



AFRL-RH-FS-TR-2021-0005

**Survey of Numerical Methods for Fractional
Differential Equations for Directed Energy
Applications**

**Victor M. Chen
Oak Ridge Institute for Science and Education**

**Andrew W. Wharmby
711th Human Performance Wing
Airman Systems Directorate
Bioeffects Division
Optical Radiation Bioeffects Branch**

18 May 2021

Interim Report - 1 Jun 2020 to 31 May 2021

Distribution Statement A: Approved for public release; distribution is unlimited. PA Case #502ABW-2021-0026. The opinions expressed on this document, electronic or otherwise, are solely those of the author(s). They do not represent an endorsement by or the views of the United States Air Force, the Department of Defense, or the United States Government.

**Air Force Research Laboratory
711th Human Performance Wing
Airman Systems Directorate
Bioeffects Division
Optical Radiation Bioeffects Branch
JBSA Fort Sam Houston, Texas 78234**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report was cleared for public release by the 502nd ABW Public Affairs Office and is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (<http://www.dtic.mil>).

Survey of Numerical Methods for Fractional Differential Equations for Directed Energy Applications

(AFRL-RH-FS-TR- 2021 - 0005) has been reviewed and is approved for publication in accordance with assigned distribution statement.

ROCKWELL.BENJA
MIN.A.1231305358

Digitally signed by
ROCKWELL.BENJAMIN.A.1231305358
Date: 2021.06.01 12:58:34 -05'00'

BENJAMIN A. ROCKWELL, PhD
Chief, Optical Radiation Bioeffects Branch

RILEY.LARA.LEILAN
I.1039605704

Digitally signed by
RILEY.LARA.LEILANI.1039605704
Date: 2021.07.20 08:45:52 -05'00'

LARA L. RILEY, LtCol, USAF, BSC
Deputy Chief, Bioeffects Division
Airman Systems Directorate
711th Human Performance Wing
Air Force Research Laboratory

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute an official position of the U.S. Government.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. **PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.**

1. REPORT DATE (DD-MM-YYYY) 18/05/2021		2. REPORT TYPE Interim Report		3. DATES COVERED (From — To) 1 Jun 2020 to 31 May 2021	
4. TITLE AND SUBTITLE Survey of Numerical Methods for Fractional Differential Equations for Directed Energy Applications				5a. CONTRACT NUMBER	
				5b. GRANT NUMBER	
				5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S) Victor M. Chen, Andrew W. Wharmby				5d. PROJECT NUMBER	
				5e. TASK NUMBER	
				5f. WORK UNIT NUMBER H10L	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Research Laboratory 711th Human Performance Wing Airman Systems Directorate Bioeffects Division Optical Radiation Bioeffects Branch				8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory 711th Human Performance Wing Airman Systems Directorate Bioeffects Division Optical Radiation Bioeffects Branch 4141 Petroleum Dr JB SA Fort Sam Houston TX				10. SPONSOR/MONITOR'S ACRONYM(S) 711 HPW/RHDO	
				11. SPONSOR/MONITOR'S REPORT NUMBER(S) AFRL-RH-FS-TR-2021-0005	
12. DISTRIBUTION / AVAILABILITY STATEMENT Distribution Statement A: Approved for public release; distribution is unlimited. PA Case #502ABW-2021-0026. The opinions expressed on this document, electronic or otherwise, are solely those of the author(s). They do not represent an endorsement by or the views of the United States Air Force, the Department of Defense, or the United States Government.					
13. SUPPLEMENTARY NOTES					
14. ABSTRACT Nonlocal operators of the fractional calculus have demonstrated the ability to model physical phenomena that have been shown to be problematic for models based on traditional integer-order calculus. Though these models based on the fractional calculus can produce highly accurate results, they come with the price of increased complexity and high computational cost. In some cases, the increased level of complexity is so high that numerical methods are sometimes necessary to implement even though an analytical solution for the problem may exist, making the importance of numerical methods for solving fractional differential equations more pronounced than perhaps even their integer order counterparts. In this survey, numerical methods popular with the fractional calculus community have been gathered, coded, and scrutinized for practical use in the multidisciplinary field of directed energy. Numerical methods stemming from basic definitions of fractional derivatives, methods based on predictor-corrector algorithms, and matrix methods in the context of fractional partial differential equations for viscoelastic, diffusion, and heating problems are considered.					
15. SUBJECT TERMS fractional calculus, fractional differential equations, numerical methods, directed energy					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Michael L. Denton
U	U	U	U	111	19b. TELEPHONE NUMBER (include area code) (210) 539-8069

This Page Intentionally Left Blank

TABLE OF CONTENTS

Section	Page
LIST OF FIGURES	iii
LIST OF TABLES	iii
LIST OF EQUATIONS	iv
ACKNOWLEDGEMENTS	vii
1.0 SUMMARY	1
2.0 INTRODUCTION	1
2.1 Grünwald-Letnikov Fractional Derivative and Integral	3
2.2 Riemann-Liouville Fractional Derivative and Integral	4
2.2.1 Equivalence of the Riemann-Liouville and Grünwald-Letnikov Fractional Derivatives	6
2.3 Caputo Fractional Derivative	6
3.0 NUMERICAL METHODS FOR SOLVING FRACTIONAL ORDINARY DIFFERENTIAL EQUATIONS	8
3.1 Grünwald-Letnikov Fractional Derivative Numerical Method	8
3.2 Diethelm Product-Trapezoidal Quadrature Coefficients	9
3.3 Matrix Method	12
3.3.1 Solving Fractional Ordinary Differential Equations	14
3.4 Applied Examples	16
3.4.1 Relaxation-Oscillation	16
3.4.2 Bagley-Torvik Equation	25
3.4.3 Fractional Lumped Capacitance Model	34
4.0 NUMERICAL METHODS FOR SOLVING FRACTIONAL PARTIAL DIFFERENTIAL EQUATIONS	40
4.1 Time-Fractional Diffusion Equations	40
4.2 Diethelm-Ford-Weilbeer Method for Time-Fractional Diffusion-Wave Equations with Dirichlet-Neumann Boundary Conditions	47
4.2.1 Definition of problem	47
4.2.2 Discretization of the problem	48
4.2.3 Complete discretization problem	50
4.2.4 Time-fractional sub-diffusion problem	50
4.2.5 Time-fractional diffusion-wave equation	54
5.0 FOUR-TERM FRACTIONAL WAVE EQUATION	58
5.1 Discretization of the Four-Term Fractional Wave Equation	58
5.2 Computational results	60
6.0 CONCLUSIONS	66
7.0 RECOMMENDATIONS	67
8.0 REFERENCES	68
APPENDIX A - SPECIAL FUNCTIONS	71
APPENDIX B - SOFTWARE CODE AND SPECIFICATIONS	73
B.1 Relaxation-Oscillation Equation	73
B.2 Bagley-Torvik Equation	78
B.3 Fractional Lumped Capacitance Model	83
B.4 Fractional Partial Differential Equations	86

B.5	Time-Fractional Diffusion-Wave Equation	89
B.6	Four-Term Fractional Wave Equation	93
	ATTACHMENT 1 - DATA FILES FOR FRACTIONAL LUMPED CAPACITANCE CODE	96
	LIST OF SYMBOLS, ABBREVIATIONS, AND ACRONYMS	101

LIST OF FIGURES

	Page
Figure 1	Relaxation-oscillation equation solution for varying α 17
Figure 2	Grünwald-Letnikov numerical solution and analytical solution for relaxation-oscillation for decreasing step sizes of h ($\alpha = 1.5$) 19
Figure 3	Matrix method numerical solution and analytical solution for relaxation-oscillation for decreasing step sizes of h ($\alpha = 1.5$) 22
Figure 4	Computation time comparison of Grünwald-Letnikov, matrix, and analytical solutions (relaxation-oscillation equation) 24
Figure 5	Computation error comparison of numerical methods (relaxation-oscillation equation) 24
Figure 6	Thin plate submerged in Newtonian fluid suspended by a spring 25
Figure 7	GL method numerical and analytical solutions for Bagley-Torvik for varying step sizes h 28
Figure 8	Matrix method numerical and analytical solutions for Bagley-Torvik for varying step sizes h 31
Figure 9	Computation time comparison of Grünwald-Letnikov, matrix, and analytical solutions (Bagley-Torvik Equation) 33
Figure 10	Computational error comparison of numerical methods (Bagley-Torvik equation) 33
Figure 11	Fractional lumped capacitance experimental configuration 34
Figure 12	Riemann-Liouville fractional integral of the Heaviside function for select orders of β 35
Figure 13	Experimental data fit using analytical solution to the fractional lumped capacitance model 39
Figure 14	Experimental data fit using numerical solution to the fractional lumped capacitance model 39
Figure 15	Time-fractional PDE example 1 45
Figure 16	Time-fractional PDE example 2 47
Figure 17	Time-fractional sub-diffusion ($0 < \alpha \leq 0.5$) 52
Figure 18	Time-fractional sub-diffusion ($0.5 < \alpha \leq 1$) 53
Figure 19	Time-fractional diffusion-wave ($1 < \alpha \leq 1.5$) 56
Figure 20	Time-fractional sub-diffusion ($1.5 < \alpha \leq 2$) 57
Figure 21	Ramped sine wave source 60
Figure 22	Numerical simulation of the four-term fractional wave equation with wave source at $x = 10$ and $\alpha = 0.25$ at various time steps 62
Figure 23	Numerical simulation of the four-term fractional wave equation with wave source at $x = 10$ and $\alpha = 0.5$ at various time steps 63
Figure 24	Numerical simulation of the four-term fractional wave equation with wave source at $x = 10$ and $\alpha = 0.75$ at various time steps 64
Figure 25	Numerical simulation of the four-term fractional wave equation with wave source at $x = 10$ and $\alpha = 1.0$ at various time steps 65

LIST OF TABLES

	Page
Table 1	Computation time results for the relaxation-oscillation equation 23
Table 2	Computation time results for the Bagley-Torvik equation 32
Table 3	Constants for analytical fit 38
Table 4	Constants for numerical fit 38
Table 1-1	Laser heating/cooling experimental data 96
Table 1-2	Pennes bioheat model fitting results 98

LIST OF EQUATIONS

	Page
Equation 1	n-fold derivative 3
Equation 2	Binomial coefficient in terms of gamma function 3
Equation 3	Derivative of arbitrary order α 4
Equation 4	Grünwald-Letnikov weights 4
Equation 5	Grünwald-Letnikov fractional integral 4
Equation 6	Grünwald-Letnikov fractional derivative 4
Equation 7	Cauchy formula 5
Equation 8	Riemann-Liouville fractional integral 5
Equation 9	Riemann-Liouville fractional derivative 5
Equation 10	Riemann-Liouville/Grünwald-Letnikov fractional derivative equivalency . . 6
Equation 11	Caputo fractional derivative 7
Equation 12	Basic fractional differential equation 8
Equation 13	Grünwald-Letnikov discretization scheme 8
Equation 14	Explicit scheme GL numerical solution for equation 9
Equation 15	Example GL numerical solution for Equation 12 9
Equation 16	Diethelm discretization scheme 9
Equation 17	Diethelm product-trapezoidal weights 10
Equation 18	Modified Diethelm product-trapezoidal weights 10
Equation 19	Explicit scheme numerical solution with Diethelm weights for Equation 12 11
Equation 20	Example GL numerical solution for equation 11
Equation 21	Lower triangular matrix with Grünwald-Letnikov weights matrix 12
Equation 22	Lower triangular matrix with Diethelm product-trapezoidal quadrature weights 12
Equation 23	Matrix method discretization 13
Equation 24	Backward difference first-order derivative 13
Equation 25	Backward difference second-order derivative 14
Equation 26	m -term linear FDE 14
Equation 27	Matrix method discretization of m -term linear FDE 15
Equation 28	Eliminator matrices 15
Equation 29	Matrix method eigenvalue problem for m -term linear FODE 16
Equation 30	Relaxation-oscillation problem 16
Equation 31	Relaxation-oscillation analytical solution 16

Equation 32	Discretization of relaxation-oscillation problem	18
Equation 33	Grünwald-Letnikov solution relaxation-oscillation	18
Equation 34	Matrix method scheme for the relaxation-oscillation equation	20
Equation 35	Matrix method relaxation-oscillation eigenvalue problem (expanded)	21
Equation 36	Mean squared error (MSE)	23
Equation 37	Bagley-Torvik equation	25
Equation 38	Analytical solution to BT equation	25
Equation 39	Fractional Green's function	26
Equation 40	k -th derivative of the two-parameter Mittag-Leffler function	26
Equation 41	Source term for the Bagley-Torvik Equation	26
Equation 42	Discretization of the Bagley-Torvik equation	26
Equation 43	Grünwald-Letnikov numerical solution to the Bagley - Torvik equation . . .	27
Equation 44	Matrix method BT	29
Equation 45	Matrix method BT (expanded)	30
Equation 46	RC circuit voltage response	34
Equation 47	Lumped capacitance model	34
Equation 48	Fractional lumped capacitance model	35
Equation 49	Riemann-Liouville fractional integral of the unit step function	35
Equation 50	Fractional lumped capacitance analytical solution	36
Equation 51	Fractional lumped capacitance model constants	36
Equation 52	Heating term	36
Equation 53	Cooling term	36
Equation 54	Grünwald-Letnikov numerical scheme for the fractional lumped capacitance equation	36
Equation 55	Explicit numerical scheme of the fractional lumped capacitance equation .	37
Equation 56	Fractional lumped capacitance source term	37
Equation 57	Numerical solution of the fractional lumped capacitance equation	37
Equation 58	Time-fractional diffusion equation	40
Equation 59	Grid points for the 1D finite difference solution	40
Equation 60	Relationship between RL and Caputo derivatives	41
Equation 61	Taylor polynomial of degree $n - 1$	41
Equation 62	Relationship between RL and Caputo derivatives with Taylor polynomial definition	41
Equation 63	Equation 62 after evaluating Riemann-Liouville derivative of the Taylor polynomial	41
Equation 64	B weights	42
Equation 65	Numerical for fractional diffusion equation FPDE	43
Equation 66	Numerical scheme time-fractional diffusion equation	43
Equation 67	Vector H_k	43
Equation 68	Matrix G_k	44
Equation 69	Eigenvalue problem for time-fractional diffusion equation	44
Equation 70	FPDE example 1	45
Equation 71	FPDE example 2	46
Equation 72	Fractional diffusion-wave equation in n -dimensions	47
Equation 73	Fractional diffusion-wave equation (one space dimension)	48

Equation 74	Mixed Dirichlet-Neumann boundary conditions	48
Equation 75	Central difference matrix discretization	48
Equation 76	Lower triangular matrix discretization for a fractional derivative	48
Equation 77	Fractional diffusion-wave equation with RL derivative	49
Equation 78	RL derivative for different α -values	49
Equation 79	RHS of discretization scheme	49
Equation 80	d, o, r -constants	50
Equation 81	Numerical discretization time-fractional diffusion-wave equation	50
Equation 82	Time-fractional sub-diffusion problem	50
Equation 83	Time-fractional diffusion-wave equation	54
Equation 84	Lower triangular matrix with modified Grünwald-Letnikov weights	54
Equation 85	Modified Grünwald-Letnikov weights for matrix form	54
Equation 86	Starting weights for Grünwald-Letnikov weights	55
Equation 87	Four-term fractional wave equation	58
Equation 88	Frequency-dependent complex permittivity	58
Equation 89	Discretization of second-order spatial derivative within the four-term fractional wave equation	58
Equation 90	Discretization of fractional mixed partial derivative within four-term fractional wave equation	59
Equation 91	Discretization of second-order time derivative within the four-term fractional wave equation	59
Equation 92	Discretization of fractional time derivative within four-term fractional wave equation	59
Equation 93	Discretization of four-term fractional wave equation	59
Equation 94	Explicit numerical solution for the four-term fractional wave equation	60
Equation 95	Ramped sine wave source	60
Equation A-1	Gamma function	71
Equation A-2	Heaviside unit step function	71
Equation A-3	One-parameter Mittag-Leffler function	71
Equation A-4	Two-parameter Mittag-Leffler function	71
Equation A-5	General one-term FDE with constant coefficients	71
Equation A-6	Fractional Green's function for one-term FDE with constant coefficients	71
Equation A-7	General two-term FDE with constant coefficients	71
Equation A-8	Fractional Green's function for two-term FDE with constant coefficients	72
Equation A-9	General three-term FDE with constant coefficients	72
Equation A-10	Fractional Green's function for three-term FDE with constant coefficients	72
Equation A-11	k -th derivative of the two-parameter Mittag-Leffler function	72

ACKNOWLEDGEMENTS

The authors would like to thank Professor Alan D. Freed of Texas A& M University and Professor Kai Diethelm of the University of Applied Sciences Würzburg-Schweinfurt for their guidance in interpreting the surveyed literature, assistance in code and algorithm implementation, and insight into the theories and philosophies behind the methods investigated throughout the course of this work.

Additional thanks goes to Ms. Alina M. Arakal, Ms. Brenda J. Novar, and 1st Lt Landon J. Roddenberry for their assistance in checking the Python code integrity.

1.0 SUMMARY

The practical application of the fractional calculus has significantly grown over the last century. In spite of this, the field still remains somewhat inaccessible to many due to the mathematical background and intensity required to properly apply these analytical tools. Thus, the fractional calculus is still underutilized in many areas of science and engineering. In addition to being a survey of popular numerical methods for solving fractional differential equations, this document is also meant to be used as an entry point for newcomers of the topic by building on classical definitions of integer-order derivatives and integrals and expanding these definitions into their generalized fractional-order counterparts.

Unfortunately, when it comes to solving a fractional differential equation (FDE), the analytical solution is sometimes difficult to find and unwieldy if found at all. This necessitates the development and maturation of numerical methods in order for the fractional calculus to be a practical tool for solving real-world problems. The numerical methods considered in this document are based off of the two rudimentary algorithms: the Grünwald-Letnikov definition (with Grünwald-Letnikov weights and Diethelm weights) and the matrix method. These two methods also provide a good visual representation of the inherent non-local properties of the fractional calculus. These two methods are applied to classical examples such as the relaxation-oscillation equation and the Bagley-Torvik equation in addition to some newer applied problems specific to directed energy applications such as the fractional lumped capacitance model [1] and Maxwell's equations incorporating causal modifications from complex permittivity functions that contain non-local operators [2].

Potential pitfalls, limitations, advantages, and disadvantages at the time of writing are noted and discussed throughout the document. An error analysis is performed comparing numerical results to the appropriate analytical solution where available. It is also noted when additional work will be needed to verify and validate numerical results due to availability of solutions and/or computational limitations.

The numerical method code contained within this report is implemented in Python and was used to generate all of the figures. The code is enclosed in Appendix B of the report along with software specifications used at the time of writing with pertinent notes included as comments within the text of the code.

2.0 INTRODUCTION

Fractional calculus, or the mathematical study of arbitrary-order differentials and integrals, is a field that was theorized back in 1695 by mathematician G. W. Leibniz in his notes with G. A. L'Hopital when contemplating the possibility of a half-order derivative, $\frac{d^{1/2}}{dx^{1/2}}$. In 1819, the French mathematician S. F. Lacroix introduced the first mention of arbitrary-order derivatives in text. Shortly after in 1823, N. H. Abel demonstrated the first application of the arbitrary-order in solving the tautochrone problem. Around the same time, J. Liouville began work in earnest on the definition of the fractional derivative as it is known today. In 1847, B. Riemann, built upon Liouville's work with their combined efforts resulting in what is known today as the Riemann-Liouville fractional integral [3].

Another definition of the fractional derivative was devised by Grünwald and Letnikov circa 1867, now known as the Grünwald-Letnikov fractional derivative [3]. The Grünwald-Letnikov fractional derivative, based off of the difference quotient, became the basis of many numerical methods developed to approximate the fractional derivative [4].

A more recent contribution of notable significance to applied fractional calculus came in 1967 with M. Caputo's work in reformulating the Riemann-Liouville fractional derivative [5]. Originally, the Riemann-Liouville and Caputo fractional derivatives appeared alongside each other in Liouville's works, although they were not recognized as distinct operators until Caputo's contributions [6]. One distinction between the two fractional derivatives was that Riemann-Liouville fractional derivative did not accept classical initial conditions. This meant that when applied to physical problems, the Riemann-Liouville fractional derivative required initial conditions that were not easy to physically interpret and are most often assumed to be zero for this reason. Caputo's applied definition of the fractional derivative provided the means for addressing this issue. The aptly named Caputo derivative allows the application of a fractional derivative to use traditional initial conditions.

The past few decades have shown a marked increase of interest in this topic with its applications taking hold in various research communities [7]. In 1974, "The First Conference on Fractional Calculus and its Applications" was held in New Haven, CT. That same year, a collaboration between Oldham and Spanier [3], became one of the first well-known texts on fractional calculus titled, "The Fractional Calculus: Theory and Applications of Differentiation and Integration to Arbitrary-Order," which subsequently led to the publication of a number of additional textbooks that are now considered to be an industry standard. Some of these works include those by A. C. McBride [8], K. S. Miller and B. Ross [9], S. G. Samko et al. [10], V. Kiryakova [11], B. Rubin [12], A. Carpinteri and F. Mainardi [13], I. Podlubny [14], R. Hilfer [15], A. A. Kilbas et. al [16], S. Das [17], and J. Sabatier et. al [18].

The surge of interest in the fractional calculus is wholly due to the discovery and/or realization that fractional derivatives and integrals naturally reside in the mathematical descriptions of a number of phenomena in a variety of disciplines. Examples of these include, but are not limited to the areas of study of viscoelasticity [19], control theory, dielectrics [2], circuits, bioengineering [20], and finance. The inherent generalized nature of the fractional calculus is a key feature of its application. That is, the fractional calculus has the ability to capture and/or describe patterns and behaviors that traditional integer-order calculus cannot due to operator orders being restricted to integer values preventing them from compensating for activity that deviate from traditional theories and models.

An additional feature that arises from the mathematical formulation of the fractional operators is non-locality. Unlike an integer-order derivative which operates on a single point of a function, a fractional derivative has the ability to look at points either behind or ahead of the point of interest which can be physically interpreted as the function's history or memory depending on the problem. The basis for the methods considered in this work consist of the three different most commonly used definitions of fractional derivatives mentioned above: the Grünwald-Letnikov, the Riemann-Liouville, and the Caputo derivative for solving fractional differential equations. Though other definitions of fractional derivatives exist besides these, the problem context of their application is

considered to be beyond the scope of this study.

2.1 Grünwald-Letnikov Fractional Derivative and Integral

The Grünwald-Letnikov fractional derivative is the most basic of the definitions considered in this study. Its derivation is simply a generalization of the well-known difference quotient of a derivative. The development is reproduced here from [14] for convenience.

Consider a continuous function $f(t)$. To derive an integer n -fold derivative, start from difference quotient of a first-order derivative.

$$f'(t) = \frac{df}{dt} = \lim_{h \rightarrow 0} \frac{f(t) - f(t-h)}{h}$$

where h is the interval or step size between two points in which $f(t)$ and $f(t-h)$ were evaluated. The first-order derivative approximation improves as values of h approach zero. Continuing to the second-order derivative:

$$\begin{aligned} f''(t) &= \frac{d^2f}{dt^2} \lim_{h \rightarrow 0} \frac{f'(t) - f'(t-h)}{h} \\ &= \lim_{h \rightarrow 0} \frac{1}{h} \left(\frac{f(t) - f(t-h)}{h} - \frac{f(t-h) - f(t-2h)}{h} \right) \\ &= \lim_{h \rightarrow 0} \frac{f(t) - 2f(t-h) + f(t-2h)}{h^2} \end{aligned}$$

Repeating for the third-order derivative:

$$f'''(t) = \frac{d^3f}{dt^3} = \lim_{h \rightarrow 0} \frac{f(t) - 3f(t-h) + 3f(t-2h) - f(t-3h)}{h^3}$$

and so on. The n -fold derivative can then be extrapolated as

$$f^{(n)}(t) = \frac{d^n f}{dt^n} = \lim_{h \rightarrow 0} \frac{1}{h^n} \sum_{r=0}^n (-1)^r \binom{n}{r} f(t-rh) \quad (1)$$

Equation 1. n -fold derivative

where $\binom{n}{k}$ is the binomial coefficient defined as $\frac{n!}{k!(n-k)!}$. This n -fold derivative can be further generalized when expressed in terms of the continuous gamma function:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{\Gamma(n+1)}{\Gamma(n-k+1)\Gamma(k+1)} \quad (2)$$

Equation 2. Binomial coefficient in terms of gamma function

where Γ is defined in Equation A-1 of Appendix A and accepts both real and imaginary numbers (for integer values n , $\Gamma(n-1) = n!$). For any real $\alpha \leq n$ and combining Equation 1 and Equation 2 yields the definition of the arbitrary-order derivative:

$${}_aD_t^\alpha[f(t)] = \lim_{h \rightarrow 0} \frac{1}{h^\alpha} \sum_{j=0}^n \omega_j^\alpha f(t-jh) \quad (3)$$

Equation 3. Derivative of arbitrary order α

where ω_j^α are weights defined as

$$\omega_j^\alpha = (-1)^j \binom{\alpha}{j} \quad (4)$$

Equation 4. Grünwald-Letnikov weights

The notations used in Equation 3 are as follows: a and t are the lower and upper terminals of operation respectively and α is the order of the derivative. The upper limit of the summation is defined as $n = \frac{t-a}{h}$. While derivatives are associated with a single point on a continuous function, integrals are associated with the area under a range of two points on a continuous function. Therefore this upper limit property for the summation essentially serves as the mathematical means to allow the Grünwald-Letnikov derivative definition to extend to both derivatives (for positive values of α) and integrals (for negative values of α) as well. Thus, evaluating the limit for $\alpha < 0$ where $\alpha = -p$ yields the Grünwald-Letnikov integral

$${}_aD_t^{-p}[f(t)] = \sum_{k=0}^m \frac{f^{(k)}(a)(t-a)^{p+k}}{\Gamma(p+k+1)} + \frac{1}{\Gamma(p+k+1)} \int_a^t (t-\tau)^{p+m} f^{(m+1)}(\tau) d\tau \quad (5)$$

Equation 5. Grünwald-Letnikov fractional integral

for a function $f(t)$ with $m+1$ continuous derivatives. Conversely, evaluating the limit for $\alpha > 0$ where $\alpha = p$, yields the Grünwald-Letnikov derivative

$${}_aD_t^p[f(t)] = \sum_{k=0}^m \frac{f^{(k)}(a)(t-a)^{-p+k}}{\Gamma(-p+k+1)} + \frac{1}{\Gamma(-p+m+1)} \int_a^t (t-\tau)^{m-p} f^{(m+1)}(\tau) d\tau \quad (6)$$

Equation 6. Grünwald-Letnikov fractional derivative

for the smallest integer m that satisfies $m < p < m+1$.

2.2 Riemann-Liouville Fractional Derivative and Integral

The Riemann-Liouville fractional derivative arises from the unification of integer-order derivatives and integrals which is derived by iterated integration [9], [14]. Given a function $f(\tau)$ that is continuous and integrable on the interval $[a, t]$ and where τ is an arbitrary variable for integration,

then the first-order integral can be written as:

$$f^{(-1)}(t) = \int_a^t f(\tau) d\tau$$

with the two-fold integral resulting in

$$\begin{aligned} f^{(-2)}(t) &= \int_a^t d\tau_1 \int_a^{\tau_1} f(\tau) d\tau = \int_a^t f(\tau) d\tau \int_{\tau}^t d\tau_1 \\ &= \int_a^t (t - \tau) f(\tau) d\tau \end{aligned}$$

and integrating once more yields the three-fold integral of $f(\tau)$:

$$\begin{aligned} f^{(-3)}(t) &= \int_a^t d\tau_1 \int_a^{\tau_1} \int_a^{\tau_2} f(\tau_3) d\tau_3 \\ &= \int_a^t d\tau_1 \int_a^{\tau_1} (\tau_1 - \tau) f(\tau) d\tau \\ &= \frac{1}{2} \int_a^t (t - \tau)^2 f(\tau) d\tau \end{aligned}$$

Through induction for the general n -th fold integral gives the Cauchy formula

$$f^{(-n)}(t) = \frac{1}{(n-1)!} \int_a^t (t - \tau)^{n-1} f(\tau) d\tau \quad (7)$$

Equation 7. Cauchy formula

Replacing the factorial term with the equivalent gamma function representation and replacing n with a real $\alpha > 0$ results in the Riemann-Liouville fractional integral:

$${}_a D_t^{-\alpha} [f(t)] = \frac{1}{\Gamma(\alpha)} \int_a^t (t - \tau)^{\alpha-1} f(\tau) d\tau \quad (8)$$

Equation 8. Riemann-Liouville fractional integral

The Riemann-Liouville fractional derivative is defined directly from the Riemann-Liouville fractional integral by applying an appropriate integer-order derivative to the Riemann-Liouville fractional integral. For the n -th integer-order derivative with a real α such that $n - \alpha > 0$, the operator is defined as:

$${}_a D_t^{n-\alpha} [f(t)] = \frac{1}{\Gamma(\alpha)} \frac{d^n}{dt^n} \int_a^t (t - \tau)^{\alpha-1} f(\tau) d\tau, \quad (0 < \alpha \leq 1) \quad (9)$$

Equation 9. Riemann-Liouville fractional derivative

The integral $\int_a^t (t - \tau)^{\alpha-1} f(\tau)$ in the Riemann-Liouville fractional derivative and integral is also known as the weakly singular Volterra integral with the singularity resulting from the exponential term $\alpha - 1$ being negative for $0 < \alpha \leq 1$. Thus, solving the Volterra at the upper terminal t produces a weak singularity due to $|\alpha - 1| < 1$. Evaluating Riemann-Liouville fractional derivatives and integrals can sometimes be complicated due to this singularity, necessitating numerical approximations depending on the problem.

