac{\partial^{\nu} u(x,t)}{\partial x^{\nu}} - c(x,t)u(x,t) = q(x,t), \tag{4.1}$$

where

$$0 \leq x \leq R, \qquad 0 < t \leq T,$$

with fractional derivative order $\nu \in (1,2]$, and with initial and boundary conditions

$$\begin{aligned} u(x,0) &= f(x), \qquad & 0 \leq x \leq R, \\ u(0,t) &= g_0(t), \qquad & 0 < t \leq T, \\ u(R,t) &= g_R(t), \qquad & 0 < t \leq T. \end{aligned} \tag{4.2}$$

The anomalous diffusion, or subdiffusion, is represented by $\frac{\partial^{\nu} u(x,t)}{\partial x^{\nu}}$, which is a fractional spatial derivative, but noting the presence of initial and boundary conditions in (4.2), it is defined in the Caputo sense [51], as presented in (2.17), but with lower terminal 0:

$$\frac{\partial^\nu u(x,t)}{\partial x^\nu} = {}_0^C\mathbf{D}_x^\nu u(x,t) = \frac{1}{\Gamma(\nu - n)} \int_0^x \frac{u^{(n)}(\tau,t)d\tau}{(x-\tau,t)^{\nu+1-n}}, \qquad (n-1 \leq \nu < n), \tag{4.3}$$

The coefficient function $a(x,t)$ denotes the drift, or advective velocity, $b(x,t)$ denotes the anomalous diffusion, and $c(x,t)$ and $q(x,t)$ are known smooth functions [51]. Setting $c = 0$ in (4.1) obtains the space-fractional advection-dispersion equation, setting $a = q = 0$ obtains the space-fractional reaction-disperson equation, and setting $q = 0$ and $\nu = 2$ obtains the Fokker-Planck equation [51].

4.1 Shifted Legendre-Gauss-Lobatto Collocation

Given our fully specified boundary conditions, we will benefit from the collocation points at the boundaries inherent to Gauss-Lobatto quadrature, as presented in § 3.2.3. For shifted Legendre polynomials, we adjust our $N+1$ collocation points x_n by the same transformation used in the polynomials, that is, for the shifted collocation points $x_{R,0}, ..., x_{R,N}$ we have

$$x_{R,n} = \frac{R}{2}(x_n + 1), \qquad 0 \leq n \leq N, \tag{4.4}$$

while the weights $w_{R_1}(x)$ still equal 1. However, the Legendre-Gauss-Lobatto quadrature weights $\omega_{R,j}$, are adjusted from ω_j which solve (3.16) by the following transformation; we observe that the ω_j solve [65]

$$\omega_j = \int_{-1}^{1} \frac{L_n(x)}{L_n'(x_j)(x-x_j)} dx, \qquad j = 1, 2, ..., n, \tag{4.5}$$

and so we have

$$\begin{aligned}\omega_{R,j} &= \int_0^R \frac{L_{R,n}(x)}{L'_{R,n}(x_j)(x-x_j)}dx, \qquad j=1,2,...,n,\\ &= \frac{R}{2}\int_{-1}^{1}\frac{L_n(x)}{L'_n(x_j)(x-x_j)}dx \\ &= \frac{R}{2}\omega_j.\end{aligned} \tag{4.6}$$

and so we have that $\omega_{R,j} = \frac{R}{2}\omega_j$. This ensures that for (3.16) we have [70]

$$\begin{aligned}\int_0^R L_R(x)w_R(x)dx &= \frac{R}{2}\int_{-1}^{1} L_R(x_R)w(x)dx, \qquad \forall\, L_R \in \mathbb{P}_{2N-1}.\\ &= \frac{R}{2}\sum_{j=0}^{N} L_R(x_{R,j})\omega_j \\ &= \sum_{j=0}^{N} L_R(x_{R,j})\omega_{R,j},\end{aligned} \tag{4.7}$$

and so our shifted Legendre-Gauss-Lobatto quadrature approximates as desired. Significantly, this allows us to solve N-truncated (3.53) expanded into the time dimension;

$$u_N(x_{R,j},t) = \sum_{n=0}^{N}\hat{u}_n(t)L_{R,n}(x_{R,j}), \tag{4.8}$$

with coefficients

$$\begin{aligned}\hat{u}_n(t) &= \frac{2n+1}{R}\int_0^R u(x,t)L_{R,n}(x)dx \\ &= \frac{2n+1}{R}\sum_{j=0}^{N}u(x_{R,j},t)L_{R,n}(x_{R,j})\omega_{R,j}.\end{aligned} \tag{4.9}$$

Substituting back into (4.8) and simplifying [51], we obtain

$$u_N(x,t) = \sum_{j=0}^{N}\left[\sum_{n=0}^{N}\frac{2n+1}{R}L_{R,n}(x_{R,j})L_{R,n}(x_{R,j})\omega_{R,j}\right]u(x_{R,j},t). \tag{4.10}$$

For determining the 1st degree spatial derivative $\frac{\partial u(x,t)}{\partial x}$, henceforth represented as $u_x(x,t)$, we turn to the convenient relation due to the linear independence of the

Lengendre polynomials [21];

$$u_x(x,t) = \sum_{n=0}^{\infty} \hat{u}_n(t) L'_n(x), \tag{4.11}$$

which after N-truncating, and substituting in (4.9) for the coefficients and (3.36) for the derivative of order $m = 1$, we obtain at the collocation points

$$\begin{aligned} u_{x,N}(x_{R,k},t) &= \sum_{j=0}^{N} \left[\sum_{n=0}^{N} \frac{2n+1}{R} L_{R,n}(x_{R,j}) L'_{R,n}(x_{R,k}) \omega_{R,j} \right] u(x_{R,j},t) \\ &= \sum_{j=0}^{N} \left[\sum_{n=0}^{N} \frac{(2n+1)(n+1)}{2R} L_{R,n}(x_{R,j}) P^{(1,1)}_{R,n-1}(x_{R,k}) \omega_{R,j} \right] u(x_{R,j},t) \\ &= \sum_{j=0}^{N} \mathcal{D}^1_{j,k} u(x_{R,j},t), \end{aligned} \tag{4.12}$$

where

$$\mathcal{D}^1_{j,k} = \sum_{n=0}^{N} \frac{(2n+1)(n+1)}{2R} L_{R,n}(x_{R,j}) P^{(1,1)}_{R,n-1}(x_{R,k}) \omega_{R,j}. \tag{4.13}$$

For the fractional spatial derivative, $\frac{\partial^\nu u(x,t)}{\partial x^\nu}$, henceforth represented as $\mathbf{D}^\nu u(x,t)$, we have

$$\mathbf{D}^\nu u(x,t) = \sum_{n=0}^{\infty} \hat{u}_n(t) \mathbf{D}^\nu L_{R,n}(x), \tag{4.14}$$

where we consider the fractional derivative of the Legendre polynomial represented by (3.51), to obtain

$$\begin{aligned} \mathbf{D}^\nu L_{R,n}(x) &= \mathbf{D}^\nu \sum_{j=0}^{n} (-1)^{n+j} \frac{(n+j)!\, x^j}{(n-j)!\,(j!\,)^2 R^j} \\ &= \sum_{j=0}^{n} (-1)^{n+j} \frac{(n+j)!\, \mathbf{D}^\nu x^j}{(n-j)!\,(j!\,)^2 R^j}, \end{aligned} \tag{4.15}$$

where we have by (2.24)

$$\mathbf{D}^\nu x^j = \frac{\Gamma(j+1)}{\Gamma(j-\nu+1)} x^{j-\nu}, \tag{4.16}$$

and thus we have

$$\mathbf{D}^{\nu} L_{R,n}(x) = \sum_{j=0}^{n} (-1)^{n+j} \frac{(n+j)!\, x^{j-\nu}}{\Gamma(j-\nu+1)(n-j)!\, j!\, R^j}, \qquad n = \lceil \nu \rceil, \lceil \nu \rceil + 1, \ldots \tag{4.17}$$

where we note that for $n < \lceil \nu \rceil$, we have by the Caputo derivative definition (2.17) that $\mathbf{D}^{\nu} L_{R,n}(x,t) = 0$, since for $x^j, j < \lceil \nu \rceil$, we have $(x^j)^{(\lceil \nu \rceil)} = 0$. Next, we can represent $x^{j-\nu}$ in terms of shifted Legendre polynomials, and so using (3.53) we have

$$x^{j-\nu} = \sum_{k=0}^{\infty} \hat{\theta}_{j,k} L_{R,k}(x), \tag{4.18}$$

which, by (3.54) has coefficients

$$\begin{aligned}
\hat{\theta}_{j,k} &= \frac{2k+1}{R} \int_0^R x^{j-\nu} L_{R,k}(x) dx \\
&= \frac{2k+1}{R} \int_0^R x^{j-\nu} \sum_{r=0}^{k} (-1)^{k+r} \frac{(k+r)!\, x^r}{(k-r)!\, (r!\,)^2 R^r} dx \\
&= \frac{2k+1}{R} \sum_{r=0}^{k} (-1)^{k+r} \frac{(k+r)!}{(k-r)!\, (r!\,)^2 R^r} \int_0^R x^{j+r-\nu} dx \\
&= (2k+1) R^{j-\nu} \sum_{r=0}^{k} (-1)^{k+r} \frac{(k+r)!}{(k-r)!\, (r!\,)^2 (j+r-\nu+1)}.
\end{aligned} \tag{4.19}$$

Now, we substitute (4.19) into (4.18), which is then substituted into (4.17), to obtain

$$\begin{aligned}
&\mathbf{D}^{\nu} L_{R,n}(x) \\
&= \sum_{j=0}^{n} (-1)^{n+j} \frac{(n+j)! \sum_{k=0}^{\infty} \hat{\theta}_{j,k} L_{R,k}(x)}{\Gamma(j-\nu+1)(n-j)!\, j!\, R^j} \\
&= \sum_{j=0}^{n} (-1)^{n+j} \frac{(n+j)! \sum_{k=0}^{\infty} (2k+1) R^{j-\nu} \sum_{r=0}^{k} (-1)^{k+r} \frac{(k+r)!}{(k-r)!(r!)^2(j+r-\nu+1)} L_{R,k}(x)}{\Gamma(j-\nu+1)(n-j)!\, j!\, R^j} \\
&= \sum_{j=0}^{n} \Pi_{\nu}(n,j) L_{R,j}(x), \qquad n = \lceil \nu \rceil, \lceil \nu \rceil + 1, \ldots
\end{aligned} \tag{4.20}$$

where, after much simplification [71], we have

$$\Pi_\nu(n,j) = \sum_{k=\lceil\nu\rceil}^{n} \frac{(-1)^{n+k}(2j+1)(n+k)!\,(k-j-\nu+1)_j}{R^\nu(n-k)!\,k!\,\Gamma(k-\nu+1)(k-\nu+1)_{j+1}}, \tag{4.21}$$

where $(x)_j$ denotes the falling factorial as presented in (2.6). With these results, we are finally able to represent a calculable form of (4.14) under the Legendre-Gauss-Lobatto quadrature, evaluated at collocation points $x_{R,0},...,x_{R,N}$;

$$\begin{aligned}\mathbf{D}^\nu u_N(x_{R,k},t) &= \sum_{j=0}^{N}\left[\sum_{n=0}^{N}\frac{2n+1}{R}L_{R,n}(x_{R,j})\mathbf{D}^\nu L_{R,n}(x_{R,k})\omega_{R,j}\right]u(x_{R,j},t)\\ &= \sum_{j=0}^{N}\mathcal{D}^\nu_{j,k}u(x_{R,j},t),\end{aligned} \tag{4.22}$$

where

$$\mathcal{D}^\nu_{j,k} = \sum_{n=0}^{N}\sum_{i=0}^{N}\frac{2n+1}{R}L_{R,n}(x_{R,j})\Pi_\nu(n,i)L_{R,i}(x_{R,k})\omega_{R,j}. \tag{4.23}$$

With suitable expressions for the approximations of our derivatives and functions, we are finally able to arrange them in such a way as to efficiently and accurately solve our FPDE. Letting $u_n(t) = u(x_{R,n},t)$, and denoting the other functions in (4.1) and (4.2) in a similar fashion, we substitute (4.10), (4.12) and (4.22) into (4.1) to obtain

$$\frac{\partial u_n(t)}{\partial t} = \dot{u}(t,u) = -a_n(t)\sum_{j=0}^{N}\mathcal{D}^1_{n,j}u_j(t) - b_n(t)\sum_{j=0}^{N}\mathcal{D}^\nu_{n,j}u_j(t) - c_n(t)u_n(t) + q_n(t), \tag{4.24}$$

for $n = 1,...,N-1$, with initial condition

$$u_n(0) = f_n, \qquad 0 \le n \le N, \tag{4.25}$$

and noting that for $n = 0$ and $n = N$, we have boundary conditions

$$\begin{aligned} u_0(t) &= g_0(t), \qquad 0 < t \le T,\\ u_N(t) &= g_R(t), \qquad 0 < t \le T.\end{aligned} \tag{4.26}$$

Conveniently, (4.24) $\rightarrow$ (4.26) represent a system of ordinary differential equations in time, and since we have an initial condition, the system can be solved explicity, marching forward through time, from $t = 0$ to $t = T$. This system was originally solved implicitly [51], but this will not be necessary for reasonably accurate results, and so our approach differs here. The particular method to be used is the explicit fourth-order Runge-Kutta [72], where we have

$$u_n(t + \Delta t) = u_n(t) + \frac{\Delta t}{6}(K_1 + 2K_2 + 2K_3 + K_4), \tag{4.27}$$

where

$$\begin{aligned} K_1 &= \dot{u}(t, u) \\ K_2 &= \dot{u}\left(t + \frac{\Delta t}{2}, u + K_1\frac{\Delta t}{2}\right) \\ K_3 &= \dot{u}\left(t + \frac{\Delta t}{2}, u + K_2\frac{\Delta t}{2}\right) \\ K_4 &= \dot{u}(t + \Delta t, u + K_3\Delta t) \end{aligned} \tag{4.28}$$

This method should provide an accurate approximation for any space-fractional advection diffusion equation that can be represented in the form of (4.1), so we will consider some particular numerical examples.

4.2 A Numerical Example

In this example, we will consider the equation

$$\frac{\partial u(x,t)}{\partial t} = -t\sin(2x)\frac{\partial u(x,t)}{\partial x} + t^3x^3\frac{\partial^\nu u(x,t)}{\partial x^\nu} - u(x,t) + q(x,t), \tag{4.29}$$

for $x \in [0, 2]$ and $t \in (0, T]$ where

$$q(x,t) = \sin(2x)te^{-t}\left(8x - 12x^2 + 4x^3\right) - x^3t^3e^{-t}\left(\frac{24x^{4-\nu}}{\Gamma(5-\nu)} - \frac{24x^{3-\nu}}{\Gamma(4-\nu)} + \frac{8x^{3-\nu}}{\Gamma(5-\nu)}\right), \tag{4.30}$$

with initial and boundary conditions

$$\begin{aligned} u(x,0) &= x^2(2-x)^2, & 0 \le x \le 2,\\ u(0,t) &= 0, & 0 < t \le T, \\ u(R,t) &= 0, & 0 < t \le T. \end{aligned} \tag{4.31}$$

This space-fractional advection-diffusion equation has exact solution

$$u(x,t) = e^{-t}x^2(2-x)^2 \tag{4.32}$$

With fractional degree $\nu = 1.45$, $N = 10$ collocation points, and setting $\Delta t = 0.01$, according to the MATLAB® code in Appendix A, we can see in Figure 4.1 that our approximation matches the exact solution in form, shown in detail in Figure 4.2.

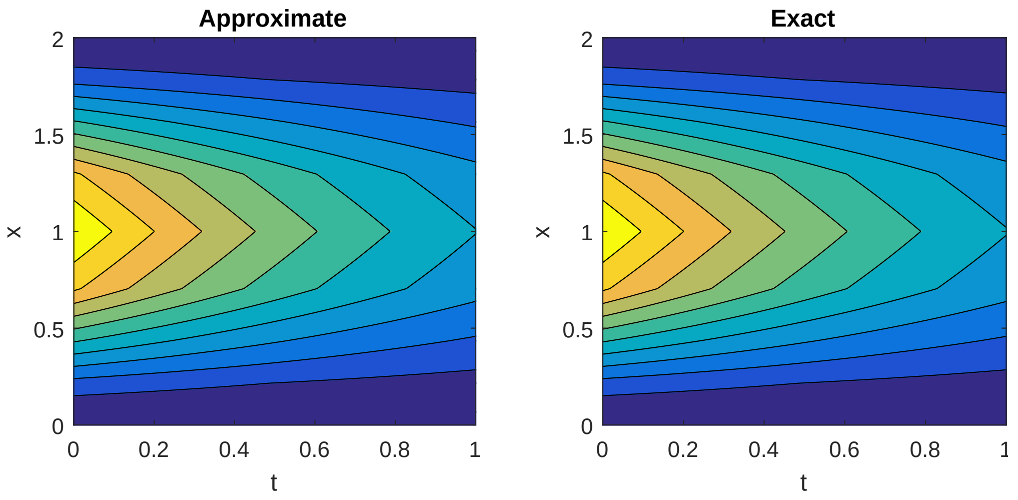


Figure 4.1. Contour Plots of Approximate and Exact Solution, $N = 10$

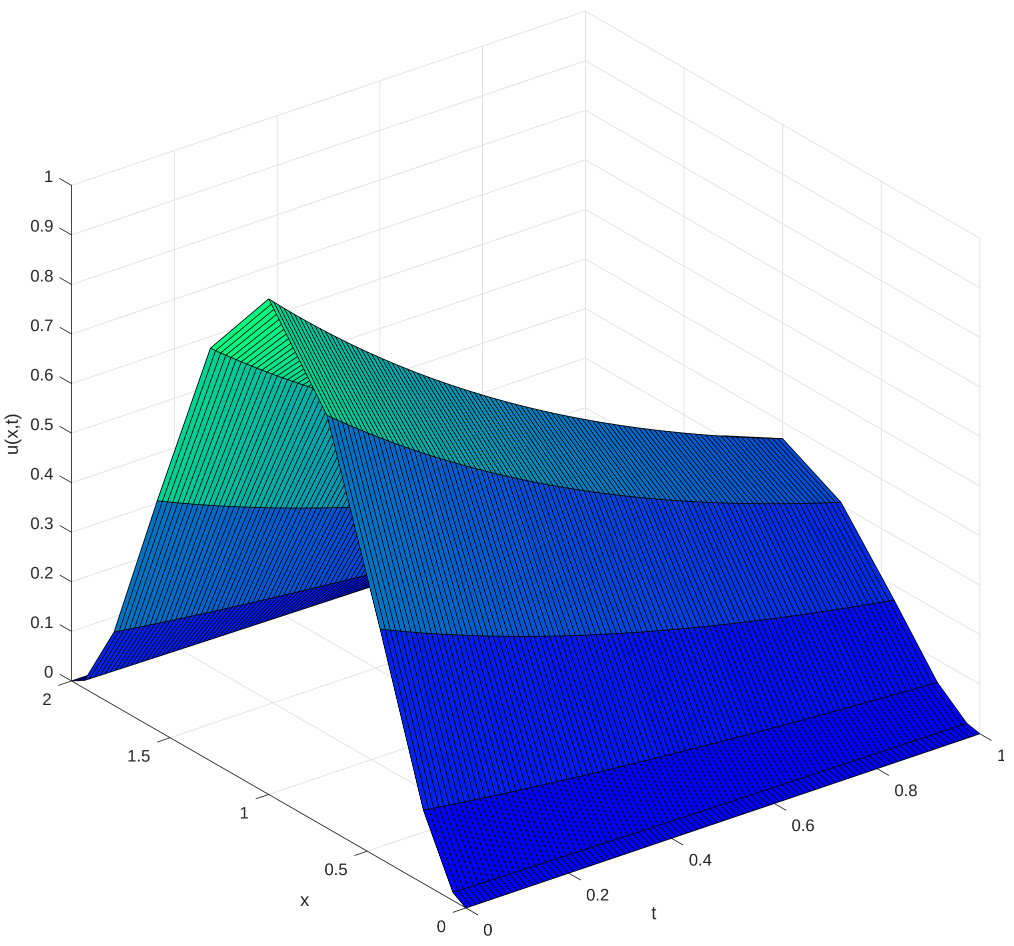


Figure 4.2. Approximate solution for $u(x,t)$, $N = 10$

With absolute error defined $|u(x,t) - u_N(x,t)|$, we observe the specific absolute error surface in Figure 4.3, showing that error increases in time.

Taking the maximum absolute error across both the space and time domain, we see in Table 4.1 that our maximum error generally diminishes as we increase N, but ticks up after $N = 9$, ostensibly due to the well-known ill-conditioned matrixes associated with spectral methods with larger values of N [73, 74] . We have demonstrated here an inferiority of explicit Runge-Kutta to its implicit counterpart [51] for the solution of the ODEs of this particular example, but nevertheless, it is clear that the Legendre-Gauss-Lobatto collocation produces viable results.

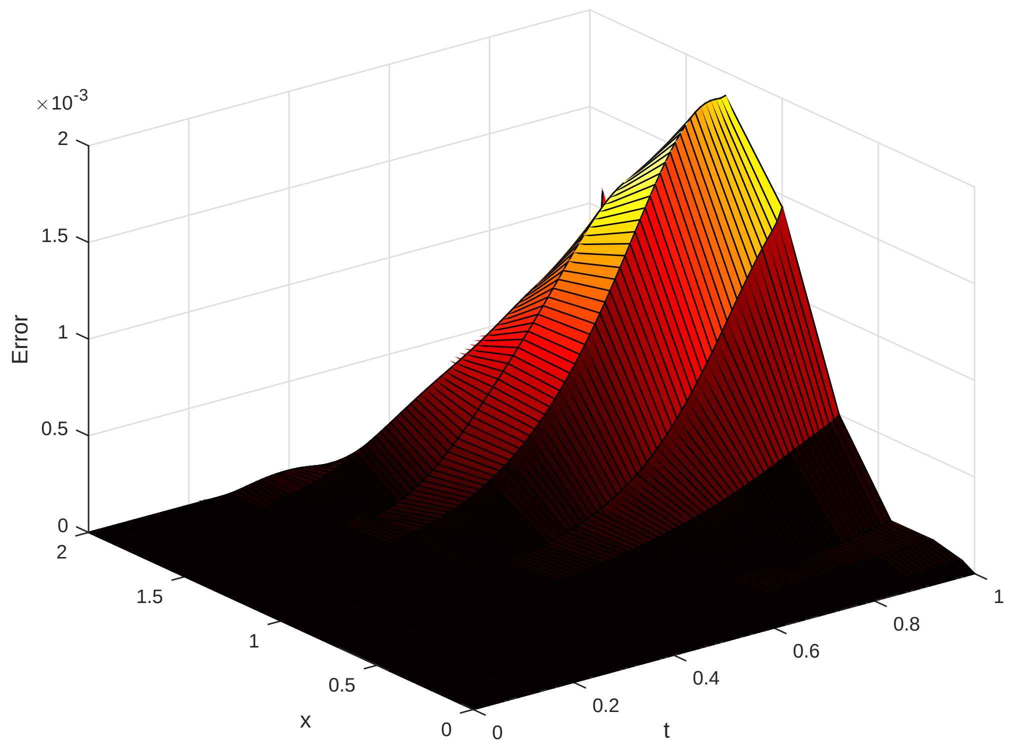


Figure 4.3. Absolute Error, $N = 10$

Table 4.1. Maximum Error for $N = 2, ..., 10$

N	Maximum Error
2	0.200850292024174
3	0.107872354548346
4	0.129915856434108
5	0.003916531162723
6	0.003795611223056
7	0.003943065660182
8	0.002273506909105
9	0.001610731333610
10	0.001879018065213

4.3 Concluding Remarks

In this chapter, we numerically solved a space-fractional advection diffusion equation using shifted Legendre polynomials under a Guass-Lobatto quadrature. Using the Caputo definition of the fractional derivative, and the technique of Legendre-Gauss-Lobatto collocation, we were able to represent the FPDE as a system of ODEs in the time variable. This system was then solved using explicit RK4, where reasonably accurate results were obtained, although not as good as those possible with an implicit method.

Chapter 5

Two-dimensional Space-Fractional Sub-Diffusion Equations

This chapter will build upon the discussion and techniques we have already explored, by extending the method of the previous chapter into two spatial dimensions, for a similar class of FPDE, and is based on the work of Bhrawy [52]. We consider two-dimensional sub-diffusion equations with fractional derivatives in both the x and y space. Building off of the technique in Chapter 4, the method involves Legendre-Gauss-Lobatto collocation of the fractional spatial derivatives, resulting in a system of ODEs in time, to be solved with explicit RK4.

In this example, we will consider equations of the form

$$\frac{\partial u(x,y,t)}{\partial t} = a(x,y)\frac{\partial^{\nu_1} u(x,y,t)}{\partial x^{\nu_1}} + b(x,y)\frac{\partial^{\nu_2} u(x,y,t)}{\partial y^{\nu_2}} + q(x,y,t), \tag{5.1}$$

where

$$0 \leq x \leq R_1, \qquad 0 \leq y \leq R_2, \qquad 0 < t \leq T,$$

with fractional derivative orders $\nu_{1,2} \in (1,2]$, and with initial and boundary conditions

$$\begin{aligned}
u(x,y,0) &= f(x,y), & 0 \le x \le R_1, && 0 \le y \le R_2, \\
u(0,y,t) &= g_{1,0}(y,t), & 0 \le y \le R_2, && 0 < t \le T, \\
u(R_1,y,t) &= g_{1,R_1}(y,t), & 0 \le y \le R_2, && 0 < t \le T, \\
u(x,0,t) &= g_{2,0}(x,t), & 0 \le x \le R_1, && 0 < t \le T, \\
u(x,R_2,t) &= g_{2,R_2}(x,t), & 0 \le x \le R_1, && 0 < t \le T.
\end{aligned} \tag{5.2}$$

We observe the anomalous diffusions $\frac{\partial^{\nu_1} u(x,y,t)}{\partial x^{\nu_1}}$ and $\frac{\partial^{\nu_2} u(x,y,t)}{\partial y^{\nu_2}}$ are again defined in the Caputo sense [52], presented in (2.17), allowing their effective treatment at the boundaries. We note the lower boundaries of 0, giving, for the fractional x derivative, the expression

$$\frac{\partial^{\nu_1} u(x,y,t)}{\partial x^{\nu_1}} = {}^{C}_{0}\mathbf{D}^{\nu_1}_{x} u(x,y,t) = \frac{1}{\Gamma(\nu_1 - n)} \int_0^x \frac{u^{(n)}(\tau,y,t)d\tau}{(x-\tau,y,t)^{\nu_1+1-n}}, \qquad (n{-}1 \le \nu_1 < n), \tag{5.3}$$

with a similar expression available for the fractional y derivative. Diffusion equations are one of the most fundamental varieties of PDEs, playing important roles in modelling such phenomena as heat conduction, flows of viscous fluids and through porous media and many more [22], with fractional diffusions able to better capture the non-locality that occasionally defines the real world [28].

5.1 Shifted Legendre-Gauss-Lobatto Collocation in Two Dimensions

As in Chapter 4, we have fully specified boundary conditions, so we will again benefit from the collocation points at the boundaries inherent to Gauss-Lobatto quadrature, and for our shifted Legendre polynomials, we adjust our $N+1$ collocation points x_n, $M+1$ collocation points y_m and quadrature weights as before:

$$\begin{aligned}
x_{R_1,n} &= \frac{R_1}{2}(x_n+1), && 0 \le n \le N,\\
y_{R_2,m} &= \frac{R_2}{2}(y_m+1), && 0 \le m \le M,\\
\omega_{R_1,j} &= \frac{R_1}{2}\omega_j, && j = 1,2,...,n,\\
\omega_{R_2,j} &= \frac{R_2}{2}\omega_j, && j = 1,2,...,m.
\end{aligned} \tag{5.4}$$

We appeal again to (3.16), noting that in two dimensions, we now have

$$\begin{aligned}
&\int_0^{R_1}\int_0^{R_2} L_{R_1}(x)w_{R_1}(x)L_{R_2}(y)w_{R_2}(y)dy\,dx\\
&\quad = \sum_{i=0}^{N}\sum_{j=0}^{M} L_{R_1}(x_{R_1,i})\omega_{R_1,i}L_{R_2}(y_{R_2,j})\omega_{R_2,j},
\end{aligned} \tag{5.5}$$

and so our shifted Legendre-Gauss-Lobatto quadrature still approximates as desired. Again, we consider equation (3.53) to express our solution as a shifted Legendre series, but adjust for the two-dimensional case [52]:

$$u(x,y,t) = \sum_{n=0}^{\infty}\sum_{m=0}^{\infty} \hat{u}_{n,m}(t)L_{R_1,n}(x)L_{R_2,m}(y), \tag{5.6}$$

which, after truncating in both dimensions, becomes

$$u_{N,M}(x_{R_1,i}, y_{R_2,j}, t) = \sum_{n=0}^{N}\sum_{m=0}^{M} \hat{u}_{n,m}(t)L_{R_1,n}(x_{R_1,i})L_{R_2,m}(y_{R_2,j}), \tag{5.7}$$

with coefficients

$$\begin{aligned}
\hat{u}_{n,m}(t) &= \frac{2n+1}{R_1}\frac{2m+1}{R_2}\int_0^{R_1}\int_0^{R_2} u(x,y,t)L_{R_1}(x)w_{R_1}(x)L_{R_2}(y)w_{R_2}(y))dy\,dx\\
&= \frac{2n+1}{R_1}\frac{2m+1}{R_2}\sum_{i=0}^{N}\sum_{j=0}^{M} u(x_{R_1,i}, y_{R_2,j}, t)L_{R_1,n}(x_{R_1,i})\omega_{R_1,i}L_{R_2,m}(y_{R_2,j})\omega_{R_2,j}.
\end{aligned} \tag{5.8}$$

Substituting back into (5.7), we obtain

$$u_{N,M}(x_{R_1,k}, y_{R_2,l}, t) = \sum_{n=0}^{N}\sum_{m=0}^{M}\sum_{i=0}^{N}\sum_{j=0}^{M}\Bigg[\frac{2n+1}{R_1}\frac{2m+1}{R_2}L_{R_1,n}(x_{R_1,k})L_{R_2,m}(y_{R_2,l}) \times u(x_{R_1,i}, y_{R_2,j}, t)L_{R_1,n}(x_{R_1,i})\omega_{R_1,i}L_{R_2,m}(y_{R_2,j})\omega_{R_2,j}\Bigg]. \tag{5.9}$$

Turning now to the fractional spatial derivatives, we recall the linear independence of Legendre polynomials, and denote $\frac{\partial^{\nu_1}u(x,y,t)}{\partial x^{\nu_1}}$, by $\mathbf{D}^{\nu_1}u(x,y,t)$, giving

$$\mathbf{D}^{\nu_1}u(x,y,t) = \sum_{n=0}^{\infty}\sum_{m=0}^{\infty}\hat{u}_{n,m}(t)\mathbf{D}^{\nu_1}L_{R_1,n}(x)L_{R_2,m}(y), \tag{5.10}$$

which, after substituting in for $\hat{u}_{n,m}(t)$ and truncating, gives

$$\begin{aligned}
&\mathbf{D}^{\nu_1}u_{N,M}(x_{R_1,k}, y_{R_2,l}, t)\\
&= \sum_{i=0}^{N}\sum_{j=0}^{M}\sum_{n=0}^{N}\sum_{m=0}^{M}\Bigg[\frac{2n+1}{R_1}\frac{2m+1}{R_2}\mathbf{D}^{\nu_1}L_{R_1,n}(x_{R_1,k})L_{R_2,m}(y_{R_2,l})\\
&\qquad L_{R_1,n}(x_{R_1,i})\omega_{R_1,i}L_{R_2,m}(y_{R_2,j})\omega_{R_2,j}u(x_{R_1,i}, y_{R_2,j}, t)\Bigg]\\
&= \sum_{i=0}^{N}\sum_{j=0}^{M}{}_{k,l}\mathcal{D}^{\nu_1}_{i,j}u(x_{R_1,i}, y_{R_2,j}, t)
\end{aligned} \tag{5.11}$$

where

$${}_{k,l}\mathcal{D}^{\nu_1}_{i,j} = \sum_{n=0}^{N}\sum_{m=0}^{M}\Bigg[\frac{2n+1}{R_1}\frac{2m+1}{R_2}\mathbf{D}^{\nu_1}L_{R_1,n}(x_{R_1,k})L_{R_2,m}(y_{R_2,l}) \times L_{R_1,n}(x_{R_1,i})\omega_{R_1,i}L_{R_2,m}(y_{R_2,j})\omega_{R_2,j}\Bigg]. \tag{5.12}$$

and where we recall from (4.17) that the fractional derivative of the Legendre polynomial is

$$\mathbf{D}^{\nu_1}L_{R_1,n}(x_{R_1,k}) = \sum_{j=0}^{n}(-1)^{n+j}\frac{(n+j)!\,x_{R_1,k}^{j-\nu_1}}{\Gamma(j-\nu_1+1)(n-j)!\,j!\,R_1^j}, \qquad n = \lceil\nu_1\rceil, \lceil\nu_1\rceil+1, ... \tag{5.13}$$

Similarly, in the y dimension, denoting $\frac{\partial^{\nu_2} u(x,y,t)}{\partial y^{\nu_2}}$, by $\mathbf{D}^{\nu_2} u(x,y,t)$ we have

$$\mathbf{D}^{\nu_2} u(x,y,t) = \sum_{n=0}^{\infty} \sum_{m=0}^{\infty} \hat{u}_{n,m}(t) L_{R_1,n}(x) \mathbf{D}^{\nu_2} L_{R_2,m}(y), \tag{5.14}$$

which gives

$$\begin{aligned} &\mathbf{D}^{\nu_2} u_{N,M}(x_{R_1,k}, y_{R_2,l}, t) \\ &= \sum_{i=0}^{N} \sum_{j=0}^{M} \sum_{n=0}^{N} \sum_{m=0}^{M} \left[\frac{2n+1}{R_1} \frac{2m+1}{R_2} L_{R_1,n}(x_{R_1,k}) \mathbf{D}^{\nu_2} L_{R_2,m}(y_{R_2,l}) \right. \\ &\qquad \left. \times L_{R_1,n}(x_{R_1,i}) \omega_{R_1,i} L_{R_2,m}(y_{R_2,j}) \omega_{R_2,j} u(x_{R_1,i}, y_{R_2,j}, t) \right] \\ &= \sum_{i=0}^{N} \sum_{j=0}^{M} {}_{k,l}\mathcal{D}^{\nu_2}_{i,j} u(x_{R_1,i}, y_{R_2,j}, t) \end{aligned} \tag{5.15}$$

where

$$\begin{aligned} {}_{k,l}\mathcal{D}^{\nu_2}_{i,j} = \sum_{n=0}^{N} \sum_{m=0}^{M} &\left[\frac{2n+1}{R_1} \frac{2m+1}{R_2} L_{R_1,n}(x_{R_1,k}) \mathbf{D}^{\nu_2} L_{R_2,m}(y_{R_2,l}) \right. \\ &\qquad \left. \times L_{R_1,n}(x_{R_1,i}) \omega_{R_1,i} L_{R_2,m}(y_{R_2,j}) \omega_{R_2,j} \right]. \end{aligned} \tag{5.16}$$

with the fractional derivative of the Legendre polynomial being

$$\mathbf{D}^{\nu_2} L_{R_2,m}(y_{R_1,l}) = \sum_{j=0}^{m} (-1)^{m+j} \frac{(m+j)!\, x_{R_2,l}^{j-\nu_2}}{\Gamma(j-\nu_2+1)(m-j)!\, j!\, R_2^j}, \quad m = \lceil \nu_2 \rceil, \lceil \nu_2 \rceil + 1, ... \tag{5.17}$$

Having obtained calculable expressions for the fractional derivatives in both dimensions, we are able to express our fractional PDE as a system of ordinary differential equations evaluated at the collocation points. Letting $u(x_{R_1,n}, y_{R_2,m}, t) = u_{n,m}(t)$, and denoting the other functions in (5.23) and (7.9) similarly, we substitute (5.11) and (5.15) into (5.23):

$$\begin{aligned}\frac{\partial u_{n,m}(t)}{\partial t} &= a_{n,m}\sum_{i=0}^{N}\sum_{j=0}^{M} {}_{n,m}\mathcal{D}^{\nu_1}_{i,j}u_{n,m}(t) + b_{n,m}\sum_{i=0}^{N}\sum_{j=0}^{M} {}_{n,m}\mathcal{D}^{\nu_2}_{i,j}u_{n,m}(t) + q_{n,m}(t)\\ &= \dot{u}(t,u),\end{aligned} \tag{5.18}$$

for $n = 1, ..., N-1$, and $m = 1, ..., M-1$, with initial condition

$$u_{n,m}(0) = f_{n,m}, \qquad 0 \le n \le N, \qquad 0 \le m \le M, \tag{5.19}$$

and noting that for $n = 0$, $n = N$, $m = 0$, and $m = M$ we have boundary conditions

$$\begin{aligned} u_{0,m}(t) &= g_{1,0,m}(t), & 0 \le m \le M, && 0 < t \le T,\\ u_{N,m}(t) &= g_{1,R_1,m}(t), & 0 \le m \le M, && 0 < t \le T,\\ u_{n,0}(t) &= g_{2,0,n}(t), & 0 \le n \le N, && 0 < t \le T,\\ u_{n,M}(t) &= g_{2,R_2,n}(t), & 0 \le n \le N, && 0 < t \le T. \end{aligned} \tag{5.20}$$

We now have a system of $(N+1)\times(M+1)$ ordinary differential equations for each step in time, which will benefit from explicit RK4, where we have

$$u_{n,m}(t+\Delta t) = u_{n,m}(t) + \frac{\Delta t}{6}(K_1 + 2K_2 + 2K_3 + K_4), \tag{5.21}$$

where

$$\begin{aligned} K_1 &= \dot{u}(t,u)\\ K_2 &= \dot{u}\left(t + \frac{\Delta t}{2}, u + K_1\frac{\Delta t}{2}\right)\\ K_3 &= \dot{u}\left(t + \frac{\Delta t}{2}, u + K_2\frac{\Delta t}{2}\right)\\ K_4 &= \dot{u}(t+\Delta t, u + K_3\Delta t) \end{aligned} \tag{5.22}$$

as before. This scheme will allow us to march through time as we solve an evolving 2D surface. To demonstrate the effectiveness of this method, we will consider a particular numerical example.