2.2.1 Equivalence of the Riemann-Liouville and Grünwald-Letnikov Fractional Derivatives.

Under specific conditions the Riemann-Liouville and Grünwald-Letnikov fractional derivatives are equivalent. The conditions are that the derivatives $f(t)$ is $(n - 1)$ -times continuously differentiable in the interval $[a, T]$ and that $f^{(n)}(t)$ is integrable on $[a, T]$. Then for every p ($0 < p < n$) the Riemann-Liouville fractional derivative ${}^{RL}_a D_t^p[f(t)]$ and the Grünwald-Letnikov fractional derivative ${}^{GL}_a D_t^p[f(t)]$ exist and are equivalent, and if $0 \leq m - 1 \leq p \leq m \leq n$, then for $a < t < T$ the following is true:

$${}^{RL}_a D_t^p[f(t)] = {}^{GL}_a D_t^p[f(t)] = \sum_{k=0}^{m-1} \frac{f^{(k)}(a)(t-a)^{k-p}}{\Gamma(1+k-p)} + \frac{1}{\Gamma(m-p)} \int_a^t \frac{f^{(m)}(\tau)d\tau}{(t-\tau)^{p-m+1}} \quad (10)$$

Equation 10. Riemann-Liouville/Grünwald-Letnikov fractional derivative equivalency

with the right hand side being equal to Equation 6. It can alternatively be written as

$$\frac{d^m}{dt^m} \left\{ \sum_{k=0}^{m-1} \frac{f^{(k)}(a)(t-a)^{m+k-p}}{\Gamma(1+m+k-p)} + \frac{1}{\Gamma(2m-p)} \int_a^t (t-\tau)^{2m-p-1} f^{(m)}(\tau)d\tau \right\}$$

where after m integrations by parts yields the definition of the Riemann-Liouville fractional derivative:

$$\frac{d^m}{dt^m} \left\{ \frac{1}{\Gamma(m-p)} \int_a^t (t-\tau)^{m-p-1} f(\tau)d\tau \right\} = \frac{d^m}{dt^m} \left\{ {}_a D_t^{-(m-p)}[f(t)] \right\} = {}^{RL}_a D_t^p[f(t)].$$

An important property of Equation 10 is that the existence of the derivative of order $p > 0$ also implies the exist of q -order derivatives between $0 < q < p$.

2.3 Caputo Fractional Derivative

The Riemann-Liouville fractional derivative can sometimes be problematic to apply to physical problems where non-zero initial conditions are necessary. This is due to the Riemann-Liouville approach leading to initial conditions containing the limit values of the Riemann-Liouville frac-

tional derivative at the lower terminal $t = a$:

$$\begin{aligned}\lim_{t \rightarrow a} {}_a D_t^{\alpha-1} [f(t)] &= b_1, \\ \lim_{t \rightarrow a} {}_a D_t^{\alpha-2} [f(t)] &= b_1, \\ &\vdots \\ \lim_{t \rightarrow a} {}_a D_t^{\alpha-n} [f(t)] &= b_n\end{aligned}$$

where $k = 1, 2, \dots, n$ and b_k are given constants [14]. A solution proposed by Caputo resulted in the following definition:

$${}_a^C D_t^\alpha [f(t)] = \frac{1}{\Gamma(\alpha - n)} \int_a^t \frac{f^{(n)}(\tau) d\tau}{(t - \tau)^{\alpha+1-n}}, \quad (n-1 < \alpha < n) \quad (11)$$

Equation 11. Caputo fractional derivative

A key feature to notice in the Caputo fractional derivative is the n -th order derivative $f^{(n)}(t)$ being contained within the Volterra integrals of Equation 9. As $\alpha \rightarrow n$, where n is an integer, the Caputo fractional derivative neatly becomes the conventional n -th integer-order derivative.

$$\begin{aligned}\lim_{\alpha \rightarrow n} {}_a^C D_t^\alpha [f(t)] &= \lim_{\alpha \rightarrow n} \left(\frac{f^{(n)}(a)(t-a)^{n-\alpha}}{\Gamma(n-\alpha+1)} \right. \\ &\quad \left. + \frac{1}{\Gamma(n-\alpha+1)} \int_a^t (t-\tau)^{n-\alpha} f^{(n+1)}(\tau) d\tau \right) \\ &= f^{(n)}(a) + \int_a^t f^{(n+1)}(\tau) d\tau = f^{(n)}(t), \quad n = 1, 2, \dots\end{aligned}$$

As a result, the initial conditions for fractional differential equations with Caputo fractional derivatives initial conditions assume the same form as for conventional integer-order differential equations.

3.0 NUMERICAL METHODS FOR SOLVING FRACTIONAL ORDINARY DIFFERENTIAL EQUATIONS

This section of the document will show that numerical solutions to fractional ordinary differential equations (FODEs) can be convenient, efficient, and in some cases necessary when compared to their corresponding analytical solutions. The two basic principles that will be considered as the basis for formulating numerical methods to solve FODEs are the Grünwald-Letnikov fractional derivative definition and matrix-based schemes. It will be shown that the Grünwald-Letnikov definition lends itself well as an explicit numerical scheme for FODEs while matrix-based methods demonstrate an implicit numerical scheme that also help illustrate the non-local properties of fractional-order operators.

3.1 Grünwald-Letnikov Fractional Derivative Numerical Method

From Equation 6 the Grünwald-Letnikov scheme can be discretized into a finite difference algorithm. Consider the following FODE

$${}_aD_t^\alpha[y(t)] = f(t) \quad (12)$$

Equation 12. Basic fractional differential equation

.

On a grid of discrete points, where $k = 0, 1, 2, \dots, \lceil \frac{t-a}{h} \rceil$ and from Equation 3 where $t_k = kh$, $y_k = y(t_k)$, $f_k = f(t_k)$

$${}_aD_t^\alpha[y_k] = \frac{1}{h^\alpha} \sum_{j=0}^k \omega_j^\alpha y_{k-j} = f_k \quad (13)$$

Equation 13. Grünwald-Letnikov discretization scheme

and from Equation 4

$$\omega_j^\alpha = (-1)^j \binom{\alpha}{j}.$$

Algebraic manipulation of Equation 13 yields an explicit scheme that can be used to directly solve for y_k :

$$\frac{1}{h^\alpha} \left(\omega_0^\alpha y_k + \sum_{j=1}^k \omega_j^\alpha y_{k-j} \right) = f_k$$

For all real α values, if $j = 0$, then the Grünwald-Letnikov weight ω_0^α becomes

$$\omega_0^\alpha = (-1)^0 \binom{\alpha}{0} = 1$$

Substituting this value

$$\frac{1}{h^\alpha} \left(y_k + \sum_{j=1}^k \omega_j^\alpha y_{k-j} \right) = f_k$$

and solving for y_k gives the explicit scheme

$$y_k = h^\alpha f_k - \sum_{j=1}^k \omega_j^\alpha y_{k-j} \quad (14)$$

Equation 14. Explicit scheme GL numerical solution for equation

This explicit scheme relies on all known values evaluated prior to y_k , meaning y_k for $k = k-1, k-2, \dots, 0$. Since f_k is already defined this means y_k can be determined explicitly. The solution for the first few iterations of $k = 0, \dots, 5$ are shown in Equation 14 with an initial condition of $y_0 = 0$.

$$\begin{aligned} y_0 &= 0 \\ y_1 &= h^\alpha f_1 - \omega_1^\alpha y_0 \\ y_2 &= h^\alpha f_2 - \omega_1^\alpha y_1 - \omega_2^\alpha y_0 \\ y_3 &= h^\alpha f_3 - \omega_1^\alpha y_2 - \omega_2^\alpha y_1 - \omega_3^\alpha y_0 \\ y_4 &= h^\alpha f_4 - \omega_1^\alpha y_3 - \omega_2^\alpha y_2 - \omega_3^\alpha y_1 - \omega_4^\alpha y_0 \\ y_5 &= h^\alpha f_5 - \omega_1^\alpha y_4 - \omega_2^\alpha y_3 - \omega_3^\alpha y_2 - \omega_4^\alpha y_1 - \omega_5^\alpha y_0 \end{aligned} \quad (15)$$

Equation 15. Example GL numerical solution for Equation 12

3.2 Diethelm Product-Trapezoidal Quadrature Coefficients

An alternative development to the Grünwald-Letnikov scheme discussed in the previous section was proposed by K. Diethelm [21] that modifies the weight ω and is shown to be

$${}_a D_t^\alpha [y_k] = \frac{1}{h^\alpha} \sum_{j=0}^k \omega_{j,k}^\alpha y_{k-j} = f_k, \quad \text{for } 0 < \alpha \leq 1 \quad (16)$$

Equation 16. Diethelm discretization scheme

.

This discretization scheme shares the same development as Equation 13 but defines the weights $\omega_{j,k}^\alpha$. In this case, the definition of the weights stem from the trapezoidal Riemann sum approxima-

tion of the integral in the Riemann-Liouville fractional derivative seen as Equation 9. An important limitation to note is that these weights are only valid for a specific range of $0 < \alpha \leq 1$.

The weights are defined in [21] as

$$\omega_{k,k-j}^\alpha = \frac{1}{\Gamma(2-\alpha)} \times \begin{cases} 1 & \text{for } j = 0 \\ (j-1)^{1-\alpha} - 2j^{1-\alpha} + (j+1)^{1-\alpha} & \text{for } 1 \leq k \leq j-1 \\ (k-1)^{1-\alpha} - (\alpha-1)k^{-\alpha} - k^{1-\alpha} & \text{for } k = j \geq 1 \end{cases} \quad (17)$$

Equation 17. Diethelm product-trapezoidal weights

When implemented into the discretization scheme in Equation 16, the weights are adjusted so that they can be directly substituted for the Grünwald-Letnikov weights while maintaining the same algorithm in the Grünwald-Letnikov discretization scheme of Equation 13. Hence, Equation 17 becomes

$$\omega_{j,k}^\alpha = \frac{1}{\Gamma(2-\alpha)} \times \begin{cases} 1 & \text{for } j = 0 \\ (j-1)^{1-\alpha} - 2j^{1-\alpha} + (j+1)^{1-\alpha} & \text{for } 1 \leq j \leq k-1 \\ (k-1)^{1-\alpha} + (1-\alpha-k)k^{-\alpha} & \text{for } k = j \geq 1 \end{cases} \quad (18)$$

Equation 18. Modified Diethelm product-trapezoidal weights

and like the Grünwald-Letnikov discretization scheme discussed previously, algebraic manipulation of Equation 16 yields an explicit scheme that can be used to directly solve for y_k :

$$\frac{1}{h^\alpha} \left(\omega_{0,k}^\alpha y_k + \sum_{j=1}^k \omega_{j,k}^\alpha y_{k-j} \right) = f_k$$

For any real α value, if $j = 0$, then the Diethelm weight $\omega_{0,k}^\alpha$ becomes

$$\omega_{0,k}^\alpha = \frac{1}{\Gamma(2-\alpha)}$$

and finally substituting this value

$$\frac{1}{h^\alpha} \left(\frac{y_k}{\Gamma(2-\alpha)} + \sum_{j=1}^k \omega_{j,k}^\alpha y_{k-j} \right) = f_k$$

and solving for y_k gives

$$y_k = \Gamma(2 - \alpha) \left(h^\alpha f_k - \sum_{j=1}^k \omega_{j,k}^\alpha y_{k-j} \right) \quad (19)$$

Equation 19. Explicit scheme numerical solution with Diethelm weights for Equation 12

The same example demonstrated for Equation 14 is solved again for $k = 0, \dots, 5$ iterations with an initial condition $y_0 = 0$.

$$\begin{aligned} y_0 &= 0 \\ y_1 &= \Gamma(2 - \alpha) (h^\alpha f_1 - \omega_{1,1}^\alpha y_0) \\ y_2 &= \Gamma(2 - \alpha) (h^\alpha f_2 - \omega_{1,2}^\alpha y_1 - \omega_{2,2}^\alpha y_0) \\ y_3 &= \Gamma(2 - \alpha) (h^\alpha f_3 - \omega_{1,3}^\alpha y_2 - \omega_{2,3}^\alpha y_1 - \omega_{3,3}^\alpha y_0) \\ y_4 &= \Gamma(2 - \alpha) (h^\alpha f_4 - \omega_{1,4}^\alpha y_3 - \omega_{2,4}^\alpha y_2 - \omega_{3,4}^\alpha y_1 - \omega_{4,4}^\alpha y_0) \\ y_5 &= \Gamma(2 - \alpha) (h^\alpha f_5 - \omega_{1,5}^\alpha y_4 - \omega_{2,5}^\alpha y_3 - \omega_{3,5}^\alpha y_2 - \omega_{4,5}^\alpha y_1 - \omega_{5,5}^\alpha y_0) \end{aligned} \quad (20)$$

Equation 20. Example GL numerical solution for equation

and looks similar to Equation 14 as expected. It is important to note here that the Diethelm weights provide a computational advantage over the Grünwald-Letnikov weights that require the costly evaluation of the binomial coefficient that contains three gamma functions whereas the Diethelm weights have only one.

3.3 Matrix Method

The matrix method employs the Grünwald-Letnikov numerical method in Equation 13 as an implicit scheme eigenvalue problem for each time step. This means that the solution to an FODE at y_k is dependent on the values calculated in time steps prior to y_k ($y_{k-1}, \dots, y_0$) in addition to y_k itself. The method discussed here follows the example provided in [22]. Given the Grünwald-Letnikov fractional derivative approximation described in subsection 3.1, the implicit scheme can be written as

$$\begin{aligned} {}_a D_{t_k}^\alpha [f(t)] &= h^{-\alpha} \sum_{j=0}^k (-1)^j \binom{\alpha}{j} f_{k-j} \\ &= h^{-\alpha} \sum_{j=0}^k \omega_j f_{k-j}, \quad k = 0, 1, 2, \dots, N \end{aligned}$$

where the Grünwald-Letnikov fractional derivative becomes a lower triangular matrix containing the Grünwald-Letnikov weights, ω_k . The modified weights from Diethelm's method, $\omega_{j,k}^\alpha$, can also be substituted in for the Grünwald-Letnikov (GL) weights here. The resulting matrices in each case are shown here for convenience. To improve readability, the α superscript for the weights were omitted.

$$B_N^\alpha = \frac{1}{h^\alpha} \begin{bmatrix} \omega_0 & 0 & 0 & 0 & \dots & 0 \\ \omega_1 & \omega_0 & 0 & 0 & \dots & 0 \\ \omega_2 & \omega_1 & \omega_0 & 0 & \dots & 0 \\ \ddots & \ddots & \ddots & \ddots & \dots & 0 \\ \omega_{N-1} & \ddots & \omega_2 & \omega_1 & \omega_0 & 0 \\ \omega_N & \omega_{N-1} & \dots & \omega_2 & \omega_1 & \omega_0 \end{bmatrix} \quad (21)$$

Equation 21. Lower triangular matrix with Grünwald-Letnikov weights matrix

$$B_N^\alpha = \frac{1}{h^\alpha} \begin{bmatrix} \omega_{0,0} & 0 & 0 & 0 & \dots & 0 \\ \omega_{1,1} & \omega_{0,1} & 0 & 0 & \dots & 0 \\ \omega_{2,2} & \omega_{1,2} & \omega_{0,2} & 0 & \dots & 0 \\ \ddots & \ddots & \ddots & \ddots & \dots & 0 \\ \omega_{N-1,N-1} & \ddots & \omega_{2,N-1} & \omega_{1,N-1} & \omega_{0,N-1} & 0 \\ \omega_{N,N} & \omega_{N-1} & \dots & \omega_{2,N} & \omega_{1,N} & \omega_{0,N} \end{bmatrix} \quad (22)$$

Equation 22. Lower triangular matrix with Diethelm product-trapezoidal quadrature weights

The Grünwald-Letnikov weight matrix will be used for the matrix method example since it can be generalized for a greater range of α values. Thus, the lower triangular matrix shown in Equation 21 is used to solve the eigenvalue problem of

$$[{}_aD_t^\alpha[F]]_N = B_N^\alpha F_N.$$

The discretization of the problem is then

$$\begin{bmatrix} {}_aD_{t_0}^\alpha[f(t_0)] \\ {}_aD_{t_1}^\alpha[f(t_1)] \\ {}_aD_{t_2}^\alpha[f(t_2)] \\ \vdots \\ {}_aD_{t_{N-1}}^\alpha[f(t_{N-1})] \\ {}_aD_{t_N}^\alpha[f(t_N)] \end{bmatrix} = \frac{1}{h^\alpha} \begin{bmatrix} \omega_0 & 0 & 0 & 0 & \dots & 0 \\ \omega_1 & \omega_0 & 0 & 0 & \dots & 0 \\ \omega_2 & \omega_1 & \omega_0 & 0 & \dots & 0 \\ \ddots & \ddots & \ddots & \ddots & \dots & 0 \\ \omega_{N-1} & \ddots & \omega_2 & \omega_1 & \omega_0 & 0 \\ \omega_N & \omega_{N-1} & \dots & \omega_2 & \omega_1 & \omega_0 \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_{N-1} \\ f_N \end{bmatrix} \quad (23)$$

Equation 23. Matrix method discretization

At this point, the reader is reminded that finite difference methods used to numerically solve traditional ordinary differential equations (ODEs) produce banded matrices. For example, for a first-order derivative ($\alpha = 1$) backward difference scheme for

$$f'(t_k) = \frac{1}{h}(f_k - f_{k-1}), \quad k = 1, \dots, N$$

the corresponding eigenvalue problem would be

$$[{}_aD_t^{\alpha=1}[F]]_N = B_N^{\alpha=1} F_N$$

with the discretization producing

$$\begin{bmatrix} f(t_0) \\ f'(t_1) \\ f'(t_2) \\ \vdots \\ f'(t_{N-1}) \\ f'(t_N) \end{bmatrix} = \frac{1}{h} \begin{bmatrix} 1 & 0 & 0 & 0 & \dots & 0 \\ -1 & 1 & 0 & 0 & \dots & 0 \\ 0 & -1 & 1 & 0 & \dots & 0 \\ 0 & 0 & \ddots & \ddots & \dots & 0 \\ 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & \dots & 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_{N-1} \\ f_N \end{bmatrix} \quad (24)$$

Equation 24. Backward difference first-order derivative

which corresponds to the fractional lower triangular matrix configuration as $\omega_0^{\alpha=1} = 1$ and $\omega_1^{\alpha=1} = -1$. For a second-order derivative ($\alpha = 2$) backward difference scheme,

$$f''(t_k) = \frac{1}{h^2}(f_k - 2f_{k-1} + f_{k-2}), \quad k = 2, \dots, N$$

the corresponding eigenvalue problem is then

$$[{}_a D_t^{\alpha=2}[F]]_N = B_N^{\alpha=2} F_N$$

and the discretization producing

$$\begin{bmatrix} f(t_0) \\ (-2f(t_0) - f(t_1)) \\ f''(t_2) \\ \vdots \\ f''(t_{N-1}) \\ f''(t_N) \end{bmatrix} = \frac{1}{h^2} \begin{bmatrix} 1 & 0 & 0 & 0 & \dots & 0 \\ -2 & 1 & 0 & 0 & \dots & 0 \\ 1 & -2 & 1 & 0 & \dots & 0 \\ 0 & \ddots & \ddots & \ddots & \dots & 0 \\ 0 & 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & \dots & 1 & -2 & 1 \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_{N-1} \\ f_N \end{bmatrix} \quad (25)$$

Equation 25. Backward difference second-order derivative

which corresponds to the fractional lower triangular configuration with the Grünwald-Letnikov weights as $\omega_0^{\alpha=2} = 1$, $\omega_1^{\alpha=2} = -2$, and $\omega_2^{\alpha=2} = 1$.

Therefore, to evaluate the integer-order derivative requires only two weights for the first-order derivative, $\{1, -1\}$ and two function value terms $\{f(t_k), f(t_{k-1})\}$, and only three weights for the second-order derivative, $\{1, -2, 1\}$ and three function value terms $\{f(t_k), f(t_{k-1}), f(t_{k-2})\}$. On the other hand, the fractional derivative ${}_a D_t^\alpha f(t_k)$ requires weights from $\{\omega_0, \dots, \omega_k\}$ and function values from $\{f(t_0), \dots, f(t_k)\}$, meaning that the entire history of the function is contained at all time steps up to t_k . Thus, the lower diagonal conformation of the difference matrix for approximating a fractional derivative is the numerical manifestation of the inherent non-local property of the operator.

3.3.1 Solving Fractional Ordinary Differential Equations.

With the matrix method defined to approximate fractional derivatives, the method can now be applied towards solving FODEs by summing the appropriate weight matrices for each term. This condenses the problem into a single eigenvalue problem to evaluate and solve for the solution to the FODE. As an example, consider an m -term linear FODE

$$\sum_{l=1}^m p_l(t) D^{\alpha_l}[y(t)] = f(t), \quad 0 \leq \alpha_1 < \alpha_2 < \dots < \alpha_k, \quad n-1 < \alpha_l < n \quad (26)$$

Equation 26. m -term linear FDE

where D^{α_l} is the Riemann-Liouville or Caputo fractional derivative of order α_l and n is the smallest integer such that the α_l value is contained between $n-1$ and n . The following discretizations are then made:

$$P_N^{(l)} = \text{diag}(p_l(t_0), p_l(t_1), \dots, p_l(t_N)) = \begin{bmatrix} p_l(t_0) & 0 & \dots & 0 \\ 0 & p_l(t_1) & 0 & \vdots \\ \vdots & 0 & \ddots & 0 \\ 0 & \dots & 0 & p_l(t_N) \end{bmatrix}$$

$$Y_N = (y(t_0), \dots, y(t_N))^T, \quad F_N = (f(t_0), \dots, f(t_N))^T$$

where *diag* is a function that takes in a vector of dimension N and places the vector values on the main diagonal of an $N \times N$ matrix. With the terms defined above, the matrix method discretization of Equation 26 becomes the eigenvalue problem

$$\sum_{l=1}^m P_N^{(l)} B_N^{\alpha_l} Y_N = F_N \quad (27)$$

Equation 27. Matrix method discretization of m -term linear FDE

where $B_N^{\alpha_l}$ is the appropriate backward difference matrix for the α_l -order derivative (integer- or fractional-order). In order to properly apply the matrix method to solve FODEs with physically intuitive initial conditions, the Riemann-Liouville and Caputo fractional derivative must be equivalent. These conditions are when $n - 1 < \alpha_l \leq n$ and the initial conditions are equal to zero for a function $y(t)$ and its $(n - 1)$ derivatives.

$$y^{(l)}(t_0) = 0, \quad l = 0, 1, \dots, n - 1$$

The backward difference approximation translates numerically as

$$y(t_0) = y(t_1) = \dots = y(t_{n-1}) = 0$$

To account for the zero initial conditions, eliminator matrices are implemented, denoted by S . With an identity matrix I , the subscript for the eliminator matrix designates which row is removed from I , instances of which are shown below.

$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad S_0 = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad S_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad S_{0,1} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \quad (28)$$

Equation 28. Eliminator matrices

Matrix multiplication of the eliminator matrix and the vector Y_N satisfies the requirements of initial conditions equal to zero for $y(t)$ and its $(n - 1)$ derivatives. The complete notation for the matrix method for FODEs takes the form:

$$\sum_{l=1}^m P_l B_{N-n}^{\alpha_l} \{S_{0,1,\dots,n-1} Y_N\} = S_{0,1,\dots,n-1} F_N \quad (29)$$

Equation 29. Matrix method eigenvalue problem for m -term linear FODE

A matrix inversion is applied to solve for the result of the eliminator matrix operation on Y_N , which is a new vector with the $0, 1, \dots, n-1$ vector elements removed.

3.4 Applied Examples

This section contains applied examples of the Grünwald-Letnikov and Matrix based numerical methods developed throughout section 3 for solving various basic fractional order differential equations.

3.4.1 Relaxation-Oscillation.

The relaxation-oscillation problem involves a derivative of order $1 \leq \alpha \leq 2$.

$$\begin{aligned} {}_0D_t^\alpha[y(t)] + Ay(t) &= f(t), \quad (t > 0) \\ y^{(m)}(0) &= 0, \quad (m = 0, 1, \dots, n-1) \end{aligned} \quad (30)$$

Equation 30. Relaxation-oscillation problem

where $n-1 < \alpha \leq n$ and ${}_0D_t^\alpha$ is the fractional derivative of order α on the interval of 0 to t . The second expression in Equation 30 is the initial condition where derivatives of order 0 to $n-1$ are set to zero. The derivative implemented for this problem is the Riemann-Liouville fractional derivative and the analytical solution is shown to be [14]:

$$y(t) = \int_0^t G_2(t-\tau) f(\tau) d\tau, \quad G_2(t) = t^{\alpha-1} E_{\alpha,\alpha}(-At^\alpha) \quad (31)$$

Equation 31. Relaxation-oscillation analytical solution

where $E_{\alpha,\alpha}$ is the two-parameter Mittag-Leffler function which can be thought of as the FODE analog of the exponential function for ODEs and is defined by Equation A-4 in Appendix A. The function $G_2(t)$ is the fractional Green's function for the two-term FODE with constant coefficients, with the relaxation-oscillation problem being a specific case of the two-term FODE and is defined by Equation A-8 in Appendix A. The source term $f(t)$ was set to the Heaviside unit step function defined in Equation A-2 in Appendix A for all examples in this section.

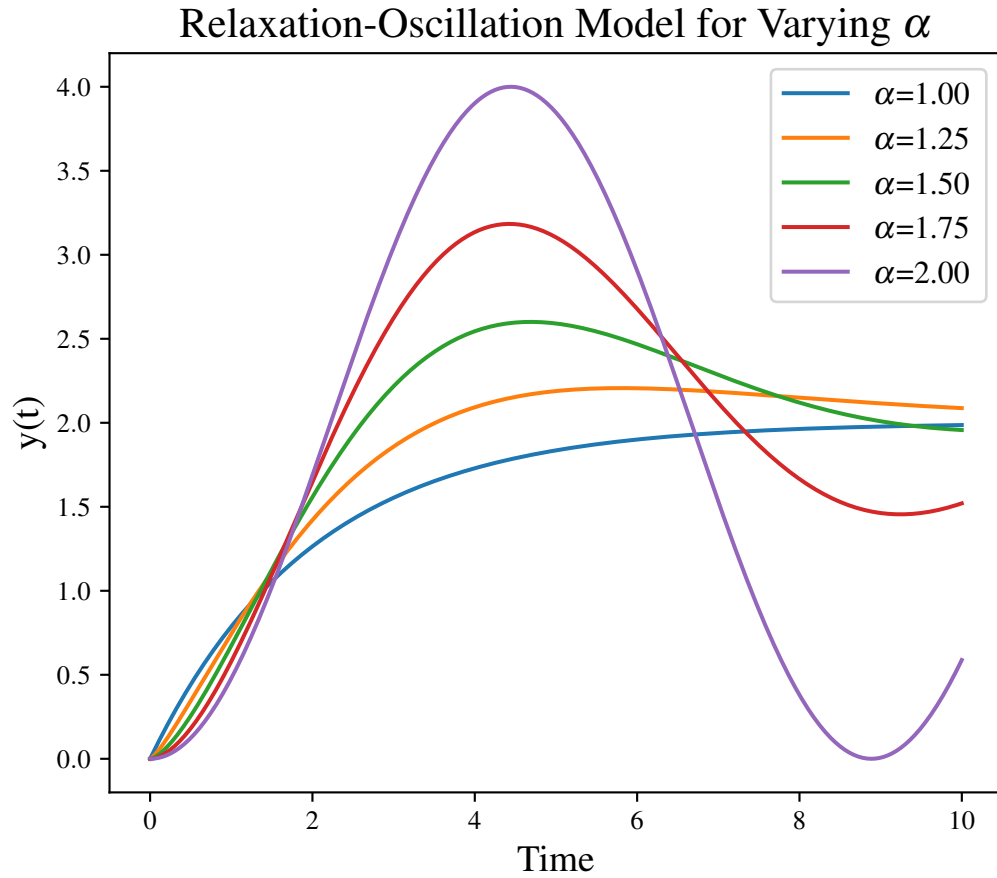


Figure 1. Relaxation-oscillation equation solution for varying α

Figure 1 illustrates the effect of varying α values within the interval of $1 \leq \alpha \leq 2$. At $\alpha = 1$, the relaxation of the system is apparent as the curve flattens out while at $\alpha = 2$, the oscillation of the system is revealed as an oscillating curve. When the value of α lies between these two values, qualities of both relaxation and oscillation behavior can be seen.

Grünwald-Letnikov Method

The discretization of the fractional derivative in the relaxation-oscillation problem results in the following configuration:

$$\begin{aligned} h^{-\alpha} \sum_{j=0}^k \omega_j^\alpha y_{k-j} + Ay_k &= f_k, \quad (k = 1, 2, \dots); \quad y_0 = 0 \\ t_k &= kh, y_k = y(t_k), f_k = f(t_k), \quad (k = 0, 1, 2, \dots); \\ \omega_j^\alpha &= (-1)^j \binom{\alpha}{j}, \quad (j = 0, 1, 2, \dots) \end{aligned} \tag{32}$$

Equation 32. Discretization of relaxation-oscillation problem

Following the same procedure as in subsection 3.1, this discretization leads to the algorithm for determining the explicit scheme numerical solution.

$$\begin{aligned} h^{-\alpha} \left(\omega_0^\alpha y_k + \sum_{j=1}^k \omega_j^\alpha y_{k-j} \right) + Ay_k &= f_k \\ y_k + h^\alpha Ay_k &= h^\alpha f_k - \sum_{j=1}^k \omega_j^\alpha y_{k-j} \end{aligned}$$

By a first-order approximation, the term $Ay_k \approx Ay_{k-1}$, and the Grünwald-Letnikov numerical solution is determined to be:

$$\begin{aligned} y_m &= 0, \quad (m = 1, 2, \dots, n-1) \\ y_k &= -Ah^\alpha y_{k-1} - \sum_{j=1}^k \omega_j^\alpha y_{k-j} + h^\alpha f_k, \quad (k = n, n+1, \dots) \end{aligned} \tag{33}$$

Equation 33. Grünwald-Letnikov solution relaxation-oscillation

where $n-1 < \alpha \leq n$ for an integer n , and the y_k values initialize the initial n points to zero to satisfy the initial conditions as defined by the relaxation-oscillation problem. Figure 2 shows the Grünwald-Letnikov numerical solution and analytical solutions for relaxation-oscillation for decreasing step sizes of h ($\alpha = 1.5$). As one would expect, accuracy improves for smaller step sizes. The outputs in Figure 2 were generated by the code in subsection B.1.