5.2 A Numerical Example

In this example, we will consider the equation

$$\frac{\partial u(x,y,t)}{\partial t} = a(x,y)\frac{\partial^{\nu_1} u(x,y,t)}{\partial x^{\nu_1}} + b(x,y)\frac{\partial^{\nu_2} u(x,t)}{\partial y^{\nu_2}} + q(x,y,t), \tag{5.23}$$

for $x \in [0,1]$, $y \in [0,1]$, and $t \in (0,T]$, where

$$\begin{aligned} a(x,y) &= \frac{(3-2x)\Gamma(3-\nu_1)}{2} \\ b(x,y) &= \frac{(4-y)\Gamma(4-\nu_2)}{6} \\ q(x,y,t) &= e^{-t}\Big(x^2(-y^{\frac{3}{2}} + y - 4)y^{\frac{3}{2}} + \sqrt{x}(2x-3)y^3\Big), \end{aligned} \tag{5.24}$$

with initial and boundary conditions

$$\begin{aligned} u(x,y,0) &= x^2y^3, && 0 \le x \le 1, && 0 \le y \le 1, \\ u(0,y,t) &= 0, && 0 \le y \le 1, && 0 < t \le T, \\ u(1,y,t) &= e^{-t}y^3, && 0 \le y \le 1, && 0 < t \le T, \\ u(x,0,t) &= 0, && 0 \le x \le 1, && 0 < t \le T, \\ u(x,1,t) &= e^{-t}x^2, && 0 \le x \le 1, && 0 < t \le T. \end{aligned} \tag{5.25}$$

This two-dimensional space-fractional sub-diffusion equation has exact solution

$$u(x,y,t) = e^{-t}x^2y^3. \tag{5.26}$$

With fractional derivative degrees $\nu_1 = \nu_2 = 1.5$, $N = 10$ collocation points, and $\Delta t = 0.001$, implemented in MATLAB as shown in Appendix D, we see in Figure 5.1 that the approximation matches the exact solution in shape, with the detailed approximation surface in Figure 5.2.

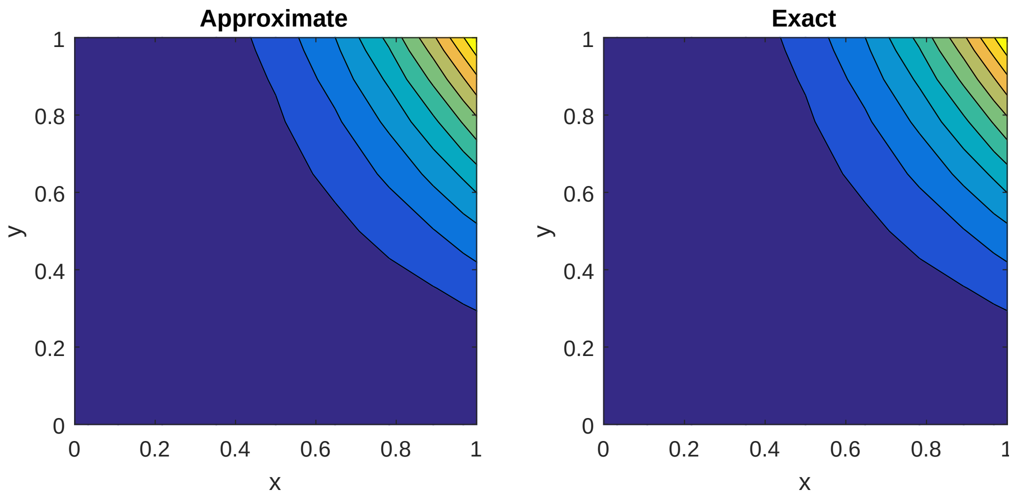


Figure 5.1. Contours of Approximate and Exact Solution for $u(x, y, 1)$, $N = 10$

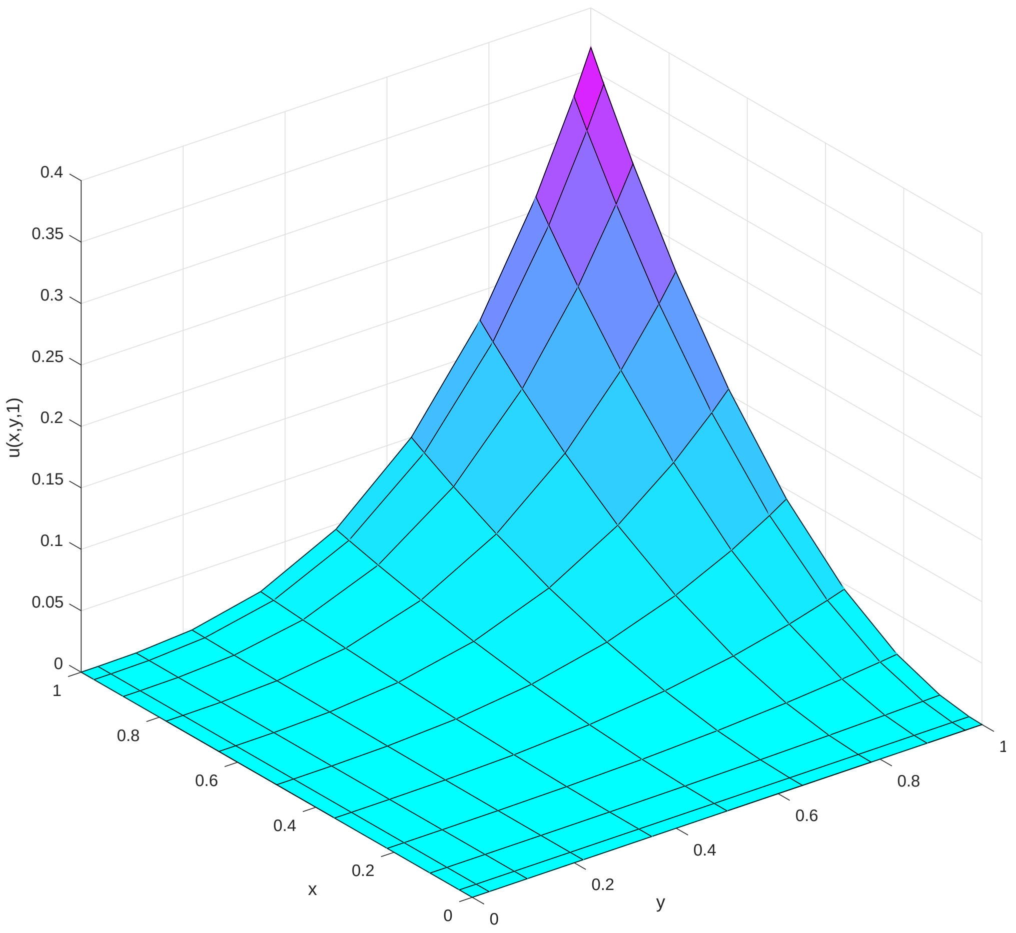


Figure 5.2. Approximate solution for $u(x, y, 1)$, $N = 10$

Defining absolute error for $t = 1$ as $|u(x, y, 1) - u_{N,M}(x, y, 1)|$, we observe the specific absolute error surface in Figure 5.3, where we note that the error is concentrated at the upper boundaries.

Considering the maximum absolute error across all three domains, defined by

$$\max_{0\leq x\leq 1, 0\leq y\leq 1, 0<t\leq 1} |u(x, y, t) - u_{N,M}(x, y, t)|,$$

presented in Table 5.1, we observe that there is a significant jump in accuracy from $N = 4$, where the scheme in fact has the smallest maximum error at less than 1.2×10^{-14}. Thereafter, accuracy slowly declines, but is still particularly good up to and beyond $N = 10$. Notably, these results compare favorably to those obtained with implicit Runge-Kutta in prior work [52].

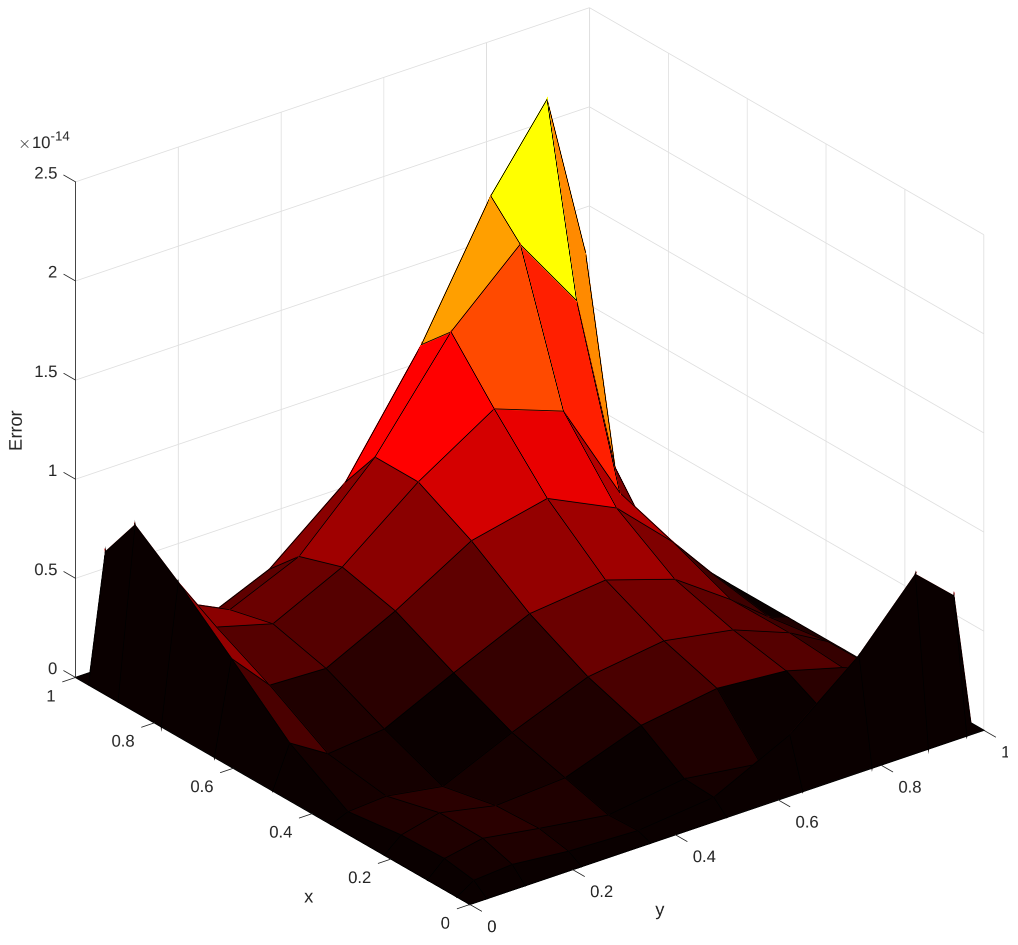


Figure 5.3. Absolute Error, $N = 10$

Table 5.1. Maximum Error for $N = M = 2, \ldots, 10$

N	Maximum Error
2	0.013226556903634
3	0.005283940097415
4	$1.193489751472043 \times 10^{-14}$
5	$1.612598943268040 \times 10^{-14}$
6	$2.431388423929093 \times 10^{-14}$
7	$3.064215547965432 \times 10^{-14}$
8	$4.007905118896815 \times 10^{-14}$
9	$4.607425552194400 \times 10^{-14}$
10	$5.428990590417016 \times 10^{-14}$

5.3 Concluding Remarks

In this chapter, we numerically solved a two-dimensional space-fractional diffusion equation using shifted Legendre polynomials under a Guass-Lobatto quadrature. Using the Caputo definition of the fractional derivative, and the technique of Legendre-Gauss-Lobatto collocation, we were able to represent the FPDE as a system of ODEs in the time variable. This system was then solved using explicit RK4, where particularly accurate results were obtained, comparable to those in prior literature obtained using the more expensive implicit schemes.

Chapter 6

One-dimensional Space- and Time- Fractional Equations

This chapter is based on the work of Doha, Bhrawy and Ezz-Eldien [50, 53], wherein they investigated ordinary differential equations, and time-fractional diffusion equations, with fractional derivatives of order less than 1. We will adapt this method to consider a variety of partial differential equations, linear and non-linear, with a single dimension in space, that have fractional derivatives in either space or time of order up to 2, and may additionally have derivative boundary conditions. The solution will involve shifted Jacobi polynomials, integrated under a Gauss-Lobatto quadrature, for the calculation of both the integer derivatives and the fracional derivatives. This spectral scheme will provide us with a system of non-linear equations, which we solve with a Levenberg-Marquardt algorithm.

In this chapter, we will consider time-fractional partial differential equations that take the following form:

$$f\left(x, t, u, \frac{\partial u}{\partial x}, \frac{\partial u}{\partial t}, \frac{\partial^2 u}{\partial x^2}, \frac{\partial^2 u}{\partial t^2}, \frac{\partial^2 u}{\partial x\, \partial t}, ..., \frac{\partial^\nu u}{\partial t^\nu}, ...\right) = 0 \tag{6.1}$$

where the operator f can be linear or non-linear, homogeneous or nonhomogeneous. These equations will be well-posed, with appropriate initial and boundary conditions. We note that the fractional derivative $\frac{\partial^\nu u}{\partial t^\nu}$ is defined in the Caputo sense, so that it will be amenable to the imposed initial conditions. We set the lower limit of integration to be 0, giving us, for the fractional time derivative, the following expression:

$$\frac{\partial^\nu u(x,t)}{\partial t^\nu} = {}_0^C\mathbf{D}_t^\nu u(x,t) = \frac{1}{\Gamma(\nu-n)}\int_0^t \frac{u^{(n)}(x,\tau)d\tau}{(x,t-\tau)^{\nu+1-n}}, \qquad (n-1 \le \nu < n), \tag{6.2}$$

with corresponding expressions available for the fractional derivatives in the spatial dimensions.

6.1 Shifted Jacobi-Gauss-Lobatto Collocation Matrices

In this section, we will expound upon the derivation of a Jacobi operational matrix for the solution of fractional PDEs, first presented in Doha, Bhrawy and Ezz-Eldien [50], attempting where necessary to fill in detail omitted from the original proof. Recalling § 3.4.1, we represent a Jacobi polynomial of degree n on domain $[0,R]$ by

$$P_{R,n}(x)^{(\alpha,\beta)} = \sum_{j=0}^{n}(-1)^{n+j}\frac{\Gamma(n+\alpha+1)\Gamma(n+j+\alpha+\beta+1)}{\Gamma(\alpha+j+1)\Gamma(n+\alpha+\beta+1)(n-j)!\,j!\,R^j}x^j. \tag{6.3}$$

The shifted weight function is $w_R^{(\alpha,\beta)} = x^\alpha(R-x)^\beta$, and we have the orthogonality of the polynomials by

$$\int_0^R P_{R,k}(x)^{(\alpha,\beta)}P_{R,m}(x)^{(\alpha,\beta)}w_R^{(\alpha,\beta)}dx = h_{R,m} \tag{6.4}$$

where, if $m \neq k$, we have $h_{R,m} = 0$, but if $m = k$, we have

$$h_{R,m} = \frac{R^{\alpha+\beta+1}\Gamma(m+\alpha+1)\Gamma(m+\beta+1)}{(2m+\alpha+\beta+1)m!\,\Gamma(m+\alpha+\beta+1)} \tag{6.5}$$

This gives shifted Jacobi series

$$u(x) = \sum_{n=0}^{\infty}\hat{u}_n P_{R,n}(x)^{(\alpha,\beta)}(x), \tag{6.6}$$

with coefficients

$$\hat{u}_n = \frac{(2n+\alpha+\beta+1)n!\,\Gamma(n+\alpha+\beta+1)}{R^{\alpha+\beta+1}\Gamma(n+\alpha+1)\Gamma(n+\beta+1)} \int_0^R u(x) P_{R,n}^{(\alpha,\beta)}(x) x^\alpha (R-x)^\beta dx. \quad (6.7)$$

Now, letting

$$\mathbf{U} = [\hat{u}_0, \hat{u}_1, ..., \hat{u}_N], \quad (6.8)$$

and

$$\mathbf{P}_{R_1}(x) = [P_{R,0}(x)^{(\alpha,\beta)}, P_{R,1}(x)^{(\alpha,\beta)}, ..., P_{R,N}(x)^{(\alpha,\beta)}]', \quad (6.9)$$

we have

$$u(x) = \mathbf{U}\mathbf{P}_{R_1}(x) \quad (6.10)$$

We seek an operational matrix that will permit derivatives of $u(x)$ of fractional order, where we denote $\frac{\partial^\nu u(x,t)}{\partial x^\nu}$ by $\mathbf{D}^\nu u(x,t)$. Before we can do this, we must prove the following lemma:

Lemma 6.1 *Let $P_{R,n}(x)^{(\alpha,\beta)}$ be a shifted Jacobi polynomial. Then,*

$$\mathbf{D}^\nu P_{R,n}(x)^{(\alpha,\beta)} = 0, \qquad n = 0, 1, ..., \lceil \nu \rceil - 1. \quad (6.11)$$

Proof: We observe from (2.25) that for integers γ and m, if we have $m < \nu \leq m+1$, we have

$$\mathbf{D}^\nu x^\gamma = \begin{cases} 0 & \text{if } \gamma < m+1 \\ \frac{\Gamma(\gamma+1)}{\Gamma(\gamma-\nu+1)} x^{\gamma-\nu}, & \text{if } \gamma \geq m+1 \end{cases} \quad (6.12)$$

Considering additionally the linearity of fractional derivatives [1], we observe from (6.3) that, for $n = 0, 1, ..., \lceil \nu \rceil - 1$,

$$\begin{aligned} \mathbf{D}^\nu P_{R,n}(x)^{(\alpha,\beta)} &= \sum_{j=0}^{n} (-1)^{n+j} \frac{\Gamma(n+\alpha+1)\Gamma(n+j+\alpha+\beta+1)}{\Gamma(\alpha+j+1)\Gamma(n+\alpha+\beta+1)(n-j)!\,j!\,R^j} \mathbf{D}^\nu x^j. \\ &= 0, \end{aligned} \quad (6.13)$$

and we have the result, as desired. $\square$

Theorem 6.1 *Let $\mathbf{P}_{R_1}(x)$ be a vector of Jacobi polynomials as defined in (6.9), and $\mu > 0$. Then,*

$$\mathbf{D}^{\nu}\mathbf{P}_{R_1}(x) \simeq \mathcal{D}_R^{\nu}\mathbf{P}_{R_1}(x), \tag{6.14}$$

where $\mathcal{D}_R^{\nu}$ is the $(N+1)\times(N+1)$ operational matrix of ν-ordered derivatives in the Caputo sense, defined by

$$\mathcal{D}_R^{\nu} = \begin{bmatrix} 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 0 \\ \Delta^{\nu}(\lceil\nu\rceil,0) & \Delta^{\nu}(\lceil\nu\rceil,1) & \Delta^{\nu}(\lceil\nu\rceil,2) & \dots & \Delta^{\nu}(\lceil\nu\rceil,N) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \Delta^{\nu}(n,0) & \Delta^{\nu}(n,1) & \Delta^{\nu}(n,2) & \dots & \Delta^{\nu}(n,N) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \Delta^{\nu}(N,0) & \Delta^{\nu}(N,1) & \Delta^{\nu}(N,2) & \dots & \Delta^{\nu}(N,N) \end{bmatrix}$$

where

$$\begin{aligned} &\Delta^{\nu}(n,k) \\ &= \sum_{j=\lceil\nu\rceil}^{n}\left[\frac{(-1)^{n-j}R^{\alpha+\beta-\nu+1}\Gamma(j+\alpha+1)\Gamma(n+\alpha+1)\Gamma(n+j+\alpha+\beta+1)}{h_{R,j}\Gamma(j+\alpha+\beta+1)\Gamma(j+\alpha+1)\Gamma(n+\alpha+\beta+1)\Gamma(j-\nu+1)(n-j)!}\right. \\ &\qquad\left.\times\sum_{r=0}^{k}\frac{(-1)^{k-r}\Gamma(k+r+\alpha+\beta+1)\Gamma(\beta+1)\Gamma(r+k+\alpha-\nu+1)}{\Gamma(r+\alpha+1)\Gamma(r+k+\alpha+\beta-\nu+2)(k-r)!\,r!}\right] \end{aligned} \tag{6.15}$$

Proof: We recall (6.3) and apply the fractional derivative of degree ν, recalling additionally the linearity of the fractional derivative and the expression of (6.12):

$$\begin{aligned}
&\mathbf{D}^{\nu} P_{R,n}(x)^{(\alpha,\beta)} \\
&\quad = \sum_{j=0}^{n} (-1)^{n+j} \frac{\Gamma(n+\alpha+1)\Gamma(n+j+\alpha+\beta+1)}{\Gamma(\alpha+j+1)\Gamma(n+\alpha+\beta+1)(n-j)!\, j!\, R^{j}} \mathbf{D}^{\nu} x^{j}. \\
&\quad = \sum_{\lceil \nu \rceil = 0}^{n} (-1)^{n+j} \frac{\Gamma(n+\alpha+1)\Gamma(n+j+\alpha+\beta+1)}{\Gamma(\alpha+j+1)\Gamma(n+\alpha+\beta+1)(n-j)!\, \Gamma(j-\nu+1) R^{j}} x^{j-\nu}
\end{aligned} \tag{6.16}$$

Now, we approximate $x^{j-\nu}$ as an expansion of Jacobi polynomials:

$$x^{j-\nu} = \sum_{k=0}^{\infty} \hat{\theta}_{j,k} P_{R,k}(x)^{(\alpha,\beta)}(x), \tag{6.17}$$

which, by (6.7) has coefficients

$$\begin{aligned}
\hat{\theta}_{j,k} = &\ \frac{1}{h_{R,k}} \int_0^R x^{j-\nu} P_{R,k}^{(\alpha,\beta)}(x) x^{\alpha} (R-x)^{\beta} dx. \\
= &\ \frac{1}{h_{R,k}} \int_0^R x^{j-\nu} x^{\alpha} (R-x)^{\beta} \\
&\ \times \sum_{r=0}^{k} (-1)^{k+r} \frac{\Gamma(k+\alpha+1)\Gamma(k+r+\alpha+\beta+1)}{\Gamma(\alpha+r+1)\Gamma(k+\alpha+\beta+1)(k-r)!\, r!\, R^{r}} x^{r} dx \\
= &\ \frac{1}{h_{R,k}} \sum_{r=0}^{k} (-1)^{k-r} \frac{\Gamma(k+\alpha+1)\Gamma(k+r+\alpha+\beta+1)}{\Gamma(\alpha+r+1)\Gamma(k+\alpha+\beta+1)(k-r)!\, r!\, R^{r}} \\
&\ \times \int_0^R x^{\alpha+j+r-\nu} (R-x)^{\beta} dx \qquad (6.18) \\
= &\ \frac{1}{h_{R,k}} \sum_{r=0}^{k} \Bigg[(-1)^{k-r} \frac{\Gamma(k+\alpha+1)\Gamma(k+r+\alpha+\beta+1)}{\Gamma(\alpha+r+1)\Gamma(k+\alpha+\beta+1)(k-r)!\, r!\, R^{r}} \\
&\ \times \frac{\Gamma(\beta+1)\Gamma(j+r+\alpha-\nu+1)}{\Gamma(j+r+\alpha+\beta-\nu+2)} R^{\alpha+\beta+j+r-\nu+1} \Bigg] \\
= &\ \frac{\Gamma(k+\alpha+1) R^{\alpha+\beta+j-\nu+1}}{\Gamma(k+\alpha+\beta+1) h_{R,k}} \\
&\ \times \sum_{r=0}^{k} (-1)^{k-r} \frac{\Gamma(\beta+1)\Gamma(k+r+\alpha+\beta+1)\Gamma(j+r+\alpha-\nu+1)}{\Gamma(\alpha+r+1)\Gamma(j+r+\alpha+\beta-\nu+2)(k-r)!\, r!}
\end{aligned}$$

Now, substituting (6.18) into (6.16), we observe that

$$\mathbf{D}^{\nu} P_{R,n}(x)^{(\alpha,\beta)} = \sum_{k=0}^{N} \Delta^{\nu}(n,k) P_{R,k}(x)^{(\alpha,\beta)}, \qquad n = \lceil \nu \rceil, \lceil \nu \rceil + 1, ..., N, \tag{6.19}$$

which gives, in vector form,

$$\mathbf{D}^{\nu} P_{R,n}(x)^{(\alpha,\beta)} \simeq [\Delta^{\nu}(n,0), \Delta^{\nu}(n,1), ..., \Delta^{\nu}(n,N)]\, \mathbf{P}_{R_1}(x),\ n = \lceil \nu \rceil, \lceil \nu \rceil + 1, ..., N. \tag{6.20}$$

Additionally, we note that from Lemma 6.1, we have

$$\mathbf{D}^{\nu} P_{R,n}(x)^{(\alpha,\beta)} \simeq [0, 0, ..., 0]\, \mathbf{P}_{R_1}(x), \qquad n = 0, 1, ..., \lceil \nu \rceil - 1. \tag{6.21}$$

Thus, we have from (6.20) and (6.21) that

$$\mathbf{D}^{\nu} \mathbf{P}_{R_1}(x) \simeq \mathcal{D}_R^{\nu} \mathbf{P}_{R_1}(x), \tag{6.22}$$

and the theorem is proven [1]. □

Now, in order to implement a space- or time-fractional derivative, we will have to expand $u(x,t)$ in both the time and space dimensions as a series of Jacobi polynomials [53], similar to the process used in (5.6). This provides the following approximation of $u(x,t)$:

$$u(x,t) = \sum_{n=0}^{\infty} \sum_{m=0}^{\infty} \hat{u}_{m,n} P_{R,n}^{\alpha,\beta}(x) P_{T,m}^{\alpha,\beta}(t), \tag{6.23}$$

which, after truncating in both dimensions, becomes

$$u_{M,N}(x_{R,i}, t_{T,j}) = \sum_{n=0}^{N} \sum_{m=0}^{M} \hat{u}_{m,n} P_{R,n}^{\alpha,\beta}(x_{R,i}) P_{T,m}^{\alpha,\beta}(t_{T,j}), \tag{6.24}$$

with coefficients

$$\hat{u}_{m,n} = \frac{1}{h_{R,n}} \frac{1}{h_{T,m}} \int_0^R \int_0^T u(x,t) P_{R,n}^{\alpha,\beta}(x) w_R^{(\alpha,\beta)} P_{T,m}^{\alpha,\beta}(t) w_T^{(\alpha,\beta)} dt\, dx. \tag{6.25}$$

[1] Consult Appendix H for the MATLAB implementation of $\Delta^{\nu}(n,j)$

Then, defining matrix

$$\mathbf{U} = \begin{bmatrix} \hat{u}_{0,0} & \hat{u}_{0,1} & \dots & \hat{u}_{0,N} \\ \hat{u}_{1,0} & \hat{u}_{1,1} & \dots & \hat{u}_{1,N} \\ \vdots & \vdots & \ddots & \vdots \\ \hat{u}_{M,0} & \hat{u}_{M,1} & \dots & \hat{u}_{M,N} \end{bmatrix}$$

we observe that

$$u_{M,N}(x_{R,i}, t_{T,j}) = [\mathbf{P}_T(t)]' \, \mathbf{U} \, \mathbf{P}_{R_1}(x). \tag{6.26}$$

Now, if we apply Theorem 6.1 to (6.26), we observe that the derivatives of (6.1) can be represented by

$$\begin{aligned} \frac{\partial u(x,t)}{\partial x} &\simeq [\mathbf{P}_T(t)]' \, \mathbf{U} \, \mathcal{D}_R^1 \mathbf{P}_R(x) \\ \frac{\partial u(x,t)}{\partial t} &\simeq \left[\mathcal{D}_T^1 \mathbf{P}_T(t)\right]' \mathbf{U} \, \mathbf{P}_R(x) \\ \frac{\partial^2 u(x,t)}{\partial x^2} &\simeq [\mathbf{P}_T(t)]' \, \mathbf{U} \, \mathcal{D}_R^2 \mathbf{P}_R(x) \\ \frac{\partial^2 u(x,t)}{\partial t^2} &\simeq \left[\mathcal{D}_T^2 \mathbf{P}_T(t)\right]' \mathbf{U} \, \mathbf{P}_R(x) \\ \frac{\partial^2 u(x,t)}{\partial x \, \partial t} &\simeq \left[\mathcal{D}_T^1 \mathbf{P}_T(t)\right]' \mathbf{U} \, \mathcal{D}_R^1 \mathbf{P}_R(x) \\ &\vdots \\ \frac{\partial^\nu u(x,t)}{\partial t^\nu} &\simeq [\mathcal{D}_T^\nu \mathbf{P}_T(t)]' \, \mathbf{U} \, \mathbf{P}_R(x) \\ &\vdots \end{aligned} \tag{6.27}$$

Now, by substituting (6.26) and (6.27) into (6.1) and its initial and boundary conditions, we obtain a system of non-linear equations. A Levenberg-Marquardt algorithm, taking **U** as its variable, is used to minimise these equations as a set of non-linear least squares problems, with the robustness of this approach making it more desirable than similar methods, such as Gauss-Newton, considering the absence of a sufficiently close initial guess that might be necessary for a less sophisticated algorithm [75]. This $\mathbf{U}$ is then used in (6.26) to acquire our approximate surface of $u(x,t)$. Having discussed the methodology, we will test its veracity by deriving the schemes for a number of particular space- and time-fractional PDEs, and then implement these schemes on appropriate numerical examples. We note here that while consistency of presentation will be prioritised, the error between exact and approximate solutions will be reported as close to the form used in the specific prior literature from which each example sourced as possible, for the sake of fair comparison between our methods and those of previous investigations.

6.2 Non-homogenous Sub-diffusion Equations

In this section, we will consider equations of the form

$$\frac{\partial^\nu u(x,y,t)}{\partial t^\nu} = A\frac{\partial^2 u(x,t)}{\partial x^2} + q(x,t), \tag{6.28}$$

where

$$0 \le x \le R, \qquad 0 < t \le T,$$

with A being the generalised diffusion constant, ν being the fractional differentiation constant in the region $(0,1]$, and $q(x,t)$ being a non-homogeneous source function. We have initial and boundary conditions

$$\begin{aligned} u(x,0) &= f(x), && 0 \le x \le R, \\ u(0,t) &= g_0(t), && 0 < t \le T, \\ u(R,t) &= g_R(t), && 0 < t \le T. \end{aligned} \tag{6.29}$$

We begin by recalling (6.27), and observe that the derivatives of (6.28) can be represented by

$$\begin{aligned}\frac{\partial^{1-\nu}u(x,t)}{\partial t^{1-\nu}} &\simeq \left[\mathcal{D}_T^{1-\nu}\mathbf{P}_T(t)\right]'\mathbf{U}\,\mathbf{P}_R(x),\\ \frac{\partial^2 u(x,t)}{\partial x^2} &\simeq \left[\mathbf{P}_T(t)\right]'\mathbf{U}\,\mathcal{D}_R^2\mathbf{P}_R(x).\end{aligned} \tag{6.30}$$

Now, by substituting (6.26) and (6.30) into (6.28), we obtain

$$\left[\mathbf{P}_T(t_{T,j})\right]'\left(\left[\mathcal{D}_T^{1-\nu}\right]'\mathbf{U} - \mathbf{U}\,A\mathcal{D}_R^2\right)\mathbf{P}_R(x_{R,i}) - q\left(x_{R,i}, t_{T,j}\right) = 0, \tag{6.31}$$

for $i = 1, 2, ..., N-1$, and $j = 1, 2, ..., M$. For the initial and boundary conditions, we have

$$\begin{aligned}\left[\mathbf{P}_T(0)\right]'\mathbf{U}\,\mathbf{P}_R(x_{R,i}) - f(x_{R,i}) &= 0, & 0 \le i \le N,\\ \left[\mathbf{P}_T(t_{T,j})\right]'\mathbf{U}\,\mathbf{P}_R(0) - g_0(t_{T,j}) &= 0, & 0 < j \le M,\\ \left[\mathbf{P}_T(t_{T,j})\right]'\mathbf{U}\,\mathbf{P}_R(R) - g_R(t_{T,j}) &= 0, & 0 < j \le M.\end{aligned} \tag{6.32}$$

Combining (6.31) and (6.32), we obtain a system of $(M+1)\times(N+1)$ equations. We use the Levenberg-Marquardt algorithm, taking $\mathbf{U}$ as its variable, to minimise (6.31) and (6.32). This $\mathbf{U}$ is then used in (6.26) to calculate our approximation of $u(x,t)$.

6.2.1 A Numerical Example

We consider here a time-fractional sub-diffusion implemented by Gao and Sun [76]:

$$\frac{\partial^\nu u(x,t)}{\partial t^\nu} = \frac{\partial^2 u(x,t)}{\partial x^2} + e^x\Gamma(2+\nu)t - t^{1+\nu}, \tag{6.33}$$

where

$$0 \le x \le 1, \qquad 0 < t \le 1,$$

with ν being the fractional differentiation constant in the region $(0, 1]$. We have initial and boundary conditions

$$\begin{aligned} u(x,0) &= 0, && 0 \le x \le 1, \\ u(0,t) &= t^{1+\nu}, && 0 < t \le 1, \\ u(1,t) &= e^1 t^{1+\nu}, && 0 < t \le 1. \end{aligned} \tag{6.34}$$

This non-homogeneous time-fractional sub-diffusion equation has exact solution

$$u(x,t) = e^x t^{1+\nu}. \tag{6.35}$$

With fractional time derivative degree $\nu = 0.75$, and $N = M = 12$ collocation points, implemented in MATLAB as shown in Appendix I, we see in Figure 6.1 that the approximation matches the exact solution in shape, with the detailed approximation surface in Figure 6.2.

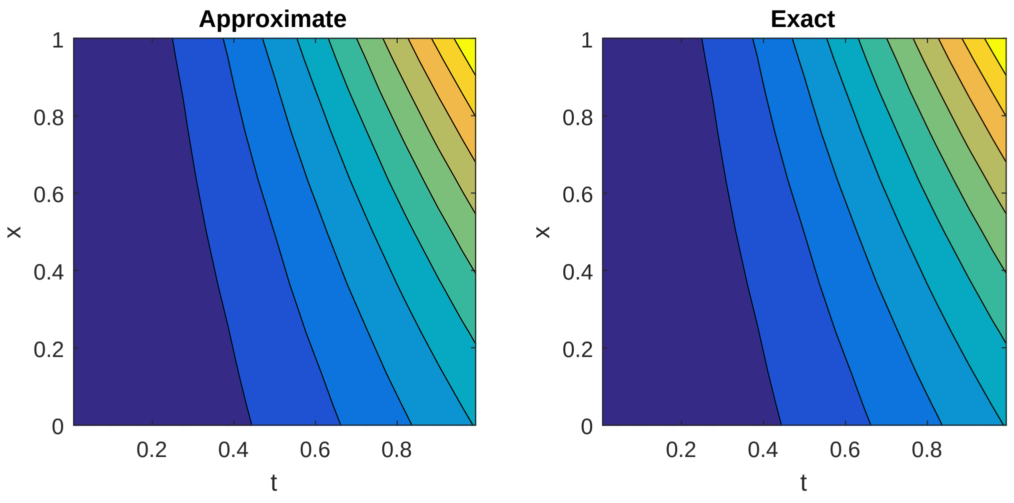


Figure 6.1. Contours of Approximate and Exact Solution, $N = M = 12$

With absolute error given by $|u(x,t) - u_{M,N}(x,t)|$, we observe the specific absolute error surface in Figure 6.3, where we note that the error is concentrated in the middle for the spatial dimension, and early on in the time dimension, diminishing as the solution evolves.

Defining the maximum error at the terminal time by $\max|u(x,1) - u_{M,N}(x,1)|$, we observe in Table 6.1 that accuracy generally improves as we increase the number of collocation points. We note that for our particular implementation, we obtain significantly lesser error values than those presented in prior research [76], with far fewer points in the discretization; even with 10^4 spatial steps, and 160 steps in time, the compact finite difference method is unable to beat Jacobi spectral collocation with as little as $N = M = 4$. It should be noted, however, that the compact finite difference is capable of far larger grid sizes, with 2×10^5 steps in time eventually beating the performance of our implementation.

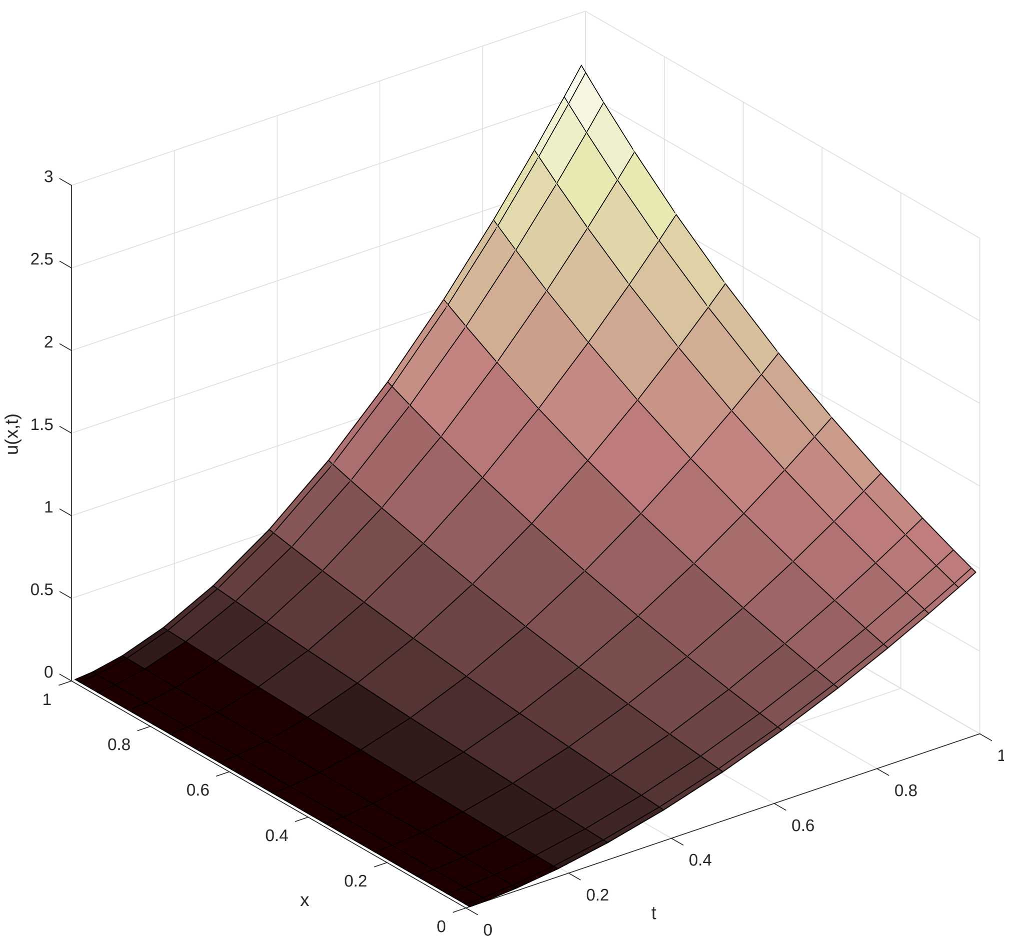


Figure 6.2. Approximate solution for $u(x,t)$, $N = M = 12$

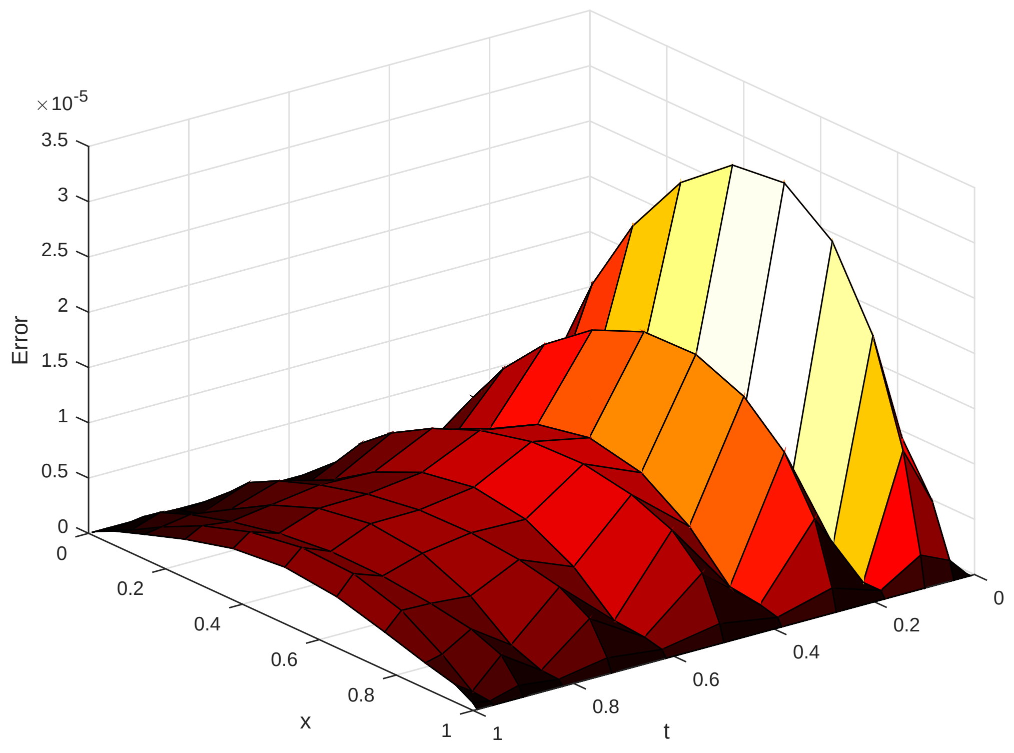


Figure 6.3. Absolute Error, $N = M = 12$

Table 6.1. Maximum Error for $N = M = 6, \ldots, 14$

N	Maximum Error
6	$0.502057417623991 \times 10^{-4}$
7	$0.188216226004734 \times 10^{-4}$
8	$0.176507481599586 \times 10^{-4}$
9	$0.112064583490668 \times 10^{-4}$
10	$0.083106997454951 \times 10^{-4}$
11	$0.076117530762865 \times 10^{-4}$
12	$0.048655461668545 \times 10^{-4}$
13	$0.041058070416877 \times 10^{-4}$
14	$0.043124542108419 \times 10^{-4}$

6.3 Non-linear Reaction-Diffusion Equations

For this exercise, we will consider equations of the form

$$\frac{\partial u(x,t)}{\partial t} = \frac{\partial^{1-\nu}}{\partial t^{1-\nu}}\left[A\frac{\partial^2 u(x,t)}{\partial x^2} - Bu(x,t)\right] + q\Big(u(x,t),x,t\Big), \tag{6.36}$$

where

$$0 \le x \le R, \qquad 0 < t \le T,$$

with A and B being constants, ν being a constant in the region $(0,1]$, and $q\Big(u(x,t),x,t\Big)$ being a non-linear source function. We have initial and boundary conditions

$$\begin{aligned} u(x,0) &= f(x), && 0 \le x \le R, \\ u(0,t) &= g_0(t), && 0 < t \le T, \\ u(R,t) &= g_R(t), && 0 < t \le T. \end{aligned} \tag{6.37}$$

First, we recall (6.27), and observe that the derivatives of (6.36) can be represented by

$$\begin{aligned} \frac{\partial u(x,t)}{\partial t} &\simeq \left[\mathcal{D}_T^1 \mathbf{P}_T(t)\right]' \mathbf{U}\,\mathbf{P}_R(x), \\ \frac{\partial^{1-\nu} u(x,t)}{\partial t^{1-\nu}} &\simeq \left[\mathcal{D}_T^{1-\nu} \mathbf{P}_T(t)\right]' \mathbf{U}\,\mathbf{P}_R(x), \\ \frac{\partial^{1-\nu}}{\partial t^{1-\nu}}\left[\frac{\partial^2 u(x,t)}{\partial x^2}\right] &\simeq \left[\mathcal{D}_T^{1-\nu} \mathbf{P}_T(t)\right]' \mathbf{U}\,\mathcal{D}_R^2 \mathbf{P}_R(x). \end{aligned} \tag{6.38}$$

Now, by substituting (6.26) and (6.38) into (6.36), we obtain

$$\begin{aligned} &[\mathbf{P}_T(t_{T,j})]' \Big(\left[\mathcal{D}_T^1\right]' \mathbf{U} - A\left[\mathcal{D}_T^{1-\nu}\right]' \mathbf{U}\,\mathcal{D}_R^2 + B\left[\mathcal{D}_T^{1-\nu}\right]' \mathbf{U}\Big)\,\mathbf{P}_R(x_{R,i}) \\ &- q\left([\mathbf{P}_T(t_{T,j})]' \mathbf{U}\,\mathbf{P}_R(x_{R,i}), x_{R,i}, t_{T,j}\right) = 0, \end{aligned} \tag{6.39}$$

for $i = 1, 2, ..., N-1$, and $j = 1, 2, ..., M$. For the initial and boundary conditions, we have

$$\begin{aligned} [\mathbf{P}_T(0)]'\,\mathbf{U}\,\mathbf{P}_R(x_{R,i}) - f(x_{R,i}) &= 0, & 0 \le i \le N, \\ [\mathbf{P}_T(t_{T,j})]'\,\mathbf{U}\,\mathbf{P}_R(0) - g_0(t_{T,j}) &= 0, & 0 < j \le M, \\ [\mathbf{P}_T(t_{T,j})]'\,\mathbf{U}\,\mathbf{P}_R(R) - g_R(t_{T,j}) &= 0, & 0 < j \le M. \end{aligned} \tag{6.40}$$

Combining (6.39) and (6.40), we obtain a system of $(M+1) \times (N+1)$ non-linear equations. We use the Levenberg-Marquardt algorithm, taking **U** as its variable, to minimise (6.39) and (6.40). This **U** is then used in (6.26) to calculate our approximation of $u(x, t)$.

6.3.1 A Numerical Example

For this exercise, we will consider the following equation [53]:

$$\frac{\partial u(x,y,t)}{\partial t} = \frac{\partial^{1-\nu}}{\partial t^{1-\nu}}\left[\frac{\partial^2 u(x,t)}{\partial x^2} - u(x,t)\right] + q\Big(u(x,t), x, t\Big), \tag{6.41}$$

where

$$0 \le x \le 1, \qquad 0 < t \le 1,$$

with ν being a constant in the region $(0, 1]$, and

$$q\Big(u(x,t), x, t\Big) = u^3(x,t) + \cos(\pi x)\left[2t + (\pi^2+1)\left(\frac{2t^{1+\nu}}{\Gamma(2+\nu)}\right) - t^6\cos^2(\pi x)\right] \tag{6.42}$$

being the non-linear source function. We have initial and boundary conditions

$$\begin{aligned} u(x,0) &= 0, & 0 \le x \le 1, \\ u(0,t) &= t^2, & 0 < t \le 1, \\ u(1,t) &= -t^2, & 0 < t \le 1. \end{aligned} \tag{6.43}$$

This non-linear time-fractional reaction sub-diffusion equation has exact solution

$$u(x,t) = t^2 \cos(\pi x). \tag{6.44}$$

With fractional time derivative degree $\nu = 0.35$, and $N = M = 10$ collocation points, implemented in MATLAB as shown in Appendix K, we see in Figure 6.4 that the approximation matches the exact solution in shape, with the detailed approximation surface in Figure 6.5.

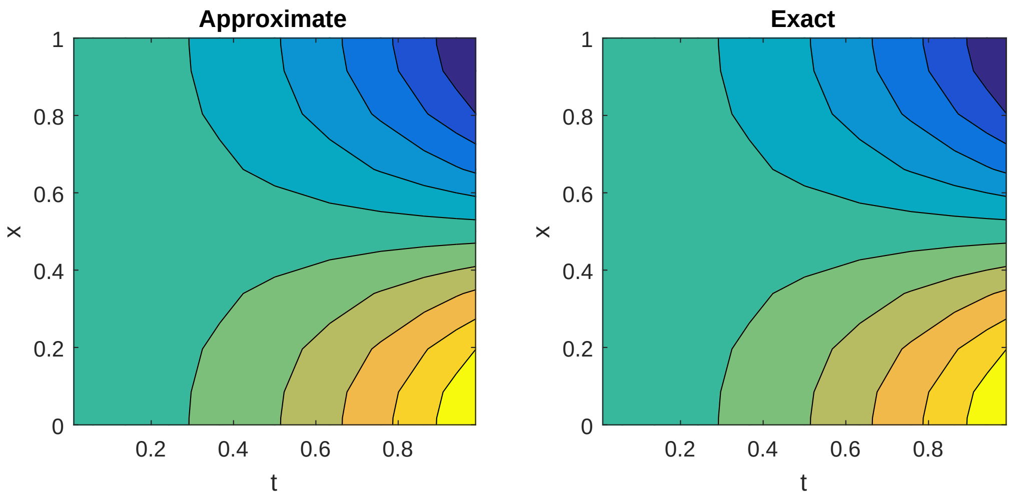


Figure 6.4. Contours of Approximate and Exact Solution, $N = 10$

With absolute error given by $|u(x,t) - u_{M,N}(x,t)|$, we observe the specific absolute error surface in Figure 6.6, where we note that the error is vaguely symmetrical.

Defining maximum error as

$$\max_{0\le x\le 1, 0<t\le 1} |u(x,t) - u_{M,N}(x,t)|,$$

we observe in Table 6.2 that accuracy generally improves as we increase the number of collocation points. We note that for our particular implementation, we obtain lesser error values than those presented in prior research [53], for values of

$N = M \leq 6$, with overall error values being considerably small for all grid sizes investigated.

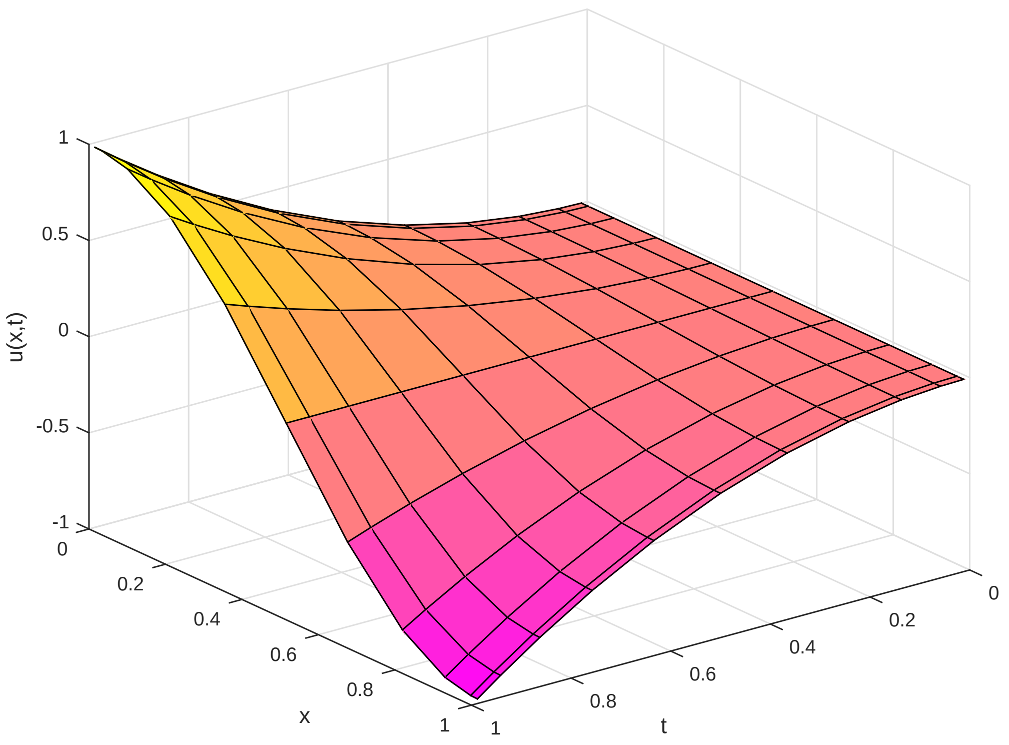


Figure 6.5. Approximate solution for $u(x,t)$, $N = M = 10$

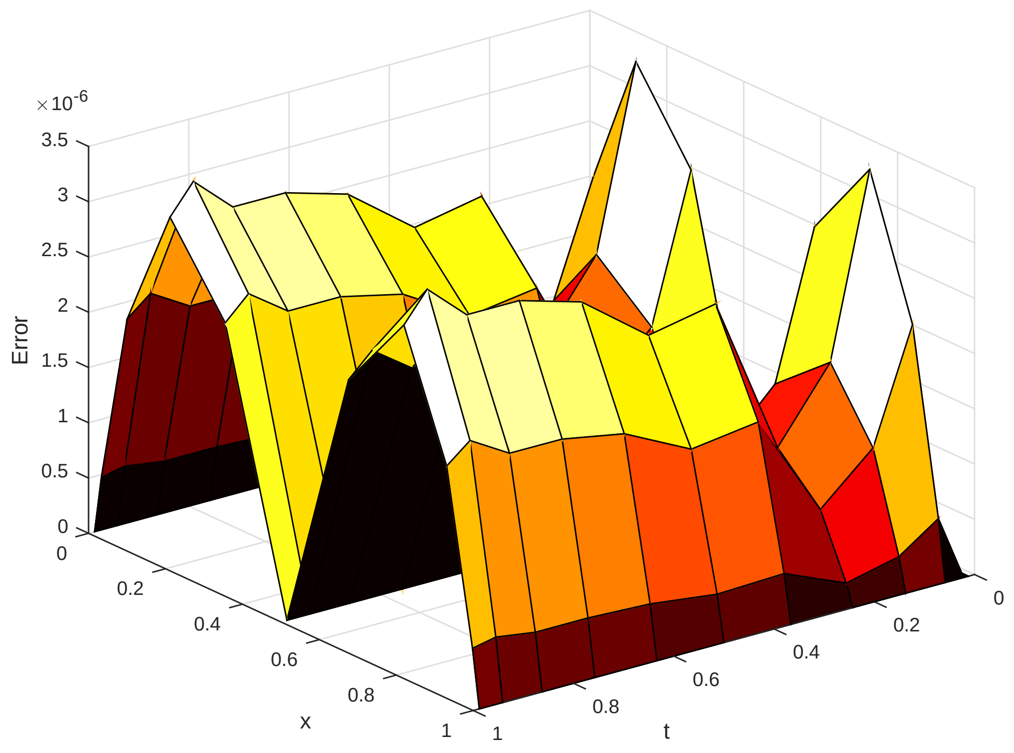


Figure 6.6. Absolute Error, $N = M = 10$

Table 6.2. Maximum Error for $N = M = 3, ..., 10$

N	Maximum Error
3	0.004631605140797
4	0.007998226534767
5	0.000313778354766
6	0.000058613008892
7	0.000010671829670
8	0.000006659873219
9	0.000005067846941
10	0.000003542928553

6.4 Hyperbolic Equations with Derivative Boundary Conditions

In this section, we will consider equations of the form

$$\frac{\partial^2 u(x,t)}{\partial t^2} = a(x,t)\frac{\partial^\nu u(x,t)}{\partial x^\nu} + q(x,t), \tag{6.45}$$

where

$$0 \le x \le R, \qquad 0 < t \le T,$$

with ν being the fractional differentiation constant in the region $(1, 2]$, and $q(x,t)$ being a source function. We have initial and boundary conditions

$$\begin{aligned} u(x,0) &= f(x), & 0 \le x \le R,\\ u_t(x,0) &= \hat{f}(x), & 0 \le x \le R,\\ u(0,t) &= g_0(t), & 0 < t \le T,\\ u(R,t) &= g_R(t), & 0 < t \le T. \end{aligned} \tag{6.46}$$

We observe that we have a derivative boundary condition, and will thus have to extend the current methodology slightly to accomodate this condition. Recalling (6.27), and observing the derivatives of (6.45) and (6.46), we note that

$$\begin{aligned} \frac{\partial u(x,t)}{\partial t} &\simeq \left[\mathcal{D}_T^1 \mathbf{P}_T(t)\right]' \mathbf{U}\,\mathbf{P}_R(x),\\ \frac{\partial^2 u(x,t)}{\partial t^2} &\simeq \left[\mathcal{D}_T^2 \mathbf{P}_T(t)\right]' \mathbf{U}\,\mathbf{P}_R(x),\\ \frac{\partial^2 u(x,t)}{\partial x^\nu} &\simeq \left[\mathbf{P}_T(t)\right]' \mathbf{U}\,\mathcal{D}_R^\nu \mathbf{P}_R(x). \end{aligned} \tag{6.47}$$

Now, by substituting (6.26) and (6.47) into (6.45), we obtain

$$\left[\mathbf{P}_T(t_{T,j})\right]' \left(\left[\mathcal{D}_T^2\right]' \mathbf{U} - a(x_{R,i}, t_{T,j})\mathbf{U}\,\mathcal{D}_R^\nu\right) \mathbf{P}_R(x_{R,i}) - q\left(x_{R,i}, t_{T,j}\right) = 0, \tag{6.48}$$

for $i = 1, 2, ..., N-1$, and $j = 1, 2, ..., M$. For the initial and boundary conditions, we have

$$\begin{aligned}
[\mathbf{P}_T(0)]' \,\mathbf{U}\,\mathbf{P}_R(x_{R,i}) - f(x_{R,i}) &= 0, && 0 \leq i \leq N, \\
\left[\mathcal{D}_T^1 \mathbf{P}_T(0)\right]' \,\mathbf{U}\,\mathbf{P}_R(x_{R,i}) - \hat{f}(x_{R,i}) &= 0, && 0 \leq i \leq N, \\
[\mathbf{P}_T(t_{T,j})]' \,\mathbf{U}\,\mathbf{P}_R(0) - g_0(t_{T,j}) &= 0, && 0 < j \leq M, \\
[\mathbf{P}_T(t_{T,j})]' \,\mathbf{U}\,\mathbf{P}_R(R) - g_R(t_{T,j}) &= 0, && 0 < j \leq M,
\end{aligned} \tag{6.49}$$

where we note the inclusion of the first time derivative as a second initial condition.

Combining (6.48) and (6.49), we obtain a system with the familiar $(M+1) \times (N+1)$ equations, plus an additional $N+1$ equations to accomodate the derivative boundary condition, giving a total of $(M+2) \times (N+1)$ equations. We again take advantage of the Levenberg-Marquardt algorithm, taking $\mathbf{U}$ as its variable, to minimise (6.48) and (6.49). This $\mathbf{U}$ is then used in (6.26) to calculate our approximation of $u(x, t)$.

6.4.1 A Numerical Example

We consider here a space-fractional hyperbolic PDE with a source term and derivative boundary conditions [59]:

$$\frac{\partial^2 u(x,t)}{\partial t^2} = \frac{\sqrt{x}}{\Gamma(0.5)} \frac{\partial^\nu u(x,t)}{\partial x^\nu} + 4x^2 + 2x^3 - 2.546x^2t^2 + 2.546xt^2, \tag{6.50}$$

where

$$0 \leq x \leq 1, \qquad 0 < t \leq 0.4,$$

and $\nu = 1.5$. We have initial and boundary conditions

$$\begin{aligned} u(x,0) &= 0, && 0 \le x \le 1, \\ u_t(x,0) &= 0, && 0 \le x \le 1, \\ u(0,t) &= 0, && 0 < t \le 0.4, \\ u(1,t) &= -t^2, && 0 < t \le 0.4. \end{aligned} \tag{6.51}$$

This space-fractional hyperbolic PDE has exact solution

$$u(x,t) = x^2(x-2)t^2. \tag{6.52}$$

With $N = M = 16$ collocation points, implemented in MATLAB as shown in Appendix M, we see in Figure 6.7 that the approximation matches the exact solution in shape, with the detailed approximation surface in Figure 6.8.

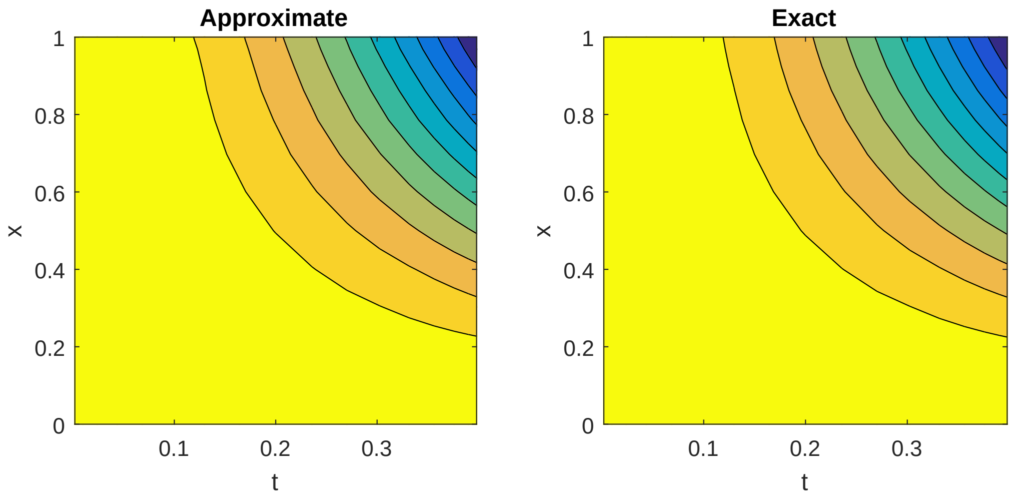


Figure 6.7. Contours of Approximate and Exact Solution, $N = M = 16$

With absolute error defined as $|u(x,t) - u_{M,N}(x,t)|$, observe the specific absolute error surface in Figure 6.9, where we note that the error tends to increase with both time and space.

We define maximum error as

$$\max_{0\leq x\leq 1, 0<t\leq 1} |u(x,t) - u_{M,N}(x,t)|,$$

and considering Table 6.3, we observe that accuracy generally improves as we increase the number of collocation points. We note that for our particular implementation, we achieve expectedly less accuracy than that of an analytical approximation [59], but performance is good nonetheless, and so it is evident that derivative boundary conditions are easily implemented with reasonable accuracy.

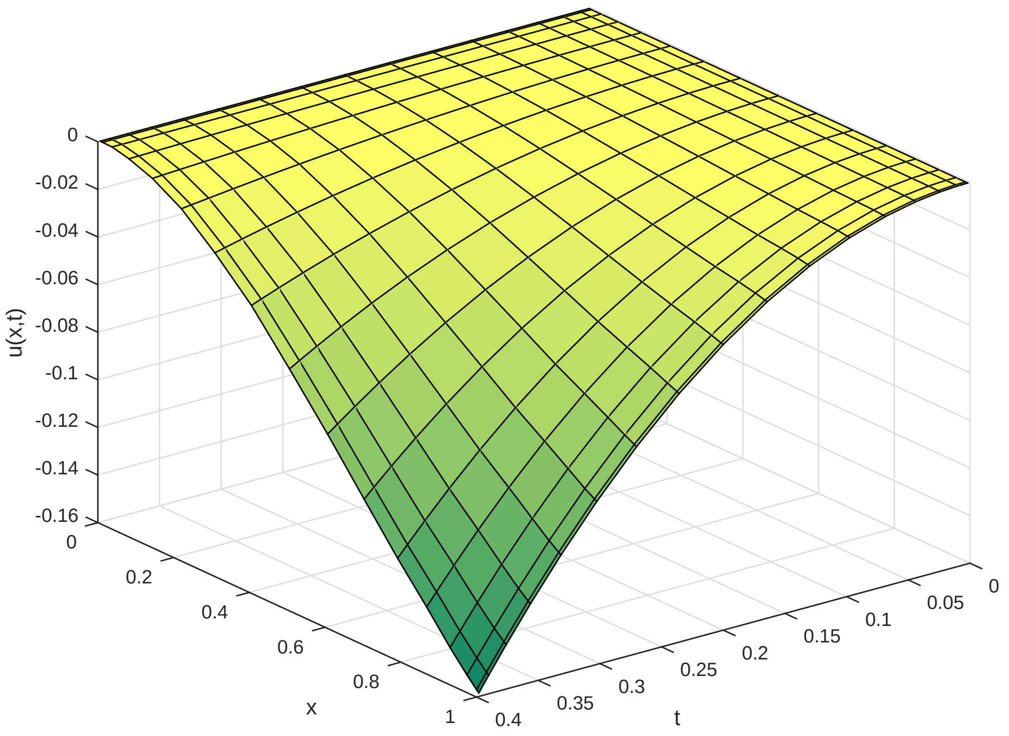


Figure 6.8. Approximate solution for $u(x,t)$, $N = M = 16$

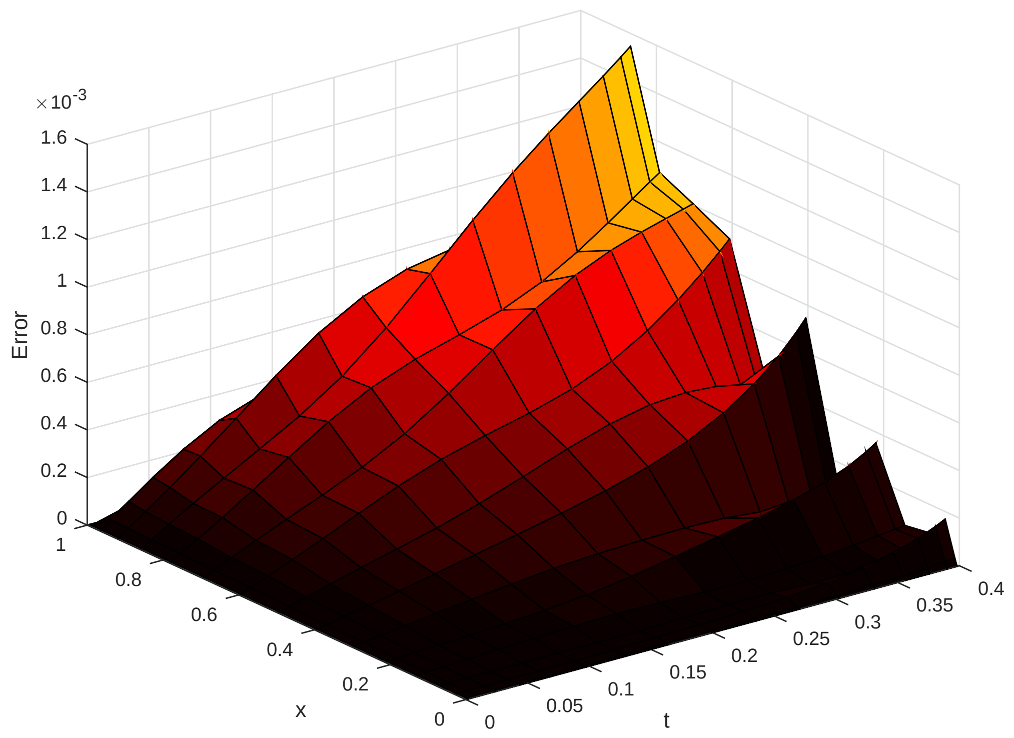


Figure 6.9. Absolute Error, $N = M = 16$

Table 6.3. Maximum Error for $N = M = 8, \ldots, 16$

N	Maximum Error
8	0.004065125666787
9	0.003119940619825
10	0.002949853822576
11	0.002687998301935
12	0.002194548679129
13	0.001828041330492
14	0.007206587528312
15	0.001570167608101
16	0.001553389994894

6.5 Concluding Remarks

Using the Caputo definition of the fractional derivative, and the technique of Jacobi-Gauss-Lobatto collocation, we were able to obtain an operational matrix representation of partial differential equations with one spatial dimension, and their various derivatives, in both time and space, and of both integer and fractional order, providing a system of possibly non-linear equations. This system was then solved using the Levenberg-Marquardt algorithm.

This approach was used to solve a non-homogeneous sub-diffusion equation, where the method here was able to achieve greater accuracy than prior methods, especially for the smaller grid sizes. It was observed, however, that increasing the grid size beyond a certain point stopped yielding benefits, which is not necessarily true of other methods, that may still slowly improve upon their accuracy with far larger grid sizes.

Then, the method was used to numerically solve a non-linear time-fractional diffusion equation, where it was found that this particular implementation was again able to provide more accurate results at smaller grid sizes than those in prior literature.

Lastly, the method was used to solve a space-fractional hyperbolic equation with derivative boundary conditions. It was found that the method was easily adapted to incorporate the derivative boundary condition, and provided reasonably accurate results. While no other numerical method was available for comparison, the method was expectedly less accurate than an analytical approximation.

Chapter 7

Two-dimensional Space- and Time-Fractional Equations

In this chapter, we extend the one dimensional shifted Jacobi polynomial operation matrix method to solve problems with two dimensions in space, in addition to the time dimension, in the fashion developed by Bhrawy [53], where he considered time-fractional diffusion equations of order less than 1. We will consider a variety of partial differential equations for this chapter, investigating fractional derivatives in both time and space, of orders less than or equal to 2, and may additionally have derivative boundary conditions, or be in coupled systems. We maintain use of the Caputo fractional derivative, with a lower boundary of 0, giving us the following expression for the fractional time derivative:

$$\frac{\partial^{\nu} u(x,y,t)}{\partial t^{\nu}} = {}_0^C\mathbf{D}_t^{\nu} u(x,y,t) = \frac{1}{\Gamma(\nu-n)} \int_0^t \frac{u^{(n)}(x,y,\tau)d\tau}{(x,y,t-\tau)^{\nu+1-n}}, \qquad (n-1 \le \nu < n), \tag{7.1}$$

with corresponding expressions available for the fractional space derivatives.

The solution will again involve shifted Jacobi polynomials, integrated under a Gauss-Lobatto quadrature, where we accommodate the increased dimensionality. This spectral scheme will provide us with a system of equations, which we again solve with a Levenberg-Marquardt algorithm.