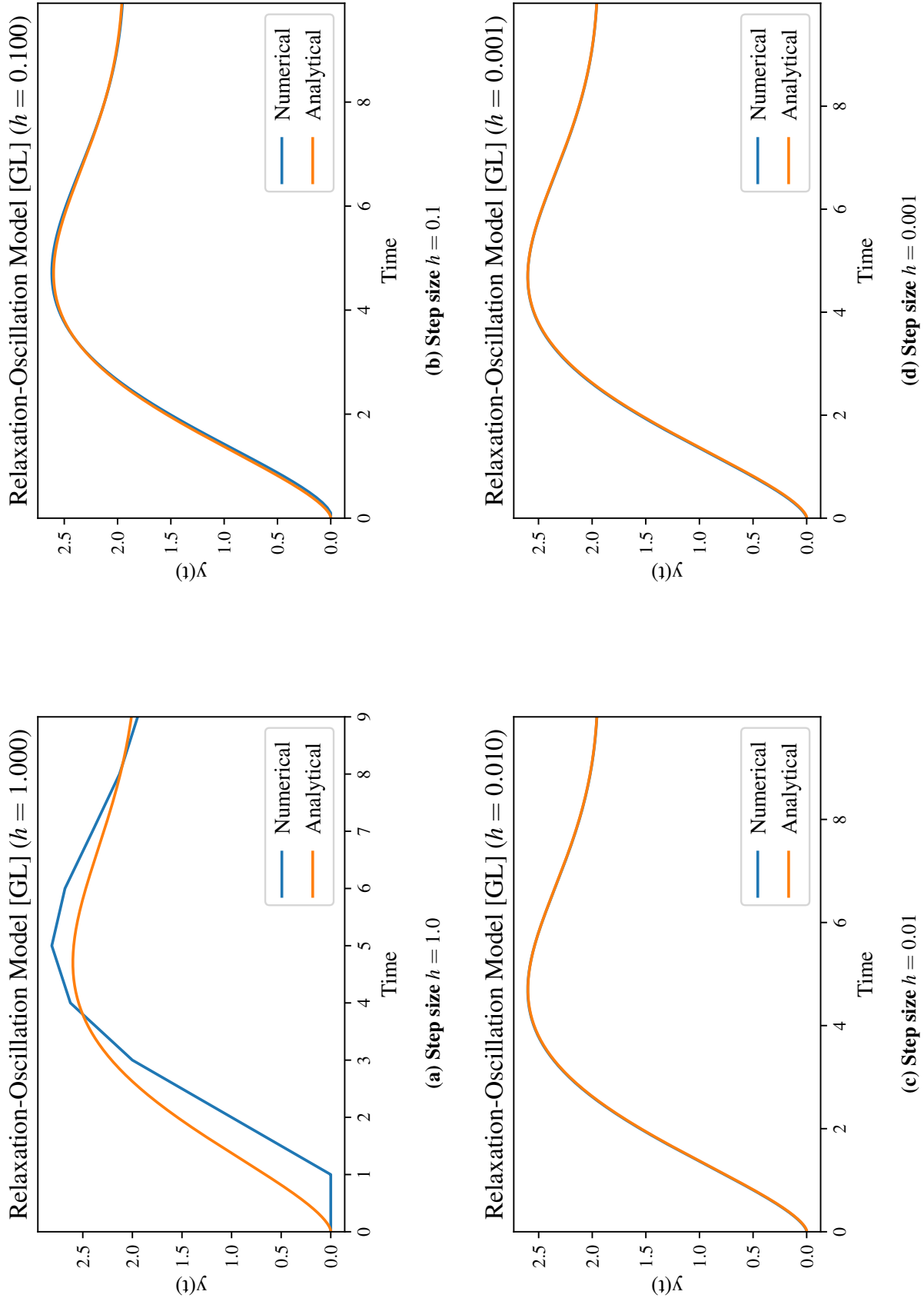


Figure 2. Grünwald-Letnikov numerical solution and analytical solution for relaxation-oscillation for decreasing step sizes of h ($\alpha = 1.5$)

Matrix Method

Starting from the eigenvalue problem in Equation 29 and taking $\alpha = 1.5$, the constants and matrices defined by the relaxation-oscillation problem in Equation 30 are as follows.

- Total number of terms: $m = 2$
- $\alpha_1 = 1.5, \alpha_2 = 0, n = \lceil \max(\{\alpha_1, \alpha_2\}) \rceil = 2$
- $p_1 = 1, p_2 = A$
- $B_{N-n}^{1.5} = B_{N-2}^{1.5}$
- $B_{N-n}^{\alpha_2} = A * I$ (where A is the constant, I is identity matrix)
- $F_N = (\underbrace{1, 1, \dots, 1}_N)^T$ (vector form of Heaviside unit step function)

Following the steps in subsubsection 3.3.1, the matrix method scheme for the relaxation-oscillation problem becomes:

$$(B_{N-2}^{\alpha} + A * I)(S_{0,1}Y_N) = S_{0,1}F_N \quad (34)$$

Equation 34. Matrix method scheme for the relaxation-oscillation equation

Equation 34 is rewritten here with the matrices and vectors expanded for the reader's convenience:

$$\left(\frac{1}{h^{\alpha}} \begin{bmatrix} \omega_0 & \dots & 0 \\ \vdots & \ddots & \vdots \\ \omega_{N-2} & \dots & \omega_0 \end{bmatrix} + A \begin{bmatrix} 1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 1 \end{bmatrix} \right) \begin{bmatrix} 0 & 0 & 1 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & \dots & \dots & 1 \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{N-1} \\ y_N \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & \dots & \dots & 1 \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_{N-1} \\ f_N \end{bmatrix}$$

After performing the matrix operations, the eigenvalue problem for the relaxation-oscillation equation emerges:

$$\begin{bmatrix} \frac{\omega_0}{h^\alpha} + A & \dots & 0 \\ \vdots & \ddots & \vdots \\ \frac{\omega_{N-2}}{h^\alpha} & \dots & \frac{\omega_0}{h^\alpha} + A \end{bmatrix} \begin{bmatrix} y_2 \\ \vdots \\ y_N \end{bmatrix} = \begin{bmatrix} f_2 \\ \vdots \\ f_N \end{bmatrix} \quad (35)$$

Equation 35. Matrix method relaxation-oscillation eigenvalue problem (expanded)

Figure 3 shows the matrix method numerical solution and analytical solutions for relaxation-oscillation for decreasing step sizes of h ($\alpha = 1.5$). Once again accuracy improves for smaller step sizes as expected. The outputs in Figure 3 were generated by the code in subsection B.1.

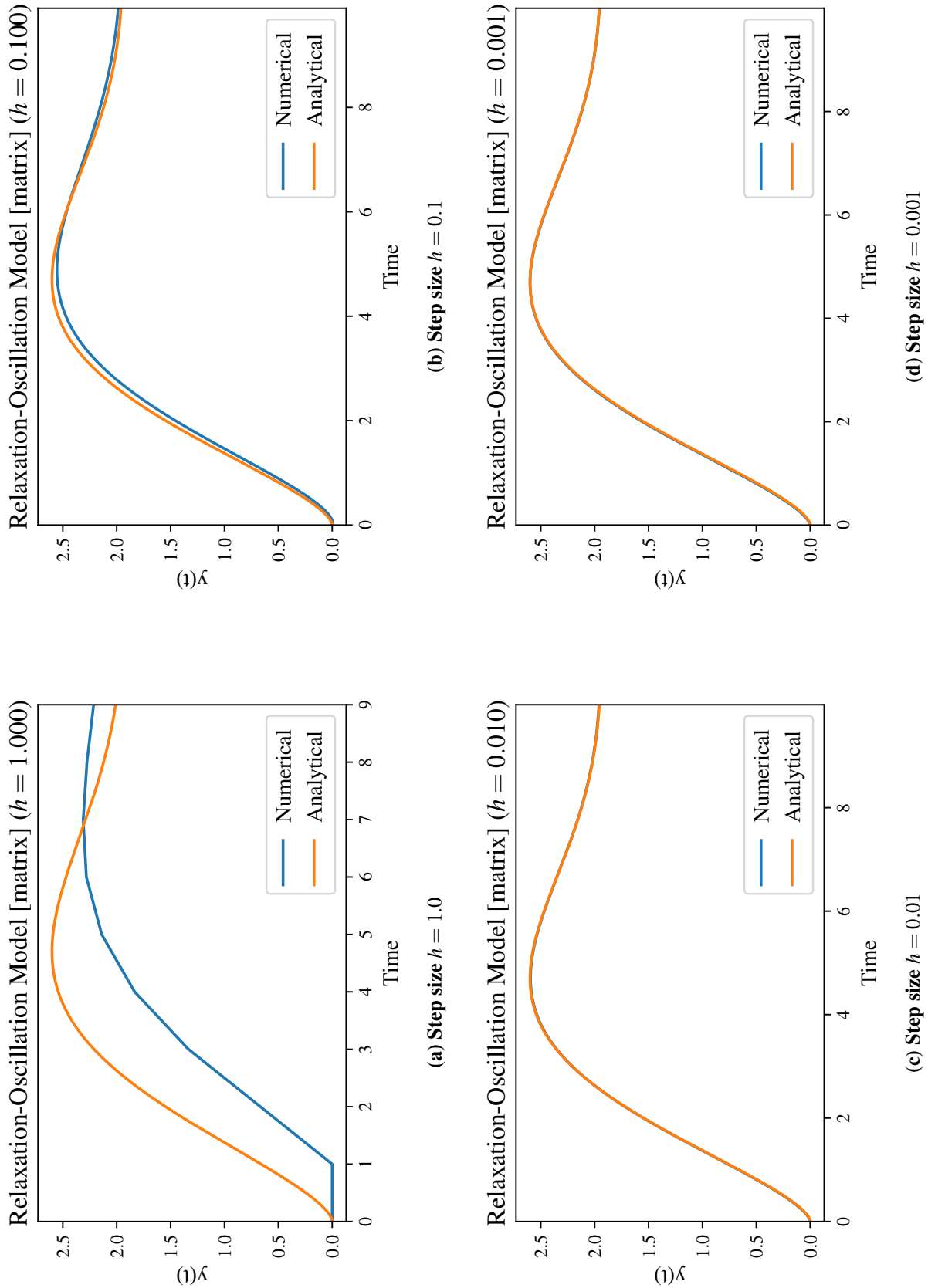


Figure 3. Matrix method numerical solution and analytical solution for relaxation-oscillation for decreasing step sizes of $h(\alpha = 1.5)$

Computation Time and Numerical Error

Each of the Grünwald-Letnikov, matrix, and analytical solutions were timed and compared against each other. The numerical solutions were each calculated for step sizes from 0.001 to 1.0 in increments of 0.001. For each step size, the solutions were calculated and the computation time was measured 10 times and averaged with the results shown in Figure 4. Results for select step sizes are shown in Table 1. Analytical solution compute times were measured for 40 runs for 4 different time steps and averaged with the result being indicated by the four points in Figure 4. The analytical solution calculated with a step size of 0.01 provided the best resolution for a relatively lower computational time cost.

Table 1. Computation time results for the relaxation-oscillation equation

Step Size	GL (s)	Matrix (s)	Analytical (s)
0.001	0.1884	0.9069	3.1111
0.01	0.0106	0.0062	0.2766
0.1	0.0010	0.0028	0.0297
1.0	0.0001	0.0017	0.0029

In Figure 4, the matrix method overtook the Grünwald-Letnikov method in terms of computation time at a step size of around 0.01. For step sizes any smaller, the matrix method computation time increased drastically almost becoming as time intensive as the analytical solution. Overall, the analytical solution remained the most time costly way to solve the relaxation-oscillation equation. This is the opposite of what is usually expected for numerical solutions to differential equations but highlights their importance for solving fractional differential equations.

The mean squared error (MSE) was calculated by squaring the difference between the analytical and numerical solutions at overlapping grid points and dividing by the total number of overlapping grid points as per the equation

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n \left(y_i^{\text{analyt}} - y_i^{\text{num}} \right)^2. \quad (36)$$

Equation 36. Mean squared error (MSE)

In terms of error, the Grünwald-Letnikov and matrix methods achieved the minimum error at two different points. The Grünwald-Letnikov method had the smallest error at step size of 0.02 whereas the matrix method had the smallest error at step size of 0.01. While the computation time for the matrix method is faster than the GL method at these two points, the differences are extremely marginal. Thus, for a fairly simple problem and a small time domain like the relaxation-oscillation equation, there is no clear best choice between these two numerical methods in terms of computation time and numerical error.

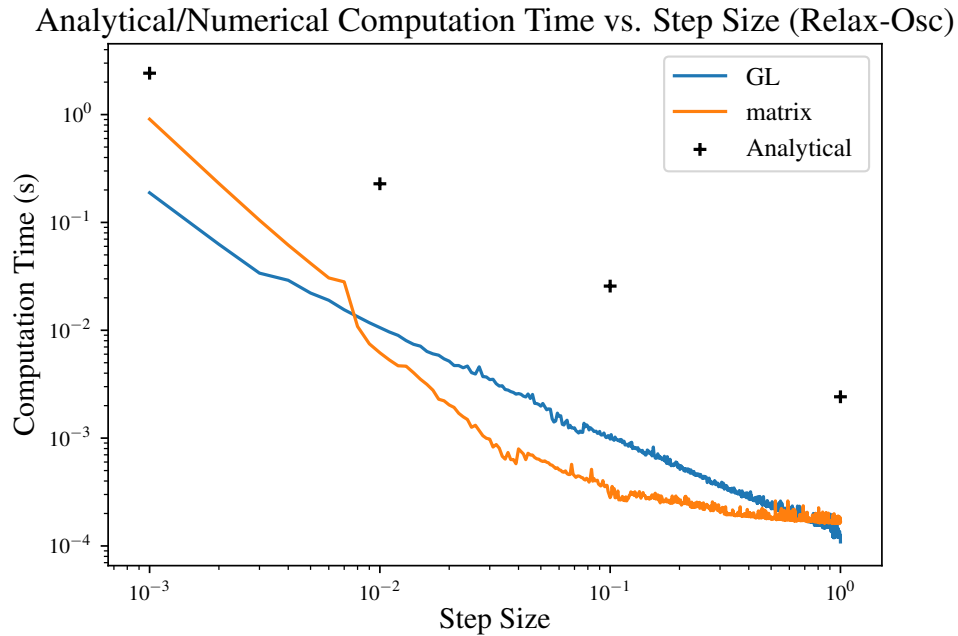


Figure 4. Computation time comparison of Grünwald-Letnikov, matrix, and analytical solutions (relaxation-oscillation equation)

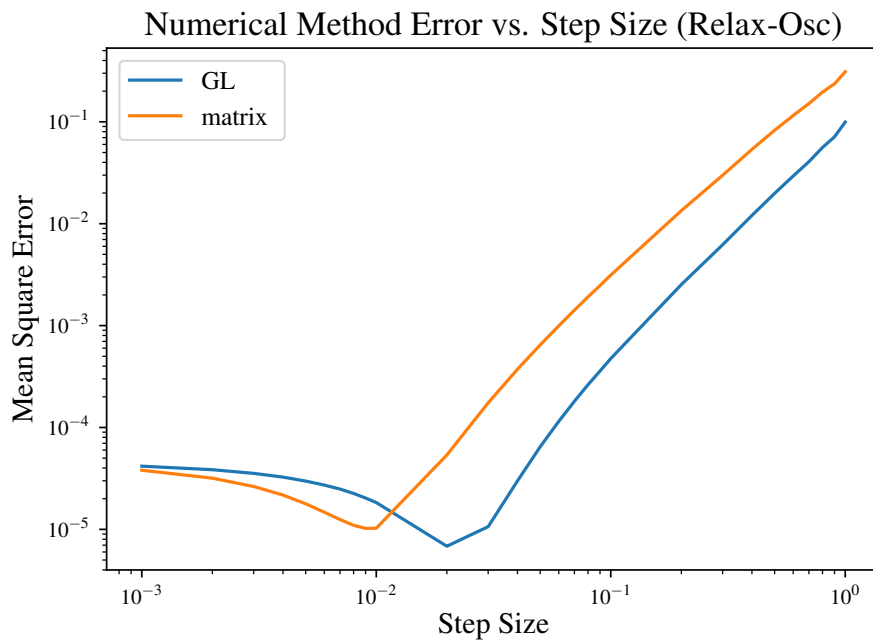


Figure 5. Computation error comparison of numerical methods (relaxation-oscillation equation)

3.4.2 Bagley-Torvik Equation.

The FODE that describes the motion of a thin plate suspended in a Newtonian fluid by a spring illustrated in Figure 6 was formulated by R. L. Bagley and P. J. Torvik [19] and is reproduced here along with physical constant definitions and initial conditions:

$$\begin{aligned} Ay''(t) + {}_0D_t^{3/2}[y(t)] + Ky(t) &= f(t), \quad (t > 0) \\ A = M, \quad B = 2S\sqrt{\mu\rho}, \quad C &= K \\ y(0) &= 0, \quad y'(0) = 0 \end{aligned} \quad (37)$$

Equation 37. Bagley-Torvik equation

where M is the mass of the plate, S is the area of the plate, μ is the viscosity, ρ is the fluid density, and K is the stiffness (spring constant) of the spring.

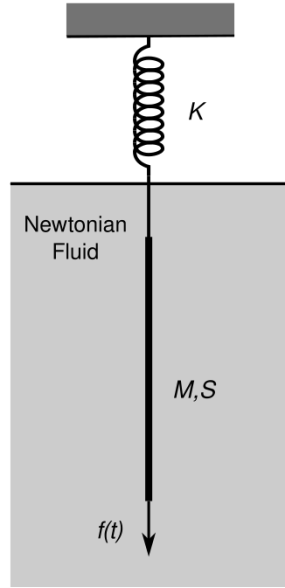


Figure 6. Thin plate submerged in Newtonian fluid suspended by a spring

The analytical solution of the initial value problem of the Bagley-Torvik equation is found to be [14]:

$$y(t) = \int_0^t G_3(t - \tau)f(\tau)d\tau \quad (38)$$

Equation 38. Analytical solution to BT equation

where $G_3(t)$ is the fractional Green's function for a specific case of the three-term FODE with constant coefficients defined by the Bagley-Torvik equation

$$G_3(t) = \frac{1}{A} \sum_{k=0}^{\infty} \frac{(-1)^k}{k!} \left(\frac{C}{A} \right)^k t^{2k+1} E_{\frac{1}{2}, 2+\frac{3k}{2}}^{(k)} \left(-\frac{B}{A} \sqrt{t} \right) \quad (39)$$

Equation 39. Fractional Green's function

that contains the k -th derivative of the Mittag-Leffler function

$$E_{\alpha, \beta}^{(k)}(y) \equiv \frac{d^k}{dy^k} E_{\alpha, \beta}(y) = \sum_{j=0}^{\infty} \frac{(j+k)! y^j}{j! \Gamma(\alpha j + \alpha k + \beta)}, \quad (k = 0, 1, 2, \dots) \quad (40)$$

Equation 40. k -th derivative of the two-parameter Mittag-Leffler function

These expressions are reproduced here from Appendix A for convenience. For the figures provided in this section, the follow function is used as the source term:

$$f(t) = \begin{cases} 8, & (0 \leq t < 1) \\ 0, & (t > 1) \end{cases} \quad (41)$$

Equation 41. Source term for the Bagley-Torvik Equation

The constants for all of the figures in this section were arbitrarily set as $A = 1$, $B = 0.5$, and $C = 0.5$ with figures generated using the code found in subsection B.2 adapted from the algorithm found in [14].

Grünwald-Letnikov Method

Discretizing of the fractional derivative in the Bagley-Torvik equation results in the following problem:

$$\begin{aligned} Ah^{-2}(y_k - 2y_{k-1} + y_{k-2}) + Bh^{-3/2} \sum_{j=0}^k \omega_j^{(3/2)} y_{k-j} + Cy_k &= f_k \\ y_0 &= 0, \quad \frac{y_1 - y_0}{h} = 0, \quad (k = 2, 3, \dots) \\ y_k &= y(kh), \quad f_k = f(kh), \quad (k = 0, 1, 2, \dots) \end{aligned} \quad (42)$$

Equation 42. Discretization of the Bagley-Torvik equation

Following the same procedure as in subsection 3.1, this discretization leads to the algorithm for determining the numerical solution using an explicit scheme .

$$\begin{aligned} Ah^{-2}(y_k - 2y_{k-1} + y_{k-2}) + Bh^{-3/2} \left(\omega_0^\alpha y_k + \sum_{j=1}^k \omega_j^{3/2} y_{k-j} \right) + Cy_k &= f_k \\ A(y_k - 2y_{k-1} + y_{k-2}) + h^{1/2} B \left(y_k + \sum_{j=1}^k \omega_j^{3/2} y_{k-j} \right) + h^2 Cy_k &= h^2 f_k \end{aligned}$$

$$Ay_k - A(2y_{k-1} - y_{k-2}) + B\sqrt{h}y_k + B\sqrt{h}\left(\sum_{j=1}^k \omega_j^{3/2} y_{k-j}\right) + h^2 Cy_k = h^2 f_k$$

$$Ay_k + B\sqrt{h}y_k = h^2 f_k - h^2 Cy_k + A(2y_{k-1} - y_{k-2}) + B\sqrt{h}\left(\sum_{j=1}^k \omega_j^{3/2} y_{k-j}\right)$$

$$(A + B\sqrt{h})y_k = h^2(f_k - Cy_k) + A(2y_{k-1} - y_{k-2}) + B\sqrt{h}\left(\sum_{j=1}^k \omega_j^{3/2} y_{k-j}\right)$$

By a first-order approximation and the term $Cy_k \approx Cy_{k-1}$, the Grünwald-Letnikov numerical solution is determined to be:

$$y_k = \frac{h^2(f_k - Cy_{k-1}) + A(2y_{k-1} - y_{k-2}) - B\sqrt{h}\sum_{j=1}^k \omega_j^{(3/2)} y_{k-j}}{A + B\sqrt{h}} \quad (43)$$

($k = 2, 3, \dots$)

Equation 43. Grünwald-Letnikov numerical solution to the Bagley - Torvik equation

where y_0 and y_1 were set to zero to satisfy the initial conditions given in Equation 37.

The output for Figure 7(a) returns flat curve of $y(t) = 0$. This result is expected for step sizes of $h = 1.0$ causing the very first time step to skip over the initial pulse term in Equation 41. The Grünwald-Letnikov algorithm therefore does not incorporate any forcing mechanism from the source term. This flat curve was detected for step sizes $h \geq 0.5$ reflecting the fact that the zero initial conditions from Equation 37 requires $y_0 = 0$, as defined by the problem, and $y_1 = 0$ as $y'(0) = \frac{y_0 - y_1}{h} = 0$. Thus, for $h < 0.5$, there was at least one grid point within the source term range with y_0 at grid point $t = 0$, y_1 at $t = h$, and thus y_2 at $t = 2h$ being the first point with no zero restriction imposed by the initial condition, is then able to detect the source function behavior as is demonstrated for the other step sizes in Figure 7. As with the previous relaxation-oscillation equation example, accuracy is seen to improve for smaller step sizes.

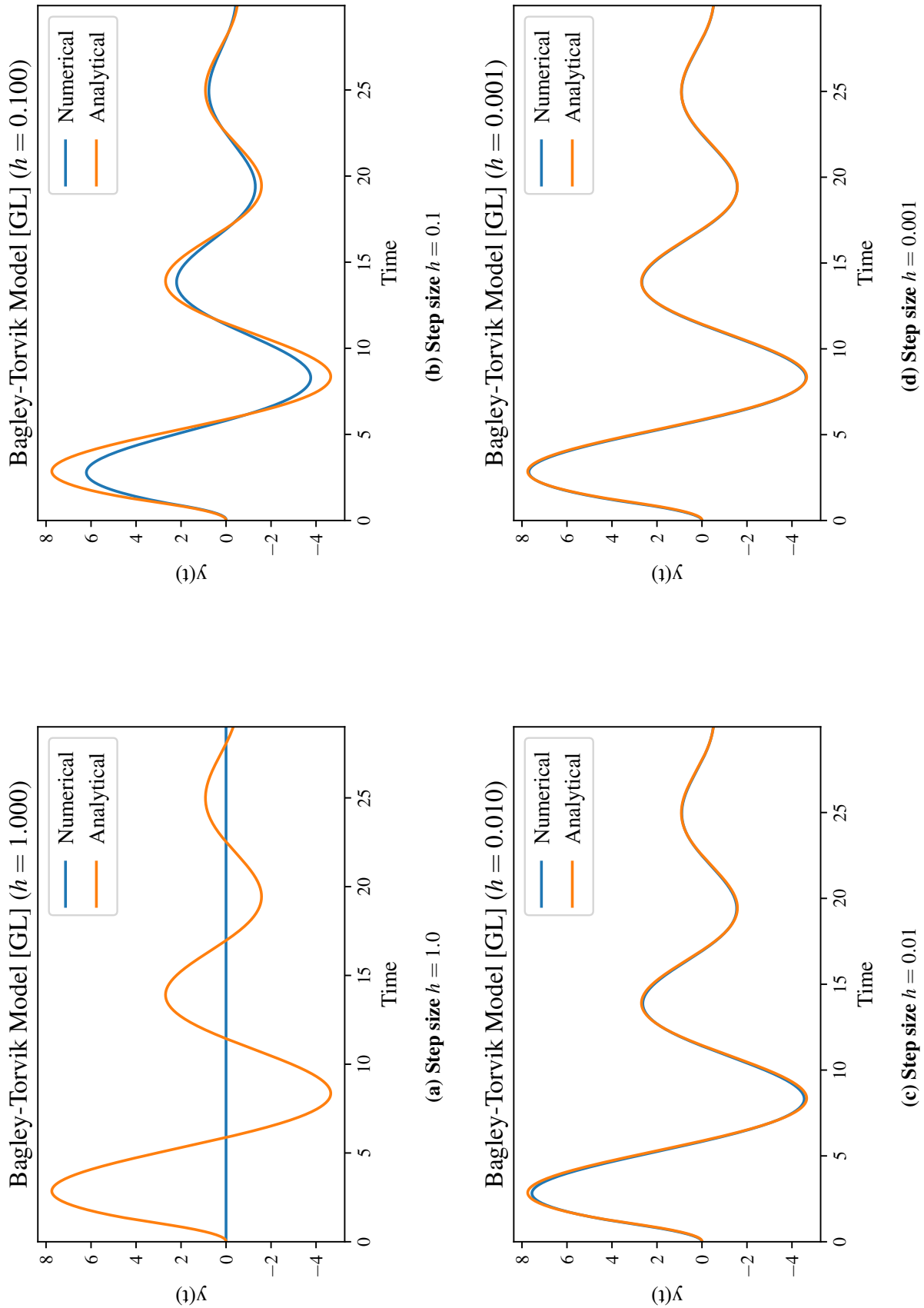


Figure 7. GL method numerical and analytical solutions for Bagley-Torvik for varying step sizes h

Matrix Method

Starting from Equation 29, the constants and matrices are defined by the Bagley-Torvik Equation 37 as follows.

- $m = 3$
- $\alpha_1 = 2, \alpha_2 = 3/2, \alpha_3 = 0, n = \lceil \max\{(\alpha_1, \alpha_2, \alpha_3)\} \rceil = 2$
- $p_1 = A, p_2 = B, p_3 = C$
- $B_{N-n}^{\alpha_1} = A * B_{N-2}^2, B_{N-n}^{\alpha_2} = B * B_{N-2}^{3/2}, B_{N-n}^{\alpha_3} = C * I$ (where I is identity matrix)
- $F_N = (\underbrace{F, F, \dots, F}_{kh < 1}, \underbrace{0, \dots, 0, 0}_{kh \geq 1})^T$ (where F is the initial impulse force, for this problem it is 8)

Thus the eigenvalue equation becomes:

$$(A * B_{N-2}^2 + B * B_{N-2}^{3/2} + C * I)(S_{0,1} Y_N) = S_{0,1} F_N \quad (44)$$

Equation 44. Matrix method BT

which, when expanded, appears as

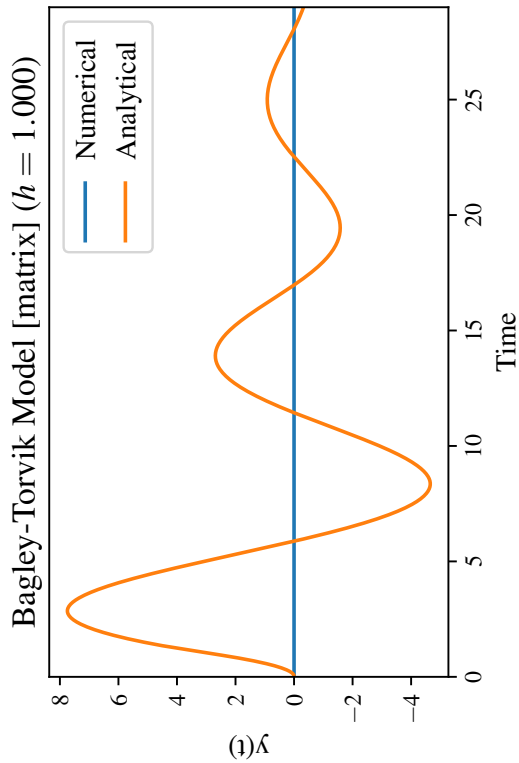
$$\left(\frac{A}{h^2} \begin{bmatrix} 1 & \dots & \dots & \dots & 0 \\ -2 & 1 & \ddots & \ddots & \vdots \\ 1 & -2 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 1 & -2 & 1 \end{bmatrix} + \frac{B}{h^{3/2}} \begin{bmatrix} \omega_0 & \dots & \dots & \dots & 0 \\ \omega_1 & \omega_0 & \ddots & \ddots & \vdots \\ \omega_2 & \omega_1 & \omega_0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \omega_{N-2} & \dots & \omega_2 & \omega_1 & \omega_0 \end{bmatrix} + C \begin{bmatrix} 1 & 0 & \dots & \dots & 0 \\ 0 & 1 & \ddots & \ddots & \vdots \\ \vdots & \ddots & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & \dots & \dots & 1 \end{bmatrix} \right) \begin{bmatrix} 0 & 0 & 1 & \dots & \dots & \dots & 0 \\ 0 & 0 & 0 & 1 & \dots & \dots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & \dots & \dots & \dots & \dots & 1 \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{N-2} \\ y_{N-1} \\ y_N \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & \dots & \dots & \dots & 0 \\ 0 & 0 & 0 & 1 & \dots & \dots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & \dots & \dots & \dots & \dots & 1 \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_{N-2} \\ f_{N-1} \\ f_N \end{bmatrix}$$

Performing the matrix operations results in the final eigenvalue problem for the Bagley-Torvik Equation.