In this chapter, we will consider time-fractional partial differential equations that take the following form:

$$f\left(x, y, t, u, \frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}, \frac{\partial u}{\partial t}, \frac{\partial^2 u}{\partial x^2}, \frac{\partial^2 u}{\partial y^2}, \frac{\partial^2 u}{\partial t^2}, \frac{\partial^2 u}{\partial x\,\partial y}, ..., \frac{\partial^\nu u}{\partial t^\nu}, ...\right) = 0 \tag{7.2}$$

7.1 Two-dimensional Shifted Jacobi Collocation Matrices

Now, we begin this section where § 6.1 left off, only we now consider $u(x, y, t)$ with two spatial dimensions, expressed as a triple Jacobi polynomial:

$$u(x, y, t) = \sum_{n_1=0}^{\infty} \sum_{n_2=0}^{\infty} \sum_{m=0}^{\infty} \hat{u}_{m,n_1,n_2} P^{\alpha,\beta}_{R_1,n_1}(x) P^{\alpha,\beta}_{R_2,n_2}(y) P^{\alpha,\beta}_{T,m}(t), \tag{7.3}$$

which, after truncating in all dimensions, becomes

$$\begin{aligned} &u_{M,N_1,N_2}(x_{R_1,i_1}, y_{R_2,i_2}, t_{T,j}) \\ &= \sum_{n_1=0}^{N_1} \sum_{n_2=0}^{N_2} \sum_{m=0}^{M} \hat{u}_{m,n_1,n_2} P^{\alpha,\beta}_{R_1,n_1}(x_{R,i_1}) P^{\alpha,\beta}_{R_2,n_2}(y_{R,i_2}) P^{\alpha,\beta}_{T,m}(t_{T,j}), \end{aligned} \tag{7.4}$$

with coefficients

$$\begin{aligned} &\hat{u}_{m,n_1,n_2} \\ &= \frac{1}{h_{R_1,n_1}} \frac{1}{h_{R_2,n_2}} \frac{1}{h_{T,m}} \\ &\quad \times \int_0^{R_1} \int_0^{R_2} \int_0^{T} u(x, y, t) P^{\alpha,\beta}_{R_1,n_1}(x) w^{(\alpha,\beta)}_{R_1} P^{\alpha,\beta}_{R_2,n_2}(y) w^{(\alpha,\beta)}_{R_2} P^{\alpha,\beta}_{T,m}(t) w^{(\alpha,\beta)}_{T} dt\, dy\, dx. \end{aligned} \tag{7.5}$$

Then, defining matrix

$$\mathbf{U} = \begin{bmatrix} \hat{u}_{0,0,0} & \hat{u}_{0,0,1} & \dots & \hat{u}_{0,0,N_2} & \hat{u}_{0,1,0} & \hat{u}_{0,1,1} & \dots & \hat{u}_{0,N_1,N_2} \\ \hat{u}_{1,0,0} & \hat{u}_{0,0,1} & \dots & \hat{u}_{1,0,N_2} & \hat{u}_{1,1,0} & \hat{u}_{1,1,1} & \dots & \hat{u}_{1,N_1,N_2} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \hat{u}_{M,0,0} & \hat{u}_{M,0,1} & \dots & \hat{u}_{M,0,N_2} & \hat{u}_{M,1,0} & \hat{u}_{M,1,1} & \dots & \hat{u}_{M,N_1,N_2} \end{bmatrix}$$

we observe that [53]

$$u_{M,N_1,N_2}(x_{R_1,i_1}, y_{R_2,i_2}, t_{T,j}) = [\mathbf{P}_T(t_{T,j})]' \, \mathbf{U} \, \mathbf{P}_{R_1}(x_{R_1,i_1}) \otimes \mathbf{P}_{R_2}(y_{R_2,i_2}), \tag{7.6}$$

where $\otimes$ represents the Kronecker product, that is, for matrices $A_{n\times m}$ and $B_{k\times l}$, such that

$$A = \begin{bmatrix} a_{0,0} & a_{0,1} & \dots & a_{0,m} \\ a_{1,0} & a_{1,1} & \dots & a_{1,m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,0} & a_{n,1} & \dots & a_{n,m} \end{bmatrix} \qquad B = \begin{bmatrix} b_{0,0} & b_{0,1} & \dots & b_{0,l} \\ b_{1,0} & b_{1,1} & \dots & b_{1,l} \\ \vdots & \vdots & \ddots & \vdots \\ b_{k,0} & b_{k,1} & \dots & b_{k,l} \end{bmatrix}$$

we have

$$A \otimes B = \begin{bmatrix} a_{0,0}B & a_{0,1}B & \dots & a_{0,m}B \\ a_{1,0}B & a_{1,1}B & \dots & a_{1,m}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,0}B & a_{n,1}B & \dots & a_{n,m}B \end{bmatrix}$$

where $A \otimes B$ is an $(k \times n) \times (l \times m)$ matrix.

Now, if we apply Theorem 6.1 to (7.6), we observe that the derivatives of (7.2) can be represented by

$$\begin{aligned}
\frac{\partial u(x,y,t)}{\partial x} &\simeq [\mathbf{P}_T(t)]'\,\mathbf{U}\,\left[\mathcal{D}^1_{R_1}\mathbf{P}_{R_1}(x)\right]\otimes\mathbf{P}_{R_2}(y)\\
\frac{\partial u(x,y,t)}{\partial y} &\simeq [\mathbf{P}_T(t)]'\,\mathbf{U}\,\mathbf{P}_{R_1}(x)\otimes\left[\mathcal{D}^1_{R_2}\mathbf{P}_{R_2}(y)\right]\\
\frac{\partial u(x,y,t)}{\partial t} &\simeq \left[\mathcal{D}^1_T\mathbf{P}_T(t)\right]'\,\mathbf{U}\,\mathbf{P}_{R_1}(x)\otimes\mathbf{P}_{R_2}(y)\\
\frac{\partial^2 u(x,y,t)}{\partial x^2} &\simeq [\mathbf{P}_T(t)]'\,\mathbf{U}\,\left[\mathcal{D}^2_{R_1}\mathbf{P}_{R_1}(x)\right]\otimes\mathbf{P}_{R_2}(y)\\
\frac{\partial^2 u(x,y,t)}{\partial y^2} &\simeq [\mathbf{P}_T(t)]'\,\mathbf{U}\,\mathbf{P}_{R_1}(x)\otimes\left[\mathcal{D}^2_{R_2}\mathbf{P}_{R_2}(y)\right]\\
\frac{\partial^2 u(x,y,t)}{\partial t^2} &\simeq \left[\mathcal{D}^2_T\mathbf{P}_T(t)\right]'\,\mathbf{U}\,\mathbf{P}_{R_1}(x)\otimes\mathbf{P}_{R_2}(y)\\
\frac{\partial^2 u(x,y,t)}{\partial x\,\partial y} &\simeq [\mathbf{P}_T(t)]'\,\mathbf{U}\,\left[\mathcal{D}^1_{R_1}\mathbf{P}_{R_1}(x)\right]\otimes\left[\mathcal{D}^2_{R_2}\mathbf{P}_{R_2}(y)\right]\\
&\vdots\\
\frac{\partial^\nu u(x,t)}{\partial t^\nu} &\simeq [\mathcal{D}^\nu_T\mathbf{P}_T(t)]'\,\mathbf{U}\,\mathbf{P}_{R_1}(x)\otimes\mathbf{P}_{R_2}(y)\\
&\vdots
\end{aligned} \tag{7.7}$$

Now, by substituting (7.6) and (7.7) into (7.2) and its initial and boundary conditions, we obtain a system of non-linear equations. A Levenberg-Marquardt algorithm, taking **U** as its variable, is used to minimise these equations as a set of least squares problems. This **U** is then used in (7.6) to acquire our approximate surface of $u(x,t)$. We will test the accuracy of this method by deriving the schemes for a number of particular space- and time-fractional PDEs, and then implement these schemes on appropriate numerical examples. We note for this chapter that, conveniently, all numerical examples are compared to prior literature examples with consistent presentations of error. Namely, we define the absolute error at the end time T as $|u(x,y,T)-u_{M,N_1,N_2}(x,y,T)|$, and the maximum error as

$$\max_{0\le x\le 1, 0\le y\le 1}|u(x,y,T)-u_{M,N_1,N_2}(x,y,T)|.$$

7.2 Space-Fractional Sub-Diffusion Equations

This section will consider two-dimensional space-fractional sub-diffusion equations, with fractional derivatives in both the x and y dimensions, as was considered in Chapter 5. These equations will be of the form

$$\frac{\partial u(x,y,t)}{\partial t} = a(x,y)\frac{\partial^{\nu_1} u(x,y,t)}{\partial x^{\nu_1}} + b(x,y)\frac{\partial^{\nu_2} u(x,y,t)}{\partial y^{\nu_2}} + q(x,y,t), \tag{7.8}$$

where

$$0 \le x \le R_1, \qquad 0 \le y \le R_2, \qquad 0 < t \le T,$$

with fractional derivative orders $\nu_{1,2} \in (1,2]$, and with initial and boundary conditions

$$\begin{aligned}
u(x,y,0) &= f(x,y), & 0 \le x \le R_1, && 0 \le y \le R_2,\\
u(0,y,t) &= g_{1,0}(y,t), & 0 \le y \le R_2, && 0 < t \le T,\\
u(R_1,y,t) &= g_{1,R_1}(y,t), & 0 \le y \le R_2, && 0 < t \le T,\\
u(x,0,t) &= g_{2,0}(x,t), & 0 \le x \le R_1, && 0 < t \le T,\\
u(x,R_2,t) &= g_{2,R_2}(x,t), & 0 \le x \le R_1, && 0 < t \le T.
\end{aligned} \tag{7.9}$$

We begin by recalling (7.7), and observe that the derivatives of (7.8) can be represented by

$$\begin{aligned}
\frac{\partial u(x,t)}{\partial t} &\simeq \left[\mathcal{D}_T^1 \mathbf{P}_T(t)\right]' \mathbf{U}\, \mathbf{P}_{R_1}(x) \otimes \mathbf{P}_{R_2}(y)\\
\frac{\partial^{\nu_1} u(x,t)}{\partial x^{\nu_1}} &\simeq \left[\mathbf{P}_T(t)\right]' \mathbf{U} \left[\mathcal{D}_{R_1}^{\nu_1} \mathbf{P}_{R_1}(x)\right] \otimes \mathbf{P}_{R_2}(y),\\
\frac{\partial^{\nu_2} u(x,t)}{\partial y^{\nu_2}} &\simeq \left[\mathbf{P}_T(t)\right]' \mathbf{U}\, \mathbf{P}_{R_1}(x) \otimes \left[\mathcal{D}_{R_2}^{\nu_2} \mathbf{P}_{R_2}(y)\right].
\end{aligned} \tag{7.10}$$

Now, by substituting (7.6) and (7.10) into (7.8), we obtain

$$
\begin{aligned}
&\left[\mathcal{D}_T^1 \mathbf{P}_T(t)\right]' \mathbf{U}\, \mathbf{P}_{R_1}(x) \otimes \mathbf{P}_{R_2}(y) \\
&- a(x_{R_1,i_1}, y_{R_2,i_2}) \left[\mathbf{P}_T(t)\right]' \mathbf{U} \left[\mathcal{D}_{R_1}^{\nu_1} \mathbf{P}_{R_1}(x)\right] \otimes \mathbf{P}_{R_2}(y) \\
&- b(x_{R_1,i_1}, y_{R_2,i_2}) \left[\mathbf{P}_T(t)\right]' \mathbf{U}\, \mathbf{P}_{R_1}(x) \otimes \left[\mathcal{D}_{R_2}^{\nu_2} \mathbf{P}_{R_2}(y)\right] \\
&- q(x_{R_1,i_1}, y_{R_2,i_2}, t_{T,j}) \\
&= 0
\end{aligned}
\tag{7.11}
$$

for $i_1 = 1, 2, ..., N_1 - 1$, $i_2 = 1, 2, ..., N_2 - 1$, and $j = 1, 2, ..., M$. For the initial and boundary conditions, we have

$$
\begin{aligned}
\left[\mathbf{P}_T(0)\right]' \mathbf{U}\, \mathbf{P}_{R_1}(x_{R_1,i_1}) \otimes \mathbf{P}_{R_2}(y_{R_2,i_2}), &= f(x_{R_1,i_1}, y_{R_2,i_2}), \\
&0 \le i_1 \le N_1, \quad 0 \le i_2 \le N_2, \\
\left[\mathbf{P}_T(t_{T,j})\right]' \mathbf{U}\, \mathbf{P}_{R_1}(0) \otimes \mathbf{P}_{R_2}(y_{R_2,i_2}), &= g_{1,0}(y_{R_2,i_2}, t_{T,j}), \\
&0 \le i_2 \le N_2, \quad 0 < j \le M, \\
\left[\mathbf{P}_T(t_{T,j})\right]' \mathbf{U}\, \mathbf{P}_{R_1}(R_1) \otimes \mathbf{P}_{R_2}(y_{R_2,i_2}), &= g_{1,R_1}(y_{R_2,i_2}, t_{T,j}), \\
&0 \le i_2 \le N_2, \quad 0 < j \le M, \\
\left[\mathbf{P}_T(t_{T,j})\right]' \mathbf{U}\, \mathbf{P}_{R_1}(x_{R_1,i_1}) \otimes \mathbf{P}_{R_2}(0), &= g_{2,0}(x_{R_1,i_1}, t_{T,j}), \\
&0 \le i_1 \le N_1, \quad 0 < j \le M, \\
\left[\mathbf{P}_T(t_{T,j})\right]' \mathbf{U}\, \mathbf{P}_{R_1}(x_{R_1,i_1}) \otimes \mathbf{P}_{R_2}(R_2), &= g_{2,R_2}(x_{R_1,i_1}, t_{T,j}), \\
&0 \le i_1 \le N_1, \quad 0 < j \le M.
\end{aligned}
\tag{7.12}
$$

Combining (7.11) and (7.12), we obtain a system of $(M+1) \times \Big((N_1+1) \times (N_2+1)\Big)$ equations. We use the Levenberg-Marquardt algorithm, taking $\mathbf{U}$ as its variable, with an initial guess of all zeros, to minimise (7.11) and (7.12). This $\mathbf{U}$ is then used in (7.6) to calculate our approximation of $u(x, y, t)$.

7.2.1 A Numerical Example

We will consider the two-dimensional space-fractional sub-diffusion equation investigated by Meerschaert, Scheffler and Tadjeran [32]:

$$\frac{\partial u(x,y,t)}{\partial t} = \frac{\Gamma(2.2)x^{2.8}y}{6}\frac{\partial^{1.8}u(x,y,t)}{\partial x^{1.8}} + \frac{2xy^{2.6}}{\Gamma(4.6)}\frac{\partial^{1.6}u(x,y,t)}{\partial y^{1.6}} - (1+2xy)e^{-t}x^3y^{3.6}, \tag{7.13}$$

where

$$0 \le x \le 1, \qquad 0 \le y \le 1, \qquad 0 < t \le 1,$$

with initial and boundary conditions

$$\begin{aligned}
u(x,y,0) &= x^3y^{3.6}, & 0 \le x \le 1, && 0 \le y \le 1,\\
u(0,y,t) &= 0, & 0 \le y \le 1, && 0 < t \le 1,\\
u(1,y,t) &= e^{-t}y^{3.6}, & 0 \le y \le 1, && 0 < t \le 1,\\
u(x,0,t) &= 0, & 0 \le x \le 1, && 0 < t \le 1,\\
u(x,1,t) &= e^{-t}x^3, & 0 \le x \le 1, && 0 < t \le 1.
\end{aligned} \tag{7.14}$$

This equation has exact solution

$$u(x,y,t) = e^{-t}x^3y^{3.6}. \tag{7.15}$$

With $N_1 = N_2 = M = 10$ collocation points, implemented in MATLAB as shown in Appendix O, we see in Figure 7.1 that at time $T = 1$, the approximation matches the exact solution in shape, with the detailed approximation surface in Figure 7.2.

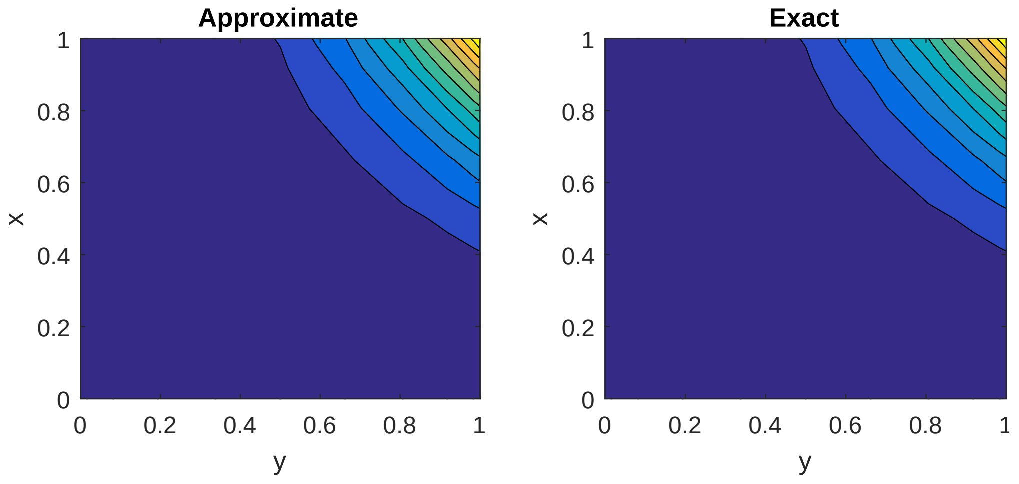


Figure 7.1. Contours of Approximate and Exact Solution, $N_1 = N_2 = M = 10$

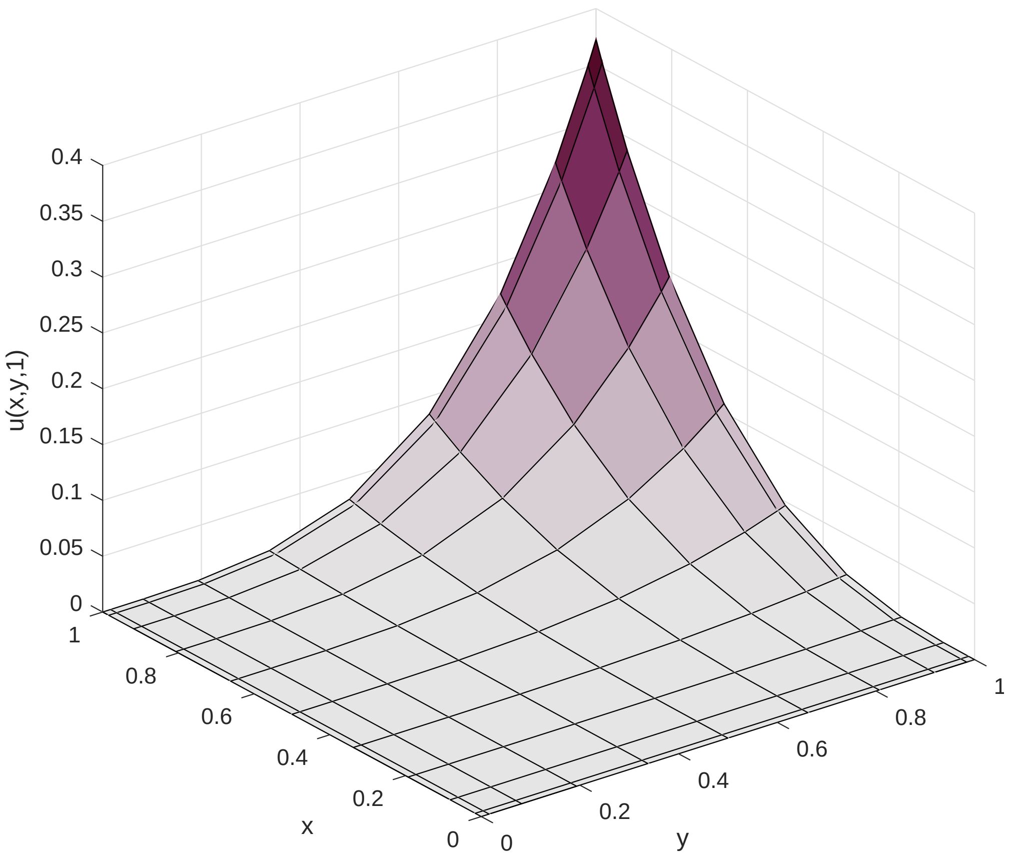


Figure 7.2. Approximate solution for $u(x,t)$, $N_1 = N_2 = M = 10$

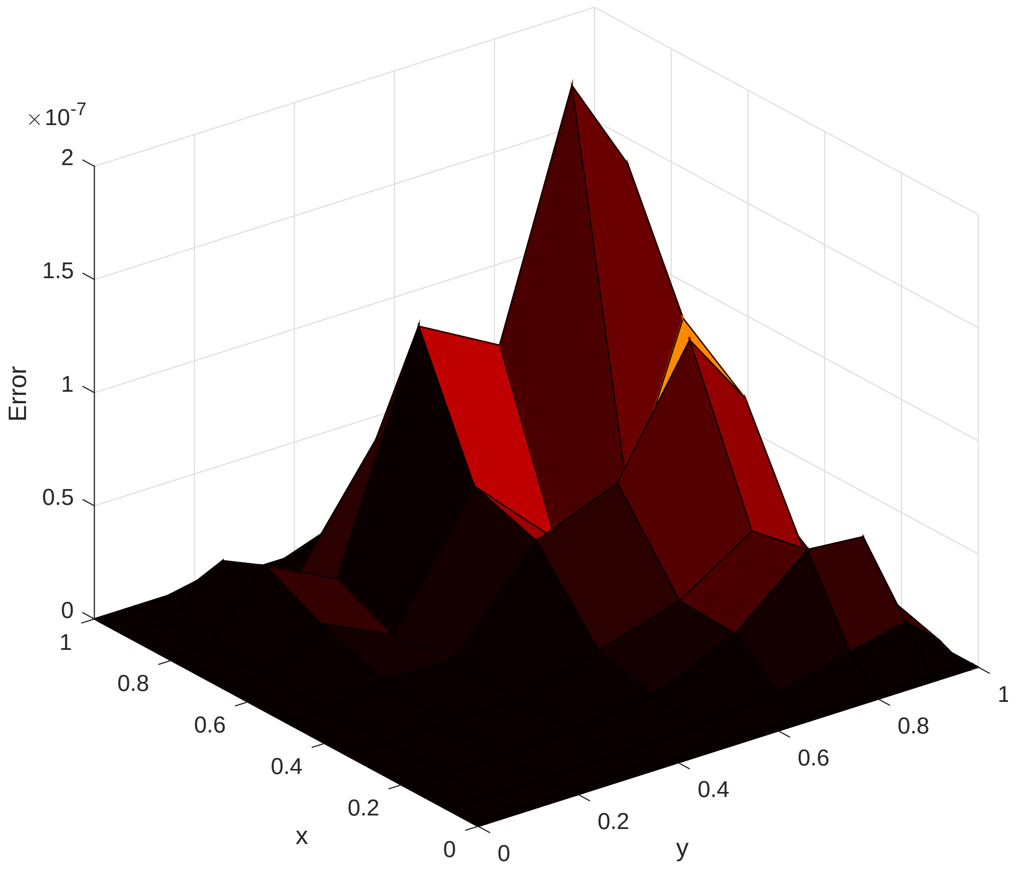


Figure 7.3. Absolute Error, $N_1 = N_2 = M = 10$

Observing the absolute error in Figure 7.3, we see that it is concentrated in the upper boundaries of x and y, however, as evident in Table 7.1, the accuracy is significant for the greater sample sizes, with error of a few orders of magnitude less than that of the finite difference method [32], and even slightly better accuracy than the shifted Legendre collocation method of Chapter 5 [52].

Table 7.1. Maximum Error for $N_1 = N_2 = M = 2, ..., 10$

N	Maximum Error
2	0.325449430043631
3	0.027003876786619
4	0.003053434501599
5	$1.346416675843468 \times 10^{-5}$
6	$2.466894411592979 \times 10^{-6}$
7	$1.947587520756411 \times 10^{-6}$
8	$7.910122618720594 \times 10^{-7}$
9	$2.863369100505886 \times 10^{-7}$
10	$1.966162568312058 \times 10^{-7}$

7.3 Time-Fractional Diffusion-Wave Equations

In this example, we will consider two-dimensional time-fractional wave equations, also referred to as diffusion-wave equations by virtue of the time derivative being inbetween a diffusion and a wave. These equations take the form

$$\frac{\partial^\nu u(x,y,t)}{\partial t^\nu} = \frac{\partial^2 u(x,y,t)}{\partial x^2} + \frac{\partial^2 u(x,y,t)}{\partial y^2} + q(x,y,t), \tag{7.16}$$

where

$$0 \le x \le R_1, \qquad 0 \le y \le R_2, \qquad 0 < t \le T,$$

with fractional derivative order $\nu \in (1,2]$, and with initial and boundary conditions

$$\begin{aligned}
u(x,y,0) &= f(x,y), && 0 \le x \le R_1, && 0 \le y \le R_2,\\
u(0,y,t) &= g_{1,0}(y,t), && 0 \le y \le R_2, && 0 < t \le T,\\
u(R_1,y,t) &= g_{1,R_1}(y,t), && 0 \le y \le R_2, && 0 < t \le T,\\
u(x,0,t) &= g_{2,0}(x,t), && 0 \le x \le R_1, && 0 < t \le T,\\
u(x,R_2,t) &= g_{2,R_2}(x,t), && 0 \le x \le R_1, && 0 < t \le T.
\end{aligned} \tag{7.17}$$

We begin by recalling (7.7), and observe that the derivatives of (7.16) can be represented by

$$\begin{aligned}
\frac{\partial^\nu u(x,t)}{\partial t^\nu} &\simeq [\mathcal{D}_T^\nu \mathbf{P}_T(t)]' \, \mathbf{U}\, \mathbf{P}_{R_1}(x) \otimes \mathbf{P}_{R_2}(y)\\
\frac{\partial^2 u(x,t)}{\partial x^2} &\simeq [\mathbf{P}_T(t)]' \, \mathbf{U} \left[\mathcal{D}_{R_1}^2 \mathbf{P}_2(x)\right] \otimes \mathbf{P}_{R_2}(y),\\
\frac{\partial^2 u(x,t)}{\partial y^2} &\simeq [\mathbf{P}_T(t)]' \, \mathbf{U}\, \mathbf{P}_{R_1}(x) \otimes \left[\mathcal{D}_{R_2}^2 \mathbf{P}_{R_2}(y)\right].
\end{aligned} \tag{7.18}$$

Now, by substituting (7.6) and (7.18) into (7.16), we obtain

$$\begin{aligned}
&[\mathcal{D}_T^\nu \mathbf{P}_T(t)]' \, \mathbf{U}\, \mathbf{P}_{R_1}(x) \otimes \mathbf{P}_{R_2}(y)\\
&- [\mathbf{P}_T(t)]' \, \mathbf{U} \left[\mathcal{D}_{R_1}^2 \mathbf{P}_{R_1}(x)\right] \otimes \mathbf{P}_{R_2}(y)\\
&- [\mathbf{P}_T(t)]' \, \mathbf{U}\, \mathbf{P}_{R_1}(x) \otimes \left[\mathcal{D}_{R_2}^2 \mathbf{P}_{R_2}(y)\right]\\
&- q(x_{R_1,i_1}, y_{R_2,i_2}, t_{T,j})\\
&= 0
\end{aligned} \tag{7.19}$$

for $i_1 = 1, 2, ..., N_1 - 1$, $i_2 = 1, 2, ..., N_2 - 1$, and $j = 1, 2, ..., M$. For the initial and boundary conditions, we have

$$
\begin{aligned}
[\mathbf{P}_T(0)]'\,\mathbf{U}\,\mathbf{P}_{R_1}(x_{R_1,i_1}) \otimes \mathbf{P}_{R_2}(y_{R_2,i_2}) &= f(x_{R_1,i_1}, y_{R_2,i_2}),\\
&\quad 0 \le i_1 \le N_1, \quad 0 \le i_2 \le N_2,\\
[\mathbf{P}_T(t_{T,j})]'\,\mathbf{U}\,\mathbf{P}_{R_1}(0) \otimes \mathbf{P}_{R_2}(y_{R_2,i_2}) &= g_{1,0}(y_{R_2,i_2}, t_{T,j}),\\
&\quad 0 \le i_2 \le N_2, \quad 0 < j \le M,\\
[\mathbf{P}_T(t_{T,j})]'\,\mathbf{U}\,\mathbf{P}_{R_1}(R_1) \otimes \mathbf{P}_{R_2}(y_{R_2,i_2}) &= g_{1,R_1}(y_{R_2,i_2}, t_{T,j}),\\
&\quad 0 \le i_2 \le N_2, \quad 0 < j \le M,\\
[\mathbf{P}_T(t_{T,j})]'\,\mathbf{U}\,\mathbf{P}_{R_1}(x_{R_1,i_1}) \otimes \mathbf{P}_{R_2}(0) &= g_{2,0}(x_{R_1,i_1}, t_{T,j}),\\
&\quad 0 \le i_1 \le N_1, \quad 0 < j \le M,\\
[\mathbf{P}_T(t_{T,j})]'\,\mathbf{U}\,\mathbf{P}_{R_1}(x_{R_1,i_1}) \otimes \mathbf{P}_{R_2}(R_2) &= g_{2,R_2}(x_{R_1,i_1}, t_{T,j}),\\
&\quad 0 \le i_1 \le N_1, \quad 0 < j \le M.
\end{aligned}
\tag{7.20}
$$

Combining (7.19) and (7.20), we obtain a system of $(M+1)\times\Big((N_1+1)\times(N_2+1)\Big)$ equations. We use the Levenberg-Marquardt algorithm, taking $\mathbf{U}$ as its variable, with an initial guess of all zeros, to minimise (7.19) and (7.20). This $\mathbf{U}$ is then used in (7.6) to calculate our approximation of $u(x, y, t)$.

7.3.1 A Numerical Example

We will implement our scheme on a two-dimensional time-fractional diffusion-wave equation investigated by Dehghan, Abbaszadeh and Mohebbi [77]:

$$
\frac{\partial^\nu u(x,y,t)}{\partial t^\nu} = \frac{\partial^2 u(x,y,t)}{\partial x^2} + \frac{\partial^2 u(x,y,t)}{\partial y^2} + \left(\frac{2t^{2-\nu}}{\Gamma(3-\nu)}\right)\sin(x+y), \tag{7.21}
$$

where

$$
0 \le x \le 1, \qquad 0 \le y \le 1, \qquad 0 < t \le 1,
$$

with fractional derivative order $\nu \in (1,2]$, and with initial and boundary conditions

$$\begin{aligned}
&u(x,y,0) = 0, && 0 \le x \le 1, && 0 \le y \le 1,\\
&u(0,y,t) = t^2 \sin(y), && 0 \le y \le 1, && 0 < t \le 1,\\
&u(1,y,t) = t^2 \sin(1+y), && 0 \le y \le 1, && 0 < t \le 1,\\
&u(x,0,t) = t^2 \sin(x), && 0 \le x \le 1, && 0 < t \le 1,\\
&u(x,1,t) = t^2 \sin(x+1), && 0 \le x \le 1, && 0 < t \le 1.
\end{aligned} \tag{7.22}$$

The exact solution to this equation is given by

$$u(x,y,t) = t^2 \sin(x+y). \tag{7.23}$$

With fractional order $\nu = 1.95$, and $M = 2$ and $N_1 = N_2 = 10$ collocation points, implemented in MATLAB as shown in Appendix Q, we see in Figure 7.4 that at time $T = 1$, the approximation matches the exact solution in shape, with the detailed approximation surface in Figure 7.5.

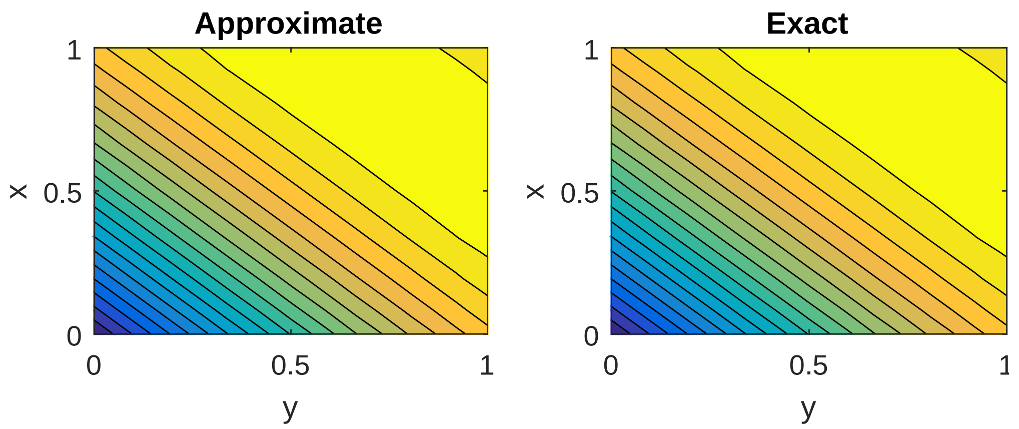


Figure 7.4. Contours of Approximate and Exact Solution, $M = 2$, $N_1 = N_2 = 10$

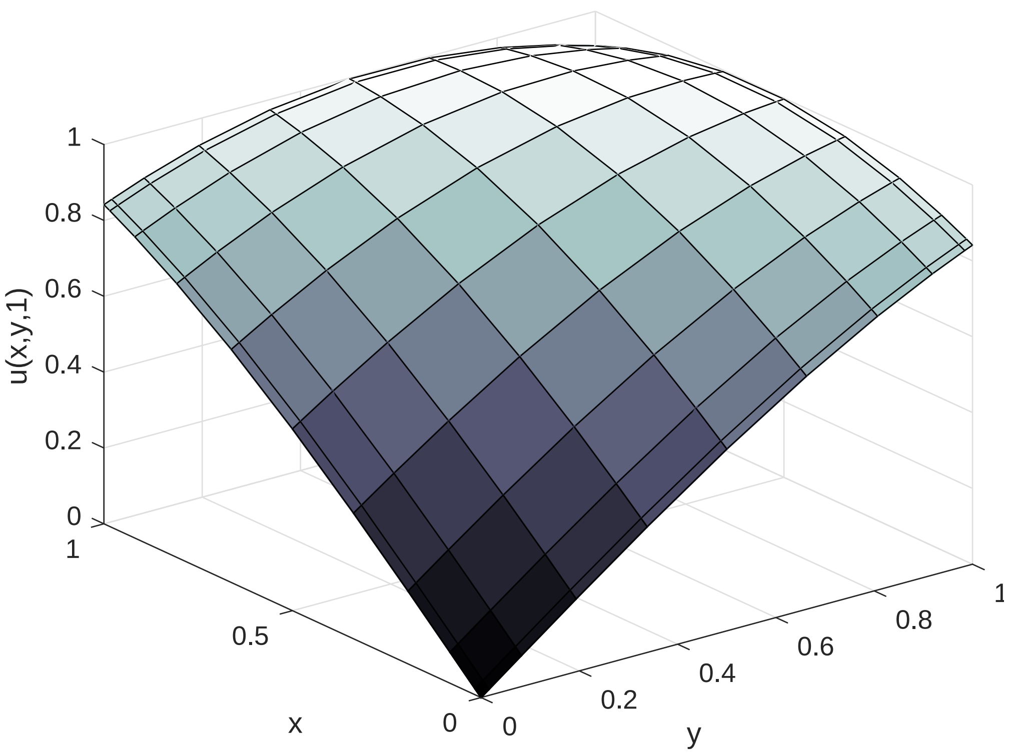


Figure 7.5. Approximate solution for $u(x,t)$, $M = 2$, $N_1 = N_2 = 10$

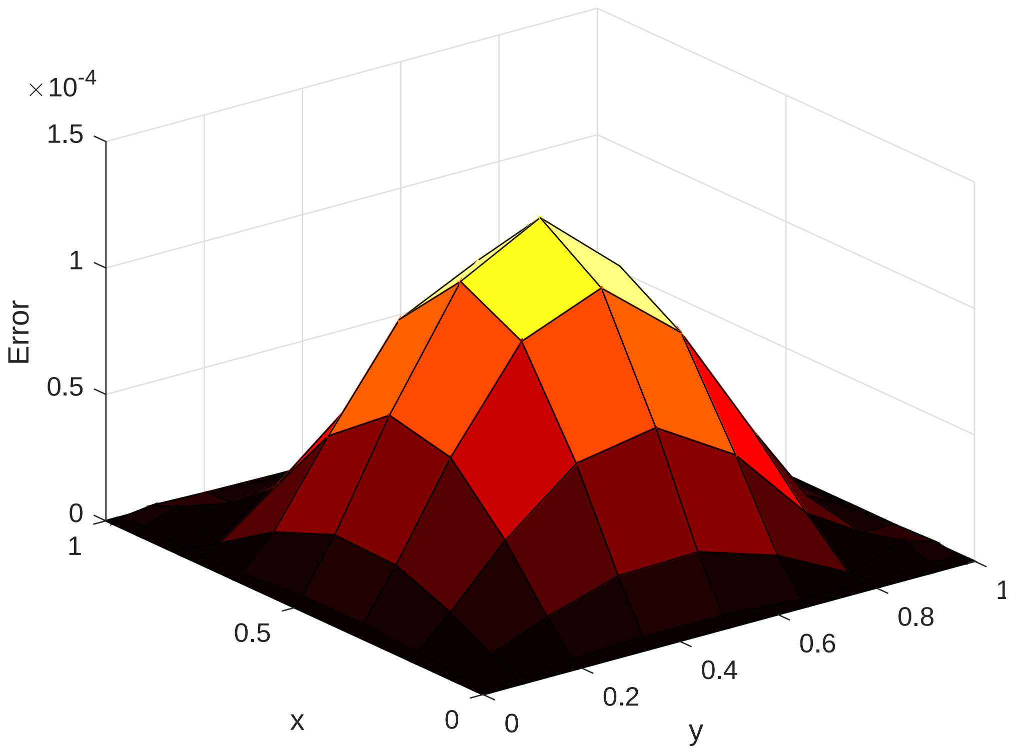


Figure 7.6. Absolute Error, $M = 2$, $N_1 = N_2 = 10$

We observe the absolute error in Figure 7.6, where we note that error is concentrated towards the spatial centre. Maximum error values are presented for $M = 2$ and $N_1 = N_2 = 2, ..., 10$, where we observe low error for all sample sizes, with the best performance occuring for $N_1 = N_2 = 5$ at just under 1×10^{-4}, and the worst performance occuring for $N_1 = N_2 = 2$ at just under 2×10^{-3}, which is still better than the best reported performance for the Finite Element method, even with as many as as 50 steps in space and 80 in time [77]. We note the value of M is only 2 for this implementation; while the method solves for higher values of M, the theoretical accuracy gained from truncating fewer terms is overwhelmed by the loss of accuracy due to rounding error in solving a large system of non-linear equations.

Table 7.2. Maximum Error for $M = 2$, $N_1 = N_2 = 2, ..., 10$

$N_1 = N_2$	Maximum Error
2	$0.163664640782224 \times 10^{-2}$
3	$0.912850678700838 \times 10^{-3}$
4	$0.113469534845367 \times 10^{-3}$
5	$0.096517630236770 \times 10^{-3}$
6	$0.127909247585545 \times 10^{-3}$
7	$0.113257309478088 \times 10^{-3}$
8	$0.127920013087590 \times 10^{-3}$
9	$0.120449697561753 \times 10^{-3}$
10	$0.127919285784039 \times 10^{-3}$

7.4 Klein-Gordon Equations With Derivative Initial Conditions

For this section, we will consider two-dimensional time-fractional non-linear Klein-Gordon equations, which take the form

$$\frac{\partial^\nu u(x,y,t)}{\partial t^\nu} = \frac{\partial^2 u(x,y,t)}{\partial x^2} + \frac{\partial^2 u(x,y,t)}{\partial y^2} + Q\Big(u(x,y,t)\Big) + q(x,y,t), \qquad (7.24)$$

where

$$0 \le x \le R_1, \qquad 0 \le y \le R_2, \qquad 0 < t \le T,$$

with fractional derivative order $\nu \in (1, 2]$, and with initial and boundary conditions

$$\begin{aligned}
u(x,y,0) &= f(x,y), && 0 \le x \le R_1, && 0 \le y \le R_2,\\
u_t(x,y,0) &= \hat{f}(x,y), && 0 \le x \le R_1, && 0 \le y \le R_2,\\
u(0,y,t) &= g_{1,0}(y,t), && 0 \le y \le R_2, && 0 < t \le T,\\
u(R_1,y,t) &= g_{1,R_1}(y,t), && 0 \le y \le R_2, && 0 < t \le T,\\
u(x,0,t) &= g_{2,0}(x,t), && 0 \le x \le R_1, && 0 < t \le T,\\
u(x,R_2,t) &= g_{2,R_2}(x,t), && 0 \le x \le R_1, && 0 < t \le T,
\end{aligned} \tag{7.25}$$

where we notice the inclusion of a condition on the derivative $u_t(x, y, 0)$, and so we will have to adapt the current method in a way similar to that of § 6.4.

To build our approximation scheme, we recall (7.7), observing that the derivatives of (7.24) and (7.31) can be represented by

$$\begin{aligned}
\frac{\partial u(x,y,t)}{\partial t} &\simeq \left[\mathcal{D}_T^1 \mathbf{P}_T(t)\right]' \mathbf{U}\, \mathbf{P}_{R_1}(x) \otimes \mathbf{P}_{R_2}(y)\\
\frac{\partial^\nu u(x,y,t)}{\partial t^\nu} &\simeq \left[\mathcal{D}_T^\nu \mathbf{P}_T(t)\right]' \mathbf{U}\, \mathbf{P}_{R_1}(x) \otimes \mathbf{P}_{R_2}(y)\\
\frac{\partial^2 u(x,y,t)}{\partial x^2} &\simeq \left[\mathbf{P}_T(t)\right]' \mathbf{U} \left[\mathcal{D}_{R_1}^2 \mathbf{P}_2(x)\right] \otimes \mathbf{P}_{R_2}(y),\\
\frac{\partial^2 u(x,y,t)}{\partial y^2} &\simeq \left[\mathbf{P}_T(t)\right]' \mathbf{U}\, \mathbf{P}_{R_1}(x) \otimes \left[\mathcal{D}_{R_2}^2 \mathbf{P}_{R_2}(y)\right].
\end{aligned} \tag{7.26}$$

Now, by substituting (7.6) and (7.26) into (7.24), we obtain

$$\begin{aligned}
&\left[\mathcal{D}_T^\nu \mathbf{P}_T(t_{T,j})\right]' \mathbf{U}\, \mathbf{P}_{R_1}(x_{R_1,i_1}) \otimes \mathbf{P}_{R_2}(y_{R_2,i_2})\\
&- \left[\mathbf{P}_T(t_{T,j})\right]' \mathbf{U} \left[\mathcal{D}_{R_1}^2 \mathbf{P}_{R_1}(x_{R_1,i_1})\right] \otimes \mathbf{P}_{R_2}(y_{R_2,i_2})\\
&- \left[\mathbf{P}_T(t_{T,j})\right]' \mathbf{U}\, \mathbf{P}_{R_1}(x_{R_1,i_1}) \otimes \left[\mathcal{D}_{R_2}^2 \mathbf{P}_{R_2}(y_{R_2,i_2})\right]\\
&- Q\Big(\left[\mathbf{P}_T(t_{T,j})\right]' \mathbf{U}\, \mathbf{P}_{R_1}(x_{R_1,i_1}) \otimes \mathbf{P}_{R_2}(y_{R_2,i_2})\Big)\\
&- q(x_{R_1,i_1}, y_{R_2,i_2}, t_{T,j})\\
&= 0
\end{aligned} \tag{7.27}$$

for $i_1 = 1, 2, ..., N_1 - 1$, $i_2 = 1, 2, ..., N_2 - 1$, and $j = 1, 2, ..., M$. For the initial and boundary conditions, we have

$$\begin{aligned}
[\mathbf{P}_T(0)]'\,\mathbf{U}\,\mathbf{P}_R(x_{R_1,i_1}) \otimes \mathbf{P}_{R_2}(y_{R_2,i_2}) &= f(x_{R_1,i_1}, y_{R_2,i_2}), \\
&0 \le i_1 \le N_1,\ 0 \le i_2 \le N_2, \\
\left[\mathcal{D}_T^1 \mathbf{P}_T(0)\right]'\,\mathbf{U}\,\mathbf{P}_{R_1}(x_{R_1,i_1}) \otimes \mathbf{P}_{R_2}(y_{R_2,i_2}) &= \hat{f}(x_{R_1,i_1}, y_{R_2,i_2}), \\
&0 \le i_1 \le N_1,\ 0 \le i_2 \le N_2, \\
[\mathbf{P}_T(t_{T,j})]'\,\mathbf{U}\,\mathbf{P}_{R_1}(0) \otimes \mathbf{P}_{R_2}(y_{R_2,i_2}) &= g_{1,0}(y_{R_2,i_2}, t_{T,j}), \\
&0 \le i_2 \le N_2,\ 0 < j \le M, \\
[\mathbf{P}_T(t_{T,j})]'\,\mathbf{U}\,\mathbf{P}_{R_1}(R_1) \otimes \mathbf{P}_{R_2}(y_{R_2,i_2}) &= g_{1,R_1}(y_{R_2,i_2}, t_{T,j}), \\
&0 \le i_2 \le N_2,\ 0 < j \le M, \\
[\mathbf{P}_T(t_{T,j})]'\,\mathbf{U}\,\mathbf{P}_{R_1}(x_{R_1,i_1}) \otimes \mathbf{P}_{R_2}(0) &= g_{2,0}(x_{R_1,i_1}, t_{T,j}), \\
&0 \le i_1 \le N_1,\ 0 < j \le M, \\
[\mathbf{P}_T(t_{T,j})]'\,\mathbf{U}\,\mathbf{P}_{R_1}(x_{R_1,i_1}) \otimes \mathbf{P}_{R_2}(R_2) &= g_{2,R_2}(x_{R_1,i_1}, t_{T,j}), \\
&0 \le i_1 \le N_1,\ 0 < j \le M.
\end{aligned} \tag{7.28}$$

Combining (7.30) and (7.28), we obtain the usual system of $(M+1) \times \Big((N_1+1) \times (N_2+1)\Big)$ equations, plus an additional $(N_1+1) \times (N_2+1)$ equations to represent the derivative condition, giving a total of $(M+2) \times \Big((N_1+1) \times (N_2+1)\Big)$ equations. We use the Levenberg-Marquardt algorithm, taking $\mathbf{U}$ as its variable, with an initial guess of all zeros, to minimise (7.30) and (7.28). This $\mathbf{U}$ is then used in (7.6) to calculate our approximation of $u(x, y, t)$.

7.4.1 A Numerical Example

In this example, we will consider a two-dimensional time-fractional non-linear Klein-Gordon equation with derivative initial condition investigated by Dehghan, Abbaszadeh and Mohebbi [78]:

$$\frac{\partial^\nu u(x,y,t)}{\partial t^\nu} = \frac{\partial^2 u(x,y,t)}{\partial x^2} + \frac{\partial^2 u(x,y,t)}{\partial y^2} + u^2(x,y,t) + u^3(x,y,t) + q(x,y,t), \quad (7.29)$$

where

$$0 \leq x \leq 1, \qquad 0 \leq y \leq 1, \qquad 0 < t \leq 2,$$

and the forcing function $q(x,y,t)$ is defined by

$$\begin{aligned} q(x,y,t) = {} & \frac{2t^{2-\nu}}{\Gamma(3-\nu)} \left(\text{sech}^2(x+y-r) + \text{sech}^2(x+y+r)\right) \\ & - 4t^2 \Big(3\text{sech}^2(x+y-r)\tanh^2(x+y-r) - \text{sech}^2(x+y-r) \\ & + 3\text{sech}^2(x+y+r)\tanh^2(x+y+r) - \text{sech}^2(x+y+r)\Big) \\ & - t^4 \left(\text{sech}^2(x+y-r) + \text{sech}^2(x+y+r)\right)^2 \\ & - t^6 \left(\text{sech}^2(x+y-r) + \text{sech}^2(x+y+r)\right)^3 . \end{aligned} \quad (7.30)$$

We have fractional derivative order $\nu \in (1,2]$, and with initial and boundary conditions

$$
\begin{aligned}
u(x, y, 0) &= 0, \\
&\qquad 0 \leq x \leq 1, \quad 0 \leq y \leq 1, \\
u_t(x, y, 0) &= 0, \\
&\qquad 0 \leq x \leq 1, \quad 0 \leq y \leq 1, \\
u(0, y, t) &= t^2 \left(\operatorname{sech}^2(y - r) + \operatorname{sech}^2(y + r)\right), \\
&\qquad 0 \leq y \leq 1, \quad 0 < t \leq 1, \\
u(1, y, t) &= t^2 \left(\operatorname{sech}^2(1 + y - r) + \operatorname{sech}^2(1 + y + r)\right), \\
&\qquad 0 \leq y \leq 1, \quad 0 < t \leq 1, \\
u(x, 0, t) &= t^2 \left(\operatorname{sech}^2(x - r) + \operatorname{sech}^2(x + r)\right), \\
&\qquad 0 \leq x \leq 1, \quad 0 < t \leq 1, \\
u(x, 1, t) &= t^2 \left(\operatorname{sech}^2(x + 1 - r) + \operatorname{sech}^2(x + 1 + r)\right), \\
&\qquad 0 \leq x \leq 1, \quad 0 < t \leq 1.
\end{aligned} \tag{7.31}
$$

The exact solution to this equation is given by

$$
u(x, y, t) = t^2 \left(\operatorname{sech}^2(x + y - r) + \operatorname{sech}^2(x + y + r)\right). \tag{7.32}
$$

With fractional derivative order $\nu = 1.95$, and $M = 16$, $N_1 = N_2 = 8$ collocation points, implemented in MATLAB as shown in Appendix S, we can see from Figure 7.7 that the approximation matches the exact solution in form, with the detailed view of the approximation at $t = 2$ given in Figure 7.8.

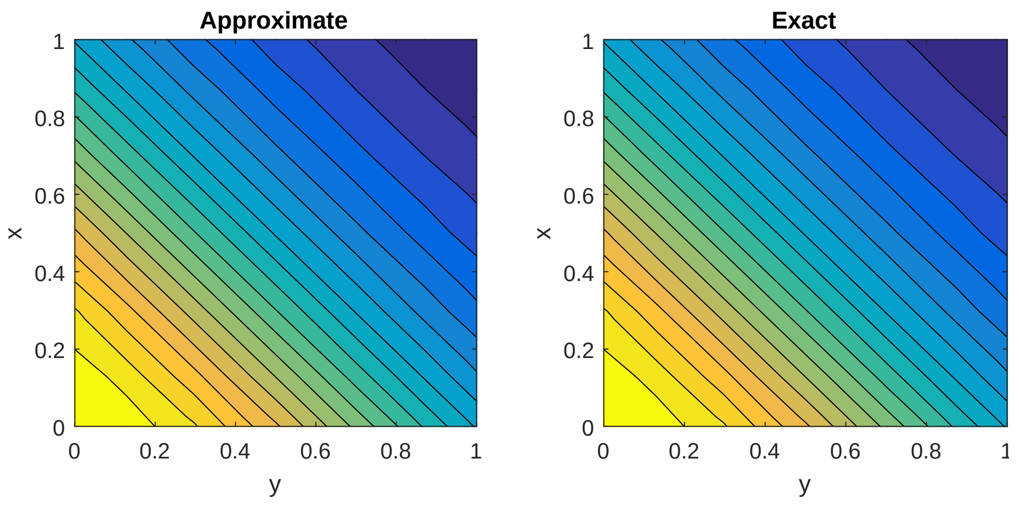


Figure 7.7. Contours of Approximate and Exact Solution, $M = 16$, $N_1 = N_2 = 8$

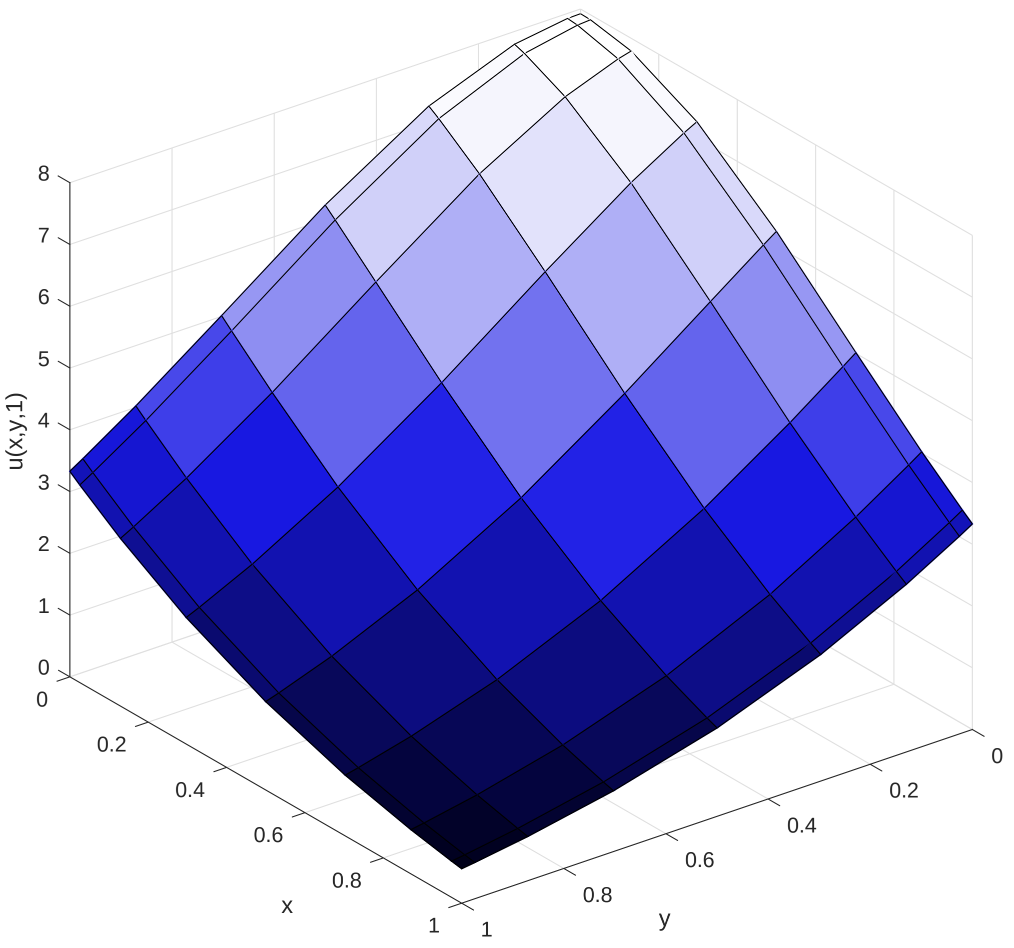


Figure 7.8. Approximate solution for $u(x, y, 2)$, $M = 16$, $N_1 = N_2 = 8$

We observe in Figure 7.9 that the error is concentrated towards the lower spatial boundary, and from Table 7.3, we see how the error compares for different numbers of collocation nodes. We note that for this problem, it is beneficial to have more nodes in time than in space for $t = 2$, and so we have twice as many nodes in time than in either spatial direction. Comparing the performance of this method to the meshless method used in prior literature [78], we see that even with as few as $M = 8$ and $N_1 = N_2 = 4$ collocation nodes, our method obtains better accuracy than their best reported result, involving 3200 steps in time. As we increase the number of nodes, our accuracy generally increases, eventually presenting over an order of magnitude less error than the best result in prior literature [78]. We do note the computational complexity of our method, however, since even relatively small numbers of collocation points quickly increase the number of equations to solve.

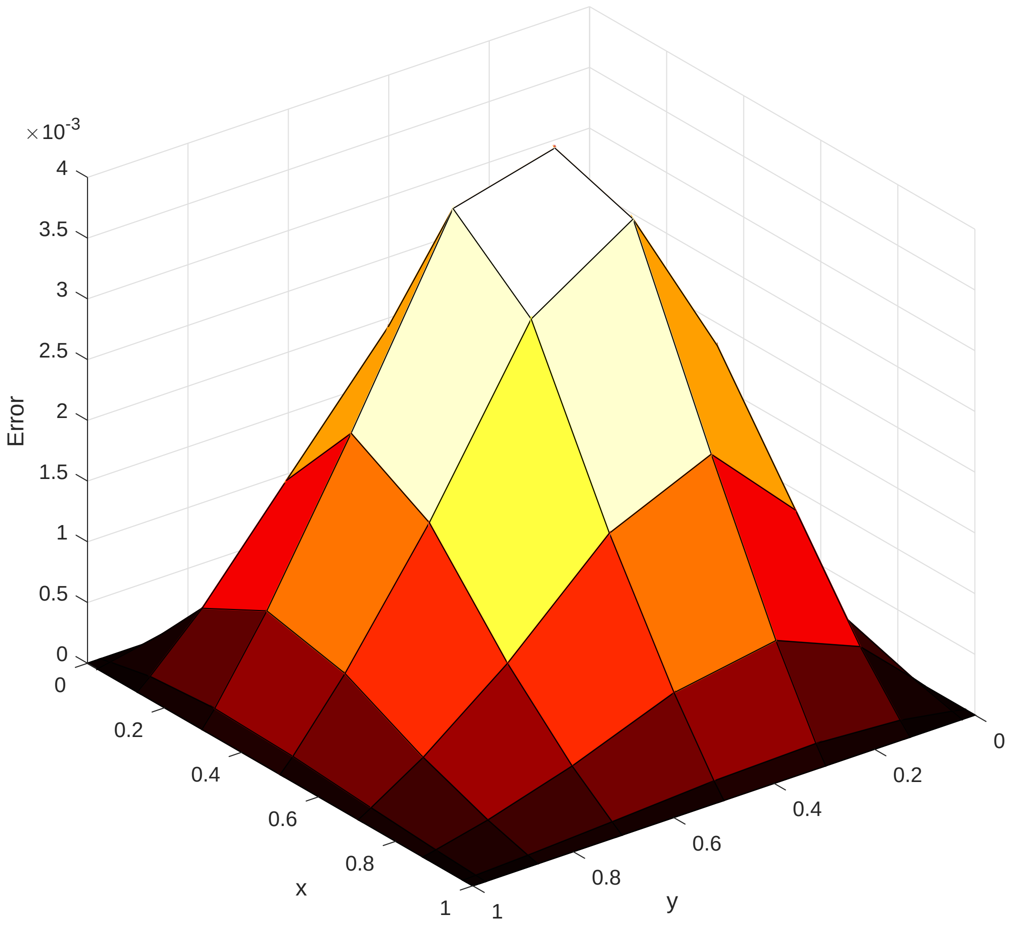


Figure 7.9. Absolute Error, $M = 16$, $N_1 = N_2 = 8$

Table 7.3. Maximum Error for $M = 2 \times N_1$, $N_1 = N_2 = 4, \ldots, 8$

M	**Maximum Error**
8	$0.272408801612141 \times 10^{-1}$
10	$0.184124513932122 \times 10^{-1}$
12	$0.215896560208835 \times 10^{-1}$
14	$0.503778538515487 \times 10^{-2}$
16	$0.379755559526007 \times 10^{-2}$

7.5 Time-fractional Non-linear Schrödinger Equations

In this section, we will consider two-dimensional time-fractional non-linear Schrödinger equations that take the form

$$i\frac{\partial^\nu \psi(x,y,t)}{\partial t^\nu} = A\frac{\partial^2 \psi(x,y,t)}{\partial x^2} + B\frac{\partial^2 \psi(x,y,t)}{\partial y^2} + C\left|\psi(x,y,t)\right|^2 \psi(x,y,t) + q(x,y,t), \tag{7.33}$$

where

$$0 \le x \le R_1, \qquad 0 \le y \le R_2, \qquad 0 < t \le T,$$

with fractional derivative order $\nu \in (0,1]$, and with initial and boundary conditions

$$\begin{aligned}
\psi(x,y,0) &= f(x,y), && 0 \le x \le R_1, && 0 \le y \le R_2,\\
\psi(0,y,t) &= g_{1,0}(y,t), && 0 \le y \le R_2, && 0 < t \le T,\\
\psi(R_1,y,t) &= g_{1,R_1}(y,t), && 0 \le y \le R_2, && 0 < t \le T,\\
\psi(x,0,t) &= g_{2,0}(x,t), && 0 \le x \le R_1, && 0 < t \le T,\\
\psi(x,R_2,t) &= g_{2,R_2}(x,t), && 0 \le x \le R_1, && 0 < t \le T.
\end{aligned} \tag{7.34}$$

To begin, we notice that our equation is in the space of complex numbers, and so we map its constituent functions as combinations of real and imaginary parts:

$$\begin{aligned}
\psi(x,y,t) &= u(x,y,t) + iv(x,y,t),\\
q(x,y,t) &= {}_u q(x,y,t) + i\,{}_v q(x,y,t),\\
f(x,y) &= {}_u f(x,y) + i\,{}_v f(x,y),\\
g_{1,0}(y,t) &= {}_u g_{1,0}(y,t) + i\,{}_v g_{1,0}(y,t),\\
g_{1,R_1}(y,t) &= {}_u g_{1,R_1}(y,t) + i\,{}_v g_{1,R_1}(y,t),\\
g_{2,0}(x,t) &= {}_u g_{2,0}(x,t) + i\,{}_v g_{2,0}(x,t),\\
g_{2,R_2}(x,t) &= {}_u g_{2,R_2}(x,t) + i\,{}_v g_{2,R_2}(x,t),
\end{aligned} \tag{7.35}$$

which when substituted into (7.33) and (7.34), provides the coupled equation

$$\begin{aligned}
\frac{\partial^\nu u(x,y,t)}{\partial t^\nu} &= A\frac{\partial^2 v(x,y,t)}{\partial x^2} + B\frac{\partial^2 v(x,y,t)}{\partial y^2}\\
&\quad + C\left(u^2(x,y,t) + v^2(x,y,t)\right)v(x,y,t) + {}_v q(x,y,t),\\
\frac{\partial^\nu v(x,y,t)}{\partial t^\nu} &= -A\frac{\partial^2 u(x,y,t)}{\partial x^2} - B\frac{\partial^2 u(x,y,t)}{\partial y^2}\\
&\quad - C\left(u^2(x,y,t) + v^2(x,y,t)\right)u(x,y,t) - {}_u q(x,y,t),
\end{aligned} \tag{7.36}$$

with initial and boundary equations

$$\begin{aligned}
u(x,y,0) &= {}_uf(x,y), && 0\le x\le R_1, && 0\le y\le R_2,\\
u(0,y,t) &= {}_ug_{1,0}(y,t), && 0\le y\le R_2, && 0<t\le T,\\
u(R_1,y,t) &= {}_ug_{1,R_1}(y,t), && 0\le y\le R_2, && 0<t\le T,\\
u(x,0,t) &= {}_ug_{2,0}(x,t), && 0\le x\le R_1, && 0<t\le T,\\
u(x,R_2,t) &= {}_ug_{2,R_2}(x,t), && 0\le x\le R_1, && 0<t\le T,\\
v(x,y,0) &= {}_vf(x,y), && 0\le x\le R_1, && 0\le y\le R_2,\\
v(0,y,t) &= {}_vg_{1,0}(y,t), && 0\le y\le R_2, && 0<t\le T,\\
v(R_1,y,t) &= {}_vg_{1,R_1}(y,t), && 0\le y\le R_2, && 0<t\le T,\\
v(x,0,t) &= {}_vg_{2,0}(x,t), && 0\le x\le R_1, && 0<t\le T,\\
v(x,R_2,t) &= {}_vg_{2,R_2}(x,t), && 0\le x\le R_1, && 0<t\le T.
\end{aligned} \tag{7.37}$$

Since we are now dealing with a pair of coupled equations, we will have to adjust the method to account for multiple approximations simultaneously. We begin our coupled approximation scheme by recalling (7.6), where we note that we describe $\mathbf{V}$ in much the same way as we described $\mathbf{U}$. Then, we recall (7.7), observing that the derivatives of (7.36) and (7.37) can be represented by

$$\begin{aligned}
\frac{\partial^\nu u(x,y,t)}{\partial t^\nu} &\simeq [\mathcal{D}_T^\nu\mathbf{P}_T(t)]'\,\mathbf{U}\,\mathbf{P}_{R_1}(x)\otimes\mathbf{P}_{R_2}(y)\\
\frac{\partial^2 u(x,y,t)}{\partial x^2} &\simeq [\mathbf{P}_T(t)]'\,\mathbf{U}\,\left[\mathcal{D}_{R_1}^2\mathbf{P}_2(x)\right]\otimes\mathbf{P}_{R_2}(y),\\
\frac{\partial^2 u(x,y,t)}{\partial y^2} &\simeq [\mathbf{P}_T(t)]'\,\mathbf{U}\,\mathbf{P}_{R_1}(x)\otimes\left[\mathcal{D}_{R_2}^2\mathbf{P}_{R_2}(y)\right],\\
\frac{\partial^\nu v(x,y,t)}{\partial t^\nu} &\simeq [\mathcal{D}_T^\nu\mathbf{P}_T(t)]'\,\mathbf{V}\,\mathbf{P}_{R_1}(x)\otimes\mathbf{P}_{R_2}(y)\\
\frac{\partial^2 v(x,y,t)}{\partial x^2} &\simeq [\mathbf{P}_T(t)]'\,\mathbf{V}\,\left[\mathcal{D}_{R_1}^2\mathbf{P}_2(x)\right]\otimes\mathbf{P}_{R_2}(y),\\
\frac{\partial^2 v(x,y,t)}{\partial y^2} &\simeq [\mathbf{P}_T(t)]'\,\mathbf{V}\,\mathbf{P}_{R_1}(x)\otimes\left[\mathcal{D}_{R_2}^2\mathbf{P}_{R_2}(y)\right],
\end{aligned} \tag{7.38}$$

Now, by substituting (7.6) and (7.38) into (7.24), we obtain

$$\begin{aligned}
&[\mathcal{D}_T^{\nu}\mathbf{P}_T(t_{T,j})]'\,\mathbf{U}\,\mathbf{P}_{R_1}(x_{R_1,i_1})\otimes\mathbf{P}_{R_2}(y_{R_2,i_2})\\
&-A\,[\mathbf{P}_T(t_{T,j})]'\,\mathbf{V}\,\left[\mathcal{D}_{R_1}^2\mathbf{P}_{R_1}(x_{R_1,i_1})\right]\otimes\mathbf{P}_{R_2}(y_{R_2,i_2})\\
&-B\,[\mathbf{P}_T(t_{T,j})]'\,\mathbf{V}\,\mathbf{P}_{R_1}(x_{R_1,i_1})\otimes\left[\mathcal{D}_{R_2}^2\mathbf{P}_{R_2}(y_{R_2,i_2})\right]\\
&-C\Bigg[\Big([\mathbf{P}_T(t_{T,j})]'\,\mathbf{U}\,\mathbf{P}_{R_1}(x_{R_1,i_1})\otimes\mathbf{P}_{R_2}(y_{R_2,i_2})\Big)^2\\
&+\Big([\mathbf{P}_T(t_{T,j})]'\,\mathbf{V}\,\mathbf{P}_{R_1}(x_{R_1,i_1})\otimes\mathbf{P}_{R_2}(y_{R_2,i_2})\Big)^2\Bigg]\\
&\times[\mathbf{P}_T(t_{T,j})]'\,\mathbf{V}\,\mathbf{P}_{R_1}(x_{R_1,i_1})\otimes\mathbf{P}_{R_2}(y_{R_2,i_2})\\
&-{}_vq(x_{R_1,i_1},y_{R_2,i_2},t_{T,j})\\
&=0,\\
&[\mathcal{D}_T^{\nu}\mathbf{P}_T(t_{T,j})]'\,\mathbf{V}\,\mathbf{P}_{R_1}(x_{R_1,i_1})\otimes\mathbf{P}_{R_2}(y_{R_2,i_2})\\
&+A\,[\mathbf{P}_T(t_{T,j})]'\,\mathbf{U}\,\left[\mathcal{D}_{R_1}^2\mathbf{P}_{R_1}(x_{R_1,i_1})\right]\otimes\mathbf{P}_{R_2}(y_{R_2,i_2})\\
&+B\,[\mathbf{P}_T(t_{T,j})]'\,\mathbf{U}\,\mathbf{P}_{R_1}(x_{R_1,i_1})\otimes\left[\mathcal{D}_{R_2}^2\mathbf{P}_{R_2}(y_{R_2,i_2})\right]\\
&+C\Bigg[\Big([\mathbf{P}_T(t_{T,j})]'\,\mathbf{U}\,\mathbf{P}_{R_1}(x_{R_1,i_1})\otimes\mathbf{P}_{R_2}(y_{R_2,i_2})\Big)^2\\
&+\Big([\mathbf{P}_T(t_{T,j})]'\,\mathbf{V}\,\mathbf{P}_{R_1}(x_{R_1,i_1})\otimes\mathbf{P}_{R_2}(y_{R_2,i_2})\Big)^2\Bigg]\\
&\times[\mathbf{P}_T(t_{T,j})]'\,\mathbf{U}\,\mathbf{P}_{R_1}(x_{R_1,i_1})\otimes\mathbf{P}_{R_2}(y_{R_2,i_2})\\
&+{}_uq(x_{R_1,i_1},y_{R_2,i_2},t_{T,j})\\
&=0.
\end{aligned}\tag{7.39}$$

for $i_1 = 1, 2, ..., N_1 - 1$, $i_2 = 1, 2, ..., N_2 - 1$, and $j = 1, 2, ..., M$. We observe that we have two sets of equations here, as opposed to the usual one. For the initial and boundary conditions, we have

$$\begin{aligned}
[\mathbf{P}_T(0)]'\,\mathbf{U}\,\mathbf{P}_R(x_{R_1,i_1}) \otimes \mathbf{P}_{R_2}(y_{R_2,i_2}) &= {}_u f(x_{R_1,i_1}, y_{R_2,i_2}), \\
&\quad 0 \le i_1 \le N_1,\ 0 \le i_2 \le N_2, \\
[\mathbf{P}_T(t_{T,j})]'\,\mathbf{U}\,\mathbf{P}_{R_1}(0) \otimes \mathbf{P}_{R_2}(y_{R_2,i_2}) &= {}_u g_{1,0}(y_{R_2,i_2}, t_{T,j}), \\
&\quad 0 \le i_2 \le N_2,\ 0 < j \le M, \\
[\mathbf{P}_T(t_{T,j})]'\,\mathbf{U}\,\mathbf{P}_{R_1}(R_1) \otimes \mathbf{P}_{R_2}(y_{R_2,i_2}) &= {}_u g_{1,R_1}(y_{R_2,i_2}, t_{T,j}), \\
&\quad 0 \le i_2 \le N_2,\ 0 < j \le M, \\
[\mathbf{P}_T(t_{T,j})]'\,\mathbf{U}\,\mathbf{P}_{R_1}(x_{R_1,i_1}) \otimes \mathbf{P}_{R_2}(0) &= {}_u g_{2,0}(x_{R_1,i_1}, t_{T,j}), \\
&\quad 0 \le i_1 \le N_1,\ 0 < j \le M, \\
[\mathbf{P}_T(t_{T,j})]'\,\mathbf{U}\,\mathbf{P}_{R_1}(x_{R_1,i_1}) \otimes \mathbf{P}_{R_2}(R_2) &= {}_u g_{2,R_2}(x_{R_1,i_1}, t_{T,j}), \\
&\quad 0 \le i_1 \le N_1,\ 0 < j \le M, \\
[\mathbf{P}_T(0)]'\,\mathbf{V}\,\mathbf{P}_R(x_{R_1,i_1}) \otimes \mathbf{P}_{R_2}(y_{R_2,i_2}) &= {}_v f(x_{R_1,i_1}, y_{R_2,i_2}), \\
&\quad 0 \le i_1 \le N_1,\ 0 \le i_2 \le N_2, \\
[\mathbf{P}_T(t_{T,j})]'\,\mathbf{V}\,\mathbf{P}_{R_1}(0) \otimes \mathbf{P}_{R_2}(y_{R_2,i_2}) &= {}_v g_{1,0}(y_{R_2,i_2}, t_{T,j}), \\
&\quad 0 \le i_2 \le N_2,\ 0 < j \le M, \\
[\mathbf{P}_T(t_{T,j})]'\,\mathbf{V}\,\mathbf{P}_{R_1}(R_1) \otimes \mathbf{P}_{R_2}(y_{R_2,i_2}) &= {}_v g_{1,R_1}(y_{R_2,i_2}, t_{T,j}), \\
&\quad 0 \le i_2 \le N_2,\ 0 < j \le M, \\
[\mathbf{P}_T(t_{T,j})]'\,\mathbf{V}\,\mathbf{P}_{R_1}(x_{R_1,i_1}) \otimes \mathbf{P}_{R_2}(0) &= {}_v g_{2,0}(x_{R_1,i_1}, t_{T,j}), \\
&\quad 0 \le i_1 \le N_1,\ 0 < j \le M, \\
[\mathbf{P}_T(t_{T,j})]'\,\mathbf{V}\,\mathbf{P}_{R_1}(x_{R_1,i_1}) \otimes \mathbf{P}_{R_2}(R_2) &= {}_v g_{2,R_2}(x_{R_1,i_1}, t_{T,j}), \\
&\quad 0 \le i_1 \le N_1,\ 0 < j \le M.
\end{aligned} \tag{7.40}$$

If we combine (7.39) and (7.40), we obtain a system of $2\times(M+1)\times\Big((N_1+1)\times(N_2+1)\Big)$ equations; this system is double the size of those we have solved previously, to account for the addition approximation this method must produce. We use the

Levenberg-Marquardt algorithm, taking in a matrix concatenated from $\mathbf{U}$ and $\mathbf{V}$ as its variable, with an initial guess of all zeros, to minimise (7.39) and (7.40). This matrix of $\mathbf{U}$ and $\mathbf{V}$ is then used in (7.6) to calculate our approximations of $u(x, y, t)$ and $v(x, y, t)$.

7.5.1 A Numerical Example

For this example, we consider the following two-dimensional time-fractional attracting non-linear Schrödinger equation, investigated by Bhrawy and Abdelkawy [79]:

$$i\frac{\partial^\nu \psi(x,y,t)}{\partial t^\nu} = -\frac{\partial^2 \psi(x,y,t)}{\partial x^2} - \frac{\partial^2 \psi(x,y,t)}{\partial y^2} - |\psi(x,y,t)|^2 \psi(x,y,t) + q(x,y,t), \tag{7.41}$$

where

$$0 \leq x \leq 1, \qquad 0 \leq y \leq 1, \qquad 0 < t \leq 1,$$

with function

$$\begin{aligned} &q(x,y,t) \\ &= t^2 \left(\frac{1}{2} \sin(x)\sin(y) \left(t^4 \cos(2x)\cos(2y) + t^4 - 4 \right) - \frac{2t^{-\nu}\cos(x)\cos(y)}{\Gamma(3-\nu)} \right) \\ &\quad + it^2 \left(\frac{1}{2} \cos(x)\cos(y) \left(t^4 \cos(2x)\cos(2y) + t^4 - 4 \right) + \frac{2t^{-\nu}\sin(x)\sin(y)}{\Gamma(3-\nu)} \right), \end{aligned} \tag{7.42}$$

with fractional derivative order $\nu \in (0, 1]$, and with initial and boundary conditions

$$\begin{aligned}
&\psi(x, y, 0) = 0, && 0 \le x \le 1, && 0 \le y \le 1,\\
&\psi(0, y, t) = it^2 \cos(y), && 0 \le y \le 1, && 0 < t \le 1,\\
&\psi(1, y, t) = t^2 \left(\sin(1) \sin(y) + i \cos(1) \cos(y)\right), && 0 \le y \le 1, && 0 < t \le 1,\\
&\psi(x, 0, t) = it^2 \cos(x), && 0 \le x \le 1, && 0 < t \le 1,\\
&\psi(x, 1, t) = t^2 \left(\sin(x) \sin(1) + i \cos(x) \cos(1)\right), && 0 \le x \le 1, && 0 < t \le 1.
\end{aligned} \tag{7.43}$$

This equation has exact solution

$$\psi(x, y, t) = t^2 \left(\sin(x) \sin(y) + i \cos(x) \cos(y)\right). \tag{7.44}$$

With fractional derivative order $\nu = 0.2$, and $M = 1$, $N_1 = N_2 = 10$ collocation points, implemented in MATLAB as shown in Appendix U, we observe from Figure 7.10 that the approximations of $u(x, y, t)$ and $v(x, y, t)$ match the exact solutions in form, with the detailed views of the approximations at $t = 1$ given in Figure 7.11.