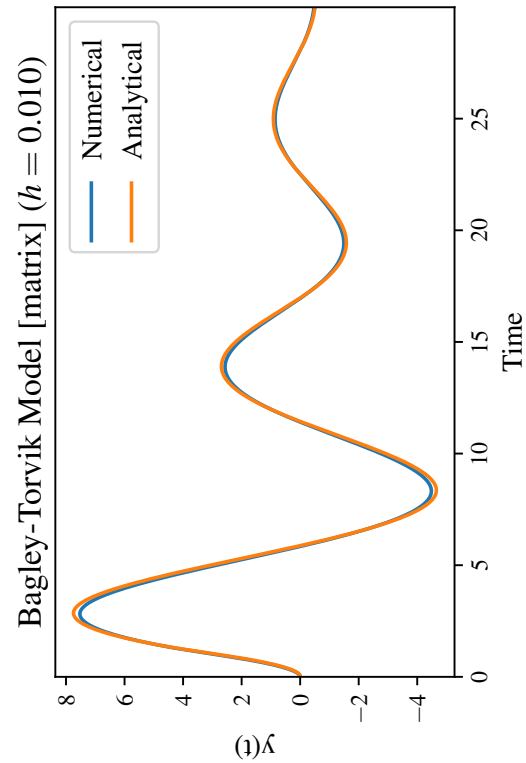
$$\begin{bmatrix}
\frac{A}{h^2} + \frac{B\omega_0}{h^{3/2}} + C & \dots & \dots & \dots & 0 \\
-\frac{2A}{h^2} + \frac{B\omega_1}{h^{3/2}} & \frac{A}{h^2} + \frac{B\omega_0}{h^{3/2}} + C & \ddots & \ddots & \vdots \\
\frac{A}{h^2} + \frac{B\omega_2}{h^{3/2}} & -\frac{2A}{h^2} + \frac{B\omega_1}{h^{3/2}} & \frac{A}{h^2} + \frac{B\omega_0}{h^{3/2}} + C & \ddots & \vdots \\
\frac{B\omega_3}{h^{3/2}} & \ddots & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \ddots & \vdots \\
\frac{B\omega_{N-2}}{h^{3/2}} & \dots & \frac{A}{h^2} + \frac{B\omega_2}{h^{3/2}} & -\frac{2A}{h^2} + \frac{B\omega_1}{h^{3/2}} & \frac{A}{h^2} + \frac{B\omega_0}{h^{3/2}} + C
\end{bmatrix}
\begin{bmatrix}
y_2 \\
y_3 \\
\vdots \\
y_{N-1} \\
y_N
\end{bmatrix}
=
\begin{bmatrix}
f_2 \\
f_3 \\
\vdots \\
f_{N-1} \\
f_N
\end{bmatrix}
\quad (45)$$

Equation 45. Matrix method BT (expanded)

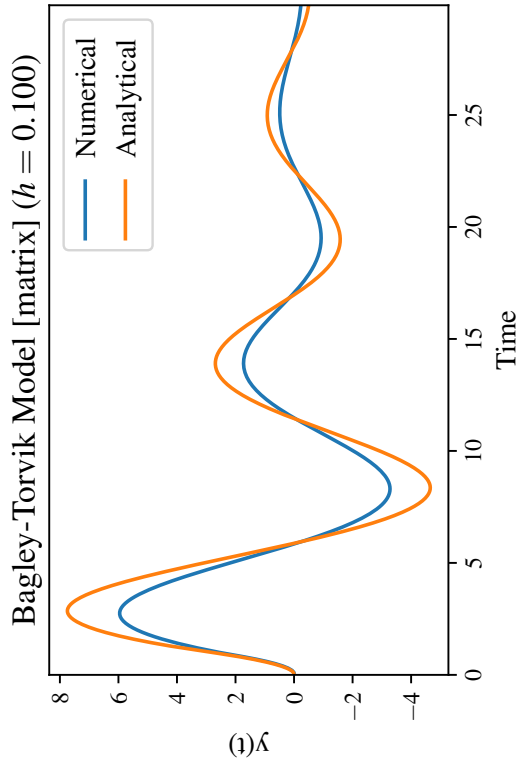
Similarly to the Grünwald-Letnikov method output plots, the output for Figure 8(a) returns a flat curve of $y(t) = 0$. The reasoning for this behavior is exactly the same in that there were no grid points within the active range of the source function in Equation 41. And thus, again for $h < 0.5$, there was at least one grid point within the source function range with y_2 at $t = 2h$, the first point with no zero restriction imposed by the initial condition, and was able to detect the source function behavior as is demonstrated for the other figures in Figure 8. The outputs for the Grünwald-Letnikov method and the matrix methods were very similar in Figure 7 and Figure 8, with accuracy again improving for smaller step sizes.



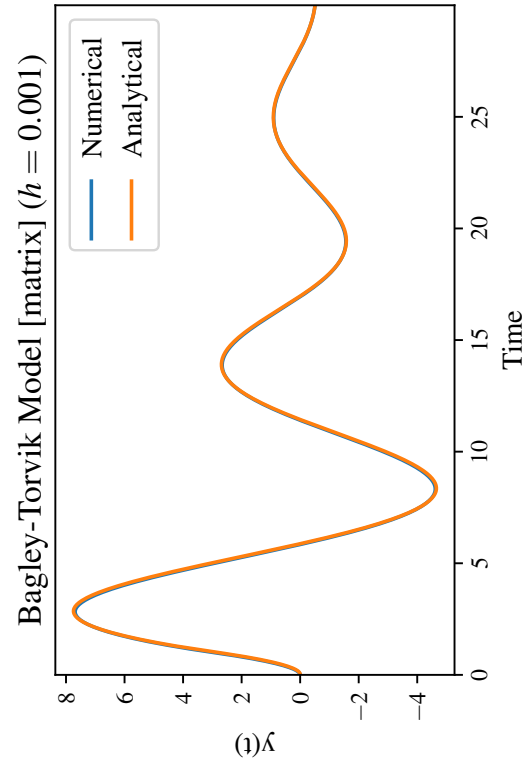
(a) Step size $h = 1.0$



(c) Step size $h = 0.01$



(b) Step size $h = 0.1$



(d) Step size $h = 0.001$

Figure 8. Matrix method numerical and analytical solutions for Bagley-Torvik for varying step sizes h

Computation Time and Numerical Error

Each of the Grünwald-Letnikov, matrix, and analytical solutions were timed and compared against each other. For each of the numerical solutions, the solutions were calculated for step sizes from 0.001 to 1.0 in increments of 0.001. Below step size of 0.005 resulted in matrix sizes that were increasingly memory intensive (time values in range [0,30] with step size 0.001 would result in matrix of size $30,000 \times 30,000$, or 6.9 gigabytes (GB) at the time of writing) sometimes causing the code to stall.

For each step size, the solutions were calculated and the computation time was measured 10 times, averaged, and plotted in Figure 9. Due to the nested summations in Equation 39, with one of them being a summation of an "infinite" series, the evaluation of the analytical solution quickly became prohibitively costly in terms of computation time. A compromise of 50 iterations for the summation of the "infinite" series was made to achieve a reasonable convergence of the analytical solution. Thus, the analytical solution computation times were measured for 40 runs and averaged with the result indicated by the four points in Figure 9. The analytical solution calculated for $N = 1000$ steps resulted in the best resolution with a relatively low computation time.

Table 2. Computation time results for the Bagley-Torvik equation

Step Size	GL (s)	Matrix (s)	Analytical (s)
0.001	1.0705	4.1456	117.4727
0.01	0.0388	0.1477	10.8770
0.1	0.0033	0.0010	1.549
1.0	0.0003	0.0001	0.1511

Like the computational time results in the relaxation-oscillation problem in Figure 4, the matrix method overtakes the Grünwald-Letnikov method in terms of computation time around a step size of 0.01.

Error was calculated using Equation 36. It can be seen in Figure 10 that the differences in mean squared error become negligible between the Grünwald-Letnikov and matrix methods for step sizes smaller than 0.01. Thus, the Grünwald-Letnikov numerical solution is the preferred method to achieve smaller error while incurring a smaller computational time cost for decreasing step sizes.

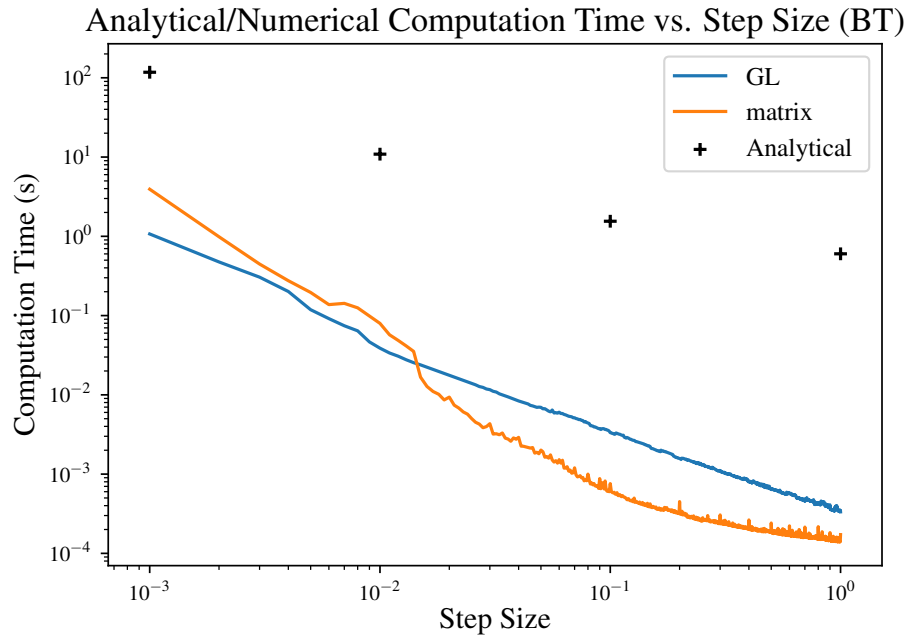


Figure 9. Computation time comparison of Grünwald-Letnikov, matrix, and analytical solutions (Bagley-Torvik Equation)

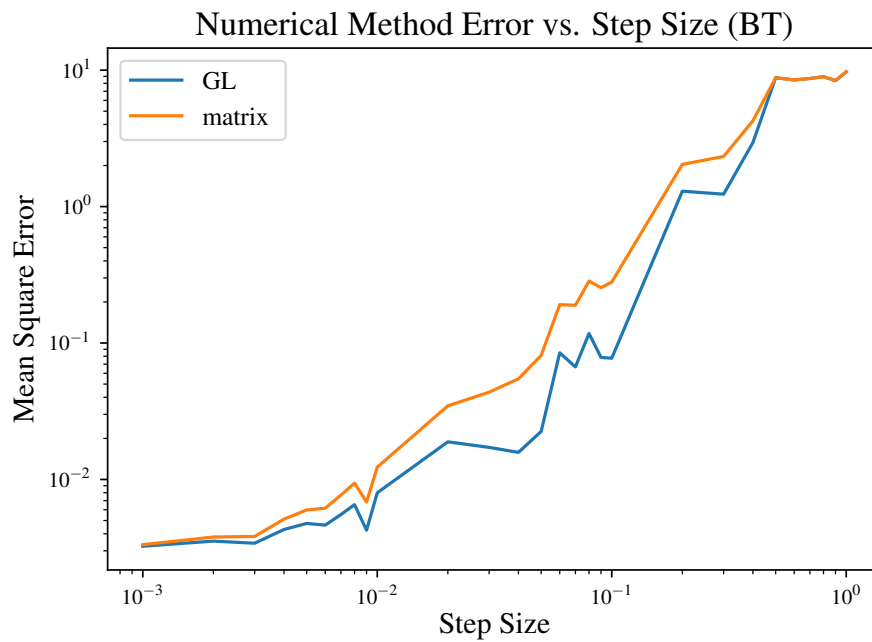


Figure 10. Computational error comparison of numerical methods (Bagley-Torvik equation)

3.4.3 Fractional Lumped Capacitance Model.

The lumped capacitance model of transient heat conduction is a differential equation that under certain conditions behaves similarly to that of a resistor-capacitor (RC) circuit described by

$$C \frac{dV(t)}{dt} + \frac{V(t)}{R} = Iu(t) \quad (46)$$

Equation 46. RC circuit voltage response

where C is the capacitance, V is the voltage, R is the resistance, I is the current, and $u(t)$ is the Heaviside unit step function defined in Equation A-2. The lumped capacitance model's temperature response is the analogue of the voltage response in the RC circuit of the form

$$\rho V c \frac{dT(t)}{dt} + \frac{k'A}{L} T(t) = \dot{E}_g \quad (47)$$

Equation 47. Lumped capacitance model

where ρ is density, c is specific heat, k' is thermal conductivity, V is volume, A is area, L is the thickness of the heated material, and $\dot{E}_g$ is the source term. In this case, $\dot{E}_g$ is the rate of thermal energy deposition in units of watts depicted in Figure 11. Solving the lumped capacitance model yields the temperature response, T , with respect to time, t .

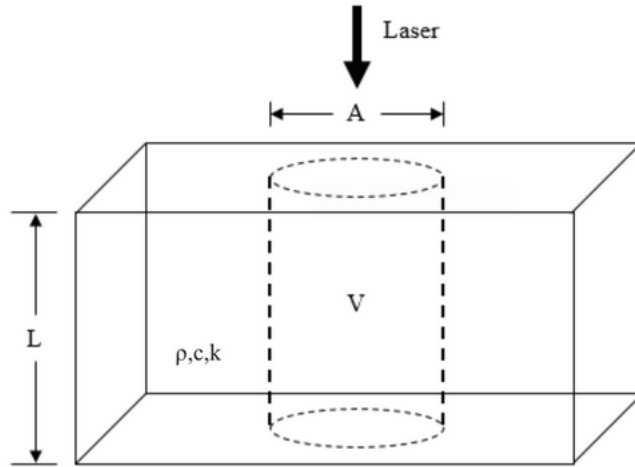


Figure 11. Fractional lumped capacitance experimental configuration

In the context of laser heating, however, thermal energy deposition is quick and confined producing heating rates of small volumes that traditional capacitance and diffusion models cannot accurately approximate under all conditions [23]. A proposed solution to address this limitation was to modify the lumped capacitance model to a more generalized model that incorporated a Riemann-Liouville fractional derivative [1]. The fractional lumped capacitance model is described by the equation

$$\rho V_C \frac{d^\alpha T(t)}{dt^\alpha} + \frac{k'A}{L} T(t) = {}_0 D_t^{-\beta} [u(t)] \dot{E}_g, \quad 0 < \alpha < 1, \quad \beta = 1 - \alpha. \quad (48)$$

Equation 48. Fractional lumped capacitance model

In the original RC circuit model, the unit step function $u(t)$ is used to mimic the instantaneous input of "switching on" the electrical current. While this description is generally accurate electric circuits, it is not realistic for heat conduction even with a heat source as fast as a laser. To compensate for this, a Riemann-Liouville fractional integral is taken of the unit step function as shown on the right hand side of Equation 48 which yields

$${}_0 D_t^{-\beta} [u(t)] = \frac{1}{\Gamma(\beta)} \int_0^t \frac{u(\tau)}{(t-\tau)^{(1-\beta)}} d\tau = \frac{t^\beta}{\Gamma(1+\beta)} = \frac{t^{1-\alpha}}{\Gamma(2-\alpha)}. \quad (49)$$

Equation 49. Riemann-Liouville fractional integral of the unit step function

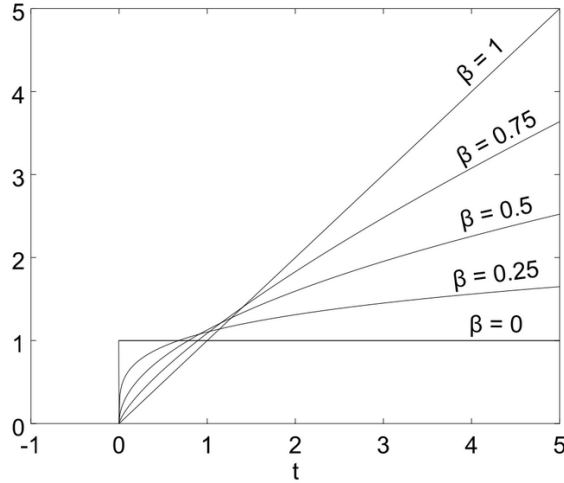


Figure 12. Riemann-Liouville fractional integral of the Heaviside function for select orders of β

Select results of the Riemann-Liouville fractional integral of the unit step function are shown in Figure 12. For $\beta = 1$, this is equivalent to the integral of the unit step function which is the ramp function. For $\beta = 0$, there is change to the unit step function. Intermediate values of β reveal fractional integration results of the unit step function that occupy the responses between that of $\beta = 0$ and $\beta = 1$ which more closely resemble realistic heating profiles [23], [24]. Being able to adjust the α and β values within the fractional lumped capacitance model to compensate for anomalous heating activity provides a powerful versatility not present in classical lumped capacitance models and diffusion models that are often used to approximate this phenomenon. Applying the Riemann-Liouville fractional integral of the unit step function is also necessary to balance the units of Equation 48. The $\frac{d^\alpha T(t)}{dt^\alpha}$ term and its coefficients produces an unusual unit of $[J/s^\alpha]$. In the second $T(t)$ term, the thermal conductivity k' is normally in units of $[W/mK]$ or $[J/smK]$. However, thermal conductivity is known to become dependent on temperature under certain conditions and is

therefore allowed to take on a fractional time dimension which produces $[J/s^\alpha mK]$ without significant consequence. This causes the units of the second term to become $[J/s^\alpha]$ which matches the units of the first term. Finally, the Riemann-Liouville fractional integral of the unit step function on the right hand side of Equation 48 results in a solution of $t^{1-\alpha}/\Gamma(2-\alpha)$ which yields a unit of $[s^{1-\alpha}]$. Combined with the unit for $\dot{E}_g$ which is in watts or $[J/s]$, the unit balance is complete with $[Js^{1-\alpha}/s]$ or $[J/s^\alpha]$.

The analytical solution for the fractional lumped capacitance model is easily obtained using the Laplace transform:

$$T(t) = btE_{\alpha,2}(-at^\alpha) + Ct^{\alpha-1}E_{\alpha,\alpha}(-at^\alpha) \quad (50)$$

Equation 50. Fractional lumped capacitance analytical solution

where

$$a = \frac{k'A}{\rho V c L}, \quad b = \frac{\dot{E}_g}{\rho V c}, \quad C = {}_0 D_t^{\alpha-1}[T(t)]_{t=0}. \quad (51)$$

Equation 51. Fractional lumped capacitance model constants

The solution represented by Equation 50 has two distinct terms. The heating term

$$T_H(t) = btE_{\alpha,2}(-at^\alpha) \quad (52)$$

Equation 52. Heating term

and the cooling term

$$T_C(t) = Ct^{\alpha-1}E_{\alpha,\alpha}(-at^\alpha) \quad (53)$$

Equation 53. Cooling term

With the governing equation and solution established for the problem, a numerical approximation can be formulated. For the rest of this section, k' denotes the thermal conductivity while k denotes the numerical time step of the GL model. Replacing the fractional derivative with the Grünwald-Letnikov numerical scheme produces a finite difference numerical approximation:

$$\rho V c h^{-\alpha} \sum_{j=0}^k \omega_j^\alpha y_{k-j} + \frac{k'A}{L} y_k = f_k. \quad (54)$$

Equation 54. Grünwald-Letnikov numerical scheme for the fractional lumped capacitance equation

Solving for y_k yields the explicit numerical solution scheme for the fractional lumped capacitance equation.

$$y_k = -ah^\alpha y_{k-1} - \sum_{j=1}^k \omega_j^\alpha y_{k-j} - f_k, \quad (k = n, n+1, \dots) \quad (55)$$

Equation 55. Explicit numerical scheme of the fractional lumped capacitance equation

where $n - 1 < \alpha < n$ for an integer n , a defined in Equation 51, and f_k defined by

$$f_k = \begin{cases} \frac{t_k^{1-\alpha}}{\Gamma(2-\alpha)} \dot{E}_g, & (t_k \leq t_{peak}) \\ 0, & (t_k > t_{peak}) \end{cases} \quad (56)$$

Equation 56. Fractional lumped capacitance source term

where t_{peak} is the time at which the heating shuts off and cooling begins. The discontinuity brought upon by zeroing the heating term during the simulation is accounted for through the finite difference scheme by assigning the heating term the parameters a_1 , α_1 and the cooling curve the parameters a_2 , α_2 :

$$y_k = \begin{cases} -a_1 h^{\alpha_1} y_{k-1} - \sum_{j=1}^k \omega_j^{\alpha_1} y_{k-j} - \frac{t_k^{1-\alpha_1}}{\Gamma(2-\alpha_1)} \dot{E}_g, & (t_k \leq t_{peak}) \\ -a_2 h^{\alpha_2} y_{k-1} - \sum_{j=1}^k \omega_j^{\alpha_2} y_{k-j} - 0, & (t_k > t_{peak}) \end{cases} \quad (57)$$

Equation 57. Numerical solution of the fractional lumped capacitance equation

The above piecewise finite difference method takes into account the function values during heating ($t_k \leq t_{peak}$) while calculating the cooling curve ($t_k > t_{peak}$) and storing it in the y_k values calculated. To verify the performance of the fractional lumped capacitance model, experimental data of measured temperature change of porcine skin exposed to a 0.01 second, 2780 Watt laser pulse was used as a test case. An additional fit using a numerical model of the Pennes bioheat transfer equation is also included for comparison [23]. Figure 13 and Figure 14 were generated using the code in subsection B.3 and the data sets in Attachment 1.

A discontinuity is apparent in the analytical fit for the lumped capacitance model in Figure 13. The heating analytical fit was performed separately on only the points in the initial rising curve while the cooling analytical fit was performed separately on only the points in the subsequent falling curve with the points offset in time so the first cooling point (the point at the peak) was at $t = 0$. The singularity in the cooling portion of the fit is a result of the $t^{\alpha-1}$ term in Equation 53. For $\alpha < 1$ the values of t near zero are undefined. This problem is nonexistent in the numerical solution in Figure 14 since, unlike the analytical fit, the solution did not require any offset and was able to provide a continuous piecewise curve fit of the experimental data. The constants used for the analytical and numerical fit parameters to the experimental data of a 2780 watt laser pulse on porcine skin are provided in Table 3 and Table 4 respectively. These values were determined at the time of coding to produce the best fit possible and were found to be within the limits of an acceptable range of values according to the literature [1], [24].

Figure 13 displays good agreement even with the singularity present at the beginning of the cooling curve using the analytical solution whereas Figure 13 eliminates the singularity entirely. Both

Table 3. Constants for analytical fit

Parameter	Heating	Cooling
a	10.9	2.625
b	3600	0
C	0	17.5
α	0.841	0.906
$\dot{E}_g$	2780	0

Table 4. Constants for numerical fit

Parameter	Heating	Cooling
a	10.9	0.1
b	3600	0
C	0	17.5
α	0.841	0.855
$\dot{E}_g$	2780	0

figures also show remarkable improvement over the Pennes bioheat transfer model for these experimental parameters.

Fractional Lumped Capacitance Model (Analytical)

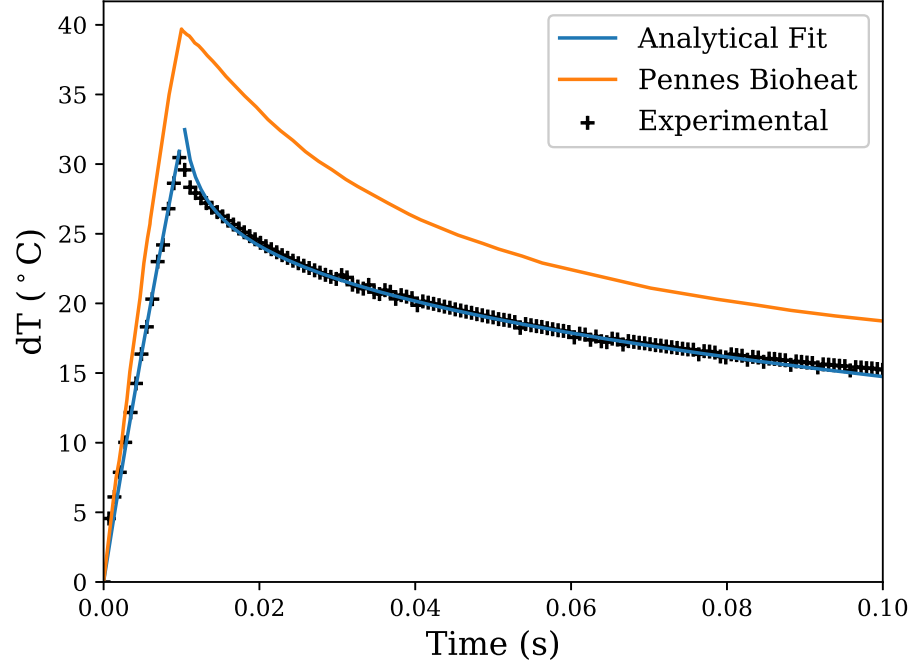


Figure 13. Experimental data fit using analytical solution to the fractional lumped capacitance model

Fractional Lumped Capacitance Model (Numerical)

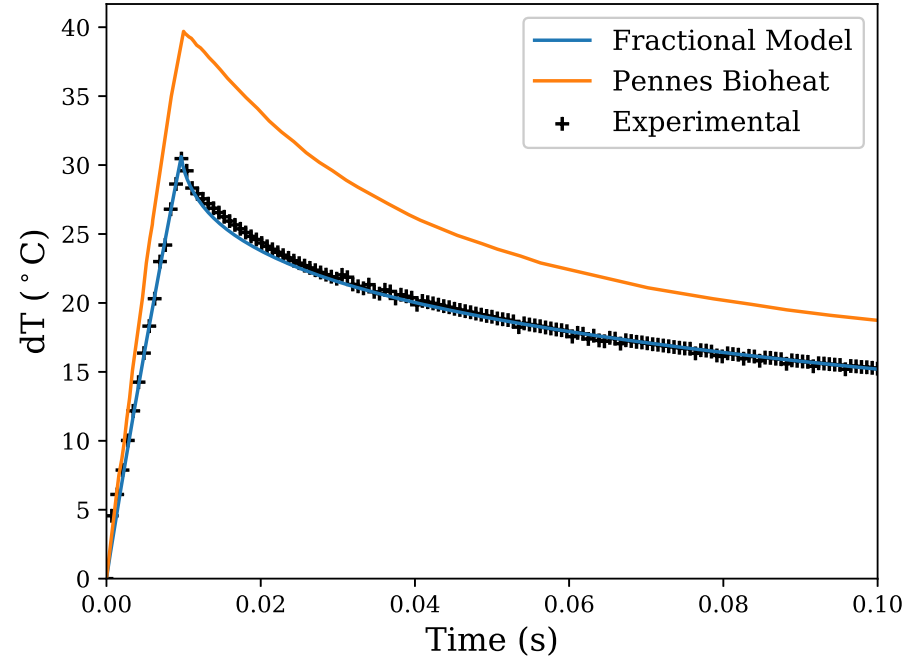


Figure 14. Experimental data fit using numerical solution to the fractional lumped capacitance model

4.0 NUMERICAL METHODS FOR SOLVING FRACTIONAL PARTIAL DIFFERENTIAL EQUATIONS

This section contains an overview of select numerical solutions to fractional partial differential equations (FPDEs) pertinent to the directed energy problems of diffusion and wave physics. Their algorithms stem from concepts introduced in section 3 similar to those of traditional FPDEs.

4.1 Time-Fractional Diffusion Equations

The time-fractional diffusion equation is an FPDE in one space dimension of the form

$${}_a^C D_t^\alpha [y(x,t)] = K(x,t) \frac{\partial^2}{\partial x^2} y(x,t) + f(x,t) \quad (58)$$

Equation 58. Time-fractional diffusion equation

for a grid bounded by $t \in [0, T]$ and $x \in [a, b]$. For this specific problem, $0 < \alpha < 1$ and K is a strictly positive function. The solutions are subject to the initial conditions

$$y(x, 0) = y_0(x) \quad \text{for } x \in [a, b]$$

and the boundary conditions

$$y(a, t) = r_1(t) \quad \text{and} \quad y(b, t) = r_2(t) \quad \text{for } t \in [0, T]$$

with the functions, $K(x, t)$, $f(x, t)$, $y_0(x)$, $r_1(t)$, and $r_2(t)$ defined by the problem.