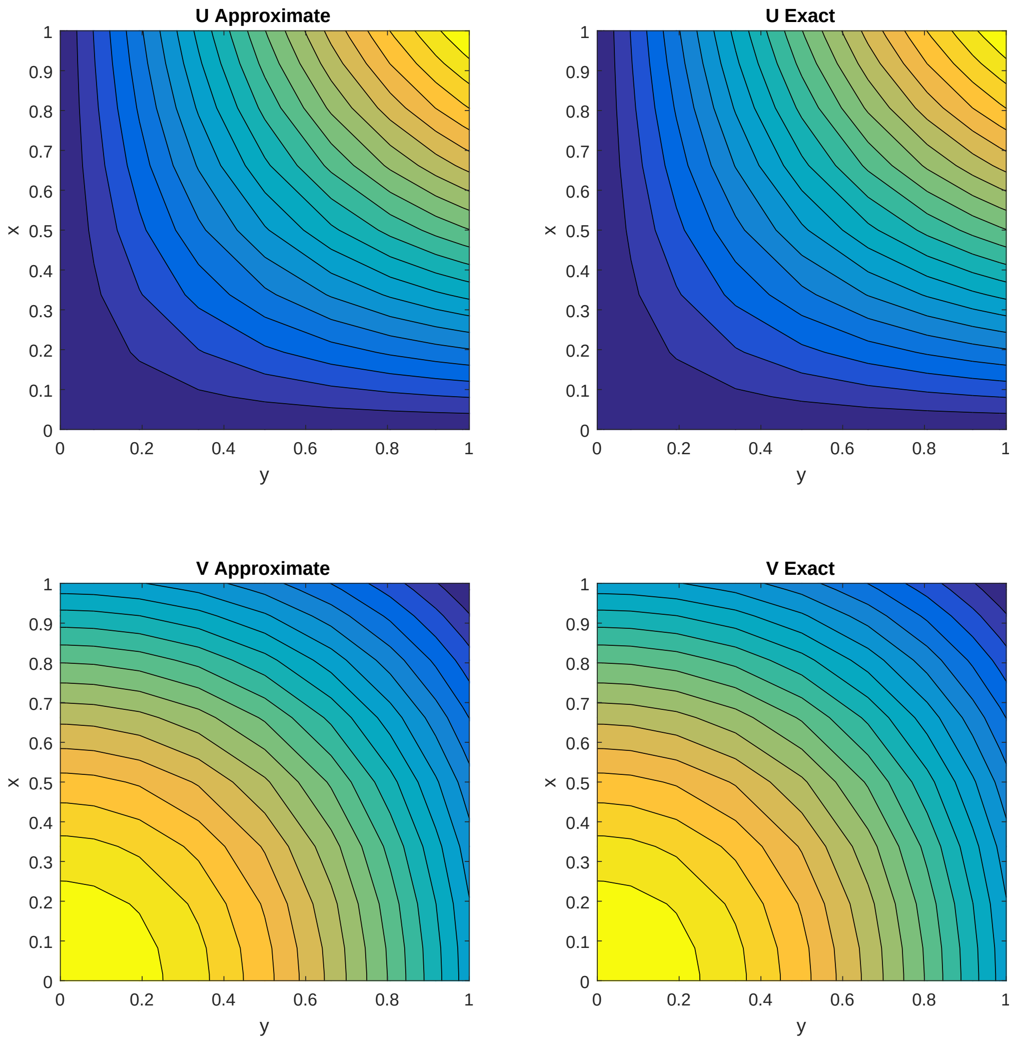
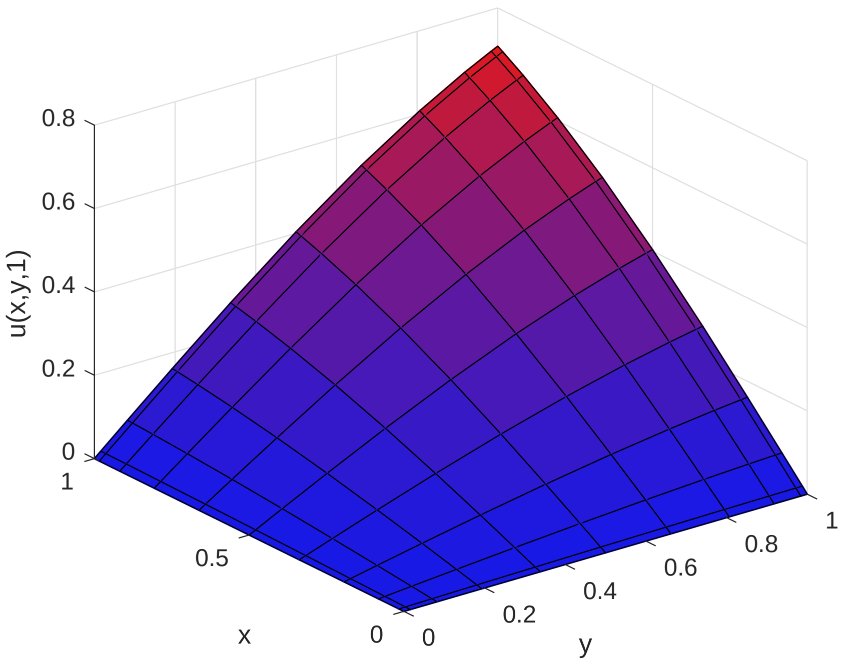


Figure 7.10. Contours of Approximate and Exact Solution, $M = 1$, $N_1 = N_2 = 10$



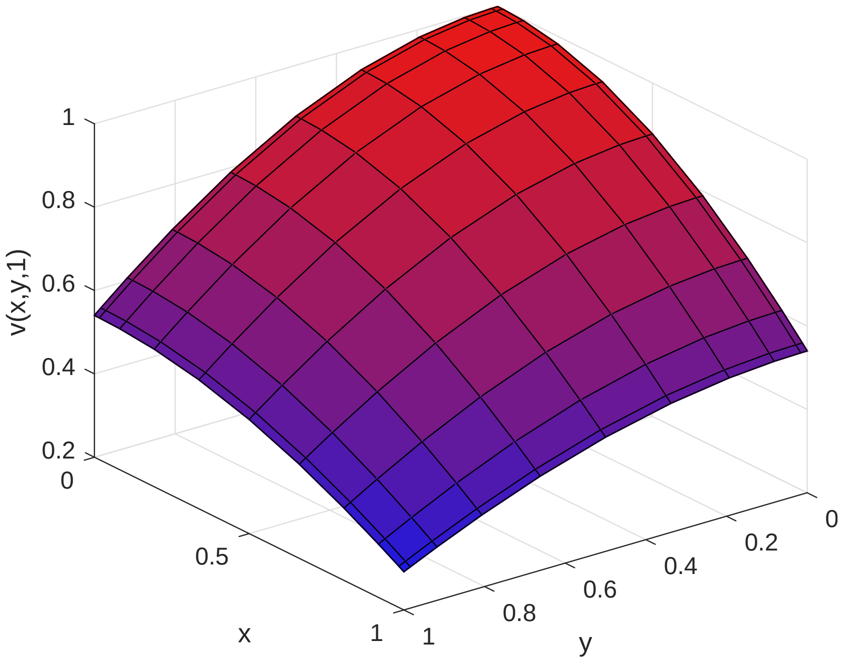
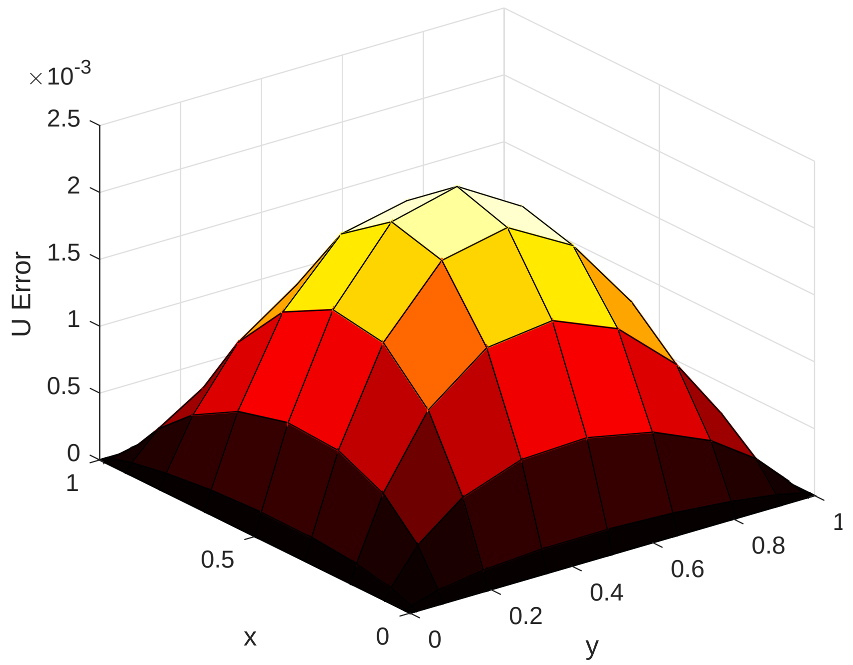


Figure 7.11. Approximate solutions for $u(x,y,1)$ and $v(x,y,1)$, $M=1$, $N_1=N_2=10$



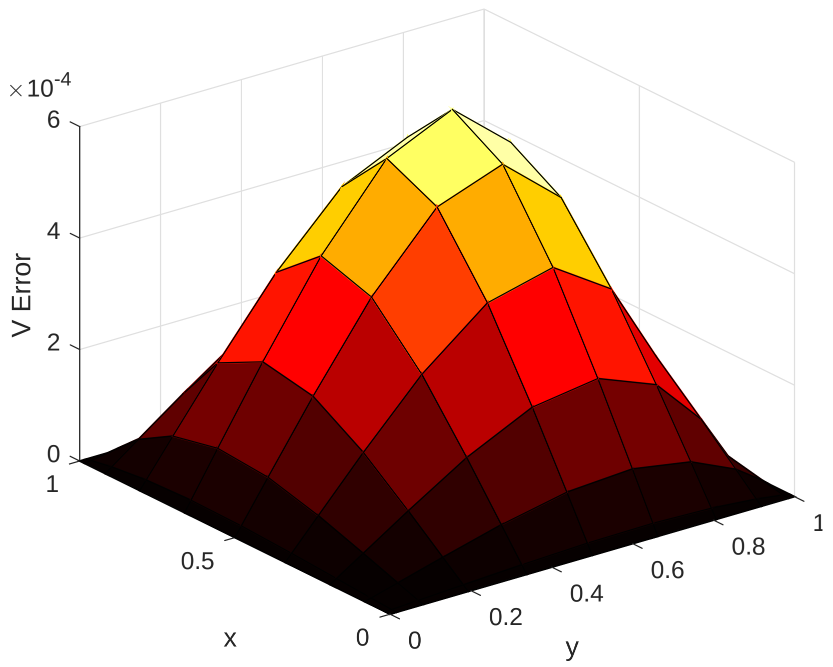


Figure 7.12. Absolute Error, $M = 1$, $N_1 = N_2 = 10$

Table 7.4. Maximum Error for $M = 1$, $N_1 = N_2 = 2, ..., 10$

$N_1 = N_2$	Maximum Error in u	Maximum Error in v
2	$0.243519883068186 \times 10^{-2}$	$0.189281200161284 \times 10^{-2}$
3	$0.119727430592679 \times 10^{-2}$	$0.135777516151370 \times 10^{-2}$
4	$0.218872556866079 \times 10^{-2}$	$0.480877636537258 \times 10^{-3}$
5	$0.192067163749554 \times 10^{-2}$	$0.583329946943656 \times 10^{-3}$
6	$0.217840551528567 \times 10^{-2}$	$0.497694481170097 \times 10^{-3}$
7	$0.209316968936274 \times 10^{-2}$	$0.580975291405439 \times 10^{-3}$
8	$0.217817028138992 \times 10^{-2}$	$0.568097615525565 \times 10^{-3}$
9	$0.215323250699784 \times 10^{-2}$	$0.569189843815598 \times 10^{-3}$
10	$0.217816886228733 \times 10^{-2}$	$0.584184914312447 \times 10^{-3}$

We see in Figure 7.12 that the error is concentrated towards the spatial centers for both $u(x, y, t)$ and $v(x, y, t)$, and considering Table 7.4, we observe that the error is fairly consistent for $N_1 = N_2 \geq 4$. We note that due to the large number of equations involved in this system, and the limits of available hardware and software, we incur rounding error for even small numbers of collocation points, and so while larger resolutions will theoretically be better approximations, it so happens that the best error achievable for this scheme is with $M = 1$, with larger values of M incurring more rounding error than the amount of truncation error that is removed. This is a known drawback of collocation methods, given the dense and often ill-conditioned systems associated to large numbers of collocation points [73, 74]. This implementation is less effective than other spectral schemes implemented on two-dimensional time-fractional Shrödinger equations [79], but it is unknown how performance would compare if it were impacted less by rounding error.

7.6 Concluding Remarks

In this chapter, we extended the method of Chapter 6 to solve fractional PDEs that have two dimensions in space, following the design of prior literature [53], wherein a method was developed for time-fractional diffusion equations. We adapted the method to deal with space-fractional derivatives, fractional derivatives of order $\nu \in (1, 2]$, derivative boundary conditions, and coupled equations, testing these adaptations on a number of relevant examples.

Our first test was a space-fractional diffusion equation, where it was found that the Jacobi collocation method tested here provided better accuracy than that of prior literature [52], and was significantly more accurate than the alternating-direction finite difference method [32].

We then considered time-fractional diffusion wave equations, where the accuracy of the method investigated here was better than that of the Finite Element method, even when comparing small collocation node numbers to high-resolution implementations of the Finite Element method [77].

Our third test was a time-fractional non-linear Klein-Gordon equation with a derivative initial condition, for which the results of the method tested here expressed less error than that of the meshless method used in prior literature, with the best reported accuracy of that method being less than this Jacobi collocation method with even small grid sizes.

Finally, we adapted the method to solve a time-fractional non-linear Shrödinger equation, represented as coupled equations in the real and imaginary spaces. Our method was able to solve the coupled equations with reasonable accuracy, but the approximations were not as accurate as other spectral methods in prior literature [79].

We observed that in solving fractional PDEs with two dimensions in space, the complexity of the system naturally increases as we increase the resolution of any of the three dimensions. This means that we have rather large systems to solve for even seemingly small numbers of collocation nodes, and so rounding error becomes an issue. Nevertheless, accurate solutions were obtained for all problems investigated in this chapter, and considering the extensions made to the method to deal with the larger variety of equations, the method has demonstrated its robustness and versatility.

Chapter 8

Conclusions

The aim of this project was to contribute to the burgeoning discussion of numerical solutions of fractional partial differential equations. The global nature of spectral methods makes them inherently well suited to the solution of FPDEs, the derivatives of which are themselves non-local. Focusing on spectal collocation methods, several schemes were investigated, with generally favourable accuracy characteristics.

In Chapter 2, we discussed the definitions and expressions necessary to represent fractional integrals and derivatives in ways that would permit numerical approximation. By unifying the notions of derivatives and integrals, we formed a consistent representation that was extendable to fractional orders. We discussed the historically significant fractional derivative definitions, and presented the Caputo definition, allowing us to treat a considerable variety of fractional differential equations. We investigated the important fractional derivative of $(t-a)^{\beta}$, which came to be useful in our subsequent methodologies.

In Chapter 3, we introduced the core concepts of spectral methods, including their defining global trial functions, and the test functions that distinguish between the three prominent varieties of spectral method, being the Galerkin, tau and collocation methods. We discussed at length the fundamentals of orthogonal polynomials, how they are obtained, and their value in approximating functions. We considered various Gaussian quadratures, how these quadratures are obtained, and how they benefit in the numerical solution of different types of problems. We then discussed Jacobi polynomials, being the unique class of polynomial arising from a singular Sturm-Liouville problem, and how Legendre and Chebyshev polynomials are variants of the general Jacobi polynomial. We went on to discuss shifted polynomials, and how they are obtained from the standard formulae.

In Chapter 4 and Chapter 5, we replicated key results from the work of Bhrawy and Baleanu, investigating both one- and two-dimensional FPDEs. We numerically solved a one-dimensional space-fractional advection diffusion equation using shifted Legendre polynomials under a Guass-Lobatto quadrature. Using the Caputo definition of the fractional derivative, and the technique of Legendre-Gauss-Lobatto collocation, we were able to represent the FPDE as a system of ODEs in the time variable. This system was then solved using explicit RK4, where reasonably accurate results were obtained, although not as good as those possible with an implicit method. We then applied a similar methodology for a two-dimensional space-fractional diffusion equation, for which we obtained accuracy results that compared favourably with prior research, especially for smaller grid sizes.

In Chapter 6, extending the work of Doha, Bhrawy and Ezz-Eldien, we were able to obtain an operational matrix representation of partial differential equations with one spatial dimension, and their various derivatives of arbitrary orders, using the Caputo definition of the fractional derivative, and the technique of Jacobi-Gauss-Lobatto collocation. This provided us with a system of equations that was then solved using the Levenberg-Marquardt algorithm. This approach was used to solve a non-homogeneous sub-diffusion equation, where the method here was able to achieve greater accuracy than prior methods, especially for the smaller grid sizes. The method was then used to numerically solve a non-linear time-fractional diffusion equation, where it was found that this particular implementation was again able to provide more accurate results at smaller grid sizes than those in prior literature. Lastly, the method was used to solve a space-fractional hyperbolic equation with derivative boundary conditions, where it was found that the method was easily adapted to incorporate the derivative boundary condition, and provided reasonably accurate results.

In Chapter 7, we benefitted from the work of Bhrawy to extend the previous method to solve fractional PDEs that have two dimensions in space. We adapted the method to deal with space-fractional derivatives, fractional derivatives of higher order, derivative boundary conditions, and coupled equations, testing these adaptations on a number of relevant examples. We first tested a space-fractional diffusion equation, where it was found that our Jacobi collocation method provided better accuracy than that of prior literature, and was significantly more accurate than Meerschaert's influential alternating-direction finite difference method. Our second test considered time-fractional diffusion wave equations, where the accuracy of the method investigated here was better than that of the Finite Element method. We then tested a time-fractional non-linear Klein-Gordon equation with a derivative initial condition, for which the results of our method expressed considerably less error than that of the meshless method used in prior literature. Finally, we adapted the method to solve a time-fractional non-linear Shrödinger equation, represented as coupled equations in the real and imaginary spaces. Our method was able to solve the coupled equations with reasonable accuracy, although with less accuracy than some prior spectral methods. We noted the known issue of spectral matrices being ill-conditioned and dense is exacerbated by the addition of extra dimensions to this method, but for small grid sizes, high accuracy was still achievable.

We acknowledge that there are limitations to the research presented in this project. Ideally, one would be able to provide rigorous analytical investigations into the stablity, consistency and convergence characteristics of the proposed schemes, but given such analysis is absent even for the initial schemes presented by the methods' originators, this researcher is unaware how to begin such an endeavour, and as such, it is outside the scope of this project, with the accuracy characteristics instead tested only for examples with known exact solutions, as is the current practice. Additionally, the differential equations for which schemes were developed in this project all possessed only left fractional derivatives, given the more natural physical interpretation of left derivatives, and their concomittant relative abundance. A more complete treatment would include two-sided, or even right-only derivatives, but this was not the focus of this project, and so resources were not spent on the development of methods for such equations. Lastly, the ill-conditioned systems that limited the accuracy of the collocation methods presented here are worth investigating, especially considering that this problem has been acknowledged by prior literature, and is thus of interest to other researchers and practitioners of such methods. All of these omittances serve as valuable areas of further research.

This project aimed to bring together two mathematical topics that are both gaining relevance in the ongoing discourse, yet are not currently overly familiar to most mathematicians and practitioners. It was the researcher's belief that both fractional calculus and spectral methods are novel topics, and the convenient appropriateness of the non-local functions inherent to spectral methods in solving fractional partial differential equations made the aim of this project both original and meaningful. Given the relative unfamiliarity of the material, care was given into arranging the theory such that limited knowledge of the underlying principles would be required to understand, derive, and apply the methods presented to FPDEs that one might reasonably encounter, even outside of abstract sciences. To this end, many example equations were considered for the development of numerical schemes, distinct to those investigated with similar methods in prior literature, demonstrating the power of spectral collocation for the solution of various FPDEs. In conjunction with the abundant code presented in the Appendices, it is the researcher's hope that this document will serve additionally as a pedagogical resource for anyone interested in solving fractional partial differential equations, with the versatile and effective methods investigated herein.

References

[1] I. Podlubny. *Fractional Differential Equations*. Academic Press, New York, 1999.

[2] M.M. Meerschaert and C. Tadjeran. Finite difference approximations for two-sided space-fractional partial differential equations. *Applied Numerical Mathematics*, 56(1):80–90, 2006.

[3] E. Barkai, R. Metzler, and J. Klafter. From continuous time random walks to the fractional fokker–planck equation. *Physical Review E*, 61(1):132, 2000.

[4] A. Blumen, G. Zumofen, and J. Klafter. Transport aspects in anomalous diffusion: Levy walks. *Physical Review A*, 40(7):3964, 1989.

[5] J.P. Bouchaud and A. Georges. Anomalous diffusion in disordered media—statistical mechanisms, models and physical applications. *Physics Reports*, 195(4-5):127–293, 1990.

[6] A. Chaves. Fractional diffusion equation to describe levy flights. *Physics Letters A*, 239(1):13–16, 1998.

[7] J. Klafter, Blumen A., and M.F. Shlesinger. Stochastic pathways to anomalous diffusion. *Physical Review A*, 35(7):3081, 1987.

[8] F. Mainardi, M. Raberto, R. Gorenflo, and E. Scalas. Fractional calculus and continuous-time finance II: the waiting-time distribution. *Physica A*, 287:468–481, 2000.

[9] R. Gorenflo, F. Mainardi, E. Scalas, and M. Raberto. Fractional calculus and continuous-time finance III: the diffusion limit. In *Mathematical Finance*, pages 171–180. Springer, 2001.

[10] M. Raberto, F. Mainardi, and E. Scalas. Waiting-times and returns in high-frequency financial data: an empirical study. *Physica A: Statistical Mechanics and its Applications*, 314(1):749–755, 2002.

[11] L. Sabatelli, S. Keating, J. Dudley, and P. Richmond. Waiting time distributions in financial markets. *The European Physical Journal B-Condensed Matter and Complex Systems*, 27(2):273–275, 2002.

[12] E. Scalas, R. Gorenflo, and F. Mainardi. Fractional calculus and continuous-time finance. *European Physical Journal B*, 284:376–384, 2000.

[13] D.A Benson, S.W. Wheatcraft, and M.M. Meerschaert. Application of a fractional advection–dispersion equation. *Physica A: Statistical Mechanics and its Applications*, 284(1):376–384, 2000.

[14] B. Baeumer, M.M. Meerschaert, D.A. Benson, and S.W. Wheatcraft. Subordinated advection–dispersion equation for contaminant transport. *Water Resources Research*, 37(6):1543–1550, 2001.

[15] D.A. Benson, R. Schumer, M.M. Meerschaert, and S.W. Wheatcraft. Fractional dispersion, levy motions, and the made tracer tests. In *Dispersion in Heterogeneous Geological Formations*, pages 211–240. Springer, 2001.

[16] R. Schumer, D.A. Benson, M.M. Meerschaert, and S.W. Wheatcraft. Eulerian derivation of the fractional advection–dispersion equation. *Journal of Contaminant Hydrology*, 48(1):69–88, 2001.

[17] R. Schumer, D.A Benson, M.M. Meerschaert, and B. Baeumer. Multiscaling fractional advection–dispersion equations and their solutions. *Water Resources Research*, 39(1), 2003.

[18] M.M. Meerschaert and C. Tadjeran. Finite difference approximations for fractional advection–dispersion flow equations. *Journal of Computational and Applied Mathematics*, 172(1):65–77, 2004.

[19] C. Angstmann, B.I. Henry, and McGann A.V. A fractional order recovery SIR model from a stochastic process. *Bulletin of Mathematical Biology*, 78(3):468–499, 2016.

[20] L. Blank. Numerical treatment of differential equations of fractional order. *Numerical Analysis Report - University of Manchester Department of Mathematics*, 1996.

[21] C. Canuto, M.Y. Hussaini, A. Quarteroni, and T.A. Zang. *Spectral Methods in Fluid Dynamics*. Springer-Verlag, Berlin, 1988.

[22] K.B. Oldham and J. Spanier. *The Fractional Calculus*. Academic Press, New York, 1974.

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

[24] M. Caputo. *Elasticità e dissipazione*. Zanichelli, 1969.

[25] M. Caputo. Vibrations of an infinite plate with a frequency independent Q. *The Journal of the Acoustical Society of America*, 60(3):634–639, 1976.

[26] R.L Bagley and P.J. Torvik. A theoretical basis for the application of fractional calculus to viscoelasticity. *Journal of Rheology (1978-present)*, 27(3):201–210, 1983.

[27] R. Metzler, W.G. Glöckle, and T.F. Nonnonmacher. Fractional model equation for anomalous diffusion. *Physica A: Statistical Mechanics and its Applications*, 211(1):13–24, 1994.

[28] R. Metzler and J. Klafter. The random walk's guide to anomalous diffusion: A fractional dynamics approach. *Physics Reports*, 339(1):1–77, 2000.

[29] S. Havlin and D. Ben-Avraham. Diffusion in disordered media. *Advances in Physics*, 36(6):695–798, 1987.

[30] S. Havlin, S.V Bulyrev, A.I Golderberger, R.N. Mantegna, S.M Ossadnik, C.K. Peng, M. Simons, and H.E. Stanley. Fractals in biology and medicine. *Chaos, Solitons & Fractals*, 6:171–201, 1995.

[31] B. Mandlebrot. *The Fractal Geometry of Nature*. Freeman, 1982.

[32] M.M. Meerschaert, H. Scheffler, and C. Tadjeran. Finite difference methods for two dimensional fractional dispersion equation. *Journal of Computational Physics*, 211(1):249–261, 2006.

[33] N. Laskin. Fractional quantum mechanics and lévy path integrals. *Physics Letters A*, 268(4):298–305, 2000.

[34] N. Laskin. Fractional quantum mechanics. *Physical Review E*, 62(3):3135, 2000.

[35] N. Laskin. Fractional schrödinger equation. *Physical Review E*, 66(5):056108, 2002.

[36] B. Jacobs. *On the Application of Partial Differential Equations and Fractional Partial Differential Equations to Images and Their Methods of Solution.* University of the Witwatersrand, Johannesburg, 2014.

[37] R.L. Bagley and P.J. Torvik. Fractional calculus-a different approach to the analysis of viscoelastically damped structures. *AIAA Journal*, 21(5):741–748, 1983.

[38] C. Lubich. Fractional linear multistep methods for abel-volterra integral equations of the second kind. *Mathematics of Computation*, 45(172):463–469, 1985.

[39] C. Lubich. Discretized fractional calculus. *SIAM Journal on Mathematical Analysis*, 17(3):704–719, 1986.

[40] C.T.H. Baker and M.S. Derakhshan. FFT techniques in the numerical solution of convolution equations. *Journal of Computational and Applied Mathematics*, 20:5–24, 1987.

[41] I. Podlubny. Numerical solution of initial value problems for fractional order differential equations. In *Proceedings of the 12th IMACS World Congress*, 1994.

[42] I. Podlubny. Analytical solution of linear differential equations of the fractional order. In *Proceedings of the 12th IMACS World Congress*, 1994.

[43] I. Podlubny. Numerical solution of ordinary fractional differential equations by the fractional difference method. In *Proceedings of the SICDEA*, 1995.

[44] K. Diethelm. An algorithm for the numerical solution of differential equations of fractional order. *Electronic Transactions on Numerical Analysis*, 5(1):1–6, 1997.

[45] K. Diethelm and N.J. Ford. Analysis of fractional differential equations. *Journal of Mathematical Analysis and Applications*, 265(2):229–248, 2002.

[46] K. Diethelm, N. J. Ford, and A.D. Freed. A predictor-corrector approach for the numerical solution of fractional differential equations. *Nonlinear Dynamics*, 29(1-4):3–22, 2002.

[47] V.E. Lynch, B.A. Carreras, D. del Castillo-Negrete, K.M. Ferreira-Mejias, and H.R. Hicks. Numerical methods for the solution of partial differential equations of fractional order. *Journal of Computational Physics*, 192(2):406–421, 2003.

[48] E.H. Doha, A.H. Bhrawy, and S.S. Ezz-Eldien. A Chebyshev spectral method based on operational matrix for initial and boundary value problems of fractional order. *Computers & Mathematics with Applications*, 62(5):2364–2373, 2011.

[49] A.H. Bhrawy, M.A. Zaky, and J.A.T. Machado. Numerical solution of the two-sided space–time fractional telegraph equation via chebyshev tau approximation. *Journal of Optimization Theory and Applications*, pages 1–21, 2016.

[50] E.H. Doha, A.H. Bhrawy, and S.S Ezz-Eldien. A new Jacobi operational matrix: An application for solving fractional differential equations. *Applied Mathematical Modelling*, 36(10):4931–4943, 2012.

[51] A.H. Bhrawy and D. Baleanu. A spectral Legendre–Gauss–Lobatto collocation method for a space-fractional advection diffusion equations with variable coefficients. *Reports on Mathematical Physics*, 72(2):219–233, 2013.

[52] A.H. Bhrawy. A new Legendre collocation method for solving a two dimensional fractional diffusion equation. *Abstract and Applied Analysis*, 2014:1–10, 2014.

[53] A.H. Bhrawy. A Jacobi spectral collocation method for solving multidimensional nonlinear fractional subdiffusion equations. *Numerical Algorithms*, pages 1–23, 2015.

[54] S.S. Ray and R.K. Bera. Solution of an extraordinary differential equation by adomian decomposition method. *Journal of Applied Mathematics*, 4(2004):331–338, 2004.

[55] C. Li and C. Tao. On the fractional adams method. *Computers & Mathematics with Applications*, 58(8):1573–1588, 2009.

[56] Z. Odibat, S. Momani, and H. Xu. A reliable algorithm of homotopy analysis method for solving nonlinear fractional differential equations. *Applied Mathematical Modelling*, 34(3):593–600, 2010.

[57] O. Abdulaziz, I. Hashim, and S. Momani. Application of homotopy-perturbation method to fractional ivps. *Journal of Computational and Applied Mathematics*, 216(2):574–584, 2008.

[58] S. Yang, A. Xiao, and H. Su. Convergence of the variational iteration method for solving multi-order fractional differential equations. *Computers & Mathematics with Applications*, 60(10):2871–2879, 2010.

[59] O.K. Jaradat, K. Al-Banawi, and A.A. Joudeh. Solving fractional hyperbolic partial differential equations by the generalized differential transform method. *World Applied Sciences Journal*, 23(12):89–96, 2013.

[60] K. Miller and M. Ross. *An Introduction to the Fractional Calaulus and Fractional Differential Equations.* John Wiley & Sons Inc, New York, 1993.

[61] P. Wang and C. Huang. Split-step alternating direction implicit difference scheme for the fractional Schrödinger equation in two dimensions. *Computers and Mathematics with Applications*, 71(5):1114–1128, 2016.

[62] L.N. Trefethen. *Spectral methods in MATLAB*, volume 10. Siam, 2000.

[63] S.A. Orszag. Comparison of pseudospectral and spectral approximation. *Studies in Applied Mathematics*, 51(3):253–259, 1972.

[64] P. Grandclément. Introduction to spectral methods. *EAS Publications Series*, 21:153–180, 2006.

[65] G. Szegö. *Orthogonal Polynomials.* American Mathematical Society, Rhode Island, 1975.

[66] R. El Attar. *Legendre Polynomials and Functions.* CreateSpace, 2009.

[67] J.P Boyd and R.G Petschek. The relationships between chebyshev, legendre and jacobi polynomials: The generic superiority of chebyshev polynomials. *Journal of Scientific Computing*, 59:1–27, 2014.

[68] A.H. Bhrawy and M.M. Al-Shomrani. A shifted Legendre spectral method for fractional-order multi-point boundary value problems. *Advances in Difference Equations*, 2012(1):1, 2012.

[69] M. Abramowitz and I. Stegun. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables.* Dover Publications, 1983.

[70] B. Guo and J. Yan. Legendre–Gauss collocation method for initial value problems of second order ordinary differential equations. *Applied Numerical Mathematics*, 59(6):1386–1408, 2009.

[71] A.H. Bhrawy, A.S. Alofi, and S.S. Ezz-Eldien. A quadrature tau method for fractional differential equations with variable coefficients. *Applied Mathematics Letters*, 24(12):2146–2152, 2011.

[72] J.C. Butcher. *Numerical Methods for Ordinary Differential Equations*. Wiley, West Sussex, 2008.

[73] S. Olver and A. Townsend. A fast and well-conditioned spectral method. *SIAM Review*, 55(3):462–489, 2013.

[74] L. Wang, M.D. Samson, and X. Zhao. A well-conditioned collocation method using a pseudospectral integration matrix. *SIAM Journal on Scientific Computing*, 36(3):A907–A929, 2014.

[75] J.J. Moré. *The Levenberg-Marquardt algorithm: Implementation and theory*, page 105–116. Springer Berlin Heidelberg, Berlin, Heidelberg, 1978.

[76] G. Gao and Z. Sun. A compact finite difference scheme for the fractional sub-diffusion equations. *Journal of Computational Physics*, 230(3):586–595, 2011.

[77] M. Dehghan, M. Abbaszadeh, and A. Mohebbi. Analysis of two methods based on Galerkin weak form for fractional diffusion wave: Meshless interpolating element free galerkin and finite element methods. *Engineering Analysis with Boundary Elements*, 64:205–221, 2016.

[78] M. Dehghan, M. Abbaszadeh, and A. Mohebbi. An implicit RBF meshless approach for solving the time fractional nonlinear sine-Gordon and Klein-Gordon equations. *Engineering Analysis with Boundary Elements*, 50:412–434, 2015.

[79] A.H. Bhrawy and M.A. Abdelkawy. A fully spectral collocation approximation for multi-dimensional fractional Schrödinger equations. *Journal of Computational Physics*, 294:462–483, 2015.