The development of the finite difference proceeds as by shown in [4] and is expanded here for the reader's convenience. In this algorithm, N is the discretization parameter for the mesh with respect to the space variables, $h = (b - a)/N$ and is the corresponding step size, M is the discretization parameter for the time mesh, and $m = T/M$ is the time step where T is the upper bound of the time interval defined for the solution. Thus, notation wise, $y_{j,k}$ means the y -value at position x_j and at time t_k . The grid points are then:

$$\{(x_j = a + jh; t_k = mk) : j = 0, 1, \dots, N; k = 0, 1, \dots, M\} \quad (59)$$

Equation 59. Grid points for the 1D finite difference solution

The term containing the second-order spatial derivative in Equation 58 is approximated by a traditional second-order central difference stencil:

$$K(x, t) \frac{\partial^2}{\partial x^2} y(x, t) \approx \frac{K(x_j, t_k)}{h^2} (y_{j+1,k} - 2y_{j,k} + y_{j-1,k})$$

which leaves just the Caputo fractional derivative to approximate. Numerical approximations developed thus far in this study have only been Grünwald-Letnikov as well as Riemann-Liouville

derivatives due to their equivalence described in subsubsection 2.2.1. Fortunately, the Caputo and Riemann-Liouville fractional derivatives are related by

$${}_a^C D_t^\alpha [y(x, t)] = {}_a^{RL} D_t^\alpha [(y - T_{[\alpha]-1}[y; a])(x, t)] \quad (60)$$

Equation 60. Relationship between RL and Caputo derivatives

where $T_{[\alpha]-1}[y; a](x, t)$ is the Taylor polynomial of degree $[\alpha] - 1$ centered at a defined as:

$$T_{n-1}[y; a](t) = \sum_{j=0}^{n-1} \frac{y^{(j)}(a)}{j!} (t - a)^j \quad (61)$$

Equation 61. Taylor polynomial of degree $n - 1$

where $n = [\alpha]$ and $T_{n-1}[y; a]$ [4]. Substituting Equation 61 into Equation 60 gives

$${}_a^C D_t^\alpha [y(t)] = {}_a^{RL} D_t^\alpha \left[y(t) - \sum_{j=0}^{n-1} \frac{y^{(j)}(a)}{j!} (t - a)^j \right] \quad (62)$$

Equation 62. Relationship between RL and Caputo derivatives with Taylor polynomial definition

Applying the linearity property of the fractional derivative to Equation 62 then yields

$${}_a^{RL} D_t^\alpha \left[y(t) - \sum_{j=0}^{n-1} \frac{y^{(j)}(a)}{j!} (t - a)^j \right] = {}_a^{RL} D_t^\alpha [y(t)] - {}_a^{RL} D_t^\alpha \left[\sum_{j=0}^{n-1} \frac{y^{(j)}(a)}{j!} (t - a)^j \right]$$

Performing the Riemann-Liouville fractional derivative on the Taylor polynomial in Equation 62 gives the result [25]:

$${}_a^C D_t^\alpha [y(t)] = {}_a^{RL} D_{a,t}^\alpha [y(t)] - \sum_{j=0}^{n-1} \frac{y^{(j)}(a)(t - a)^{j-\alpha}}{\Gamma(j + 1 - \alpha)} \quad (63)$$

Equation 63. Equation 62 after evaluating Riemann-Liouville derivative of the Taylor polynomial

For this development the basic case of $a = 0$ will be used meaning that the Taylor polynomial will be centered at 0 ($T_{n-1}[y; 0]$) which forces the lower terminal a of the Riemann-Liouville (${}_a^{RL} D_t^\alpha$) and Caputo (${}_a^C D_t^\alpha$) fractional derivative operators also to $a = 0$ and are defined henceforth by ${}_0^{RL} D_t^\alpha$ and ${}_0^C D_t^\alpha$ respectively. This section discusses the cases for $0 < \alpha \leq 1$. Cases for the Riemann-Liouville derivative of the Taylor polynomial for $1 < \alpha \leq 2$ are discussed in subsection 4.2. The formulation of the numerical solution begins with the Taylor polynomial

$$T_{n-1}[y; 0](x_j, t_k) = y_0(x_j), \quad 0 < \alpha \leq 1$$

with the resulting Riemann-Liouville fractional derivative of the Taylor polynomial for $0 < \alpha \leq 1$ being

$${}^{RL}D_t^\alpha[(T_{n-1}[y;0])(x_j, t_k)] = \frac{1-\alpha}{\Gamma(2-\alpha)} t_k^{-\alpha} y_0(x_j).$$

Substituting in this result and discretizing the operator in a fashion similar to the method discussed in subsection 3.2 yields:

$$\begin{aligned} {}^CD_t^\alpha[y(x, t)] &= {}^{RL}D_t^\alpha[y(x, t)] - \frac{1-\alpha}{\Gamma(2-\alpha)} t_k^{-\alpha} y_0(x) \\ &\approx \frac{1}{m^\alpha \Gamma(2-\alpha)} \sum_{l=0}^k B_{l,k} y_{j,l} - \frac{1-\alpha}{\Gamma(2-\alpha)} t_k^{-\alpha} y_0(x_j) \\ &\approx \frac{1}{m^\alpha \Gamma(2-\alpha)} B_{k,k} y_{j,k} + \frac{1}{m^\alpha \Gamma(2-\alpha)} \sum_{l=0}^{k-1} B_{l,k} y_{j,l} - \frac{1-\alpha}{\Gamma(2-\alpha)} t_k^{-\alpha} y_0(x_j) \end{aligned}$$

where the weights $B_{l,k}$ are defined as

$$B_{l,k} = \begin{cases} (k-1)^{1-\alpha} - (k-1+\alpha)k^{-\alpha} & \text{for } l=0 \\ (k-l+1)^{1-\alpha} - 2(k-l)^{1-\alpha} + (k-l-1)^{1-\alpha} & \text{for } 1 \leq l \leq k-1 \\ 1 & \text{for } k=l \geq 1 \end{cases} \quad (64)$$

Equation 64. B weights

which are the product trapezoidal quadrature weights defined in [26] for this algorithm. The implicit scheme is obtained by substituting the discretizations above into Equation 58:

$$\begin{aligned} \frac{1}{m^\alpha \Gamma(2-\alpha)} B_{k,k} y_{j,k} + \frac{1}{m^\alpha \Gamma(2-\alpha)} \sum_{l=0}^{k-1} B_{l,k} y_{j,l} - \frac{1-\alpha}{\Gamma(2-\alpha)} t_k^{-\alpha} y_0(x) \\ = \frac{K(x_j, t_k)}{h^2} (y_{j+1,k} - 2y_{j,k} + y_{j-1,k}) + f(x_j, t_k). \end{aligned}$$

Solving for $y_{j,k}$, given $B_{k,k} = 1$ produces

$$\begin{aligned} y_{j,k} &= \Gamma(2-\alpha) \frac{m^\alpha}{h^2} K(x_j, t_k) (y_{j+1,k} - 2y_{j,k} + y_{j-1,k}) + \Gamma(2-\alpha) m^\alpha f(x_j, t_k) \\ &\quad + (1-\alpha) t_k^{-\alpha} y_0(x_j) - \sum_{l=0}^{k-1} B_{l,k} y_{j,l}. \end{aligned}$$

Finally, accounting for the boundary condition functions $r_1(t)$ and $r_2(t)$, the implicit numerical scheme for solving the fractional diffusion equation becomes

$$\begin{aligned}
y_{0,k} &= r_1(t_k), \\
y_{j,k} &= \Gamma(2-\alpha) \frac{m^\alpha}{h^2} K(x_j, t_k) (y_{j+1,k} - 2y_{j,k} + y_{j-1,k}) \\
&\quad + \Gamma(2-\alpha) m^\alpha f(x_j, t_k) + (1-\alpha) t_k^{-\alpha} y_0(x_j) \\
&\quad - \sum_{l=0}^{k-1} B_{l,k} y_{j,l} \quad (1 \leq j < N) \\
y_{N,k} &= r_2(t_k)
\end{aligned} \tag{65}$$

Equation 65. Numerical for fractional diffusion equation FPDE

for $k = 0, 1, 2, \dots, M$. Recall that y_0 is the initial condition for $0 < \alpha \leq 1$ and r_1 and r_2 are the boundary conditions.

It is convenient to rewrite the system in matrix-vector form:

$$\begin{bmatrix} y_{0,k} \\ y_{1,k} \\ y_{2,k} \\ \vdots \\ y_{N-1,k} \\ y_{N,k} \end{bmatrix} = \Gamma(2-\alpha) \frac{m^\alpha}{h^2} G_k \begin{bmatrix} y_{0,k} \\ y_{1,k} \\ y_{2,k} \\ \vdots \\ y_{N-1,k} \\ y_{N,k} \end{bmatrix} + H_k \tag{66}$$

Equation 66. Numerical scheme time-fractional diffusion equation

with the vector H_k defined as

$$H_k = \begin{bmatrix} r_1(t_k) \\ (1-\alpha) t_k^{-\alpha} y_0(x_1) + \Gamma(2-\alpha) m^\alpha f(x_1, t_k) - \sum_{l=0}^{k-1} B_{l,k} y_{1,l} \\ (1-\alpha) t_k^{-\alpha} y_0(x_2) + \Gamma(2-\alpha) m^\alpha f(x_2, t_k) - \sum_{l=0}^{k-1} B_{l,k} y_{2,l} \\ \vdots \\ (1-\alpha) t_k^{-\alpha} y_0(x_{N-1}) + \Gamma(2-\alpha) m^\alpha f(x_{N-1}, t_k) - \sum_{l=0}^{k-1} B_{l,k} y_{N-1,l} \\ r_2(t_k) \end{bmatrix} \tag{67}$$

Equation 67. Vector H_k

and the matrix G_k defined as

$$G_k = \begin{bmatrix} 0 & 0 & \dots & \dots & \dots & 0 \\ K(x_j, t_k) & -2K(x_j, t_k) & K(x_j, t_k) & 0 & \ddots & \vdots \\ 0 & K(x_j, t_k) & -2K(x_j, t_k) & K(x_j, t_k) & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & 0 & K(x_j, t_k) & -2K(x_j, t_k) & K(x_j, t_k) \\ 0 & \dots & \dots & 0 & 0 & 0 \end{bmatrix}. \quad (68)$$

Equation 68. Matrix G_k

Note how the first and last rows in matrix G_k are set to all zero to account for the boundary conditions of the problem. The solution vector $Y_k = (y_{0,k}, y_{1,k}, \dots, y_{N,k})^T$ at the k -th time step can be obtained as the solution of the linear system:

$$\left(I - \Gamma(2 - \alpha) \frac{m^\alpha}{h^2} G_k \right) Y_k = H_k \quad (69)$$

Equation 69. Eigenvalue problem for time-fractional diffusion equation

where I is the identity matrix. Two examples with different parameters have been worked as test cases for this algorithm. The code used to generate Figure 15 and Figure 16 is provided in subsection B.4. At the time of writing, no appropriate analytical solutions were available to perform computational error and run time analysis with and have therefore been omitted.

Example 1

The specific case of the time-fractional diffusion equation appearing as

$${}^C D_t^{0.5}[u(x,t)] - \frac{\partial^2}{\partial x^2} u(x,t) = 0 \quad (70)$$

Equation 70. FPDE example 1

is solved on $[0, \pi] \times [0, 10]$ with the initial conditions

$$u(x,0) = \sin(x), \quad u_t(x,0) = 0$$

and boundary conditions $u(0,t) = 0, u(\pi,t) = 0$. In this case:

$$f(x,t) = 0$$

$$K(x,t) = 1$$

The result is given in Figure 15.

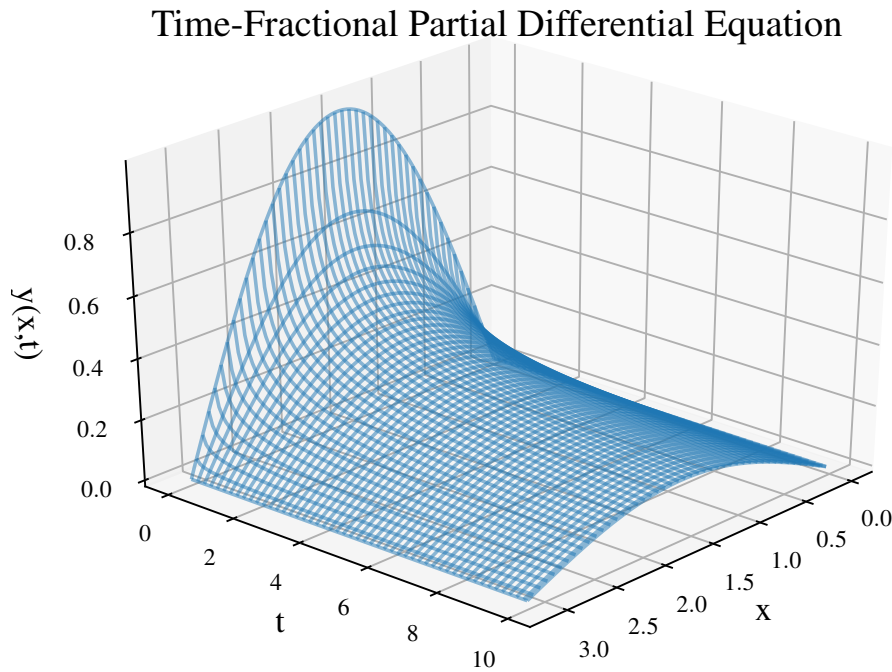


Figure 15. Time-fractional PDE example 1

Example 2

This initial-boundary value problem appears in [4] and is used as a more complicated test case compared to the previous problem. The equation and figures were reproduced for verification and are as follows:

$${}^C D_t^{0.7} y(x, t) = \left(2 + \frac{1 + x(2\pi - x)}{5 + 5t} \right) \frac{\partial^2}{\partial x^2} y(x, t) + \cos x \cos t \quad (71)$$

Equation 71. FPDE example 2

for $0 \leq t \leq 10$ and $0 \leq x \leq 2\pi$ with initial condition

$$y(x, 0) = \cos \frac{x}{2}$$

and the boundary conditions

$$y(0, t) = \exp(-t) \quad \text{and} \quad y(2\pi, t) = -\exp(-t).$$

In this case:

$$f(x, t) = \cos x \cos t$$

and

$$K(x, t) = \left(2 + \frac{1 + x(2\pi - x)}{5 + 5t} \right).$$

The result is given in Figure 15 and appears to match those found in [4].

Time-Fractional Partial Differential Equation

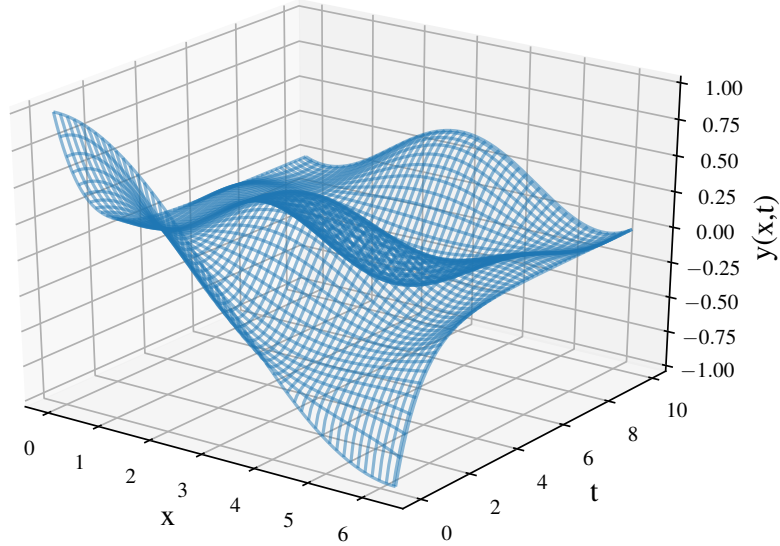


Figure 16. Time-fractional PDE example 2

4.2 Diethelm-Ford-Weilbeer Method for Time-Fractional Diffusion-Wave Equations with Dirichlet-Neumann Boundary Conditions

The time-fractional diffusion equation is revisited in this section to showcase a numerical FPDE method with Dirichlet-Neumann boundary conditions and a larger range of α values [21].

4.2.1 Definition of problem.

The time-fractional diffusion-wave equation in multiple dimensions is in the form of

$${}^C D_t^\alpha u(x,t) + \phi(x,t) \sum_{j=1}^n \frac{\partial^2}{\partial x_j^2} u(x,t) = f(x,t) \quad (72)$$

Equation 72. Fractional diffusion-wave equation in n -dimensions

for $x = (x_1, \dots, x_n) \in G \subset \mathbb{R}^n$ and $t \in [0, T]$. Here, the notation ${}^C D_t^\alpha u(x,t)$ indicates the partial derivative of Caputo type of order α of the function u with respect to t . The sub-diffusion equation is defined in the range $0 < \alpha < 1$ and the diffusion-wave equation is defined in the range $1 < \alpha < 2$. When $\alpha = 1$, the equation becomes the classical diffusion equation. When $\alpha = 2$, the equation becomes the classical wave equation. For sake of instruction, the time-fractional diffusion-wave equation in the case of one spatial dimension is considered:

$${}^C D_t^\alpha [u(x, t)] + \phi(x, t) \frac{\partial^2}{\partial x^2} u(x, t) = f(x, t) \quad (73)$$

Equation 73. Fractional diffusion-wave equation (one space dimension)

with mixed Dirichlet-Neumann boundary conditions of

$$\begin{aligned} g_1(t)u(a, t) + h_1(t) \frac{\partial}{\partial x} u(a, t) &= r_1(t) \\ g_2(t)u(b, t) + h_2(t) \frac{\partial}{\partial x} u(b, t) &= r_2(t) \end{aligned} \quad (74)$$

Equation 74. Mixed Dirichlet-Neumann boundary conditions

where a and b are the left and right boundaries of the problem $x \in [a, b]$ respectively. The initial conditions are defined as:

$$\begin{aligned} u(x, 0) &= u_0(x), & \text{for } 0 < \alpha \leq 1 \\ u(x, 0) &= u_0(x), \quad \frac{\partial}{\partial t} u(x, 0) = u_1(x), & \text{for } 1 < \alpha \leq 2 \end{aligned}$$

4.2.2 Discretization of the problem.

To find the numerical solution of the fractional diffusion, an implicit approach is taken via matrix inversion that solves the fractional diffusion-wave equation at each time step. Using the discretization on the space axis, the second derivative $u_{xx}(x, t)$ is approximated by the second order central difference stencil

$$u_{xx}(x, t) = \frac{1}{\Delta x^2} \begin{bmatrix} d_1 & o_1 & & 0 \\ 1 & -2 & 1 & \\ & \ddots & \ddots & \ddots \\ & & 1 & -2 & 1 \\ 0 & & & o_2 & d_2 \end{bmatrix} \begin{bmatrix} u(a, t) \\ u(x_1, t) \\ \vdots \\ u(x_{N-1}, t) \\ u(b, t) \end{bmatrix} \quad (75)$$

Equation 75. Central difference matrix discretization

As in section 3, the discretization of the Riemann-Liouville fractional derivative $D_t^\alpha [u(x, t)]$ can be defined by

$$D_t^\alpha [u(x, t)] = \frac{1}{\Delta t^\alpha} \begin{bmatrix} \omega_{0,0} & & 0 \\ \vdots & \ddots & \\ \omega_{M,0} & \dots & \omega_{M,M} \end{bmatrix} \begin{bmatrix} u(x, t_0) \\ \vdots \\ u(x, t_M) \end{bmatrix} \quad (76)$$

Equation 76. Lower triangular matrix discretization for a fractional derivative

As previously explained in subsection 4.1, the distinction between the Riemann-Liouville and Caputo derivative must be accounted for. Substituting the relationship from Equation 60 results in the new fractional diffusion-wave equation

$${}^{RL}D_t^\alpha u(x,t) - {}^{RL}D_t^\alpha (T_{n-1}[u;0](x,t)) + \phi(x,t) \frac{\partial^2}{\partial x^2} u(x,t) = f(x,t) \quad (77)$$

Equation 77. Fractional diffusion-wave equation with RL derivative

where $n = \lceil \alpha \rceil$. In this section, the Taylor polynomial definition and Riemann-Liouville fractional derivative of the Taylor polynomial for the range $1 < \alpha \leq 2$ are determined to be

$$\begin{aligned} T_{n-1}[u;0](x,t) &= u_0(x), & 0 < \alpha \leq 1 \\ T_{n-1}[u;0](x,t) &= u_0(x) + tu_1(x), & 1 < \alpha \leq 2 \end{aligned}$$

The expression ${}^{RL}D_t^\alpha (T_{n-1}[u;0])(x,t)$ is the Riemann-Liouville fractional derivative of the Taylor polynomial with respect to the time variable. Below are the results of the Riemann-Liouville fractional derivative of the Taylor polynomial for pertinent values of α [25]:

$$\begin{aligned} {}^{RL}D_t^\alpha [(T_{n-1}[u;0])(x,t)] &= \frac{1-\alpha}{\Gamma(2-\alpha)} t^{-\alpha} u_0(x), & 0 < \alpha \leq 1 \\ {}^{RL}D_t^\alpha [(T_{n-1}[u;0])(x,t)] &= \frac{1-\alpha}{\Gamma(2-\alpha)} t^{-\alpha} u_0(x) + \frac{1}{\Gamma(2-\alpha)} t^{1-\alpha} u_1(x), & 1 < \alpha \leq 2 \end{aligned} \quad (78)$$

Equation 78. RL derivative for different α -values

The eigenvalue problem can now be formulated. For each time step t_k , an eigenvalue problem is generated to solve for values $u(x, t_k)$ for all x -values. Thus, the vector for $u(x, t_k)$ has dimensions of $N \times 1$. Known values in the discretization of the fractional diffusion-wave equation were separated and moved to the right-hand side (RHS) vector of dimension $N \times 1$ of the eigenvalue problem. These values include known source function values $f(x, t_k)$, the Riemann-Liouville fractional derivative of the Taylor polynomial ${}^{RL}D_t^\alpha T_{n-1}[u;0](x, t_k)$, and the elements of the Grünwald-Letnikov numerical scheme for $t_{k-1}, \dots, 0$. The right-hand side (RHS) of the discretization scheme then becomes

$$\text{RHS} = \begin{bmatrix} r_1 \\ f(x_1, t_k) \\ \vdots \\ f(x_{N-1}, t_k) \\ r_2 \end{bmatrix} + \begin{bmatrix} 0 \\ {}^{RL}D_t^\alpha [(T_{n-1}[u;0])(x_1, t_k)] \\ \vdots \\ {}^{RL}D_t^\alpha [(T_{n-1}[u;0])(x_{N-1}, t_k)] \\ 0 \end{bmatrix} - \begin{bmatrix} 0 \\ \sum_{v=0}^{k-1} \omega_{v,k} u(x_1, t_v) \\ \vdots \\ \sum_{v=0}^{k-1} \omega_{v,k} u(x_{N-1}, t_v) \\ 0 \end{bmatrix}. \quad (79)$$

Equation 79. RHS of discretization scheme

The matrix entries d_1, o_1, d_2, o_2 and the vector entries r_1, r_2 are determined by the initial and boundary conditions using Taylor series approximation. Denoting $\xi_1 = a$ and $\xi_2 = b$, they are given for $i = 1, 2$ at the time step t_k by

$$\begin{aligned}
d_i &= \frac{g_i(t_k)}{\Delta x} + h_i(t_k)(-1)^i \left[\frac{1}{\Delta x^2} - \frac{\omega_{k,k}}{2\phi(\xi_i, t_k)\Delta t^\alpha} \right] \\
o_i &= (-1)^{i+1} \frac{h_i(t_k)}{\Delta x^2} \\
r_i &= \frac{r_i(t_k)}{\Delta x} + \frac{(-1)^{i+1} h_i(t_k)}{2\phi(\xi_i, t_k)} \\
&\times \left[f(\xi_i, t_k) - \frac{1}{\Delta t^\alpha} \sum_{v=0}^{k-1} \omega_{v,k} u(\xi_i, t_v) + {}^{RL}D_t^\alpha (T_{n-1}[u; 0])(\xi_i, t_k) \right]
\end{aligned} \tag{80}$$

Equation 80. d, o, r -constants

4.2.3 Complete discretization problem.

Putting both the LHS and RHS of the discretization together, the resulting discretized problem below can be solved using any available linear algebra program.

$$\begin{aligned}
&\left(\frac{\phi(x_i, t_k)}{\Delta x^2} \begin{bmatrix} d_1 & o_1 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & & o_2 & d_2 \end{bmatrix} + \frac{1}{\Delta t^\alpha} \begin{bmatrix} \omega_{0,0} & & \cdots & & 0 \\ & \omega_{0,1} & & & \\ \vdots & & \ddots & & \vdots \\ 0 & & & \omega_{0,N-1} & \\ & \cdots & & & \omega_{0,N} \end{bmatrix} \right) \begin{bmatrix} u(a, t_k) \\ u(x_1, t_k) \\ \vdots \\ u(x_{N-1}, t_k) \\ u(b, t_k) \end{bmatrix} = \\
&\begin{bmatrix} r_1 \\ f(x_1, t_k) \\ \vdots \\ f(x_{N-1}, t_k) \\ r_2 \end{bmatrix} + \begin{bmatrix} 0 \\ {}^{RL}D_t^\alpha [(T_{n-1}[u; 0])(x_1, t_k)] \\ \vdots \\ {}^{RL}D_t^\alpha [(T_{n-1}[u; 0])(x_{N-1}, t_k)] \\ 0 \end{bmatrix} - \begin{bmatrix} 0 \\ \sum_{v=0}^{k-1} \omega_{v,k} u(x_1, t_v) \\ \vdots \\ \sum_{v=0}^{k-1} \omega_{v,k} u(x_{N-1}, t_v) \\ 0 \end{bmatrix}
\end{aligned} \tag{81}$$

Equation 81. Numerical discretization time-fractional diffusion-wave equation

The discretization in Equation 81 is generated and evaluated by matrix inversion at every time step t_k for $k = 0, \dots, M$. Note that the matrix dimensions and vector dimensions are based off of the number of grid points in the space dimension, $N \times N$ and $N \times 1$, respectively.

4.2.4 Time-fractional sub-diffusion problem.

The time-fractional sub-diffusion problem involves the fractional diffusion-wave equation where $0 < \alpha \leq 1$. For the sub-diffusion problem, the Diethelm weights were implemented here due to computational efficiency for these values of α .

$${}^C D_t^\alpha u(x, t) + \phi(x, t) \frac{\partial^2}{\partial x^2} u(x, t) = f(x, t), \quad 1 < \alpha \leq 2 \tag{82}$$

Equation 82. Time-fractional sub-diffusion problem

For this example, the equation is solved with $f(x, t) = 0$ in $x \in [0, \pi]$ and $t \in [0, 10]$ with the initial condition

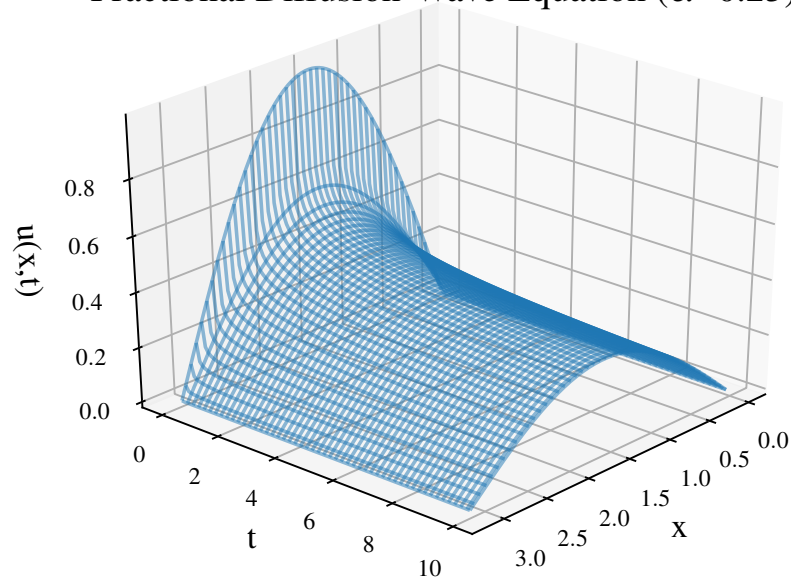
$$u(x, 0) = \sin(x)$$

and boundary conditions

$$u(0, t) = 0, \quad u(\pi, t) = 0$$

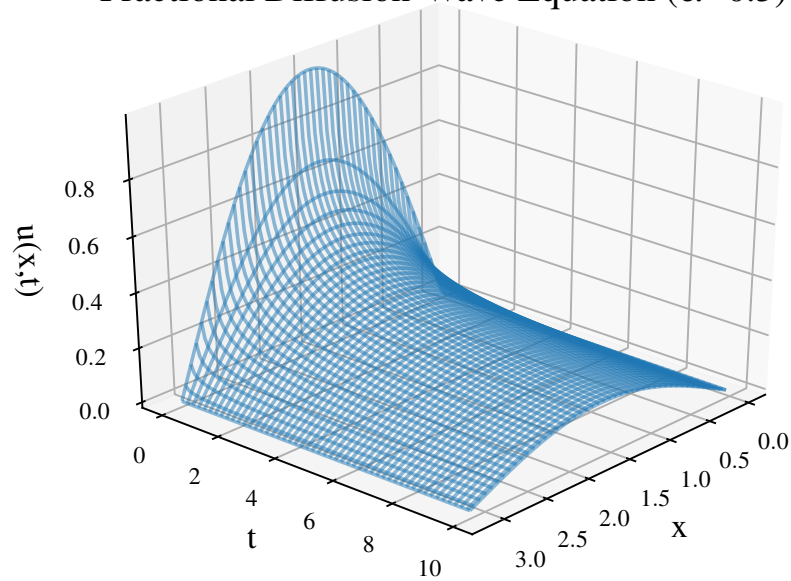
The code appearing in subsection B.5 was used to obtain the results in Figure 17 and Figure 18 which agreed the results found in [21].