Appendices

Appendix A

Chapter 4 Example MATLAB Code

```matlab
% Chapter 4 Example 1: u_t = -t*sin(2x)*u_x + t^3*x^3*u_(x^nu) + u + q
% q = sin(2*x)*t*exp(-t)*(8*x - 12*x^2 + 4*x^3) -...
%        t^3*x^3*exp(-t)*((24*x^(4-nu))/gamma(5-nu) - ...
%       (24*x^(3-nu))/gamma(4-nu) + (8*x^(2-nu))/gamma(3-nu) )
%
% IC: u(x,0) = x^2(2-x)^2
% DnuC: u(0,t) = 0;   u(1,t) = 0;
%
% exact solution: exp(-t)*(x^2)*(2-x)^2;

% set up workspace
clear all
clc
set(0,'DefaultFigureWindowStyle', 'docked')
format long

% initialize vector of maximum errors for table
max_error = zeros(9,1);

%% Loop through varying numbers of collocation points
for N=2:10

% set constants
nu = 1.45;
L = 2;
T = 1;
dt = 0.01; time_frac = 2; dt_frac = dt/time_frac;
t = 0:dt_frac:T;
```

```
%% other calculated values
h_Lj = L./(2*(0:N)+1);

% obtain x nodes and Christoffel numbers
[X,CHRIS] = LegGauLob(N);

% shift from [-1,1] to [0,L];
x = (L/2)*(double(X) + 1); Chris = (L/2)*double(CHRIS);

%% Polynomials and matrices
% shifted Legendre polynomials
P_Lj = zeros(N+1,N-1);
PJ = P_Lj;
for j = 0:N
    syms y
    P_L = legendreP(j,y);
    y = X(2:end-1);
    P_Lj(j+1,:) = subs(P_L);
end

% shifted Jacobi polynomials
for j = 1:N
    syms y
    PJacobi = jacobiP(j-1,1,1,y);
    y = X(2:end-1);
    PJ(j+1,:) = subs(PJacobi);
end

% Dnu matrix
Dnu = zeros( N-1 );
for n = 1:N-1
    for i = 1:N-1
        for j = 0:N
            for l = 0:N
                Pi_nu = Pinu(j,l,nu,L);
                Dnu(n,i) = Dnu(n,i)+(1/h_Lj(j+1))*P_Lj(j+1,i)*Pi_nu*...
                    P_Lj(l+1,n)*Chris(i+1);

            end
        end
    end
```

```
end

% D1 matrix
D1 = zeros( N-1 );
for n = 1:N-1
    for i = 1:N-1
        for j = 0:N
            D1(n,i) = D1(n,i)+((j+1)/(2*h_Lj(j+1)))*P_Lj(j+1,i)*...
                PJ(j+1,n)*Chris(i+1);
        end
    end
end

%% Other component functions
[tt,xx] = meshgrid(t,x);

a_n = -tt.*sin(2*xx);

b_n = (tt.^3).*xx.^3;

q_n = sin(2*xx).*tt.*exp(-tt).*(8.*xx - 12.*xx.^2 + 4.*xx.^3) -...
        tt.^3.*xx.^3.*exp(-tt).*((24.*xx.^(4-nu))/gamma(5-nu) - ...
       (24.*xx.^(3-nu))/gamma(4-nu) + (8.*xx.^(2-nu))/gamma(3-nu) );

% initial conditions
f = (x.^2).*(2-x).^2;
u = zeros(N+1,length(t));
u(:,1) = f;

%% explicit RK4: u_dot = a*D1*u + b*Dnu*u - u + q

for time = 1:time_frac:length(t)-1
    k_1 = dt*(a_n(2:end-1,time).*sum(D1.*repmat(u(2:end-1,time),1,N-1)',2)...
        + b_n(2:end-1,time).*sum(Dnu.*repmat(u(2:end-1,time),1,N-1)',2)...
        - u(2:end-1,time) + q_n(2:end-1,time));

    k_2 = dt*(a_n(2:end-1,time+1).*sum(D1.*repmat(u(2:end-1,time)+...
        (k_1/2),1,N-1)',2) + b_n(2:end-1,time+1).*sum(Dnu.*...
        repmat(u(2:end-1,time)+(k_1/2),1,N-1)',2) -...
        (u(2:end-1,time)+(k_1/2)) + q_n(2:end-1,time+1));
    k_3 = dt*(a_n(2:end-1,time+1).*sum(D1.*repmat(u(2:end-1,time)+...
        (k_2/2),1,N-1)',2) + b_n(2:end-1,time+1).*sum(Dnu.*...
        repmat(u(2:end-1,time)+(k_2/2),1,N-1)',2) -...
        (u(2:end-1,time)+(k_2/2)) + q_n(2:end-1,time+1));

    k_4 = dt*(a_n(2:end-1,time+2).*sum(D1.*repmat(u(2:end-1,time)+...
        (k_3),1,N-1)',2) + b_n(2:end-1,time+2).*...
        sum(Dnu.*repmat(u(2:end-1,time)+(k_3),1,N-1)',2) -...
        (u(2:end-1,time)+(k_3)) + q_n(2:end-1,time+2));

        u(2:end-1,time+time_frac) = u(2:end-1,time) +...
            (1/6)*(k_1 + 2*k_2 + 2*k_3 + k_4);
end

u(:,time_frac:time_frac:length(t))=[];

%% Exact solution and error
u_exact = exp(-tt).*(xx.^2).*(2-xx).^2;
u_exact(:,time_frac:time_frac:length(t))=[];
tt(:,time_frac:time_frac:length(t))=[];
xx(:,time_frac:time_frac:length(t))=[];
error = abs(u-u_exact);
max_error(N-1) = max(max(error));

%% Plots
if N == 10

% Contour Plot of the approximate and exact solution (side by side)
figure(1);
subplot(2,2,1)
contourf(tt,xx,u,10), xlabel('t'), ylabel('x'),title('Approximate');
subplot(2,2,2)
contourf(tt,xx,u_exact,10), xlabel('t'), ylabel('x'), title('Exact');
% print('C4_Ex1_contour','-depsc2','-r600');

% Surface plot of the solution u(x,t)
figure(2)
surf(tt,xx,u), colormap winter;
xlabel('t'),ylabel('x'),zlabel('u(x,t)')
print('C4_Ex1_surface','-depsc2','-r600');
```

```
% Surface plot of error
figure(3);
surf(tt,xx,error), colormap hot;
xlabel('t'), ylabel('x'), zlabel('Error');
print('C4_Ex1_error','-depsc2','-r600');

display(max_error)

end

end
```

Appendix B

Legendre-Gauss-Lobatto Nodes and Weights Function

```
function [x,Chris] = LegGauLob(N)

syms x
L = legendreP(N,x);
L_1 = diff(L,x,1);

x = vpasolve(L_1 == 0);
x = [-1;x;1];
Chris = 2./(N*(N+1)*(subs(L)).^2);

end
```

Appendix C

Π_ν Function

```
function [ result ] = Pinu( i,l, nu, L )

k = ceil(nu):i;

result = sum( ( (-1).^(i+k)*(2*l + 1).*factorial(i+k).*(gamma(k-nu+1)./...
    gamma(k-l-nu+1)) ) ./ ( (L^nu).*factorial(i-k).*factorial(k).*...
    gamma(k - nu + 1).*(gamma(k-nu+2+l)./gamma(k-nu+1)) ));

end
```

Appendix D

Chapter 5 Example MATLAB Code

```
% Chapter 5 Example 1: u_t = g1*u_(x^nu1) + g2*u_(y^nu2) + f
% f = exp(-t).*(x.^2.*(-y.^(3/2)+y-4).*y^(3/2) + sqrt(x).*(2x-3).*y.^3)
% g1 = ((3-2*x).*gamma(3-nu1))./2
% g2 = ((4-y).*gamma(4-nu2))./6
%
% IC: u(x,y,0) = x^2 .* y.^3
%
% Boundary Conditions:
% u(0,y) = 0,  u(1,y) = exp(-t).*y.^3
% u(x,0) = 0,  u(x,1) = exp(-t).*x.^2
%
% exact solution: exp(-t).*(x.^2).*(2-x).^2;

% set up workspace
clear all
clc
set(0,'DefaultFigureWindowStyle', 'docked')
format long

% initialize vector of maximum errors for table
max_error = zeros(9,1);

%% Loop through varying numbers of collocation points
for N=2:10

% set constants
M = N; % for simplicity
nu1 = 1.5;
nu2 = nu1; % for simplicity
```

```
R1 = 1;
R2 = R1; % for simplicity
T = 1;
dt = 0.001; time_frac = 2; dt_frac = dt/time_frac;
t = 0:dt_frac:T;

%% other calculated values
h_L1j = R1./(2*(0:N)+1);
h_L2j = R1./(2*(0:M)+1);

% obtain x and y nodes and Christoffel numbers
[X,CHRISX] = LegGauLob(N);
% [Y,CHRISY] = LegGauLob(M); % not needed if M = N
% shift from [-1,1] to [0,L];
x = (R1/2)*(double(X) + 1); Chrisx = (R1/2)*double(CHRISX);
y = x; Chrisy = Chrisx;

%% polynomials and matrices
% shifted Legendre polynomials
P_Lj = zeros(N+1,N+1);
for j = 0:N
    syms z
    P_L = legendreP(j,z);
    z = X;
    P_Lj(j+1,:) = subs(P_L);
end

Dnu1 = zeros(N+1);
for i = 0:N
    for n = 0:N
        Dnu1(i+1,n+1) = Dnu(i,x(n+1),nu1,R1);
    end
end

Dnu2 = Dnu1;

d_nu1 = zeros(N+1,N+1,N+1,N+1);
d_nu2 = d_nu1;

for n = 0:N
    for m = 0:M
```

```
        for l = 0:N
            for k = 0:M
                d_nu1(l+1,k+1,n+1,m+1) =...
                    dnu1fun( N,n,m,l,k,h_L1j,P_Lj,Chrisx,Dnu1 );
                d_nu2(l+1,k+1,n+1,m+1) =...
                    dnu2fun( N,n,m,l,k,h_L1j,P_Lj,Chrisx,Dnu2 );
            end
        end
    end
end

%% other component functions
[yy,xx] = meshgrid(y,x);

a_nm = ((3-2*xx).*gamma(3-nu1))./2;

b_nm = ((4-yy).*gamma(4-nu2))./6;

q_nm = zeros(N+1,M+1,length(t));

u = zeros(N+1,M+1,length(t));

% Initial condition
u(:,:,1) = xx.^2 .* yy.^3;

% boundary conditions and function f
for i = 1:length(t)
    u(end,:,i) = exp(-t(i)).*y.^3;
    u(:,end,i) = exp(-t(i)).*x.^2;
    q_nm(:,:,i) = exp(-t(i)).*(xx.^2.*(-(yy.^(3/2))+yy-4).*yy.^(3/2) +...
        (xx.^0.5).*(2*xx-3).*yy.^3);
end

%% explicit RK: v_dot = g1 sum sum rho1 v + g2 sum sum rho2 v + f

for time = 1:time_frac:length(t)-1

    k_1 = zeros(N+1);
    k_2 = zeros(N+1);
    k_3 = zeros(N+1);
    k_4 = zeros(N+1);
    for n = 1:N+1
        for m = 1:N+1
            k_1(n,m) = dt*( a_nm(n,m)*sum(sum(d_nu1(:,:,n,m).*u(:,:,time)))...
                + b_nm(n,m)*sum(sum(d_nu2(:,:,n,m).*u(:,:,time))) +...
                q_nm(n,m,time) );
        end
    end
    for n = 1:N+1
        for m = 1:N+1
            k_2(n,m) = dt*( a_nm(n,m)*sum(sum(d_nu1(:,:,n,m).*(u(:,:,time)...
                + k_1/2))) + b_nm(n,m)*sum(sum(d_nu2(:,:,n,m).*(u(:,:,time)...
                + k_1/2) )) + q_nm(n,m,time+1) );
        end
    end
    for n = 1:N+1
        for m = 1:N+1
            k_3(n,m) = dt*( a_nm(n,m)*sum(sum(d_nu1(:,:,n,m).*(u(:,:,time)...
                + k_2/2))) + b_nm(n,m)*sum(sum(d_nu2(:,:,n,m).*(u(:,:,time)...
                + k_2/2) )) + q_nm(n,m,time+1) );
        end
    end
    for n = 1:N+1
        for m = 1:N+1
            k_4(n,m) = dt*( a_nm(n,m)*sum(sum(d_nu1(:,:,n,m).*(u(:,:,time)...
                + k_3))) + b_nm(n,m)*sum(sum(d_nu2(:,:,n,m).*(u(:,:,time)...
                + k_3) )) + q_nm(n,m,time+2) );
        end
    end

        u(2:end-1,2:end-1,time+time_frac) = u(2:end-1,2:end-1,time) +...
        (1/6)*(k_1(2:end-1,2:end-1) + 2*k_2(2:end-1,2:end-1) +...
        2*k_3(2:end-1,2:end-1) + k_4(2:end-1,2:end-1));

end

u(:,:,time_frac:time_frac:length(t))=[];

%% Exact solution and error

u_exact = zeros(N+1,M+1,length(t));
```

```
for i = 1:length(t)
    u_exact(:,:,i) = exp(-t(i)).*xx.^2.*yy.^3;
end
u_exact(:,:,time_frac:time_frac:length(t))=[];
t(time_frac:time_frac:length(t))=[];
% error = abs(u(:,:,end) - u_exact(:,:,end));
error = abs(u(:,:,:) - u_exact(:,:,:));
max_error(N-1) = max(max(max(error)));

%% Plots
if N == 10

% Contour Plot of the solution (side by side)
figure(1);
subplot(2,2,1)
contourf(xx,yy,u(:,:,end)',10), xlabel('x'), ylabel('y'),title('Approximate');
subplot(2,2,2)
contourf(xx,yy,u_exact(:,:,end)',10), xlabel('x'), ylabel('y'), title('Exact');
print('C5_Ex1_contour','-depsc2','-r600');

% Surface plot of the solution u(x,y,t)
figure(2)
surf(yy,xx,u(:,:,end)), colormap cool;
xlabel('y')
ylabel('x')
zlabel('u(x,y,1)')
print('C5_Ex1_surface','-depsc2','-r600');

% Surface plot of error
figure(3)
surf(yy,xx,error(:,:,end)), colormap hot;
xlabel('y')
ylabel('x')
zlabel('Error')
print('C5_Ex1_error','-depsc2','-r600');

display(max_error)

end

end
```

Appendix E

$\mathrm{D}^{\nu}L_{R,i}(x)$ Function

```
function [ result ] = Dnu( i,x, nu, R )

if x == 0
    result = 0;
return
end

k = 0:i;

result = sum( ( (-1).^(i+k).*factorial(i+k).*(x.^(k-nu) ) )./...
    ( (R.^k).*factorial(i-k).*(factorial(k)).*gamma(k - nu + 1) ));

end
```

Appendix F

${}_{n,m}\mathcal{D}^{\nu_1}_{k,l}$ Function

```
function [ result ] = dnu1fun( N,n,m,l,k,h,P_L,Chris,Dmat )

Chris_k = Chris(k+1);
Chris_l = Chris(l+1);
P_Ljk = P_L(:,k+1);
P_Lil = P_L(:,l+1);
P_Ljm = P_L(:,m+1);
D = Dmat(:,n+1);

result = 0;
for i = 0:N
    result = result + sum( (P_Ljk*Chris_k*P_Lil(i+1)*Chris_l*...
        D(i+1).*P_Ljm) ./( h(i+1)*h' ) );
end

end
```

Appendix G

${}_{n,m}\mathcal{D}^{\nu_2}_{k,l}$ Function

```
function [ result ] = dnu2fun( N,n,m,l,k,h,P_L,Chris,Dmat )

Chris_k = Chris(k+1);
Chris_l = Chris(l+1);
P_Ljk = P_L(:,k+1);
P_Lil = P_L(:,l+1);
P_Lin = P_L(:,n+1);
D = Dmat(:,m+1);

result = 0;
for j = 0:N
    result = result + sum( (P_Ljk(j+1)*Chris_k*P_Lil*Chris_l*...
        D(j+1).*P_Lin) ./( h(j+1)*h' ) );
end

end
```

Appendix H

Chapter 6 $\Delta^{\nu}(n, j)$ function

```
function [ result ] = Deltafun( n,j, alp, beta, nu, T, h )

l = 0:j;

result = 0;

for k = ceil(nu):n
 result = result + (( (-1).^(n-k)*T^(1-nu+alp+beta)*gamma(n + beta + 1)*gamma(j +
     beta + 1)*gamma(n+k+alp+beta+1) )./...
        ( h*factorial(n-k)*gamma(j+alp+beta+1)*gamma(n+alp+beta+1)*gamma(k+beta+1)
            *gamma(k-nu+1)  ))*...
        sum( ( (-1).^(j-l).*gamma(j+l+alp+beta+1).*gamma(1+alp).*gamma(l+k+beta-nu
            +1))./...
        (factorial(l).*factorial(j-l).*gamma(l+beta+1).*gamma(l+k+alp+beta-nu+2) )
             );
end

end
```

Appendix I

Chapter 6 Example 1 MATLAB Code

```matlab
% Chapter 6 Example 1: u_(t^nu) = u_xx + exp(x)*(gamma(2+nu)*t - t^(1+nu))
% 0 <= x <= 1, 0 < t <= 1
%
% IC: u(x,0) = 0
%
% Boundary Conditions:
% u(0,t) = t^(1+nu),  u(1,t) = exp(1)*t^(1+nu)
%
% exact solution: exp(x)*t^(1+nu)

clear all
clc

% initialize global variables
global N M P_LN P_TM D_tnu D_x2 u f
set(0,'DefaultFigureWindowStyle', 'docked')  % figures docked
format long

% initialize vector of maximum errors for table
Max_N = 14;
max_error = zeros(Max_N-5,1);

%% Loop through varying numbers of collocation points

for N = 6:Max_N

% set constants
M = N;
nu = 0.75;
```

```
T = 1;
L = 1;
alp = 0;
beta = 0;

%% Jacobi polynomials

% in x
syms x
j=N-1;
k=0:j;
P_LNmin1 = sum( (  (-1).^(j-k).*gamma(j+beta+1).*gamma(j+k+beta+alp+1).*x.^k ) ./
    ...
        ( gamma(k+beta+1).*gamma(j+alp+beta+1).*factorial(j-k).*factorial(k).*L.^k
             ) );
x = double(vpasolve(P_LNmin1 == 0));
x = [0;x;L];
P_LN = zeros(N+1,N+1);

for j=0:N
    syms z
    k=0:j;
    P_LNx = sum( (  (-1).^(j-k).*gamma(j+beta+1).*gamma(j+k+beta+alp+1).*z.^k ) ./
        ...
        ( gamma(k+beta+1).*gamma(j+alp+beta+1).*factorial(j-k).*factorial(k).*L.^k
             ) );
    z = x;
    P_LN(j+1,:) = subs(P_LNx);
end

% in t
syms t
j=M+1;
k=0:j;
P_TMpls1 = sum( (  (-1).^(j-k).*gamma(j+beta+1).*gamma(j+k+beta+alp+1).*t.^k ) ./
    ...
        ( gamma(k+beta+1).*gamma(j+alp+beta+1).*factorial(j-k).*factorial(k).*T.^k
             ) );
t = double(vpasolve(P_TMpls1 == 0));

P_TM = zeros(M+1,M+1);
```

```
for j=0:M
    syms z
    k=0:j;
    P_TMt = sum( (  (-1).^(j-k).*gamma(j+beta+1).*gamma(j+k+beta+alp+1).*z.^k ) ./
        ...
        ( gamma(k+beta+1).*gamma(j+alp+beta+1).*factorial(j-k).*factorial(k).*T.^k
             ) );
    z = t;
    P_TM(j+1,:) = subs(P_TMt);
end

%% Differentiation Matrixes

% h terms
i=0:N;%
h_abLi = ((L)^(alp+beta+1).*gamma(i+alp+1).*gamma(i+beta+1) )./...
   ( (2*i+alp+beta+1).*factorial(i).*gamma(i+alp+beta+1) ) ;%

i=0:M;
h_abTi = ((T)^(alp+beta+1).*gamma(i+alp+1).*gamma(i+beta+1) )./...
   ( (2*i+alp+beta+1).*factorial(i).*gamma(i+alp+beta+1) ) ;%

% initialize matrixes
D_tnu = zeros(M+1,M+1);
D_x1 = zeros(N+1,N+1);
D_x2 = zeros(N+1,N+1);

% matrixes for time derivatives
for i = 0:M
    for j = 0:M
        D_tnu(i+1,j+1) = Deltafun( i,j, alp, beta, nu, T, h_abTi(j+1) );
    end
end

% matrixes for space derivatives
for i = 0:N
    for j = 0:N
        D_x2(i+1,j+1) = Deltafun( i,j, alp, beta, 2, L, h_abLi(j+1) );
    end
end
```

```
%% Set Up Data Structures

[xx,tt] = meshgrid(x,t);

u_exact = exp(xx).*tt.^(1+nu);

% define initial and boundary conditions
u = u_exact; u(2:end,2:end-1) = 0;

% define forcing function
f = exp(xx).*(gamma(2+nu).*tt - tt.^(1+nu));

%% Solve Equations

% initial guess of zeros
U0 = zeros(M+1,N+1);

% solver options
opts = optimoptions(@fsolve,'TolFun',1e-14,'TolX',1e-14,'Algorithm',...
    'levenberg-marquardt','Display','iter-detailed');

% solve the system
U = fsolve(@equations,U0,opts);

% assign solved values to u(x,t)
for m = 2:M+1
    for n = 2:N
        u(m,n) = P_TM(:,m)'*U*P_LN(:,n);
    end
end

% calculate error
error = abs(u-u_exact);
% max_error(N-1) = max(max(error));
max_error(N-1) = max(error(end,:));

%% Plots

if N == Max_N
```

```
% Contour Plot of the approximate and exact solution (side by side)
figure(1);
subplot(2,2,1)
contourf(tt,xx,u,10), xlabel('t'), ylabel('x'),title('Approximate');
subplot(2,2,2)
contourf(tt,xx,u_exact,10), xlabel('t'), ylabel('x'), title('Exact');
print('C6_Ex1_contour','-depsc2','-r600');

% Surface plot of the solution u(x,t)
figure(2)
surf(tt,xx,u), colormap pink;
xlabel('t'), ylabel('x'), zlabel('u(x,t)')
print('C6_Ex1_surface','-depsc2','-r600');

% Surface plot of error
figure(3);
surf(tt,xx,error), colormap hot;
xlabel('t'), ylabel('x'), zlabel('Error');
view(142.5,30)
print('C6_Ex1_error','-depsc2','-r600');

display(max_error);
end

end
```

Appendix J

Chapter 6 Example 1 Equation Function

```
function [ F ] = equations( U )

global N M P_LN P_TM D_tnu D_x2 u f

F = zeros(M+1,N+1);

for m = 2:M+1
    for n = 2:N
        F(m,n) = P_TM(:,m)'*D_tnu'*U*P_LN(:,n)...
            - (P_TM(:,m)'*U*D_x2*P_LN(:,n))...
            - f(m,n);
    end
end

% initial condition
for n = 2:N
    F(1,n) = P_TM(:,1)'*U*P_LN(:,n) - u(1,n);
end

% boundary conditions
for m = 1:M+1
    F(m,1) = P_TM(:,m)'*U*P_LN(:,1) - u(m,1);
    F(m,N+1) = P_TM(:,m)'*U*P_LN(:,N+1) - u(m,N+1);
end

end
```

Appendix K

Chapter 6 Example 2 MATLAB Code

```
% Chapter 6 Example 2: u_t = d^(1-nu)/dt^(1-nu)( u_xx - u) + q
% 0 <= x <= 1, 0 < t <= 1
% q = u^3 + cos(pi*x)*(2*t + (pi^2+1)*(2*t^(1+nu))/gamma(2+nu)
%     - t^6*cos(pi*x)^2 );
%
% IC: u(x,0) = 0
%
% Boundary Conditions:
% u(0,t) = t^2,  u(1,t) = -t^2
%
% exact solution: t^2*cos(pi*x);

clear all
clc

% initialize global variables
global N M P_LN P_TM D_tnu D_x2 D_t1 f u
set(0,'DefaultFigureWindowStyle', 'docked')  % figures docked
format long

% initialize vector of maximum errors for table
Max_N = 10;
max_error = zeros(Max_N-1,1);

%% Loop through varying numbers of collocation points

for N = 2:Max_N

% set constants
```

```
M = N;
nu = 0.35;
T = 1;
L = 1;
alp = 0.1;
beta = 0.1;

%% Jacobi polynomials

% in x
syms x
j=N-1;
k=0:j;
P_LNmin1 = sum( (  (-1).^(j-k).*gamma(j+beta+1).*gamma(j+k+beta+alp+1).*x.^k ) ./
    ...
        ( gamma(k+beta+1).*gamma(j+alp+beta+1).*factorial(j-k).*factorial(k).*L.^k
             ) );
x = double(vpasolve(P_LNmin1 == 0));
x = [0;x;L];

P_LN = zeros(N+1,N+1);
for j=0:N
    syms z
    k=0:j;
    P_LNx = sum( (  (-1).^(j-k).*gamma(j+beta+1).*gamma(j+k+beta+alp+1).*z.^k ) ./
        ...
        ( gamma(k+beta+1).*gamma(j+alp+beta+1).*factorial(j-k).*factorial(k).*L.^k
             ) );
    z = x;
    P_LN(j+1,:) = subs(P_LNx);
end

% in t
syms t
j=M+1;
k=0:j;
P_TMpls1 = sum( (  (-1).^(j-k).*gamma(j+beta+1).*gamma(j+k+beta+alp+1).*t.^k ) ./
    ...
        ( gamma(k+beta+1).*gamma(j+alp+beta+1).*factorial(j-k).*factorial(k).*T.^k
             ) );
t = double(vpasolve(P_TMpls1 == 0));
```

```

P_TM = zeros(M+1,M+1);
for j=0:M
    syms z
    k=0:j;
    P_TMt = sum( (  (-1).^(j-k).*gamma(j+beta+1).*gamma(j+k+beta+alp+1).*z.^k ) ./
        ...
        ( gamma(k+beta+1).*gamma(j+alp+beta+1).*factorial(j-k).*factorial(k).*T.^k
             ) );
    z = t;
    P_TM(j+1,:) = subs(P_TMt);
end

%% Differentiation matrices

% h terms
i=0:N;%
h_abLi = ((L)^(alp+beta+1).*gamma(i+alp+1).*gamma(i+beta+1) )./...
   ( (2*i+alp+beta+1).*factorial(i).*gamma(i+alp+beta+1) ) ;%

i=0:M;
h_abTi = ((T)^(alp+beta+1).*gamma(i+alp+1).*gamma(i+beta+1) )./...
   ( (2*i+alp+beta+1).*factorial(i).*gamma(i+alp+beta+1) ) ;%

% initialize matrices
D_tnu = zeros(M+1,M+1);
D_t1 = D_tnu;
D_x2 = zeros(N+1,N+1);

% matrices for time derivatives
for i = 0:M
    for j = 0:M
        D_tnu(i+1,j+1) = Deltafun( i,j, alp, beta, 1-nu, T, h_abTi(j+1) );
        D_t1(i+1,j+1) = Deltafun( i,j, alp, beta, 1, T, h_abTi(j+1) );
    end
end

% matrices for space derivatives
for i = 0:N
    for j = 0:N
        D_x2(i+1,j+1) = Deltafun( i,j, alp, beta, 2, L, h_abLi(j+1) );
    end
end


%% set up data structures

[xx,tt] = meshgrid(x,t);

u_exact = tt.^2.*cos(pi.*xx);

% define initial and boundary conditions
u = u_exact; u(2:end,2:end-1) = 0;

% define forcing function
f = cos(pi.*xx).*(2*tt + (pi^2+1).*(2.*tt.^(1+nu))/gamma(2+nu) -...
    tt.^6.*cos(pi.*xx).^2 );

%% Solve Equations

% initial guess of zeros
U0 = zeros(M+1,N+1);

% solver options
opts = optimoptions(@fsolve,'TolFun',1e-14,'TolX',1e-14,'Algorithm',...
    'levenberg-marquardt','Display','iter-detailed');

% solve the system
U = fsolve(@equations,U0,opts);

% assign solved values to u(x,t)
for m = 2:M+1
    for n = 2:N
        u(m,n) = P_TM(:,m)'*U*P_LN(:,n);
    end
end

% calculate error
error = abs(u-u_exact);
max_error(N-1) = max(max(error));

%% Plots
```

```matlab

if N == Max_N

% contour plot of the approximate and exact solution (side by side)
figure(1);
subplot(2,2,1)
contour(tt,xx,u,10), xlabel('t'), ylabel('x'),title('Approximate');
subplot(2,2,2)
contour(tt,xx,u_exact,10), xlabel('t'), ylabel('x'), title('Exact');
print('C6_Ex2_contour','-depsc2','-r600');

% surface plot of the solution u(x,t)
figure(2)
surf(tt,xx,u), colormap spring;
xlabel('Time (t) \rightarrow')
ylabel('{\leftarrow} Spatial co-ordinate (x)')
zlabel('Solution profile (u(x,t)) \rightarrow')
view(142.5,30)
print('C6_Ex2_surface','-depsc2','-r600');

% surface plot of error
figure(3);
surf(tt,xx,error), colormap hot;
xlabel('t'), ylabel('x'), zlabel('Error');
view(142.5,30)
print('C6_Ex2_error','-depsc2','-r600');

end

end
```

Appendix L

Chapter 6 Example 2 Equation Function

```
function [ F ] = equations( U )

global N M P_LN P_TM D_tnu D_x2 D_t1 f u

F = zeros(M+1,N+1);

for m = 2:M+1
    for n = 2:N
        F(m,n) = P_TM(:,m)'*(D_t1'*U - D_tnu'*U*D_x2 + D_tnu'*U)*P_LN(:,n)...
            - 0*P_TM(:,1)'*U*P_LN(:,n) + 0*u(1,n)...
            - (P_TM(:,m)'*U*P_LN(:,n))^3 - f(m,n);

    end
end

% initial condition
for n = 2:N
    F(1,n) = P_TM(:,1)'*U*P_LN(:,n) - u(1,n);
end

% boundary conditions
for m = 1:M+1
    F(m,1) = P_TM(:,m)'*U*P_LN(:,1) - u(m,1);
    F(m,N+1) = P_TM(:,m)'*U*P_LN(:,N+1) - u(m,N+1);
end

end
```

Appendix M

Chapter 6 Example 3 MATLAB Code

```
% Jaradat 2013:
% 0 <= x <= 1, 0 < t <= 1
%
% IC:
%
% Boundary Conditions:
%
%
% exact solution:

clear all
clc

% initialize global variables
global N M xx P_RN P_TM D_xnu D_t1 D_t2 q u
set(0,'DefaultFigureWindowStyle', 'docked')  % figures docked
format long

% initialize vector of maximum errors for table
Max_N = 16;
max_error = zeros(Max_N-1,1);

%% Loop through varying numbers of collocation points

for N = Max_N:Max_N

% set constants
M = N;
nu = 1.5;
```

```
T = 0.4;
R = 1;
alp = 0;
beta = 0;

%% Jacobi polynomials

% in x
syms x
j=N-1;
k=0:j;
P_RNmin1 = sum( (  (-1).^(j-k).*gamma(j+beta+1).*gamma(j+k+beta+alp+1).*x.^k ) ./
    ...
        ( gamma(k+beta+1).*gamma(j+alp+beta+1).*factorial(j-k).*factorial(k).*R.^k
             ) );
x = double(vpasolve(P_RNmin1 == 0));
x = [0;x;R];

P_RN = zeros(N+1,N+1);
for j=0:N
    syms z
    k=0:j;
    P_RNx = sum( (  (-1).^(j-k).*gamma(j+beta+1).*gamma(j+k+beta+alp+1).*z.^k ) ./
        ...
        ( gamma(k+beta+1).*gamma(j+alp+beta+1).*factorial(j-k).*factorial(k).*R.^k
             ) );
    z = x;
    P_RN(j+1,:) = subs(P_RNx);
end

% in t
syms t
j=M+1;
k=0:j;
P_TMpls1 = sum( (  (-1).^(j-k).*gamma(j+beta+1).*gamma(j+k+beta+alp+1).*t.^k ) ./
    ...
        ( gamma(k+beta+1).*gamma(j+alp+beta+1).*factorial(j-k).*factorial(k).*T.^k
             ) );
t = double(vpasolve(P_TMpls1 == 0));

P_TM = zeros(M+1,M+1);
```

```
for j=0:M
    syms z
    k=0:j;
    P_TMt = sum( (  (-1).^(j-k).*gamma(j+beta+1).*gamma(j+k+beta+alp+1).*z.^k ) ./
        ...
        ( gamma(k+beta+1).*gamma(j+alp+beta+1).*factorial(j-k).*factorial(k).*T.^k
             ) );
    z = t;
    P_TM(j+1,:) = subs(P_TMt);
end

%% differentiation matrices


% h terms
i=0:N;%
h_abLi = ((R)^(alp+beta+1).*gamma(i+alp+1).*gamma(i+beta+1) )./...
   ( (2*i+alp+beta+1).*factorial(i).*gamma(i+alp+beta+1) ) ;%

i=0:M;
h_abTi = ((T)^(alp+beta+1).*gamma(i+alp+1).*gamma(i+beta+1) )./...
   ( (2*i+alp+beta+1).*factorial(i).*gamma(i+alp+beta+1) ) ;%

% initialize matrices
D_tnu = zeros(M+1,M+1);
D_x1 = zeros(N+1,N+1);
D_x2 = zeros(N+1,N+1);

% matrices for time derivatives
for i = 0:M
    for j = 0:M
        D_t1(i+1,j+1) = Deltafun( i,j, alp, beta, 1, T, h_abTi(j+1) );
        D_t2(i+1,j+1) = Deltafun( i,j, alp, beta, 2, T, h_abTi(j+1) );
    end
end

% matrices for space derivatives
for i = 0:N
    for j = 0:N
        D_xnu(i+1,j+1) = Deltafun( i,j, alp, beta, nu, R, h_abLi(j+1) );
    end
end
```

```

%% set up data structures

[xx,tt] = meshgrid(x,t);

u_exact = xx.^2.*(xx-2).*tt.^2;

% define initial and boundary conditions
u = u_exact; u(2:end,2:end-1) = 0;

q = -4*xx.^2 + 2*xx.^3 - 2.546*xx.^2.*tt.^2 + 2.546*xx.*tt.^2;

%% solve equations

% initial guess of zeros
U0 = zeros(M+1,N+1);

% solver options
opts = optimoptions(@fsolve,'TolFun',1e-16,'TolX',1e-16,'Algorithm',...
    'levenberg-marquardt','Display','iter-detailed',...
    'MaxFunEvals',100000,'MaxIter',4000);

% solve the system
U = fsolve(@equations,U0,opts);

% assign solved values to u(x,t)
for m = 2:M+1
    for n = 2:N
        u(m,n) = P_TM(:,m)'*U*P_RN(:,n);
    end
end

% calculate error
error = abs(u-u_exact);
max_error(N-1) = max(max(error));

%% Plots

if N == Max_N
```

```
% contour plot of the approximate and exact solution (side by side)
figure(1);
subplot(2,2,1)
contour(tt,xx,u,10), xlabel('t'), ylabel('x'),title('Approximate');
subplot(2,2,2)
contour(tt,xx,u_exact,10), xlabel('t'), ylabel('x'), title('Exact');
print('C6_Ex3_contour','-depsc2','-r600');

% surface plot of the solution u(x,t)
figure(2)
surf(tt,xx,u), colormap summer;
xlabel('t')
ylabel('x')
zlabel('u(x,t)')
view(142.5,30)
print('C6_Ex3_surface','-depsc2','-r600');

% Surface plot of error
figure(3);
surf(tt,xx,error), colormap hot;
xlabel('t'), ylabel('x'), zlabel('Error');
print('C6_Ex3_error','-depsc2','-r600');

display(max_error);
end

end
```

Appendix N

Chapter 6 Example 2 Equation Function

```
function [ F ] = equations( U )

global N M xx P_RN P_TM D_xnu D_t1 D_t2 q u

F = zeros(M+2,N+1);

for m = 2:M+1
    for n = 2:N
        F(m,n) = P_TM(:,m)'*D_t2'*U*P_RN(:,n)...
            - 1/gamma(0.5)*xx(m,n)^0.5*(P_TM(:,m)'*U*D_xnu*P_RN(:,n)) ...
            - q(m,n);
    end
end

% boundary conditions in t
for n = 1:N+1
    % Dirichlet
    F(1,n) = P_TM(:,1)'*U*P_RN(:,n) - u(1,n);
    % Derivative
    F(m+2,n) = P_TM(:,1)'*D_t1'*U*P_RN(:,n);
end

% boundary conditions in x
for m = 1:M+1
    F(m,1) = P_TM(:,m)'*U*P_RN(:,1) - u(m,1);
    F(m,N+1) = P_TM(:,m)'*U*P_RN(:,N+1) - u(m,N+1);
end

end
```