Fractional Diffusion-Wave Equation ($\alpha=0.25$)



(a) $\alpha = 0.25$

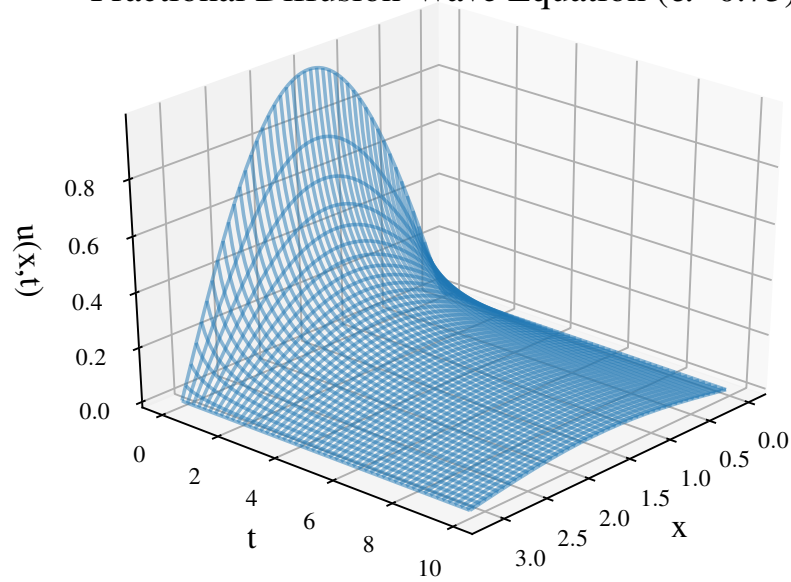
Fractional Diffusion-Wave Equation ($\alpha=0.5$)



(b) $\alpha = 0.5$

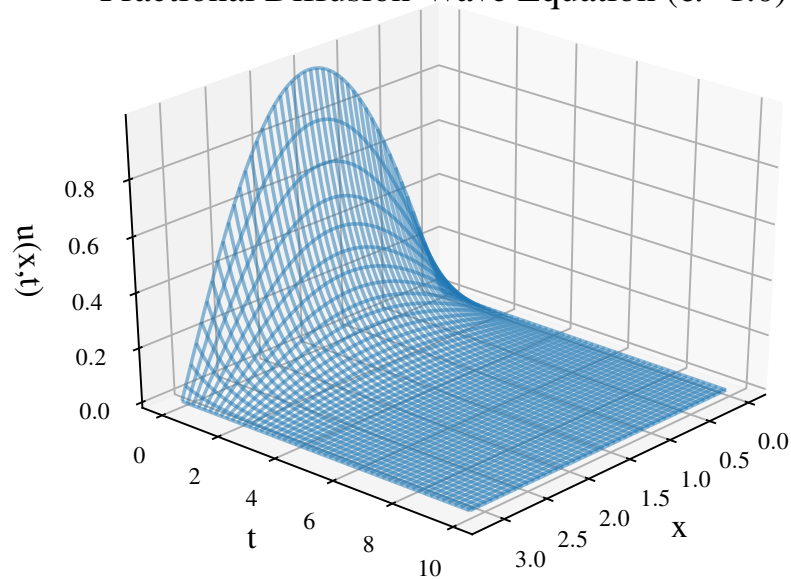
Figure 17. Time-fractional sub-diffusion ($0 < \alpha \leq 0.5$)

Fractional Diffusion-Wave Equation ($\alpha=0.75$)



(a) $\alpha = 0.75$

Fractional Diffusion-Wave Equation ($\alpha=1.0$)



(b) $\alpha = 1.0$

Figure 18. Time-fractional sub-diffusion ($0.5 < \alpha \leq 1$)

4.2.5 Time-fractional diffusion-wave equation.

The time-fractional diffusion-diffusion problem [21] involves the fractional diffusion-wave equation where $1 \leq \alpha \leq 2$. The solution was implemented with modified Grünwald-Letnikov weights as α -values for the diffusion-wave problem extend beyond the scope where the Diethelm weights provide valid results.

$${}^C D_t^\alpha u(x, t) + \phi(x, t) \frac{\partial^2}{\partial x^2} u(x, t) = f(x, t), \quad 1 < \alpha \leq 2 \quad (83)$$

Equation 83. Time-fractional diffusion-wave equation

The equation was solved with $f(x, t) = 0$ in $x \in [0, \pi]$ and $t \in [0, 10]$ with the initial conditions

$$u(x, 0) = \sin(x), \quad \frac{\partial}{\partial t} u(x, 0) = 0$$

and boundary conditions

$$u(0, t) = 0, \quad u(\pi, t) = 0$$

Modifying Grünwald-Letnikov weights

It is necessary to modify the Grünwald-Letnikov weights $\omega_{j,k}$, to use two indices in the same manner as the Diethelm weights. This is done for convenience and consistency in code and is necessary to account for initial conditions. Specifically, this applies for the case if the function $u(x, t)$ is a causal function with respect to time, i.e. $u(x, t) \equiv 0$ if $t < 0$ and if $u(x, 0) = 0$. The modification applies additional starting weights w_j^α to the first column of the lower triangular matrix defined in Equation 76 [21]:

$$\begin{bmatrix} \omega_0 + w_0^\alpha & & & 0 \\ \omega_1 + w_1^\alpha & \omega_0 & & \\ \vdots & & \ddots & \\ \omega_{M-1} + w_{M-1}^\alpha & & & \ddots \\ \omega_M + w_M^\alpha & \omega_{M-1} & \dots & \omega_0 \end{bmatrix} = \begin{bmatrix} \omega_{0,0} & & & 0 \\ \omega_{1,1} & \omega_{0,1} & & \\ \vdots & & \ddots & \\ \omega_{M-1,M-1} & & & \ddots \\ \omega_{M,M} & \omega_{M-1,M} & \dots & \omega_{0,M} \end{bmatrix} \quad (84)$$

Equation 84. Lower triangular matrix with modified Grünwald-Letnikov weights

where the Grünwald-Letnikov weights $\omega_{j,k}$ are defined as

$$\omega_{j,k} = \begin{cases} (-1)^j \binom{\alpha}{j} + \frac{j^{-\alpha}}{\Gamma(1-\alpha)} - (-1)^j \frac{\Gamma(\alpha)}{\Gamma(\alpha-j)\Gamma(j+1)}, & \text{if } k-j=0 \\ (-1)^j \binom{\alpha}{j}, & \text{else} \end{cases} \quad (85)$$

Equation 85. Modified Grünwald-Letnikov weights for matrix form

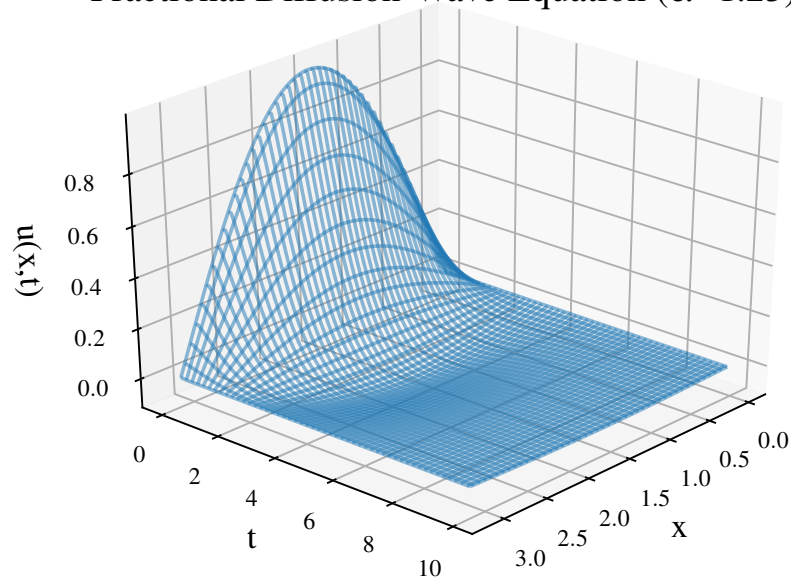
The additional starting weights w_j^α (note: w not ω) are defined as:

$$w_j^\alpha = \frac{j^{-\alpha}}{\Gamma(1-\alpha)} - (-1)^j \frac{\Gamma(\alpha)}{\Gamma(\alpha-j)\Gamma(j+1)} \quad (86)$$

Equation 86. Starting weights for Grünwald-Letnikov weights

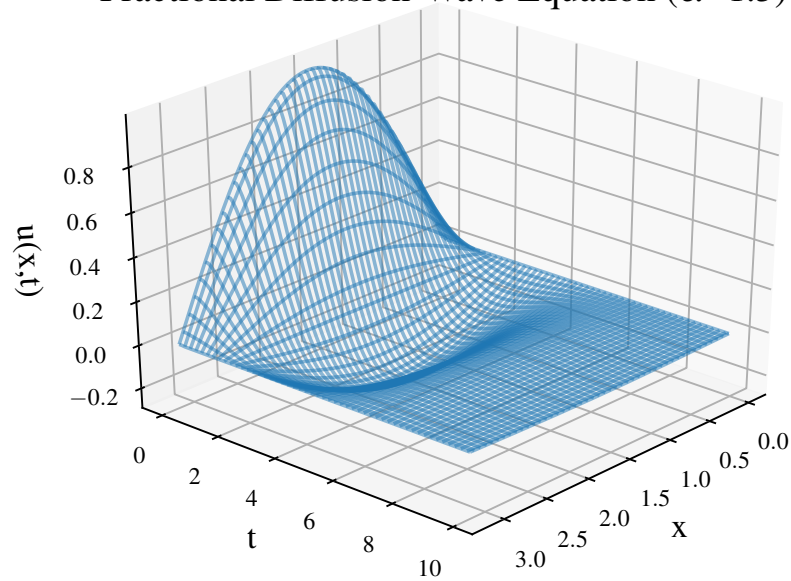
The accuracy of the numerical solution is heavily dependent on the sizes of the steps in time and space. Lowering the time step from 0.1 to 0.01 creates a huge computational time cost, but is necessary for accuracy for larger values of α . For the case $\alpha = 2$ in Figure 20 a time step of 0.01 was used which was necessary to have results remotely accurate. At the time of writing, no appropriate analytical solutions were available to perform computational error and run time analysis with and have therefore been omitted. The code used to generate Figure 19 and Figure 20 is provided in subsection B.5 of the Appendix.

Fractional Diffusion-Wave Equation ($\alpha=1.25$)



(a) $\alpha = 1.25$

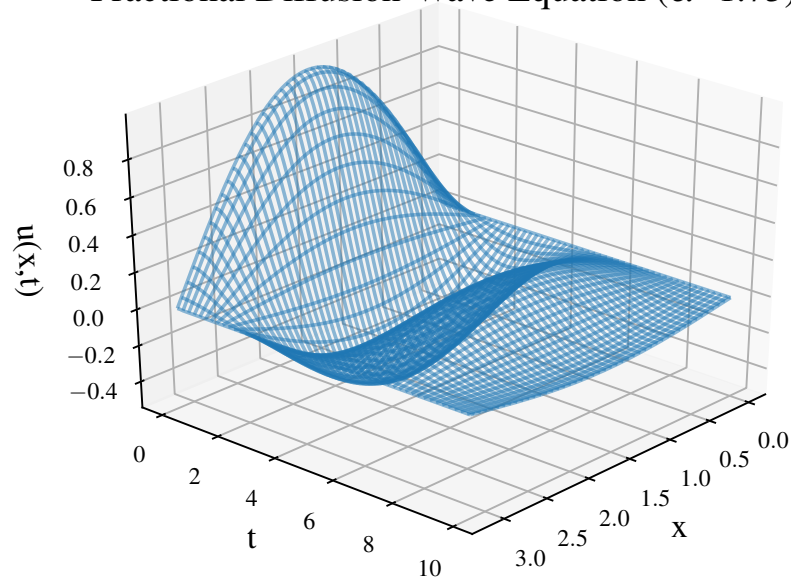
Fractional Diffusion-Wave Equation ($\alpha=1.5$)



(b) $\alpha = 1.5$

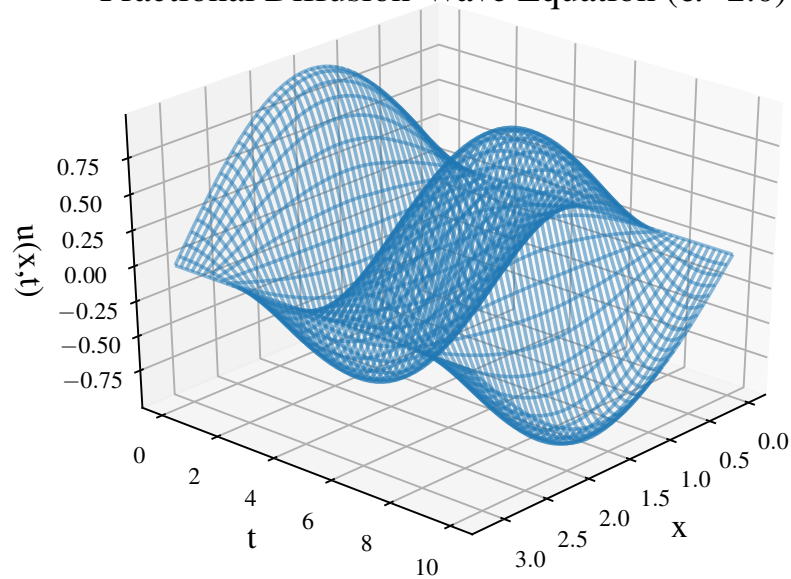
Figure 19. Time-fractional diffusion-wave ($1 < \alpha \leq 1.5$)

Fractional Diffusion-Wave Equation ($\alpha=1.75$)



(a) $\alpha = 1.75$

Fractional Diffusion-Wave Equation ($\alpha=2.0$)



(b) $\alpha = 2.0$

Figure 20. Time-fractional sub-diffusion ($1.5 < \alpha \leq 2$)

5.0 FOUR-TERM FRACTIONAL WAVE EQUATION

The four-term fractional wave equation developed in [2] appearing as

$$A_1 \frac{\partial^2 E_y}{\partial x^2} + A_2 \frac{\partial^\alpha}{\partial t^\alpha} \left(\frac{\partial^2 E_y}{\partial x^2} \right) = \mu_r \mu_0 \epsilon_0 \frac{\partial^2 E_y}{\partial t^2} + A_0 \mu_r \mu_0 \epsilon_0 \frac{\partial^\alpha}{\partial t^\alpha} \left(\frac{\partial^2 E_y}{\partial t^2} \right) \quad (87)$$

Equation 87. Four-term fractional wave equation

is the result of incorporating a frequency-dependent complex permittivity transfer function, also developed in [2], that is defined as

$$\epsilon_r^*(j\omega) = \frac{1 + A_0(j\omega)^\alpha}{A_1 + A_2(j\omega)^\alpha} \quad (88)$$

Equation 88. Frequency-dependent complex permittivity

into the classical wave equation

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2}$$

through the application of integral transforms. The two additional terms in the four-term wave equation arise from the frequency-dependent complex permittivity of a material described by Equation 88 and contain Riemann-Liouville fractional derivatives. The second term on the left hand side of Equation 87 is a fractional mixed partial derivative which means the term consists of derivatives with respect to both space and time, or x and t , respectively. Specifically, a fractional time derivative of order α is operating on a second order spatial derivative of the function E_y . The second term on the right hand side of Equation 87 is a fractional time derivative of order $2 + \alpha$ which is an order beyond what is typically found in applied problems whose numerical treatment was found to be absent at the time this survey was performed. It was demonstrated in [2], [27], [28] that the inclusion of these additional terms was necessary for the wave equation to compensate for anomalous dispersion phenomena that occur when electromagnetic waves propagate through various dielectric materials.

5.1 Discretization of the Four-Term Fractional Wave Equation

Discretization of the four-term fractional equation was attempted by using a combination of integer-order and fractional-order numerical methods. Here the central difference was applied for the second-order derivatives and the Grünwald-Letnikov approximation was employed for the fractional-order derivatives. The grid point values u_n^k represents the value E_y at grid point n in space, and k in time. Using these approximations, the first term on the left-side of Equation 87 is discretized as

$$A_1 \frac{\partial^2}{\partial x^2} u(x, t) \approx A_1 \frac{u_{n+1}^k - 2u_n^k + u_{n-1}^k}{\Delta x^2} \quad (89)$$

Equation 89. Discretization of second-order spatial derivative within the four-term fractional wave equation

The second term on the left-hand side of the equation that contains the fractional mixed partial derivative $\frac{\partial^\alpha}{\partial t^\alpha} \frac{\partial^2 E_y}{\partial x^2}$ when discretized involves combining the classical integer-order scheme and the fractional Grünwald-Letnikov scheme:

$$\begin{aligned} A_2 \frac{\partial^\alpha}{\partial t^\alpha} \left(\frac{\partial^2}{\partial x^2} \right) u(x, t) &\approx A_2 \frac{\partial^\alpha}{\partial t^\alpha} \left(\frac{u_{n+1}^k - 2u_n^k + u_{n-1}^k}{\Delta x^2} \right) \\ &= \frac{A_2}{\Delta x^2 \Delta t^\alpha} \left(\sum_{j=0}^k \omega_j^\alpha \left(u_{n+1}^{k-j} - 2u_n^{k-j} + u_{n-1}^{k-j} \right) \right) \end{aligned} \quad (90)$$

Equation 90. Discretization of fractional mixed partial derivative within four-term fractional wave equation

The first term on the right-hand side of Equation 87 is simply

$$\mu_r \mu_0 \varepsilon_0 \frac{\partial^2}{\partial t^2} u(x, t) \approx \mu_r \mu_0 \varepsilon_0 \frac{u_n^{k+1} - 2u_n^k + u_n^{k-1}}{\Delta t^2} \quad (91)$$

Equation 91. Discretization of second-order time derivative within the four-term fractional wave equation

Finally, the second term on the right hand side that contains the $(2 + \alpha)$ -th order time derivative is approximated as

$$A_0 \mu_r \mu_0 \varepsilon_0 \frac{\partial^\alpha}{\partial t^\alpha} \left(\frac{\partial^2}{\partial t^2} \right) u(x, t) \approx A_0 \mu_r \mu_0 \varepsilon_0 \Delta t^{-(2+\alpha)} \sum_{j=0}^k \omega_j^{2+\alpha} u_n^{k-j} \quad (92)$$

Equation 92. Discretization of fractional time derivative within four-term fractional wave equation

Appropriately combining these approximations results in the discretization of the four-term fractional wave equation:

$$\begin{aligned} A_1 \frac{u_{n+1}^k - 2u_n^k + u_{n-1}^k}{\Delta x^2} + \frac{A_2}{\Delta x^2} \Delta t^{-\alpha} \left(\sum_{j=0}^k \omega_j^\alpha \left(u_{n+1}^{k-j} - 2u_n^{k-j} + u_{n-1}^{k-j} \right) \right) \\ = \mu_r \mu_0 \varepsilon_0 \frac{u_n^{k+1} - 2u_n^k + u_n^{k-1}}{\Delta t^2} + A_0 \mu_r \mu_0 \varepsilon_0 \Delta t^{-(2+\alpha)} \sum_{j=0}^k \omega_j^{2+\alpha} u_n^{k-j} \end{aligned} \quad (93)$$

Equation 93. Discretization of four-term fractional wave equation

Solving this discretization for the term u_n^{k+1} , which only appears in the first term on the right-hand side, yields an explicit numerical scheme for the four-term fractional wave equation:

$$\begin{aligned}
u_n^{k+1} = & 2u_n^k - u_n^{k-1} + \left[\frac{A_1}{\mu_0 \mu_r \epsilon_0} \frac{\Delta t^2}{\Delta x^2} \left(u_{n+1}^k - 2u_n^k + u_{n-1}^k \right) \right. \\
& + \frac{A_2}{\mu_0 \mu_r \epsilon_0} \frac{\Delta t^{(2-\alpha)}}{\Delta x^2} \left(\sum_{j=0}^k \omega_j^\alpha \left(u_{n+1}^{k-j} - 2u_n^{k-j} + u_{n-1}^{k-j} \right) \right) \\
& \left. - A_0 \frac{1}{\Delta t^\alpha} \sum_{j=0}^k \omega_j^{2+\alpha} u_n^{k-j} \right]
\end{aligned} \tag{94}$$

Equation 94. Explicit numerical solution for the four-term fractional wave equation

where the constants ω_j^α are the Grünwald-Letnikov weights given in Equation 4.

5.2 Computational results

The numerical solution described in Equation 94 was implemented using the code that is found in subsection B.6 of the Appendix. A ramped sine wave function in time was used as the source term at the midpoint of the spatial grid which in this case was $x = 10$ and is defined by

$$f(t) = \begin{cases} \frac{1}{2} \left(1 + \sin \left(\frac{\pi}{2\tau} (2t - \tau) \right) \right) \sin(2\pi f_0 t), & \text{for } t < \tau \\ \sin(2\pi f_0 t) & \end{cases} \tag{95}$$

Equation 95. Ramped sine wave source

where f_0 is the frequency of the sine wave, set to 1 Hz, and τ is the ramping duration which is assumed to be $\tau = 5/f_0$ for this example. This behavior of this function is shown in Figure 21.

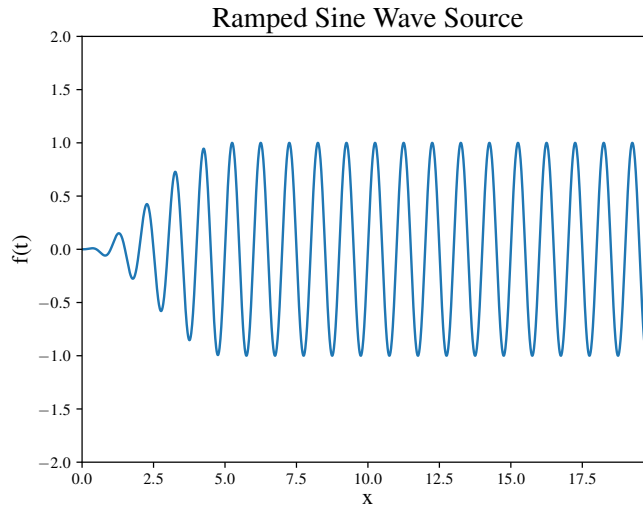


Figure 21. Ramped sine wave source

The ramped sine wave is used as the source term as opposed to the initial condition used for the analytical solution found in [2] to avoid introducing steep gradients at the start of the simulation

that incorrectly produced large peaks at both wave fronts. A workaround to compensate for this artifact was not found at the time of writing and therefore a computational error and run time analysis were not performed. The physical constants μ_0 , μ_r , and ϵ_0 were all normalized to 1 while the rest of the constant values were arbitrarily set to $A_0 = 10\text{e-}8$, $A_1 = 0.1$, and $A_2 = 10\text{e-}4$. The grid spacing in both time and space were set to 0.01. Though this specific configuration is non-physical, it was able to generate a stable solution.

The results of the simulation are shown in figures 22-25. As time progressed, the source wave at $x = 10$ began propagating outwards and decaying. Subtle differences in the attenuation rate are most obviously observed at time step 1999 for different values of α . It appears as if the attenuation rate decreases with increasing α which is the opposite of what is expected [2], [27], [28]. This could be attributed to how the source term was applied, the non-physical parameters used for the constants, how the numerical method itself was implemented, or a combination thereof. The numerical solution for the four-term fractional wave equation therefore requires further development.

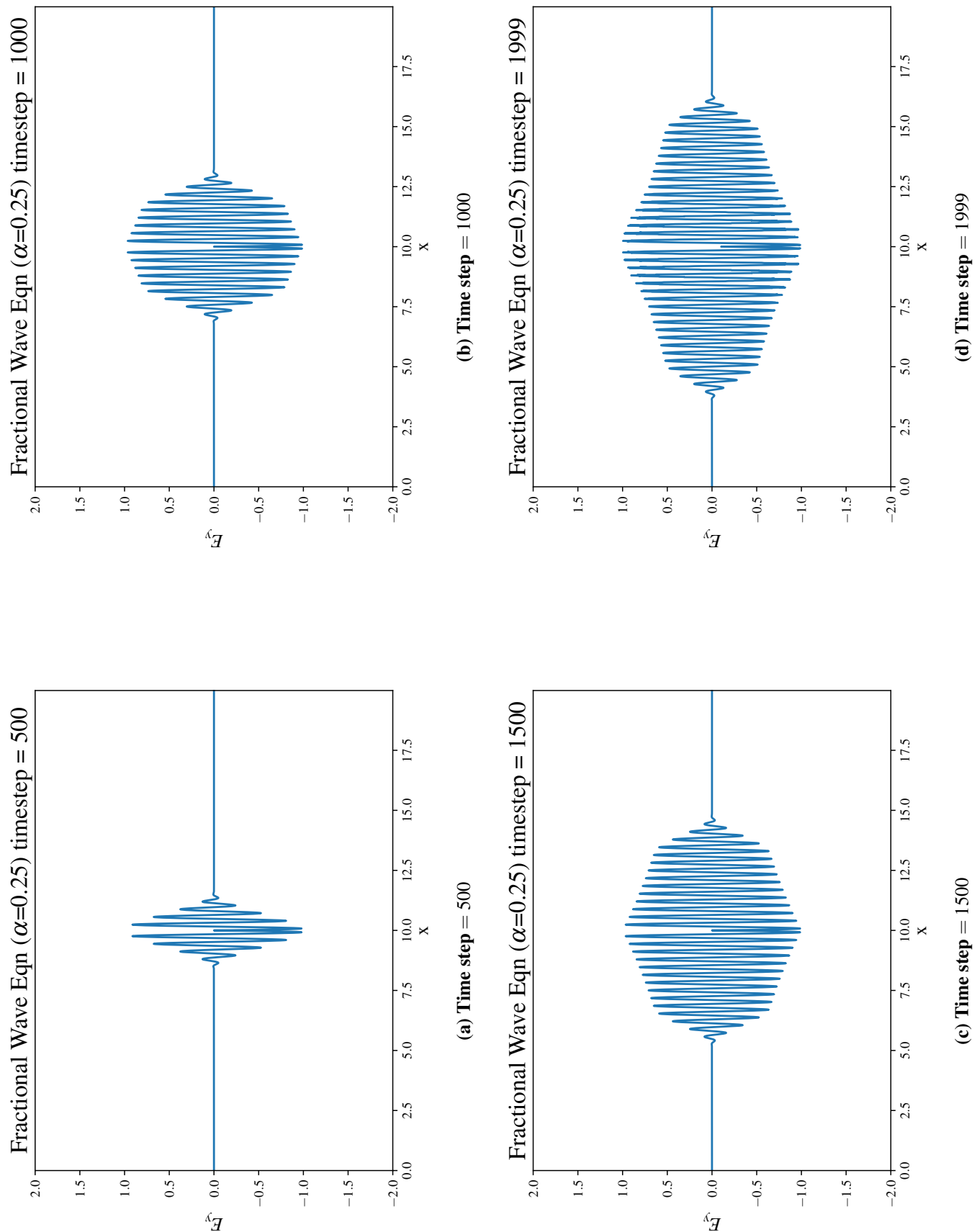
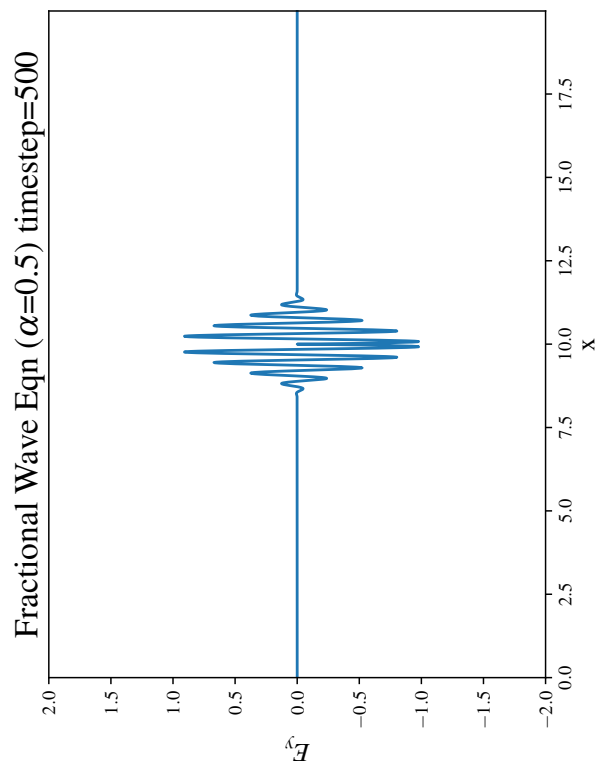
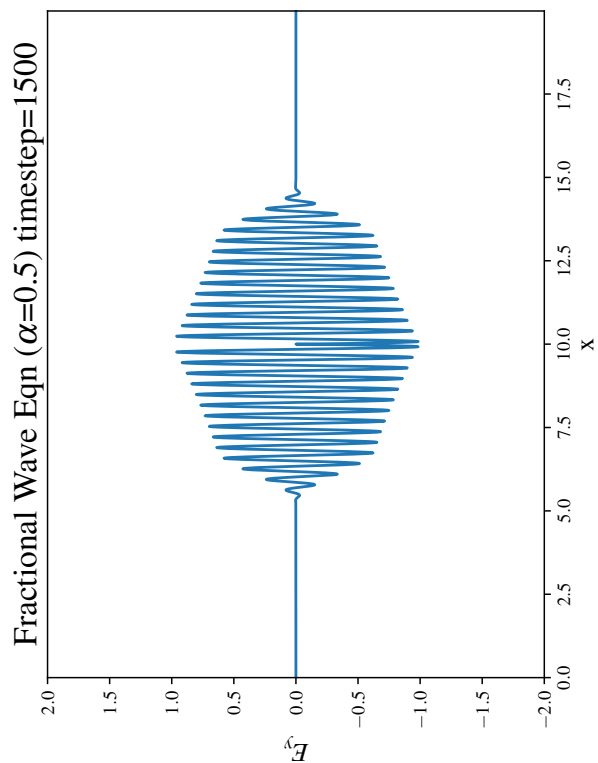


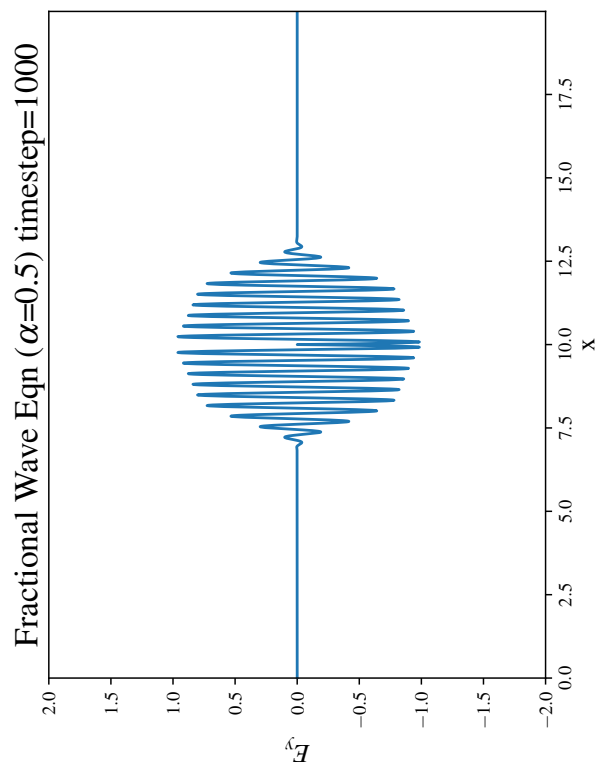
Figure 22. Numerical simulation of the four-term fractional wave equation with wave source at $x = 10$ and $\alpha = 0.25$ at various time steps



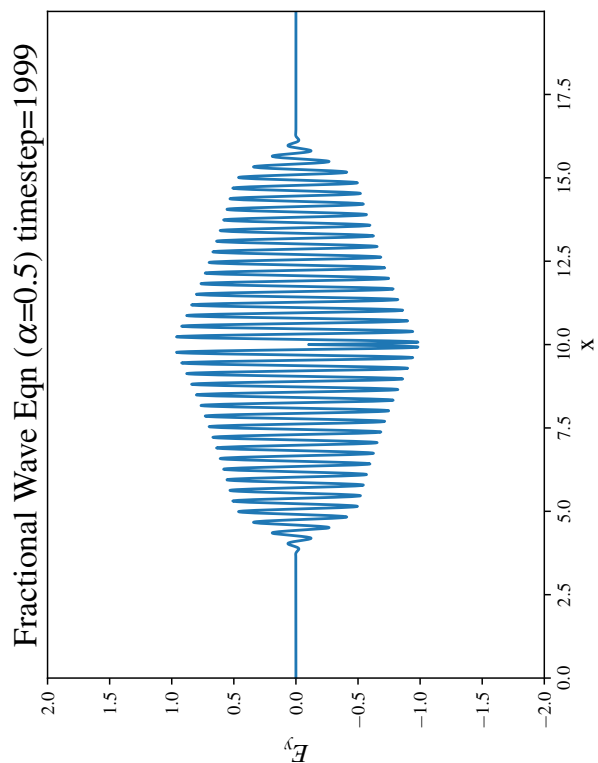
(a) Time step = 500



(c) Time step = 1500

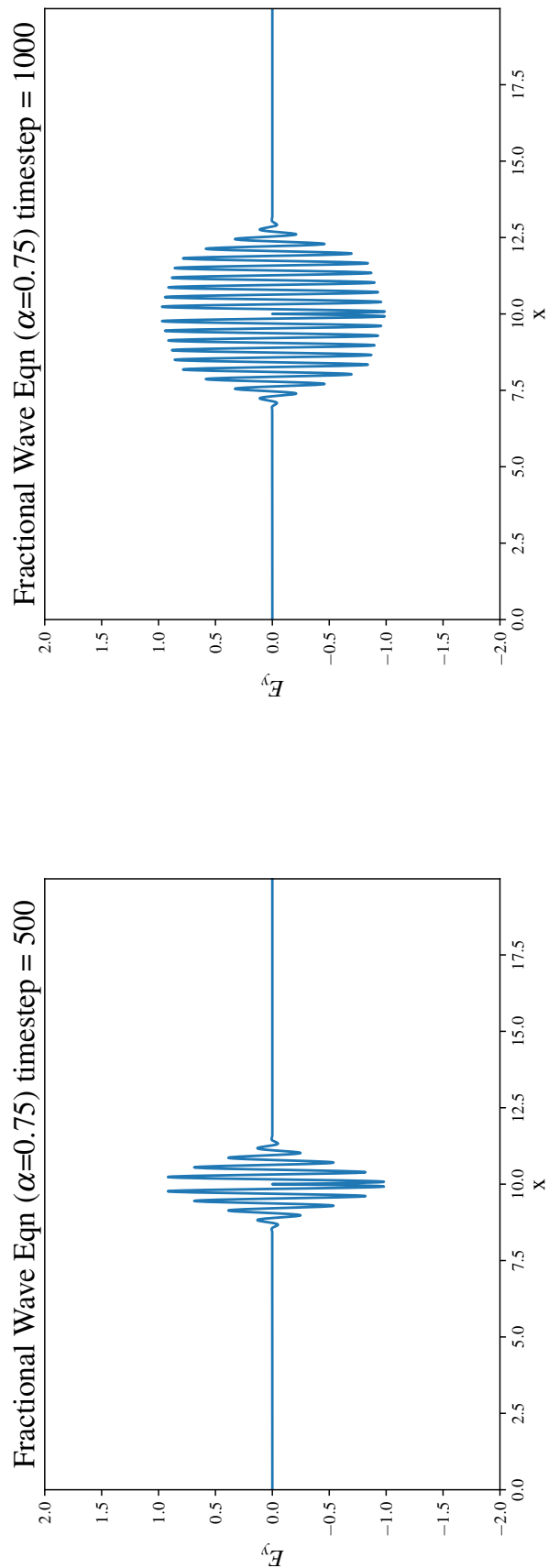


(b) Time step = 1000

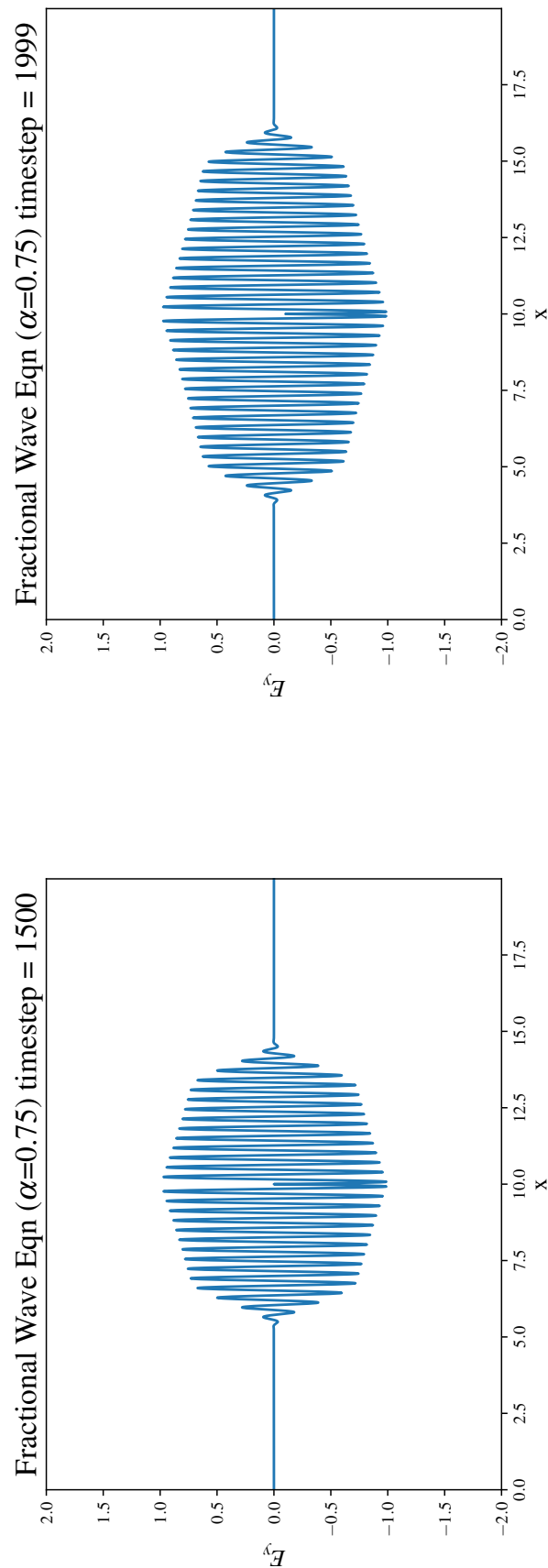


(d) Time step = 1999

Figure 23. Numerical simulation of the four-term fractional wave equation with wave source at $x = 10$ and $\alpha = 0.5$ at various time steps

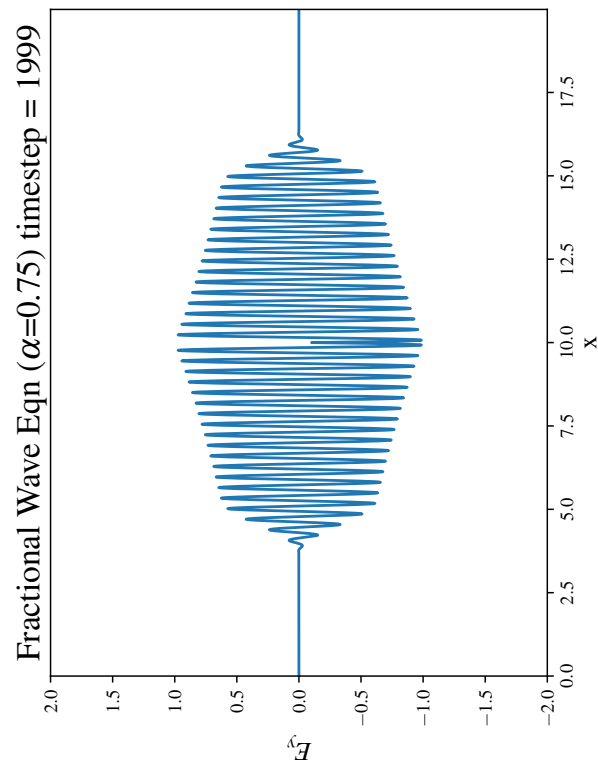


(a) Time step = 500



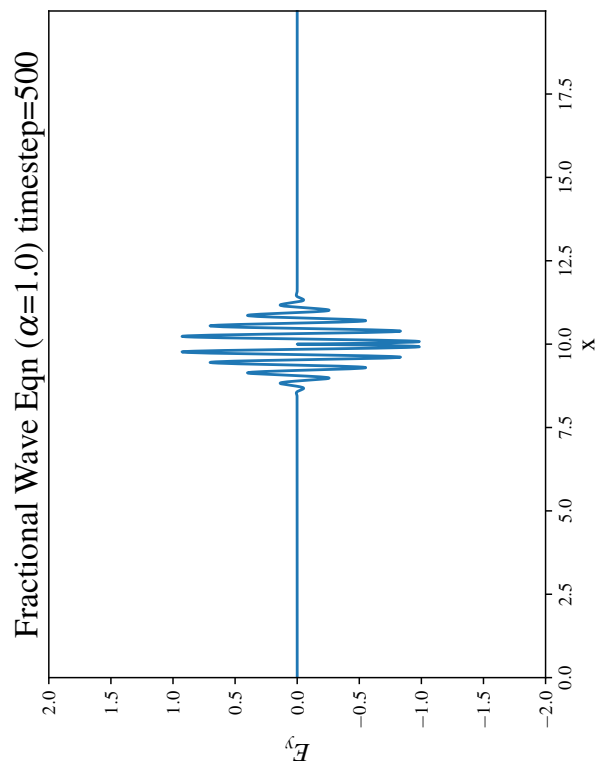
(c) Time step = 1500

(b) Time step = 1000

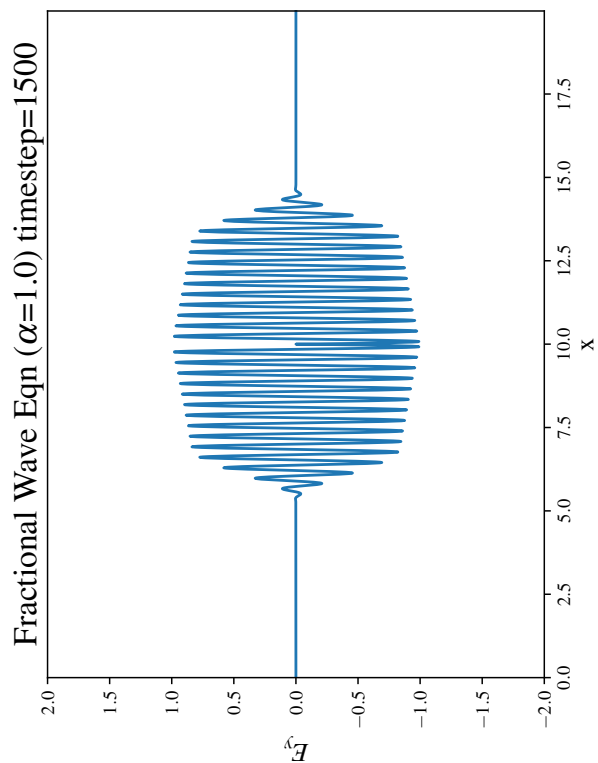


(d) Time step = 1999

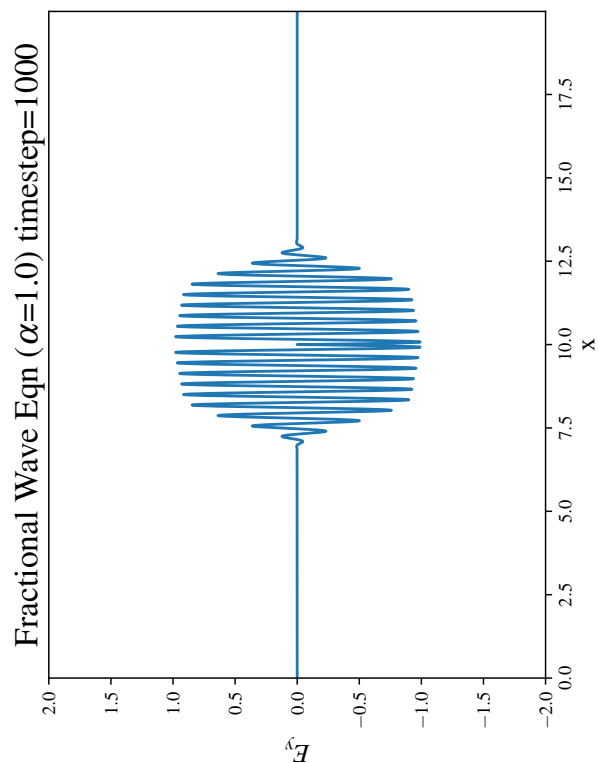
Figure 24. Numerical simulation of the four-term fractional wave equation with wave source at $x = 10$ and $\alpha = 0.75$ at various time steps



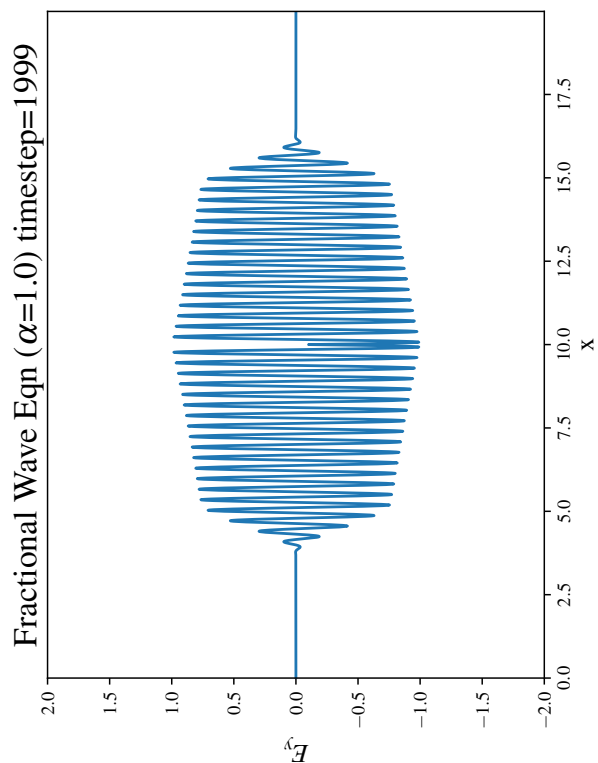
(a) Time step = 500



(c) Time step = 1500



(b) Time step = 1000



(d) Time step = 1999

Figure 25. Numerical simulation of the four-term fractional wave equation with wave source at $x = 10$ and $\alpha = 1.0$ at various time steps

6.0 CONCLUSIONS

Three different fractional derivative operators, the Grünwald-Letnikov, Riemann-Liouville, and Caputo, were considered for the development of numerical solutions for fractional differential equations (FDEs) in this study. The Grünwald-Letnikov provided a foundation difference quotient-based numerical schemes for FDEs and several weight coefficient options were also considered. The Grünwald-Letnikov weights that naturally arise from the derivation of the fractional derivative whereas the Diethelm weights stem from a product-trapezoidal scheme to approximate the weakly-singular Volterra integral of the Riemann-Liouville fractional derivative and integral. Preliminary studies showed that each had their share of strengths and weaknesses when it came to accuracy, computation time, and validity for specific ranges of values of derivative order. The Riemann-Liouville fractional derivative and integral definitions are the natural consequence of generalizing the Cauchy repeated integration formula. However, the initial conditions required for the Riemann-Liouville fractional derivative operator are difficult to physically interpret. On the other hand, the Caputo fractional derivative operator circumvents this problem with a minor formulation change that produces more traditional expressions for initial conditions making them easier to apply to a wider variety of real-world problems. Relationships between each of these three operators were also reviewed and used to address the shortcomings of each.

The applied example problems of the relaxation-oscillation and Bagley-Torvik equations were also explored and solved using both an explicit Grünwald-Letnikov method and an implicit matrix method. The explicit Grünwald-Letnikov method proved to be far less expensive in terms of memory and computation time compared to the implicit matrix method while no method stood out in terms of minimizing error. The Grünwald-Letnikov method proved to be more readily applicable and easier to implement due to the matrix method being restricted to problems with zeroed initial conditions. Therefore the problem must be redefined for the matrix method to meet this requirement for non-zero initial conditions which severely restricts its application.

Of the two numerical solutions to the fractional diffusion-wave equation that were investigated, the Diethelm-Ford-Weilbeer solution allows for the most general Neumann-Dirichlet boundary conditions and initial conditions. The code has only been implemented for derivative orders of $0 < \alpha \leq 2$, but the functionality can be expanded by properly incorporating a feature that can accept more initial conditions to define fractional partial differential equations (FPDEs) with derivative orders of $\alpha > 2$. The other solution for FPDEs presented by Baleanu et. al [4] on the other hand, while more restricted in functionality, best illustrates the mechanics behind numerically solving FPDEs.

Preliminary development and application for a numerical scheme for approximating the fractional mixed partial derivative in the four-term fractional wave equation appeared to be fairly straightforward and easy however the results that were obtained during the test case deviated from what was expected. It is clear that methods needed to solve this type of FPDE require further development.

The problems investigated in this survey began from a study of numerical methods for solving simple, well-known fractional equations (relaxation-oscillation, Bagley-Torvik). From this starting point, appropriate adjustments to the numerical solutions were then made so the methods could be applied to more complex directed energy problems (fractional lumped capacitance, four-term

fractional wave equation). Overall, the results obtained within this survey suggest that numerical methods can be and are appropriate for solving directed energy related problems whose governing equations manifest themselves as FDEs.

7.0 RECOMMENDATIONS

Throughout the course of the survey of numerical methods for solving fractional differential equations, it was noted that rigorous efforts to optimize algorithms for accuracy, computation time, and computer memory were wholly absent in the literature. Some work has been done in this field however, such as a "short-memory" principle with Gr̈unwald-Letnikov weight coefficients where one can neglect the history of a function some distance away from the lower terminal under certain conditions at the cost of accuracy [14] for instance. Another example are predictor-corrector methods [4] that attempt to minimize error at each iterative step in the algorithm to improve the rate of convergence. Though these methods prove to be effective for some problems, deducing when, where, and how to apply these methods to real-world applications is not always straightforward. As such, no conventional or industrial standards were found for the application of numerical methods for solving fractional differential equations in general. Establishing such a standard or a unified collection of optimized routines for specific classes of problems would greatly encourage more broad application of these methods in industry and research.

In some cases, established methods for approximating certain operators simply did not exist. For example, to the authors' knowledge at the time of writing, approximating the fractional mixed partial derivative operator in the four-term fractional wave equation was a novel attempt that produced less than favorable results. This outcome suggests that the formulation of numerical methods for approximating more complicated fractional derivative operators for real-world problems is not straightforward and requires formal research and development for functionality as well as optimization.

8.0 REFERENCES

1. A. Wharmby, “Fractional lumped capacitance,” *Fractional Calculus and Applied Analysis*, vol. 21, no. 4, pp. 1104–1119, 2018. DOI: [10.1515/fca-2018-0059](https://doi.org/10.1515/fca-2018-0059) 1, 34, 37.
2. A. Wharmby, “Modifying maxwell’s equations for dielectric materials based on techniques from viscoelasticity and concepts from fractional calculus,” *International Journal of Engineering Science*, vol. 79, pp. 59–80, 2014. DOI: <https://doi.org/10.1016/j.ijengsci.2014.02.004> 1, 2, 58, 60, 61.
3. K. B. Oldham and J. Spanier, *The Fractional Calculus: Theory and Applications of Differentiation and Integration to Arbitrary Order*. Mineola, NY: Dover Publications, Inc., 1974 1, 2.
4. D. Baleanu, K. Diethlem, E. Scalas, and J. J. Trujillo, *Fractional Calculus: Models and Numerical Methods*, A. C. Luo, Ed. Singapore: World Scientific Publishing Co. Pte. Ltd., 2017 2, 40, 41, 46, 66, 67.
5. M. Caputo, “Linear models of dissipation whose q is almost frequency independent—ii,” *Geophysical Journal International*, vol. 13, no. 5, pp. 529–539, 1967. DOI: [10.1111/j.1365-246X.1967.tb02303.x](https://doi.org/10.1111/j.1365-246X.1967.tb02303.x) 2.
6. K. Diethelm, N. Ford, A.D.Freed, and Y. Luchko, “Algorithms for fractional calculus: A selection of numerical methods,” *Computer Methods in Applied Mechanics and Engineering*, vol. 194, pp. 743–773, 2005. DOI: [10.1016/j.cma.2004.06.006](https://doi.org/10.1016/j.cma.2004.06.006) 2.
7. M. Lazarević, M. Rapaić, and T. Šekara, “Introduction to fractional calculus with brief historical background,” in *Advanced Topics on Applications of Fractional Calculus on Control Problems, System Stability, and Modeling*, N. M. V. Mladenov, Ed., Athens, Greece: WSEAS Press, 2014, ch. 1 2.
8. A. McBride, *Fractional calculus and integral transforms of generalized functions - (Research notes in mathematics; 31)*. London: Pitman Advanced Publishing Program, 1979 2.
9. K. S. Miller and B. Ross, *An Introduction to Fractional Calculus and Fractional Differential Equations*. New York: John Wiley & Sons, Inc., 1993 2, 4.
10. S. G. Samko, A. A. Kilbas, and O. I. Marichev, *Fractional Integrals and Derivatives: Theory and Applications*. Minsk, Belarus: Gordon and Breach Science Publishers, 1993 2.
11. V. Kiryakova, *Generalized Fractional Calculus and its Applications*. Harlow, UK & New York, NY: Longman & John Wiley, 1993 2.
12. B. Rubin, *Fractional Integrals and Potentials*. Harlow, UK: Addison Wesley Longman Ltd., 1996 2.
13. F. Mainardi, *Fractals and Fractional Calculus in Continuum Mechanics*, A. Carpinteri, Ed. Germany: Springer, 1997 2.

14. I. Podlubny, *Fractional Differential Equations*. San Diego, CA: Academic Press, 1999 2–4, 7, 16, 25, 26, 67, 72.
15. R. Hilfer, *Applications of Fractional Calculus in Physics*. Singapore: World Scientific Publishing Company, 2000 2.
16. A. Kilbas, H. Srivastava, and J. Trujillo, *Theory and Applications of Fractional Differential Equations*. Netherlands: Elsevier, 2006 2.
17. S. Das, *Functional Fractional Calculus for System Identification and Controls*. Germany: Springer, 2008 2.
18. J. Sabatier, O. Agrawal, and J. T. Machado, *Advances in Fractional Calculus: Theoretical Developments and Applications in Physics and Engineering*. Netherlands: Springer, 2007 2.
19. R. Bagley and P. Torvik, “On the appearance of the fractional derivative in the behavior of real materials,” *Journal of Applied Mechanics*, vol. 51, pp. 294–298, 1984. DOI: [10.1115/1.3167615](#) 2, 25.
20. R. L. Magin, *Fractional Calculus in Bioengineering*. Connecticut: Begell House Publishers, Inc., 2006 2.
21. K. Diethelm and M. Weilbeer, “Initial-boundary value problems for time-fractional diffusion-wave equations and their numerical solution,” in *Conference: 1st IFAC Workshop on Fractional Differentiation and its Applications*, Toulouse, France: International Federation of Automatic Controls, 2004 9, 10, 47, 51, 54.
22. I. Podlubny, “Matrix approach to discrete fractional calculus,” *Fractional Calculus & Applied Analysis*, vol. 3, no. 4, 2000 12.
23. M. P. DeLisi, N. J. Gamez, C. D. Clark, S. S. Kumru, B. A. Rockwell, and R. J. Thomas, “Computational modeling and damage threshold prediction of continuous-wave and multiple-pulse porcine skin laser exposures at 1070 nm,” *Journal of Laser Applications*, vol. 33, no. 022023, 2021. DOI: [10.2351/7.0000367](#) 34, 35, 37.
24. M. P. DeLisi, M. S. Schmidt, A. F. Hoffman, *et al.*, “Thermal damage thresholds for multiple-pulse porcine skin laser exposures at 1070 nm,” *Journal of Biomedical Optics*, vol. 25, no. 3, March 2020. DOI: [10.1117/1.JBO.25.3.035001](#) 35, 37.
25. C. Li and A. Chen, “Numerical methods for fractional partial differential equations,” *International Journal of Computer Mathematics*, vol. 95, no. 6-7, pp. 1048–1099, 2018. DOI: [10.1080/00207160.2017.1343941](#) 41, 49.
26. K. Diethelm, “An algorithm for the numerical solution of differential equations of fractional order,” *Electronic Transactions on Numerical Analysis*, vol. 5, pp. 1–6, 1997. DOI: [10.1111/j.1365-246X.1997.tb02303.x](#) 42.
27. A. W. Wharmby and R. L. Bagley, “The application of the fractional calculus model for dispersion and absorption in dielectrics i. terahertz waves,” *International Journal of Engineering Science*, vol. 93, pp. 1–12, 2015 58, 61.

28. A. W. Wharmby, "The application of the fractional calculus model for dispersion and absorption in dielectrics ii. infrared waves," *International Journal of Engineering Science*, vol. 104, pp. 62–74, 2016 58, 61.
29. R. Garrappa, "Numerical evaluation of two and three parameter mittag-leffler functions," *SIAM Journal on Numerical Analysis*, vol. 53, no. 3, 2015. DOI: [10 . 1137 / 140971191](https://doi.org/10.1137/140971191) 73.

APPENDIX A - SPECIAL FUNCTIONS

$$\Gamma(z) = \int_0^{\infty} x^{z-1} e^{-x} dx \quad (\text{A-1})$$

Equation A-1. Gamma function

$$H(t) = \begin{cases} 0, & t < 0 \\ \frac{1}{2}, & t = 0 \\ 1, & t > 0 \end{cases} \quad (\text{A-2})$$

Equation A-2. Heaviside unit step function

$$E_{\alpha}(z) = \sum_{k=0}^{\infty} \frac{z^k}{\Gamma(\alpha k + 1)} \quad (\text{A-3})$$

Equation A-3. One-parameter Mittag-Leffler function

$$E_{\alpha,\beta}(z) = \sum_{k=0}^{\infty} \frac{z^k}{\Gamma(\alpha k + \beta)} \quad (\text{A-4})$$

Equation A-4. Two-parameter Mittag-Leffler function

The fractional Green's functions are used for the analytical solutions for fractional differential equations (FDEs) with constant coefficients. The one-term, two-term, and three-term fractional Green's functions are defined below.

$$a_0 D_t^{\alpha} [y(t)] = f(t) \quad (\text{A-5})$$

Equation A-5. General one-term FDE with constant coefficients

The corresponding Green's function is solved by inverse Laplace transform of the expression:

$$g_1(p) = \frac{1}{ap^{\alpha}}$$

which yields:

$$G_1(t) = \frac{1}{a} \frac{t^{\alpha-1}}{\Gamma(\alpha)} \quad (\text{A-6})$$

Equation A-6. Fractional Green's function for one-term FDE with constant coefficients

For the two-term FDE:

$$a_0 D_t^{\alpha} [y(t)] + by(t) = f(t) \quad (\text{A-7})$$

Equation A-7. General two-term FDE with constant coefficients

The corresponding Green's function is solved by inverse Laplace transform of the expression:

$$g_2(p) = \frac{1}{ap^{\alpha} + b} = \frac{1}{a} \frac{1}{p^{\alpha} + \frac{b}{a}}$$

which yields:

$$G_2(t) = \frac{1}{a} t^{\alpha-1} E_{\alpha,\alpha} \left(-\frac{b}{a} t^\alpha \right) \quad (\text{A-8})$$

Equation A-8. Fractional Green's function for two-term FDE with constant coefficients

For the three-term FDE:

$$a_0 D_t^\beta [y(t)] + b_0 D_t^\alpha [y(t)] + cy(t) = f(t) \quad (\text{A-9})$$

Equation A-9. General three-term FDE with constant coefficients

The corresponding Green's function is solved by inverse Laplace transform of the expression:

$$g_3(p) = \frac{1}{ap^\beta + bp^\alpha + c}$$

which yields:

$$G_3(t) = \frac{1}{a} \sum_{k=0}^{\infty} \frac{(-1)^k}{k!} \left(\frac{c}{a} \right)^k t^{\beta(k+1)-1} E_{\beta-\alpha, \beta+\alpha k}^{(k)} \left(-\frac{b}{a} t^{\beta-\alpha} \right) \quad (\text{A-10})$$

Equation A-10. Fractional Green's function for three-term FDE with constant coefficients

where

$$E_{\gamma,\eta}^{(k)}(y) \equiv \frac{d^k}{dy^k} E_{\gamma,\eta}(y) = \sum_{j=0}^{\infty} \frac{(j+k)! y^j}{j! \Gamma(\gamma j + \eta)}, \quad (k = 0, 1, 2, \dots). \quad (\text{A-11})$$

Equation A-11. k -th derivative of the two-parameter Mittag-Leffler function

For the general n -term case, refer to [14] Chapter 5.6.

APPENDIX B - SOFTWARE CODE AND SPECIFICATIONS

The figures produced in this technical report used Python 3 to solve and plot numerical and analytical solutions. Note the Python indentation conventions if copying over the code from the technical report to a compiler. Refer to Python documentation for any syntax errors that may potentially arise from compilation. The details of Python version and libraries used are noted here.