Appendix O

Chapter 7 Example 1 MATLAB Code

```
% Chapter 7 Example 1: u_t = d(x,y)u_x^1.6 + e(x,y)*u_y^1.8 + q(x,y,t)
% 0 <= x <= 1, 0 <= y <= 1, 0 < t <= 1
% q = -(1+2*x*y)*exp(-t)*x^3*y^3.6;
% d = (gamma(2.2)*x^(2.8)*y)/6;
% e = (2*x*y^(2.6))/(gamma(4.6));
%
% exact solution: u(x,y,t) = e^(-t)* x^(3)* y^(3.6)

clear all
clc

% initialize global variables
global N1 N2 M P_RN1 P_RN2 P_TM D_xnu1 D_ynu2 D_t1 q d e u
set(0,'DefaultFigureWindowStyle', 'docked')
format long

% initialize vector of maximum errors for table
Max_N = 10;
max_error = zeros(Max_N-5,1);

%% Loop through varying numbers of collocation points

for N1 = 2:Max_N

% set constants
N2 = N1;
M = N1;
nu1 = 1.8;
nu2 = 1.6;
```

```
T = 1;
R1 = 2;
R2 = R1;
alp = 0;
beta = 0;

%% Jacobi polynomials

% in x
syms x
j=N1-1;
k=0:j;
P_RN1min1 = sum( (  (-1).^(j-k).*gamma(j+beta+1).*gamma(j+k+beta+alp+1).*x.^k ) ./
    ...
        ( gamma(k+beta+1).*gamma(j+alp+beta+1).*factorial(j-k).*factorial(k).*R1.^
            k ) );
x = double(vpasolve(P_RN1min1 == 0));
x = [0;x;R1];
y = x;
P_RN1 = zeros(N1+1,N1+1);
for j=0:N1
    syms z
    k=0:j;
    P_LNx = sum( (  (-1).^(j-k).*gamma(j+beta+1).*gamma(j+k+beta+alp+1).*z.^k ) ./
        ...
        ( gamma(k+beta+1).*gamma(j+alp+beta+1).*factorial(j-k).*factorial(k).*R1.^
            k ) );
    z = x;
    P_RN1(j+1,:) = subs(P_LNx);
end
P_RN2 = P_RN1;

% in t
syms t
j=M+1;
k=0:j;
P_TMpls1 = sum( (  (-1).^(j-k).*gamma(j+beta+1).*gamma(j+k+beta+alp+1).*t.^k ) ./
    ...
        ( gamma(k+beta+1).*gamma(j+alp+beta+1).*factorial(j-k).*factorial(k).*T.^k
             ) );
t = double(vpasolve(P_TMpls1 == 0));
```

```
P_TM = zeros(M+1,M+1);
for j=0:M
    syms z
    k=0:j;
    P_TMt = sum( (  (-1).^(j-k).*gamma(j+beta+1).*gamma(j+k+beta+alp+1).*z.^k ) ./
        ...
        ( gamma(k+beta+1).*gamma(j+alp+beta+1).*factorial(j-k).*factorial(k).*T.^k
             ) );
    z = t;
    P_TM(j+1,:) = subs(P_TMt);
end

%% Differentiation matrices

% h terms
i=0:N1;
h_abR1i = ((R1)^(alp+beta+1).*gamma(i+alp+1).*gamma(i+beta+1) )./...
   ( (2*i+alp+beta+1).*factorial(i).*gamma(i+alp+beta+1) ) ;%

i=0:M;
h_abTi = ((T)^(alp+beta+1).*gamma(i+alp+1).*gamma(i+beta+1) )./...
   ( (2*i+alp+beta+1).*factorial(i).*gamma(i+alp+beta+1) ) ;%

% initialize matrices
D_t1 = zeros(M+1,M+1);
D_xnu1 = zeros(N1+1,N1+1);
D_ynu2 = zeros(N2+1,N2+1);

% matrices for time derivatives
for i = 0:M
    for j = 0:M
        D_t1(i+1,j+1) = Deltafun( i,j, alp, beta, 1, T, h_abTi(j+1) );
    end
end

% matrices for space derivatives
for i = 0:N1
    for j = 0:N1
        D_xnu1(i+1,j+1) = Deltafun( i,j, alp, beta, nu1, R1, h_abR1i(j+1) );
        D_ynu2(i+1,j+1) = Deltafun( i,j, alp, beta, nu2, R1, h_abR1i(j+1) );
    end
end

%% Set up data structures
[xx,yy,tt] = meshgrid(x,y,t);

u_exact = exp(-tt).*xx.^3.*yy.^3.6;

% define initial and boundary conditions
u = u_exact; u(2:end-1,2:end-1,2:end) = 0;

% define forcing function
q = -(1+2*xx.*yy).*exp(-tt).*xx.^3.*yy.^3.6;

% define other functions
d = (gamma(2.2)*xx(:,:,1).^(2.8).*yy(:,:,1))./6;
e = (2*xx(:,:,1).*yy(:,:,1).^(2.6))./(gamma(4.6));

%% Solve equations

% initial guess of zeros
U0 = zeros(M+1,(N1+1)*(N2+1));

% solver options
opts = optimoptions(@fsolve,'TolFun',1e-14,'TolX',1e-14,'Algorithm','levenberg-
    marquardt','Display','iter-detailed');

% solve the system
U = fsolve(@equations,U0,opts);

% assign solved values to u(x,y,t)
for m = 2:M+1
    for n1 = 2:N1
        for n2 = 2:N2
            u(n2,n1,m) = P_TM(:,m)'*U*kron(P_RN1(:,n1),P_RN1(:,n2));
        end
    end
end

% calculate error
error = abs(u(:,:,end)-u_exact(:,:,end));
```

```
max_error(N1-1) = max(max(error));

%% Plots
if N1 == Max_N

% contour plot of the approximate and exact solution (side by side)
figure(1);
subplot(2,2,1)
contourf(yy(:,:,end),xx(:,:,end),u(:,:,end),12), xlabel('y'), ylabel('x'),title('
    Approximate');
subplot(2,2,2)
subplot(2,2,2)
contourf(yy(:,:,end),xx(:,:,end),u_exact(:,:,end),12), xlabel('y'), ylabel('x'),
    title('Exact');
print('C7_Ex1_contour','-depsc2','-r600');

% surface plot of the solution u(x,t)
figure(2)
surf(yy(:,:,end),xx(:,:,end),u(:,:,end)), colormap bone;
xlabel('y')
ylabel('x')
zlabel('u(x,y,1)')
print('C7_Ex1_surface','-depsc2','-r600');

% surface plot of error
figure(3);
surf(yy(:,:,end),xx(:,:,end),error(:,:,end)), colormap hot;
xlabel('y')
ylabel('x')
zlabel('Error')
print('C7_Ex1_error','-depsc2','-r600');

display(max_error);

end

end
```

Appendix P

Chapter 7 Example 1 Equation Function

```
function [ F ] = equations( U )

global  N1 N2 M P_RN1 P_RN2 P_TM D_xnu1 D_ynu2 D_t1 q d e u

F = zeros(M+1,(N1+1)*(N2+1));

for m = 2:M+1
    for n1 = 1:N1+1
        for n2 = 1:N2+1
        F(m,(N2+1)*(n1-1)+(n2)) = P_TM(:,m)'*(D_t1'*U)*kron(P_RN1(:,n1),P_RN2(:,n2
            ))...
            - d(n2,n1)*P_TM(:,m)'*U*kron(D_xnu1*P_RN1(:,n1),P_RN2(:,n2))...
            - e(n2,n1)*P_TM(:,m)'*U*kron(P_RN1(:,n1),D_ynu2*P_RN2(:,n2))...
            - q(n2,n1,m);
        end
    end
end

% initial condition
for n1 = 2:N1
    for n2 = 2:N2
        F(1,(N2+1)*(n1-1)+(n2)) = P_TM(:,1)'*U*kron(P_RN1(:,n1),P_RN2(:,n2)) - u(
            n2,n1,1);
    end
end

% boundary conditions
for m = 1:M+1
    for n2 = 1:N2+1
```

```
            F(m,n2) = P_TM(:,m)'*U*kron(P_RN1(:,1),P_RN2(:,n2)) - u(n2,1,m);
            F(m,(N1+1)*N2+n2) = P_TM(:,m)'*U*kron(P_RN1(:,end),P_RN2(:,n2)) - u(n2,end
                ,m);
        end
        for n1=2:N1
            F(m,(N1+1)*(n1-1)+1) = P_TM(:,m)'*U*kron(P_RN1(:,n1),P_RN2(:,1)) - u(1,n1,
                m);
            F(m,(N1+1)*(n1)) = P_TM(:,m)'*U*kron(P_RN1(:,n1),P_RN2(:,end)) - u(end,n1,
                m);
        end
    end

    end
```

Appendix Q

Chapter 7 Example 2 MATLAB Code

```
% Chapter 7 Example 2: u_(t^nu) = u_xx + exp(x)*(gamma(2+nu)*t - t^(1+nu))
% 0 <= x <= 1, 0 < t <= 1
% f = sin(x)*sin(y)*( t^2*gamma(nu+3)/2 + 2*t^(nu+2) );
%
% IC: u(x,0) = 0
%
% Boundary Conditions:
% u(0,t) = 0,  u(1,t) = exp(1)*t^(1+nu)
%
% exact solution: sin(x)*sin(y)*t^(nu+2);

clear all
clc

% initialize global variables
global N1 N2 M P_RN1 P_RN2 P_TM D_tnu D_x2 D_y2 f u
set(0,'DefaultFigureWindowStyle', 'docked')
format long

% initialize vector of maximum errors for table
Max_N = 10;
max_error = zeros(Max_N-5,1);

%% Loop through varying numbers of collocation points

for N1 = 2:Max_N

% set constants
N2 = N1;
```

```
M = 2;
nu = 1.95;
T = 1;
R1 = 1;
R2 = R1;
alp = 0;
beta = 0;

%% Jacobi polynomials

% in x
syms x
j=N1-1;
k=0:j;
P_RN1min1 = sum( (  (-1).^(j-k).*gamma(j+beta+1).*gamma(j+k+beta+alp+1).*x.^k ) ./
    ...
        ( gamma(k+beta+1).*gamma(j+alp+beta+1).*factorial(j-k).*factorial(k).*R1.^
            k ) );
x = double(vpasolve(P_RN1min1 == 0));
x = [0;x;R1];
y = x;
P_RN1 = zeros(N1+1,N1+1);
for j=0:N1
    syms z
    k=0:j;
    P_LNx = sum( (  (-1).^(j-k).*gamma(j+beta+1).*gamma(j+k+beta+alp+1).*z.^k ) ./
        ...
        ( gamma(k+beta+1).*gamma(j+alp+beta+1).*factorial(j-k).*factorial(k).*R1.^
            k ) );
    z = x;
    P_RN1(j+1,:) = subs(P_LNx);
end
P_RN2 = P_RN1;

% in t
T_adj = 1.127*T; % adjust so end node is at T
syms t
j=M+1;
k=0:j;
P_TMpls1 = sum( (  (-1).^(j-k).*gamma(j+beta+1).*gamma(j+k+beta+alp+1).*t.^k ) ./
    ...
```

```
        ( gamma(k+beta+1).*gamma(j+alp+beta+1).*factorial(j-k).*factorial(k).*
            T_adj.^k ) );
t = double(vpasolve(P_TMpls1 == 0));
t(1) = 0;

P_TM = zeros(M+1,M+1);
for j=0:M
    syms z
    k=0:j;
    P_TMt = sum( (  (-1).^(j-k).*gamma(j+beta+1).*gamma(j+k+beta+alp+1).*z.^k ) ./
        ...
        ( gamma(k+beta+1).*gamma(j+alp+beta+1).*factorial(j-k).*factorial(k).*
            T_adj.^k ) );
    z = t;
    P_TM(j+1,:) = subs(P_TMt);
end

%% Differentiation matrices

% h terms
i=0:N1;
h_abR1i = ((R1)^(alp+beta+1).*gamma(i+alp+1).*gamma(i+beta+1) )./...
   ( (2*i+alp+beta+1).*factorial(i).*gamma(i+alp+beta+1) ) ;%

i=0:M;
h_abTi = ((T_adj)^(alp+beta+1).*gamma(i+alp+1).*gamma(i+beta+1) )./...
   ( (2*i+alp+beta+1).*factorial(i).*gamma(i+alp+beta+1) ) ;%

% initialize matrices
D_tnu = zeros(M+1,M+1);
D_t1 = D_tnu;
D_x2 = zeros(N1+1,N1+1);

% matrices for time derivatives
for i = 0:M
    for j = 0:M
        D_tnu(i+1,j+1) = Deltafun( i,j, alp, beta, nu, T_adj, h_abTi(j+1) );
    end
end

% matrices for space derivatives
```

```
for i = 0:N1
    for j = 0:N1
        D_x2(i+1,j+1) = Deltafun( i,j, alp, beta, 2, R1, h_abR1i(j+1) );
    end
end
D_y2 = D_x2;

%% Set Up Data Structures

[xx,yy,tt] = meshgrid(x,y,t);

u_exact = tt.^(2).*sin(xx+yy);

% define initial and boundary conditions
u = u_exact; u(2:end-1,2:end-1,2:end) = 0;

% define forcing function
f = (2*tt.^(2-nu)/gamma(3-nu) +2*tt.^2 ).*sin(xx+yy);

%% Solve Equations

% initial guess of zeros
U0 = zeros(M+1,(N1+1)*(N2+1));

% solver options
opts = optimoptions(@fsolve,'TolFun',1e-14,'TolX',1e-14,'Algorithm',...
    'levenberg-marquardt','Display','iter-detailed');

% solve the system
U = fsolve(@equations,U0,opts);

% assign solved values to u(x,y,t)
for m = 2:M+1
    for n1 = 2:N1
        for n2 = 2:N2
            u(n2,n1,m) = P_TM(:,m)'*U*kron(P_RN1(:,n1),P_RN1(:,n2));
        end
    end
end

% calculate error
```

```
error = abs(u(:,:,end)-u_exact(:,:,end));
max_error(N1-1) = max(max(error));

%% Plots

if N1 == Max_N

% contour plot of the approximate and exact solution (side by side)
figure(1);
subplot(2,2,1)
contourf(yy(:,:,end),xx(:,:,end),u(:,:,end),20), xlabel('y'), ylabel('x'),title('
    Approximate');
subplot(2,2,2)
contourf(yy(:,:,end),xx(:,:,end),u_exact(:,:,end),20), xlabel('y'), ylabel('x'),
    title('Exact');
print('C7_Ex2_contour','-depsc2','-r600');

% surface plot of the solution u(x,t)
figure(2)
surf(yy(:,:,end),xx(:,:,end),u(:,:,end)), colormap bone;
xlabel('y')
ylabel('x')
zlabel('u(x,y,1)')
print('C7_Ex2_surface','-depsc2','-r600');

% surface plot of error
figure(3);
surf(yy(:,:,end),xx(:,:,end),error(:,:,end)), colormap hot;
xlabel('y')
ylabel('x')
zlabel('Error')
print('C7_Ex2_error','-depsc2','-r600');

display(max_error);

end

end
```

Appendix R

Chapter 7 Example 2 Equation Function

```
function [ F ] = equations( U )

global  N1 N2 M P_RN1 P_RN2 P_TM D_tnu D_x2 D_y2 f u

F = zeros(M+1,(N1+1)*(N2+1));

for m = 2:M+1
    for n1 = 1:N1+1
        for n2 = 1:N2+1
        F(m,(N2+1)*(n1-1)+(n2)) = P_TM(:,m)'*(D_tnu'*U)*kron(P_RN1(:,n1),P_RN2(:,
            n2))...
            - P_TM(:,m)'*(U)*kron(D_x2*P_RN1(:,n1),P_RN2(:,n2))...
            - P_TM(:,m)'*(U)*kron(P_RN1(:,n1),D_y2*P_RN2(:,n2))...
            - f(n2,n1,m);
        end
    end
end

% initial condition
for n1 = 2:N1
    for n2 = 2:N2
        F(1,(N2+1)*(n1-1)+(n2)) = P_TM(:,1)'*U*kron(P_RN1(:,n1),P_RN2(:,n2)) - u(
            n2,n1,1);
    end
end

% boundary conditions
for m = 1:M+1
    for n2 = 1:N2+1
```

```
        F(m,n2) = P_TM(:,m)'*U*kron(P_RN1(:,1),P_RN2(:,n2)) - u(n2,1,m);
        F(m,(N1+1)*N2+n2) = P_TM(:,m)'*U*kron(P_RN1(:,end),P_RN2(:,n2)) - u(n2,end
            ,m);
    end
    for n1=2:N1
        F(m,(N1+1)*(n1-1)+1) = P_TM(:,m)'*U*kron(P_RN1(:,n1),P_RN2(:,1)) - u(1,n1,
            m);
        F(m,(N1+1)*(n1)) = P_TM(:,m)'*U*kron(P_RN1(:,n1),P_RN2(:,end)) - u(end,n1,
            m);
    end
end

end
```

Appendix S

Chapter 7 Example 3 MATLAB Code

```
% Chapter 7 Example 3: u_(t^nu) = u_xx + u_yy + u^2 + u^3 + q
% 0 <= x,y <= 1, 0 < t <= 2
% q = (2*t^(2-nu)/gamma(3-nu))*(sech(x+y-r)^2 + sech(x+y+r)^2 )...
%      -4*t^2*(3*sech(x+y-r)^2*tanh(x+y-r)^2 - sech(x+y-r)^2 ...
%      + 3*sech(x+y+r)^2*tanh(x+y+r)^2 - sech(x+y+r)^2)...
%      - t^4*(sech(x+y-r)^2 + sech(x+y+r)^2 )^2 ...
%      - t^6*(sech(x+y-r)^2 + sech(x+y+r)^2 )^3;
%
% exact solution: t^2*(sech(x+y-r)^2 + sech(x+y+r)^2 );

clear all
clc

% initialize global variables
global  N1 N2 M P_RN1 P_RN2 P_TM D_tnu D_x2 D_y2 D_t1 q u
set(0,'DefaultFigureWindowStyle', 'docked')  % figures docked
format long

% initialize vector of maximum errors for table
Max_N = 8;
max_error = zeros(Max_N-1,1);

%% Loop through varying numbers of collocation points

for N1 = 7:Max_N

% set constants
N2 = N1;
M = 2*N1;
```

```
nu = 1.95;
T = 2;
R1 = 1;
R2 = R1;
alp = 0;
beta = 0;
r = 0;

%% Jacobi polynomials

% in x
syms x
j=N1-1;
k=0:j;
P_RN1min1 = sum( (  (-1).^(j-k).*gamma(j+beta+1).*gamma(j+k+beta+alp+1).*x.^k ) ./
    ...
        ( gamma(k+beta+1).*gamma(j+alp+beta+1).*factorial(j-k).*factorial(k).*R1.^
            k ) );
x = double(vpasolve(P_RN1min1 == 0));
x = [0;x;R1];
y = x;
P_RN1 = zeros(N1+1,N1+1);
for j=0:N1
    syms z
    k=0:j;
    P_LNx = sum( (  (-1).^(j-k).*gamma(j+beta+1).*gamma(j+k+beta+alp+1).*z.^k ) ./
        ...
        ( gamma(k+beta+1).*gamma(j+alp+beta+1).*factorial(j-k).*factorial(k).*R1.^
            k ) );
    z = x;
    P_RN1(j+1,:) = subs(P_LNx);
end
P_RN2 = P_RN1;

% in t
syms t
j=M+1;
k=0:j;
P_TMpls1 = sum( (  (-1).^(j-k).*gamma(j+beta+1).*gamma(j+k+beta+alp+1).*t.^k ) ./
    ...
        ( gamma(k+beta+1).*gamma(j+alp+beta+1).*factorial(j-k).*factorial(k).*T.^k
                ) );
t = double(vpasolve(P_TMpls1 == 0));

P_TM = zeros(M+1,M+1);
for j=0:M
    syms z
    k=0:j;
    P_TMt = sum( (  (-1).^(j-k).*gamma(j+beta+1).*gamma(j+k+beta+alp+1).*z.^k ) ./
        ...
        ( gamma(k+beta+1).*gamma(j+alp+beta+1).*factorial(j-k).*factorial(k).*T.^k
             ) );
    z = t;
    P_TM(j+1,:) = subs(P_TMt);
end

%% Differentiation matrices

% h terms
i=0:N1;%
h_abR1i = ((R1)^(alp+beta+1).*gamma(i+alp+1).*gamma(i+beta+1) )./...
   ( (2*i+alp+beta+1).*factorial(i).*gamma(i+alp+beta+1) ) ;%

i=0:M;
h_abTi = ((T)^(alp+beta+1).*gamma(i+alp+1).*gamma(i+beta+1) )./...
   ( (2*i+alp+beta+1).*factorial(i).*gamma(i+alp+beta+1) ) ;%

% initialize matrices
D_tnu = zeros(M+1,M+1);
D_t1 = D_tnu;
D_x2 = zeros(N1+1,N1+1);

% matrices for time derivatives
for i = 0:M
    for j = 0:M
        D_t1(i+1,j+1) = Deltafun( i,j, alp, beta, 1, T, h_abTi(j+1) );
        D_tnu(i+1,j+1) = Deltafun( i,j, alp, beta, nu, T, h_abTi(j+1) );
    end
end

% matrices for space derivatives
for i = 0:N1
```

```
    for j = 0:N1
        D_x2(i+1,j+1) = Deltafun( i,j, alp, beta, 2, R1, h_abR1i(j+1) );
    end
end
D_y2 = D_x2;

%% Set up data structures
[xx,yy,tt] = meshgrid(x,y,t);

u_exact = tt.^2.*(sech(xx+yy-r).^2 + sech(xx+yy+r).^2 );

% define initial and boundary conditions
u = u_exact; u(2:end-1,2:end-1,2:end) = 0;

% define forcing function
q = (2*tt.^(2-nu)/gamma(3-nu)).*(sech(xx+yy-r).^2 + sech(xx+yy+r).^2 )...
    -4*tt.^2.*(3*sech(xx+yy-r).^2.*tanh(xx+yy-r).^2 - sech(xx+yy-r).^2 ...
    + 3*sech(xx+yy+r).^2.*tanh(xx+yy+r).^2 - sech(xx+yy+r).^2)...
    - tt.^4.*(sech(xx+yy-r).^2 + sech(xx+yy+r).^2 ).^2 ...
    - tt.^6.*(sech(xx+yy-r).^2 + sech(xx+yy+r).^2 ).^3;

%% Solve equations

U0 = zeros(M+1,(N1+1)*(N2+1));

% solver options
opts = optimoptions(@fsolve,'TolFun',1e-16,'TolX',1e-16,'Algorithm','levenberg-
    marquardt','Display','iter-detailed',...
    'MaxFunEvals',1000000,'MaxIter',20000);
% solve the system
U = fsolve(@equations,U0,opts);

% assign solved values to u(x,y,t)
for m = 2:M+1
    for n1 = 2:N1
        for n2 = 2:N2
            u(n2,n1,m) = P_TM(:,m)'*U*kron(P_RN1(:,n1),P_RN1(:,n2));
        end
    end
end
```

```matlab
% calculate error
error = abs(u(:,:,end)-u_exact(:,:,end));
max_error(N1-1) = max(max(error));

%% Plots

if N1 == Max_N

% contour plot of the approximate and exact solution (side by side)
figure(1);
subplot(2,2,1)
contourf(yy(:,:,end),xx(:,:,end),u(:,:,end),20), xlabel('y'), ylabel('x'),title('
    Approximate');
subplot(2,2,2)
contourf(yy(:,:,end),xx(:,:,end),u_exact(:,:,end),20), xlabel('y'), ylabel('x'),
    title('Exact');
print('C7_Ex3_contour','-depsc2','-r600');

% surface plot of the solution u(x,t)
figure(2)
surf(yy(:,:,end),xx(:,:,end),u(:,:,end)), colormap bone;
xlabel('y')
ylabel('x')
zlabel('u(x,y,1)')
view([-37.5,50])
print('C7_Ex3_surface','-depsc2','-r600');

% surface plot of error
figure(3);
surf(yy(:,:,end),xx(:,:,end),error(:,:,end)), colormap hot;
xlabel('y')
ylabel('x')
zlabel('Error')
view([142.5,30])
print('C7_Ex3_error','-depsc2','-r600');

display(max_error);

end

end
```

Appendix T

Chapter 7 Example 3 Equation Function

```
function [ F ] = equations( U )

global  N1 N2 M P_RN1 P_RN2 P_TM D_tnu D_x2 D_y2 D_t1 q u

F = zeros(M+2,(N1+1)*(N2+1));

for m = 2:M+1
    for n1 = 1:N1+1
        for n2 = 1:N2+1
        F(m,(N2+1)*(n1-1)+(n2)) = P_TM(:,m)'*(D_tnu'*U)*kron(P_RN1(:,n1),P_RN2(:,
            n2))...
            - P_TM(:,m)'*(U)*kron(D_x2*P_RN1(:,n1),P_RN2(:,n2))...
            - P_TM(:,m)'*(U)*kron(P_RN1(:,n1),D_y2*P_RN2(:,n2))...
            - (P_TM(:,m)'*(U)*kron(P_RN1(:,n1),P_RN2(:,n2)))^2 ...
            - (P_TM(:,m)'*(U)*kron(P_RN1(:,n1),P_RN2(:,n2)))^3 ...
            - q(n2,n1,m);
        end
    end
end

% initial condition
for n1 = 1:N1+1
    for n2 = 1:N2+1
        F(1,(N2+1)*(n1-1)+(n2)) = P_TM(:,1)'*U*kron(P_RN1(:,n1),P_RN2(:,n2)) - u(
            n2,n1,1);
        F(M+2,(N2+1)*(n1-1)+(n2)) = P_TM(:,1)'*D_t1'*U*kron(P_RN1(:,n1),P_RN2(:,n2
            ));
    end
end
```

```
% boundary conditions
for m = 1:M+1
    for n2 = 1:N2+1
        F(m,n2) = P_TM(:,m)'*U*kron(P_RN1(:,1),P_RN2(:,n2)) - u(n2,1,m);
        F(m,(N1+1)*N2+n2) = P_TM(:,m)'*U*kron(P_RN1(:,end),P_RN2(:,n2)) - u(n2,end
            ,m);
    end
    for n1=2:N1
        F(m,(N1+1)*(n1-1)+1) = P_TM(:,m)'*U*kron(P_RN1(:,n1),P_RN2(:,1)) - u(1,n1,
            m);
        F(m,(N1+1)*(n1)) = P_TM(:,m)'*U*kron(P_RN1(:,n1),P_RN2(:,end)) - u(end,n1,
            m);
    end
end

end
```

Appendix U

Chapter 7 Example 4 MATLAB Code

```
%% Fractional sub diffusion, example 3

clear all
clc

% initialize global variables
global N1 N2 M nu P_RN1 P_RN2 alp beta P_TM D_tnu D_x2 D_y2 D_t1 fu fv u v
set(0,'DefaultFigureWindowStyle', 'docked')
format long

% initialize vector of maximum errors for table
Max_N = 10;
max_error = zeros(Max_N-1,2);

%% Loop through varying numbers of collocation points

for N1 = 2:Max_N

% set constants
N2 = N1;
M = 1;
nu = 0.2;
T = 1;
R1 = 1;
R2 = R1;
alp = 0;
beta = 0;

%% Jacobi polynomials
```

```

% in x
syms x
j=N1-1;
k=0:j;
P_RN1min1 = sum( (  (-1).^(j-k).*gamma(j+beta+1).*gamma(j+k+beta+alp+1).*x.^k ) ./
    ...
        ( gamma(k+beta+1).*gamma(j+alp+beta+1).*factorial(j-k).*factorial(k).*R1.^
            k ) );
x = double(vpasolve(P_RN1min1 == 0));
x = [0;x;R1];
y = x;
P_RN1 = zeros(N1+1,N1+1); %\Phi_{L,N}(x)
for j=0:N1
    syms z
    k=0:j;
    P_LNx = sum( (  (-1).^(j-k).*gamma(j+beta+1).*gamma(j+k+beta+alp+1).*z.^k ) ./
        ...
        ( gamma(k+beta+1).*gamma(j+alp+beta+1).*factorial(j-k).*factorial(k).*R1.^
            k ) );
    z = x;
    P_RN1(j+1,:) = subs(P_LNx);
end
P_RN2 = P_RN1;

% in t
t = [0;T];

P_TM = zeros(M+1,M+1); %\Phi_{T,M}(t)
for j=0:M
    syms z
    k=0:j;
    P_TMt = sum( (  (-1).^(j-k).*gamma(j+beta+1).*gamma(j+k+beta+alp+1).*z.^k ) ./
        ...
        ( gamma(k+beta+1).*gamma(j+alp+beta+1).*factorial(j-k).*factorial(k).*T.^k
             ) );
    z = t;
    P_TM(j+1,:) = subs(P_TMt);
end

%% differentiation matrices
```

```
% h thing
i=0:N1;%
h_abR1i = ((R1)^(alp+beta+1).*gamma(i+alp+1).*gamma(i+beta+1) )./...
   ( (2*i+alp+beta+1).*factorial(i).*gamma(i+alp+beta+1) ) ;%
% h_abR2i = h_abR1i;

i=0:M;
h_abTi = ((T)^(alp+beta+1).*gamma(i+alp+1).*gamma(i+beta+1) )./...
   ( (2*i+alp+beta+1).*factorial(i).*gamma(i+alp+beta+1) ) ;%

D_tnu = zeros(M+1,M+1);
D_t1 = D_tnu;
D_x2 = zeros(N1+1,N1+1);
% D_y2 = zeros(N2+1,N2+1);

for i = 0:M
    for j = 0:M
        D_tnu(i+1,j+1) = Deltafun( i,j, alp, beta, 1-nu, T, h_abTi(j+1) );
        D_t1(i+1,j+1) = Deltafun( i,j, alp, beta, 1, T, h_abTi(j+1) );
    end
end

for i = 0:N1
    for j = 0:N1
        D_x2(i+1,j+1) = Deltafun( i,j, alp, beta, 2, R1, h_abR1i(j+1) );
    end
end
D_y2 = D_x2;

%% set up data structures
[xx,yy,tt] = meshgrid(x,y,t);

u_exact = tt.^2.*sin(xx).*sin(yy);
v_exact = tt.^2.*cos(xx).*cos(yy);

u = u_exact; u(2:end-1,2:end-1,2:end) = 0;
v = v_exact; v(2:end-1,2:end-1,2:end) = 0;

fu = (tt.^2).*( (1/2)*sin(xx).*sin(yy).*((tt.^4).*cos(2*xx).*cos(2*yy) + (tt.^4) -
     4)...
    -( (2*tt.^(-nu)).*cos(xx).*cos(yy) )./(gamma(3-nu)) );
fv = (tt.^2).*( (1/2)*cos(xx).*cos(yy).*((tt.^4).*cos(2*xx).*cos(2*yy) + (tt.^4) -
     4)...
    +( (2*tt.^(-nu)).*sin(xx).*sin(yy) )./(gamma(3-nu)) );

%% solve equations

A0 = zeros((M+1)*2,(N1+1)*(N2+1));

% options
opts = optimoptions(@fsolve,'TolFun',1e-14,'TolX',1e-14,'Algorithm','levenberg-
    marquardt','Display','iter-detailed');
% solve
A = fsolve(@equations,A0,opts);
U = A(1:M+1,:);
V = A(M+2:end,:);

for m = 2:M+1
    for n1 = 2:N1
        for n2 = 2:N2
            u(n2,n1,m) = P_TM(:,m)'*U*kron(P_RN1(:,n1),P_RN1(:,n2));
            v(n2,n1,m) = P_TM(:,m)'*V*kron(P_RN1(:,n1),P_RN1(:,n2));
        end
    end
end

% for m=1:M+1
%     subplot(1,2,1)
%     surf(u(:,:,m))
%     axis([-inf inf -inf inf min(min(min(min(u_exact))),-0.01) max(max(max(
    u_exact)))*1.1])
%     subplot(1,2,2)
%     surf(v(:,:,m))
%     axis([-inf inf -inf inf min(min(min(min(v_exact))),0) max(max(max(v_exact)))
    *1.1])
%     pause(0.2)
% end


error_u = abs(u-u_exact);
error_v = abs(v-v_exact);
```

```
max_error(N1-1,:) = [max(max(error_u(:,:,end))) max(max(error_v(:,:,end)))];

%% Plots

if N1 == Max_N

% contour plot of the approximate and exact solution (side by side)
figure(1);
subplot(2,2,1)
contourf(yy(:,:,end),xx(:,:,end),u(:,:,end),20), xlabel('y'), ylabel('x'),title('U
     Approximate');
subplot(2,2,2)
contourf(yy(:,:,end),xx(:,:,end),u_exact(:,:,end),20), xlabel('y'), ylabel('x'),
    title('U Exact');
subplot(2,2,3)
contourf(yy(:,:,end),xx(:,:,end),v(:,:,end),20), xlabel('y'), ylabel('x'),title('V
     Approximate');
subplot(2,2,4)
contourf(yy(:,:,end),xx(:,:,end),v_exact(:,:,end),20), xlabel('y'), ylabel('x'),
    title('V Exact');
print('C7_Ex4_contour','-depsc2','-r600');

% surface plot of the solution u(x,t)
figure(2)
subplot(1,2,1)
surf(yy(:,:,end),xx(:,:,end),u(:,:,end)), colormap cool;
xlabel('y')
ylabel('x')
zlabel('u(x,y,1)')
subplot(1,2,2)
surf(yy(:,:,end),xx(:,:,end),v(:,:,end));
xlabel('y')
ylabel('x')
zlabel('v(x,y,1)')
view([142.5,30])
print('C7_Ex4_surface','-depsc2','-r600');

% surface plot of error
figure(3);
subplot(1,2,1)
surf(yy(:,:,end),xx(:,:,end),error_u(:,:,end)), colormap hot;
```

```
xlabel('y')
ylabel('x')
zlabel('U Error')
subplot(1,2,2)
surf(yy(:,:,end),xx(:,:,end),error_v(:,:,end));
xlabel('y')
ylabel('x')
zlabel('V Error')
print('C7_Ex4_error','-depsc2','-r600');

display(max_error);


end

end
```

Appendix V

Chapter 7 Example 4 Equation Function

```
function [ F ] = equations( A )

global  N1 N2 M nu T P_RN1 P_RN2 alp beta P_TM D_tnu D_x2 D_y2 D_t1 fu fv u v

U = A(1:M+1,:);
V = A(M+2:end,:);

F = zeros((M+1)*2,(N1+1)*(N2+1));

% for u
for m = 2:M+1
    for n1 = 1:N1+1
        for n2 = 1:N2+1
        F(m,(N2+1)*(n1-1)+(n2)) = P_TM(:,m)'*(D_tnu'*U)*kron(P_RN1(:,n1),P_RN2(:,
            n2))...
            + P_TM(:,m)'*(V)*kron(D_x2*P_RN1(:,n1),P_RN2(:,n2))...
            + P_TM(:,m)'*(V)*kron(P_RN1(:,n1),D_y2*P_RN2(:,n2))...
            + ( (P_TM(:,m)'*(U)*kron(P_RN1(:,n1),P_RN2(:,n2)))^2 ...
            + (P_TM(:,m)'*(V)*kron(P_RN1(:,n1),P_RN2(:,n2)))^2)...
            *(P_TM(:,m)'*(V)*kron(P_RN1(:,n1),P_RN2(:,n2)))...
            - fv(n2,n1,m);
        end
    end
end

% initial condition
for n1 = 2:N1
    for n2 = 2:N2
        F(1,(N2+1)*(n1-1)+(n2)) = P_TM(:,1)'*U*kron(P_RN1(:,n1),P_RN2(:,n2)) - u(
            n2,n1,1);
    end
end

% boundary conditions
for m = 1:M+1
    for n2 = 1:N2+1
        F(m,n2) = P_TM(:,m)'*U*kron(P_RN1(:,1),P_RN2(:,n2)) - u(n2,1,m);
        F(m,(N1+1)*N2+n2) = P_TM(:,m)'*U*kron(P_RN1(:,end),P_RN2(:,n2)) - u(n2,end
            ,m);
    end
    for n1=2:N1
        F(m,(N1+1)*(n1-1)+1) = P_TM(:,m)'*U*kron(P_RN1(:,n1),P_RN2(:,1)) - u(1,n1,
            m);
        F(m,(N1+1)*(n1)) = P_TM(:,m)'*U*kron(P_RN1(:,n1),P_RN2(:,end)) - u(end,n1,
            m);
    end
end

% for v
for m = 2:M+1
    for n1 = 1:N1+1
        for n2 = 1:N2+1
        F(m+M+1,(N2+1)*(n1-1)+(n2)) = P_TM(:,m)'*(D_tnu'*V)*kron(P_RN1(:,n1),P_RN2
            (:,n2))...
            - P_TM(:,m)'*(U)*kron(D_x2*P_RN1(:,n1),P_RN2(:,n2))...
            - P_TM(:,m)'*(U)*kron(P_RN1(:,n1),D_y2*P_RN2(:,n2))...
            - ( (P_TM(:,m)'*(U)*kron(P_RN1(:,n1),P_RN2(:,n2)))^2 ...
            + (P_TM(:,m)'*(V)*kron(P_RN1(:,n1),P_RN2(:,n2)))^2)...
            *(P_TM(:,m)'*(U)*kron(P_RN1(:,n1),P_RN2(:,n2)))...
            + fu(n2,n1,m);
        end
    end
end

% initial condition
for n1 = 2:N1
    for n2 = 2:N2
        F(M+2,(N2+1)*(n1-1)+(n2)) = P_TM(:,1)'*V*kron(P_RN1(:,n1),P_RN2(:,n2)) - v
            (n2,n1,1);
    end
```

```
end

% boundary conditions
for m = 1:M+1
    for n2 = 1:N2+1
        F(m+M+1,n2) = P_TM(:,m)'*V*kron(P_RN1(:,1),P_RN2(:,n2)) - v(n2,1,m);
        F(m+M+1,(N1+1)*N2+n2) = P_TM(:,m)'*V*kron(P_RN1(:,end),P_RN2(:,n2)) - v(n2
            ,end,m);
    end
    for n1=2:N1
        F(m+M+1,(N1+1)*(n1-1)+1) = P_TM(:,m)'*V*kron(P_RN1(:,n1),P_RN2(:,1)) - v
            (1,n1,m);
        F(m+M+1,(N1+1)*(n1)) = P_TM(:,m)'*V*kron(P_RN1(:,n1),P_RN2(:,end)) - v(end
            ,n1,m);
    end
end



end
```