- Python 3.7.6
- numpy 1.18.1
- scipy 1.4.1
- matplotlib 3.1.3
- Mittag-Leffler Python implementation: <https://github.com/khinsen/mittag-leffler>
 - Need both `mittag_leffler.py` and `ml_internal.py` files
 - Based on Garrappa's MATLAB implementation [29]:
<https://www.mathworks.com/matlabcentral/fileexchange/48154-the-mittag-leffler-function>

B.1 Relaxation-Oscillation Equation

```
import numpy as np
from scipy.special import gamma, binom
from mittagleffler import ml
import time

class Relax_Osc:
    """
    Initialize class given a dict "params"
    """
    def __init__(self, params):
        self.Acoeff = params["Acoeff"]
        self.alpha = params["alpha"]
        self.f = params["f"]
        self.ICs = params["ICs"]
        self.m = int(np.ceil(self.alpha))
        assert 0 < self.m <= 2, "ceil(alpha) must be 0 < ceil(alpha) <= 2"
        assert self.m == len(self.ICs), "Must have ceil(alpha) number of
            initial conditions defined"
        self.method = params["method"]

        self.tmin = params["tmin"]
        self.tmax = params["tmax"]
        self.dt = params["dt"]
        self.tnum = np.arange(self.tmin, self.tmax, self.dt)
        self.ynum = np.zeros(self.tnum.shape[0])
        self.wj = np.array([(-1)**j * binom(self.alpha, j) for j in range(len(
            self.tnum))])
```

```

self.analytic = params["analytic"]
if self.analytic:
    self.tanalytic = np.arange(self.tmin, self.tmax, (self.tmax-self.
        tmin)/1000.)
    self.yanalytic = np.zeros(self.tanalytic.shape[0])
self.exec_tanalyt = 0.; self.exec_tnum = 0.
self.MAE = 0.; self.MSE = 0.
self.error = params["error"]
self.write = params["write"]

def analytical(self):
    tanalyt0 = time.time()
    A = self.Acoeff
    start = min(self.tanalytic)
    print("Analytical simulation size: %2d points" %self.tanalytic.shape
        [0])
    print("Evaluating analytical solution...")
    # Define integrand for analytical solution
    G2 = lambda t: t**(self.alpha-1) * ml(-A*t**self.alpha, self.alpha, self
        .alpha)
    G2res = G2(self.tanalytic)
    # Solve for analytical solution
    f = self.f(self.tanalytic)
    for i in range(1, len(self.tanalytic)):
        self.yanalytic[i] = np.trapz(G2res[:i][::-1]*f[:i], self.tanalytic
            [:i])
    tanalyt1 = time.time()
    self.exec_tanalyt = tanalyt1 - tanalyt0
    print("DONE!")
    return self.yanalytic

def numerical(self):
    tnum0 = time.time()
    print("Numerical simulation size: %2d points" %self.tnum.shape[0])
    print("Evaluating numerical solution...")
    T = len(self.tnum)
    A = self.Acoeff
    self.ynum[0] = self.ICs[0]
    if self.m == 2:
        self.ynum[1] = self.ICs[1] * self.dt + self.ynum[0]
    if self.method == "GL":
        for i in range(self.m, T):
            term1 = A*self.dt**self.alpha * self.ynum[i-1]
            term2 = np.dot(self.wj[1:i+1], self.ynum[i-1::-1]) # Main
                fractional derivative calculation
            term3 = self.dt**self.alpha * self.f(self.tnum[i])
            self.ynum[i] = -term1 - term2 + term3
    if self.method == "matrix":
        from scipy.linalg import solve_triangular
        # Generate LHS matrix
        left = np.zeros((T-self.m, T-self.m))
        main_diag = np.diag_indices(T-self.m)

```

```

        # Lower triangular matrix with GL coefficients
        for i in range(T-self.m):
            left[i,:i+1] = self.wj[:i+1][::-1]
        left *= self.dt**-self.alpha
        left[main_diag] += A
        # Generate RHS vector values
        right = self.f(self.tnum[self.m:])
        # Solve for numerical solution
        self.ynum[self.m:] = solve_triangular(left,right,lower=True,
            check_finite=True)
    tnum1 = time.time()
    self.exec_tnum = tnum1 - tnum0
    print("DONE!")
    return self.ynum

def plot(self):
    import matplotlib.pyplot as plt
    plt.figure(figsize=(7,5))
    plt.plot(self.tnum,self.ynum,label="Numerical")
    if self.analytic:
        plt.plot(self.tanalytic,self.yanalytic,label="Analytical")
    plt.xlabel("Time")
    plt.ylabel("y(t)")
    plt.title(r"Relaxation-Oscillation Model (h = %.3f)" %self.dt)
    plt.xlim(min(self.tnum),max(self.tnum))
    plt.legend(loc=1)
    plt.show()

def run(self):
    self.get_params()
    self.numerical()
    print("Time to calculate numerical solution: %.6f" %self.exec_tnum)
    print("-----")
    if self.analytic:
        self.analytical()
        print("Time to calculated analytical solution: %.6f" %self.
            exec_tanalyt)
        if self.error:
            self.get_error()
            print("MAE: %.3e, MSE: %.3e" %(self.MAE,self.MSE))
    if (not self.analytic and self.error):
        print("Cannot calculate error without analytical solution")
    if self.write:
        store_data = "RO_results_" + self.method + ".txt"
        f = open(store_data,"a")
        f.write("%.6f,%.6f,%.6f,%.6f,%.6e,%.6e\n" %(self.alpha,self.dt,
            self.exec_tnum,self.exec_tanalyt,self.MAE,self.MSE))
        f.close()
    print("=====\n")

def get_error(self):
    """
    Since the number of points calculated for the analytical and numerical
    solutions are not necessarily the same,

```

```

we match the common time points of both analytical and numerical
solutions and average the error of all matched points.
"""
self.MAE = 0.; self.MSE = 0.
dt_analyt = (self.tmax-self.tmin)/1000. # predetermined number of step
size for analytical solution for sufficient resolution
print("Calculating error...")
j = 0; count = 0
# Matching algorithm is faster if you for loop goes through fewer
points!
# Loop through analytical points and scan through numerical points
only once for matches
if dt_analyt >= self.dt:
    for i in range(len(self.tanalytic)):
        prev = j
        while not np.isclose(self.tnum[j], self.tanalytic[i]):
            if self.tnum[j] > self.tanalytic[i] or j == len(self.tnum)
            -1:
                j = prev
                break
            j += 1
        if j != prev:
            self.MAE += np.abs(self.yanalytic[i] - self.ynum[j])
            self.MSE += (self.yanalytic[i] - self.ynum[j])**2
            count += 1
# Loop through numerical points and scan through analytical points
only once for matches
if dt_analyt < self.dt:
    for i in range(len(self.tnum)):
        prev = j
        while not np.isclose(self.tnum[i], self.tanalytic[j]):
            if self.tanalytic[j] > self.tnum[i] or j == len(self.
            tanalytic)-1:
                j = prev
                break
            j += 1
        if j != prev:
            self.MAE += np.abs(self.yanalytic[j] - self.ynum[i])
            self.MSE += (self.yanalytic[j] - self.ynum[i])**2
            count += 1
# If no matches are made, set error to 0 to avoid divide by zero
if count == 0:
    self.MAE = 0.; self.MSE = 0.
else:
    self.MAE = self.MAE/count; self.MSE = self.MSE/count
print("DONE!")
return self.MAE, self.MSE

def output(self):
    return self.t, self.ynum, self.tanalytic, self.yanalytic

def get_params(self):
    print("=====")
    print("RELAXATION-OSCILLATION PROBLEM:")

```

```

print("-----")
print("_0D_t^alpha y(t) + A*y(t) = f(t)")
print("-----")
print("PARAMS:")
print("alpha    = %.2f" %self.alpha)
print("method   = %s" %self.method)
print("t range = [%.2f,%.2f]" %(min(self.tnum),max(self.tnum)))
print("    y(0) = %.2f, y\'(0) = %.2f" %(self.ICs[0],self.ICs[1]))
print("    dt = %.3f" %self.dt)

if __name__ == "__main__":
    """
    _0D_t^alpha y(t) + Ay(t) = f(t)
    """
    params = {
        "Acoeff" : 0.5,          # A coefficient
        "alpha"   : 1.5,          # alpha-order derivative
        "f"       : lambda t : np.heaviside(t,0), # Use lambda function to
            define your source function
        "tmin"    : 0,            # Start time
        "tmax"    : 10,           # End time
        "dt"      : 0.001,        # Time step
        "ICs"     : [0,0],        # Define initial conditions, must satisfy
            existence (ceil(alpha) = # initial conditions)
        "method"  : "matrix",     # Pick numerical method, "GL" or "matrix"
        "analytic" : True,        # Solve for analytical solution? True or False
        "error"   : True,         # Evaluate both mean absolute error and mean
            square error (MAE and MSE)
        "write"   : True          # Write 6 values for: alpha, dt, numerical
            comp time, analytical comp time, MAE, MSE
    }
    # How to initialize class and run code
    #RO = Relax_Osc(params)
    #RO.run()
    #RO.plot()
    #t,ynum,tanalytic,yanalytic = RO.output()
    # How to do multiple runs
    step = np.arange(0.001,0.011,0.001)
    step = np.concatenate((step,np.arange(0.02,0.11,0.01)))
    step = np.concatenate((step,np.arange(0.2,1.1,0.1)))
    iterations = 1
    for h in step:
        params["dt"] = h
        RO = Relax_Osc(params)
        for i in range(iterations):
            RO.run()
            # RO.plot()

```

B.2 Bagley-Torvik Equation

```
import numpy as np
from scipy.special import gamma, binom
import time

class Bagley_Torvik:
    def __init__(self, params):
        self.A, self.B, self.C = params["coeffs"]
        self.alpha = params["alpha"]
        self.f = params["f"]
        self.ICs = params["ICs"]
        self.m = 2
        assert self.m == len(self.ICs), "Must have %ld number of initial
            conditions defined" %self.m
        self.method = params["method"]

        tmin = params["tmin"]
        tmax = params["tmax"]
        self.dt = params["dt"]
        self.tnum = np.arange(tmin, tmax, self.dt)
        self.ynum = np.zeros(self.tnum.shape[0])
        self.wj = np.array([(-1)**j * binom(self.alpha, j) for j in range(len(
            self.tnum))])

        self.analytic = params["analytic"]
        if self.analytic:
            self.tanalytic = np.arange(tmin, tmax, (tmax-tmin)/1000.) # 1000
                steps is recommended for best accuracy and time efficiency
            self.yanalytic = np.zeros(self.tanalytic.shape[0])
            self.tmin = tmin; self.tmax = tmax
            self.exec_tanalyt = 0.; self.exec_tnum = 0.
            self.MAE = 0; self.MSE = 0.
            self.error = params["error"]
            self.write = params["write"]

    def analytical(self):
        tanalyt0 = time.time()
        # Number of iterations for analytical solution to converge, adjust if
            needed for accuracy and/or time efficiency
        iterations = 50
        G3res = np.zeros(self.tanalytic.shape[0])
        print("Analytical simulation size: %2d points" %self.tanalytic.shape
            [0])
        print("Evaluating analytical solution...")
        # Find Green's function values at each point and store
        for i in range(len(self.tanalytic)):
            G3 = 0
            for k in range(iterations):
                mlk = 0
                for j in range(iterations):
                    mlk += (gamma(j+k+1) * ((-self.B/self.A)*self.tanalytic[i
                        ]**0.5)**j) / \
                        (gamma(j+1) * gamma(0.5*j + 0.5*k + 2 + (3.*k/2)))
                G3 += (1/self.A) * ((-1)**k)/gamma(k+1) * (self.C/self.A)**k *
```

```

        self.tanalytic[i]**(2*k+1) * mlk
    G3res[i] = G3
    # Get values for source function, f
    f = self.f(self.tanalytic)
    # Solve the convolution integral
    for i in range(1,len(self.tanalytic)):
        self.yanalytic[i] = np.trapz(G3res[:i][::-1]*f[:i],self.tanalytic
            [:i])
    tanalyt1 = time.time()
    self.exec_tanalyt = tanalyt1 - tanalyt0
    print("DONE!")
    return self.yanalytic

def numerical(self):
    print("Numerical simulation size: %2d points" %len(self.tnum))
    print("Evaluating numerical solution...")
    tnum0 = time.time()
    T = len(self.tnum)
    A = self.A; B = self.B; C = self.C
    self.ynum[0] = self.ICs[0]
    self.ynum[1] = self.ICs[1] * self.dt + self.ynum[0]
    if self.method == "GL":
        for i in range(self.m,T):
            term1 = (self.dt**2) * (self.f(self.tnum[i]) - C*self.ynum[i-1])
            term2 = A * (2*self.ynum[i-1] - self.ynum[i-2])
            term3 = B * (self.dt**0.5 * np.dot(self.wj[1:i],self.ynum[1:i]
                [::-1])) # Main fractional derivative calculation
            numer = term1 + term2 - term3
            denom = A + B*(self.dt**0.5)
            self.ynum[i] = numer/denom
    if self.method == "matrix":
        from scipy.linalg import solve_triangular
        Tm = T-self.m
        left = np.zeros((Tm,Tm))
        # Generate lower triangular matrix with GL coefficients for
        # fractional derivative
        for i in range(T-self.m):
            left[i,:i+1] = self.wj[:i+1][::-1]
        left *= B * self.dt**-self.alpha
        # Add backward difference matrix for second-order derivative
        main_diag = np.diag_indices(Tm)
        lower1_diag = (main_diag[0][1:],(main_diag[1]-1)[1:])
        lower2_diag = (main_diag[0][2:],(main_diag[1]-2)[2:])
        left[main_diag] += 1. * A * self.dt**-2.
        left[lower1_diag] += -2. * A * self.dt**-2.
        left[lower2_diag] += 1. * A * self.dt**-2.
        # Add identity matrix for term with no derivative
        left[main_diag] += C
        # Generate RHS vector
        right = self.f(self.tnum[self.m:])
        # Solve for numerical solution
        self.ynum[self.m:] = solve_triangular(left,right,lower=True,
            check_finite=True)

```

```

tnum1 = time.time()
self.exec_tnum = tnum1 - tnum0
print("DONE!")
return self.ynum

def plot(self):
    import matplotlib.pyplot as plt
    plt.figure(figsize=(7,5))
    plt.plot(self.tnum,self.ynum,label="Numerical")
    if self.analytic:
        plt.plot(self.tanalytic,self.yanalytic,label="Analytical")
    plt.xlabel("Time")
    plt.ylabel("y(t)")
    plt.title("Bagley-Torvik Model (h = %.3f)" %self.dt)
    plt.xlim(min(self.tnum),max(self.tnum))
    plt.legend(loc=1)
    plt.show()

def run(self):
    self.get_params()
    self.numerical()
    print("Time to calculate numerical solution: %.6f" %self.exec_tnum)
    print("-----")
    if self.analytic:
        self.analytical()
        print("Time to calculated analytical solution: %.6f" %self.
            exec_tanalyt)
        if self.error:
            self.get_error()
            print("MAE: %.3e, MSE: %.3e" %(self.MAE,self.MSE))
    if (not self.analytic and self.error):
        print("Cannot calculate error without analytical solution")
    if self.write:
        store_data = "BT_results_" + self.method + ".txt"
        f = open(store_data,"a")
        f.write("%.6f,%.6f,%.6f,%.6f,%.6e,%.6e\n" %(self.alpha,self.dt,
            self.exec_tnum,self.exec_tanalyt,self.MAE,self.MSE))
        f.close()
    print("=====\n")

def get_error(self):
    """
    Since the number of points calculated for the analytical and numerical
    solutions are not necessarily the same,
    we match the common time points of both analytical and numerical
    solutions and average the error of all matched points.
    """
    self.MAE = 0.; self.MSE = 0.
    dt_analyt = (self.tmax-self.tmin)/1000. # predetermined number of step
        size for analytical solution for sufficient resolution
    print("Calculating error...")
    j = 0; count = 0
    # Matching algorithm is faster if you for loop goes through fewer
    points!

```

```

# Loop through analytical points and scan through numerical points
  only once for matches
if dt_analyt >= self.dt:
    for i in range(len(self.tanalytic)):
        prev = j
        while not np.isclose(self.tnum[j], self.tanalytic[i]):
            if self.tnum[j] > self.tanalytic[i] or j == len(self.tnum)
                -1:
                    j = prev
                    break
            j += 1
        if j != prev:
            self.MAE += np.abs(self.yanalytic[i] - self.ynum[j])
            self.MSE += (self.yanalytic[i] - self.ynum[j])**2
            count += 1
# Loop through numerical points and scan through analytical points
  only once for matches
if dt_analyt < self.dt:
    for i in range(len(self.tnum)):
        prev = j
        while not np.isclose(self.tnum[i], self.tanalytic[j]):
            if self.tanalytic[j] > self.tnum[i] or j == len(self.
                tanalytic)-1:
                    j = prev
                    break
            j += 1
        if j != prev:
            self.MAE += np.abs(self.yanalytic[j] - self.ynum[i])
            self.MSE += (self.yanalytic[j] - self.ynum[i])**2
            count += 1
# If no matches are made, set error to 0 to avoid divide by zero
if count == 0:
    self.MAE = 0.; self.MSE = 0.
else:
    self.MAE = self.MAE/count; self.MSE = self.MSE/count
print("DONE!")
return self.MAE, self.MSE

def get_params(self):
    print("BAGLEY-TORVIK PROBLEM:")
    print("-----")
    print("A*y\"(t) + B*_0D_t^(3/2) y(t) + C*y(t) = f(t)")
    print("-----")
    print("PARAMS:")
    print("alpha   = %.2f" %self.alpha)
    print("method   = %s" %self.method)
    print("t range = [%.2f,%.2f]" %(min(self.tnum),max(self.tnum)))
    print("    y(0) = %.2f, y\'(0) = %.2f" %(self.ICs[0],self.ICs[1]))
    print("    dt = %.3f" %self.dt)

def output(self):
    return self.tnum, self.ynum, self.tanalytic, self.yanalytic

if __name__ == "__main__":

```

```

params = {
    "coeffs" : [1.,0.5,0.5], # A, B, C coefficients
    "alpha" : 1.5, # For Bagley-Torvik, this should always be
        1.5
    "f" : lambda t : 8. * (t < 1), # Use lambda function to define
        your source function
    "tmin" : 0, # Start time
    "tmax" : 30, # End time
    "dt" : 0.01, # Time step (for matrix, dt = 0.001 for
        (0,30) overloads scipy.solve_triangular)
    "ICs" : [0,0], # Initial conditions, must have 2 defined
        for BT problem
    "method" : "GL", # Pick numerical method
    "analytic" : True, # Solve for analytical solution? True or
        False
    "error" : True, # Evaluate both mean absolute error and
        mean square error (MAE and MSE)
    "write" : True # Write 6 values: alpha, dt, numerical
        comp time, analytical comp time, MAE, MSE
}
# How to initialize class and run code
# BT = Bagley_Torvik(params)
# BT.run()
# BT.plot()
# How to run multiple instances
step = np.arange(0.005,0.011,0.001)
step = np.concatenate((step,np.arange(0.02,0.11,0.01)))
step = np.concatenate((step,np.arange(0.2,1.1,0.1)))
iterations = 1
for h in step:
    params["dt"] = h
    BT = Bagley_Torvik(params)
    for i in range(iterations):
        BT.run()
        # BT.plot()

```

B.3 Fractional Lumped Capacitance Model

```
import numpy as np
from scipy.special import gamma, binom
from mittleleffler import ml

class Frac_Lumped:
    def __init__(self, params):
        data = np.loadtxt(params["exp"])
        ambient = np.where(data[:,1] < 30)[0][-1]
        data = data[ambient:]
        print("Ambient temperature: %.3f C" %min(data[:,1]))
        data[:,0] -= min(data[:,0])
        data[:,1] -= min(data[:,1])
        sing = np.where(data[:,1] == max(data[:,1]))[0][0]
        self.rise = np.concatenate((data[sing+1,0][:,None],data[sing+1,1][:,
            None]),axis=1)
        self.fall = np.concatenate((data[sing:,0][:,None],data[sing:,1][:,None
            ]),axis=1)
        self.fall -= min(self.fall[:,0])
        self.data = data
        self.sing = sing
        self.Eg = params["Eg"]
        self.analytic = params["analytic"]

        bioheat = np.genfromtxt(params["bioheat"],delimiter=",")
        bioheat[:,1] -= bioheat[:,1][0]
        self.bioheat = bioheat
        self.analyt_heat_coeff = params["analyt_heat_coeff"]
        self.analyt_cool_coeff = params["analyt_cool_coeff"]
        self.num_heat_coeff = params["num_heat_coeff"]
        self.num_cool_coeff = params["num_cool_coeff"]

    def heating(self,t):
        a,b,c,alpha = self.analyt_heat_coeff
        return b * t * ml(-a*t**alpha,alpha,2)

    def cooling(self,t):
        a,b,c,alpha = self.analyt_cool_coeff
        return c * t**(alpha-1) * ml(-a*t**alpha,alpha,alpha)

    def analytical_fit(self):
        shift = self.rise[:,0][self.sing]
        self.tfit = np.concatenate((self.rise[:,0],self.fall[:,0]+shift),axis
            =0)
        lumpy = np.concatenate((self.rise[:,1],self.fall[:,1]),axis=0)
        self.analyt_fit = np.concatenate((self.heating(self.rise[:,0]),self.
            cooling(self.fall[:,0])),axis=0)
        return self.analyt_fit

    def numerical_fit(self):
        peak = self.data[:,0][self.sing]
        start = 0.; end = 0.1; dt = 0.0001
        self.tnum = np.arange(start,end,dt)
```

```

A1,_,_,alpha1 = self.num_heat_coeff
A2,_,_,alpha2 = self.num_cool_coeff
laser = lambda t : self.Eg*t**(1-alpha1)/gamma(2-alpha1) * (0 < t <
    peak)
wj = []
for j in range(1,len(self.tnum)):
    if self.tnum[j] <= peak:
        wj.append(((1)**j)*binom(alpha1,j))
    elif self.tnum[j] > peak:
        wj.append(((1)**j)*binom(alpha2,j))
wj = np.array(wj)
self.ynum = np.zeros(self.tnum.shape[0])
self.ynum[0] = self.data[0][1]
for i in range(1,len(self.tnum)):
    if self.tnum[i] <= peak:
        term1 = -A1*(dt**alpha1)*self.ynum[i-1]
        term2 = -np.dot(wj[:i],self.ynum[i-1::-1])
        term3 = dt**alpha1 * laser(self.tnum[i])
    elif self.tnum[i] > peak:
        term1 = -A2*(dt**alpha2)*self.ynum[i-1]
        term2 = -np.dot(wj[:i],self.ynum[i-1::-1])
        term3 = dt**alpha2 * laser(self.tnum[i])
    self.ynum[i] = term1 + term2 + term3
return self.ynum

def plot(self):
    import matplotlib.pyplot as plt
    plt.figure(figsize=(10,7))
    plt.scatter(self.data[:,0],self.data[:,1],marker="+",label="
        Experimental",color="k",alpha=0.5)
    plt.plot(self.tnum,self.ynum,label="Fractional Model")
    if self.analytic:
        plt.plot(self.tfit,self.analyt_fit,label="Analytical Fit")
    plt.plot(self.bioheat[:,0],self.bioheat[:,1],label="Pennes Bioheat")
    plt.xlim(0,0.1)
    plt.ylim(0,)
    plt.xlabel("Time (s)")
    plt.ylabel(r"dT ($^{\circ}$C)")
    plt.title("Fractional Lumped Capacitance Model")
    plt.legend(loc=1)
    plt.show()

def run(self):
    self.get_params()
    self.numerical_fit()
    if self.analytic:
        self.analytical_fit()

def output(self):
    return self.tnum, self.ynum, self.tfit, self.analyt_fit

def get_params(self):
    print ("FRACTIONAL LUMPED CAPACITANCE MODEL")
    print ("-----")

```

```

print("PARAMS:")
print("Numerical heating curve coeffs:")
print("a = %.3f, b = %.3f, c = %.3f, alpha = %.3f" %tuple(self.
    num_heat_coeff))
print("Numerical cooling curve coeffs:")
print("a = %.3f, b = %.3f, c = %.3f, alpha = %.3f" %tuple(self.
    num_cool_coeff))
print("Analytical heating curve coeffs:")
print("a = %.3f, b = %.3f, c = %.3f, alpha = %.3f" %tuple(self.
    analyt_heat_coeff))
print("Analytical cooling curve coeffs:")
print("a = %.3f, b = %.3f, c = %.3f, alpha = %.3f" %tuple(self.
    analyt_cool_coeff))
print("Eg = %.3f" %self.Eg)

if __name__ == "__main__":
    params = {
        "exp" : "0.01s,1.037cm,1p,2780W.txt",
        "bioheat" : "0.01s,1cm,2780W-SESE.csv",
        "analyt_heat_coeff" : [10.9,3600,0,0.841],
        "analyt_cool_coeff" : [2.625,0,17.5,0.906],
        "num_heat_coeff" : [10.9,3600,0,0.841],
        "num_cool_coeff" : [0.1,0,17.5,0.855],
        "Eg" : 2780*1.28,
        "analytic" : True
    }
    FLC = Frac_Lumped(params)
    FLC.run()
    FLC.plot()

```

B.4 Fractional Partial Differential Equations

```
import numpy as np
from scipy.special import gamma, binom

class FPDE:

    ### Initialize the y matrix (stores results of numerical calculations)
    # Contains N rows for the space dimension, contains M columns for time
    dimension
    # y[space-index,time-index] is how to access y-values at a particular
    location
    # y[space-index,:] is how to access y-values at a given location in space
    # y[:,time-index] is how to access y-values at a given moment in time
    def __init__(self,params):
        # Set alpha
        self.alpha = params["alpha"]

        # Set up space and time grids
        xmin = params["xmin"]; xmax = params["xmax"]
        tmin = params["tmin"]; tmax = params["tmax"]
        dx = params["dx"]; dt = params["dt"]
        x = np.arange(xmin,xmax,dx); N = len(x)
        t = np.arange(tmin,tmax,dt); M = len(t)

        # Define initial condition function and set to y0
        self.y = np.zeros((N,M))
        self.y0 = params["ICs"]
        self.y[:,0] = self.y0(x)

        # Define coefficient functions
        self.f = params["f"]
        self.K = params["K"]

        # Define boundary conditions
        self.r1,self.r2 = params["BCs"]
        self.y[0,:] = self.r1(t); self.y[-1,:] = self.r2(t)

        # Allocate memory for Hm vector and Gm matrix
        self.Hm = np.zeros(x.shape[0])
        self.Gm = np.zeros((N,N))

        # Redefine to self
        self.x = x; self.t = t
        self.N = N; self.M = M
        self.dx = dx; self.dt = dt

    ### Calculates the B coefficient values given mu- and m-values
    def Bcoeff(self,mu,m):
        if mu == 0:
            B = (m-1)**(1-self.alpha) - (m-1+self.alpha)*m**(-self.alpha)
        else:
            B = (m-mu+1)**(1-self.alpha) - 2*(m-mu)**(1-self.alpha) + (m-mu-1)
                ** (1-self.alpha)
        return B
```

```

## Calculate the Hm vector given m-values
def Hmvec(self,m):
    self.Hm[0] = self.r1(self.t[m])
    self.Hm[-1] = self.r2(self.t[m])
    for i in range(1,self.N-1): # iterate through x values
        Hmi = (1-self.alpha)*m**(-self.alpha)*self.y0(self.x[i]) \
            + gamma(2-self.alpha)*self.dt**self.alpha * self.f(self.x[i],
                self.t[m])
        for mu in range(m):
            Hmi -= self.Bcoeff(mu,m) * self.y[i,mu]
        self.Hm[i] = Hmi
    return self.Hm

## Calculates the Gm matrix given m-values
def Gmat(self,m):
    # Find indices of upper, main, and lower diagonals
    main_diag = np.diag_indices(self.N); main,_ = main_diag
    lower_diag = (main_diag[0][1:],(main_diag[1]-1)[1:]); lower,_ =
        lower_diag
    upper_diag = (main_diag[0][: -1],(main_diag[1]+1)[: -1]); upper,_ =
        upper_diag
    # Set values of the upper, main, and lower diagonals
    self.Gm[main_diag] = -2*self.K(self.x[main],self.t[m])
    self.Gm[lower_diag] = self.K(self.x[lower],self.t[m])
    self.Gm[upper_diag] = self.K(self.x[upper],self.t[m])
    # First and last rows are all zero to account for boundary conditions
    self.Gm[0,0] = 0.; self.Gm[0,1] = 0.
    self.Gm[-1,-1] = 0.; self.Gm[-1,-2] = 0.
    return self.Gm

## Solve numerically
def numerical(self):
    self.get_params()
    from scipy.linalg import solve
    print("Numerical simulation size: %.2d x %.2d points" %(self.x.shape
        [0],self.t.shape[0]))
    print("Evaluating numerical solution...")
    I = np.eye(self.N)
    for m in range(1,self.M):
        self.Hmvec(m)
        self.Gmat(m)
        LHS = I - ((gamma(2-self.alpha)*self.dt**self.alpha)/self.dx**2) *
            self.Gm
        self.y[:,m] = solve(LHS,self.Hm)
    print("DONE!")
    return self.y

## Plot results on 3D plot
def plot(self):
    import matplotlib.pyplot as plt
    from mpl_toolkits.mplot3d import Axes3D
    fig = plt.figure(figsize=(10,7))
    ax = fig.add_subplot(111,projection="3d")

```

```

        # Create (N x M) matrices corresponding to space and time for each y(x
        ,t)
        X,T = np.meshgrid(self.x,self.t)
        # Plot results
        ax.plot_wireframe(X,T,self.y.T,alpha=0.5)
        ax.set_title("Time-Fractional Partial Differential Equation")
        ax.set_xlabel("x")
        ax.set_ylabel("t")
        ax.set_zlabel("y(x,t)")
        plt.show()

    ## Print out input parameters
    def get_params(self):
        print("-----")
        print("TIME-FRACTIONAL PARTIAL DIFFERENTIAL EQUATIONS")
        print("-----")
        print("PARAMS:")
        print("alpha = %.2f" %self.alpha)
        print("x range = [%.2f,%.2f]" %(min(self.x),max(self.x)))
        print("t range = [%.2f,%.2f]" %(min(self.t),max(self.t)))
        print("dx = %.2f, dt = %.2f" %(self.dx, self.dt))

    ## Return the result values
    def output(self):
        return self.x, self.t, self.y

if __name__ == "__main__":
    params = {
        "alpha": 0.5,
        "tmin" : 0.,
        "tmax" : 10.,
        "dt" : 0.1,
        "xmin" : 0.,
        "xmax" : np.pi,
        "dx" : 0.01,
        "BCs" : [lambda t: 0., lambda t: 0.], #[lambda t: np.exp(-t) , lambda
            t: -np.exp(-t)],
        "ICs" : lambda x: np.sin(x),
        "f" : lambda x,t : 0., #lambda x,t: np.cos(x) * np.cos(t),
        "K" : lambda x,t : 1. #lambda x,t: 2 + (1 + x*(2*np.pi - x))/(5+5*t
        )
    }
    FPDE1 = FPDE(params)
    FPDE1.numerical()
    FPDE1.plot()

```

B.5 Time-Fractional Diffusion-Wave Equation

```
import numpy as np
from scipy.special import gamma, binom

"""
Diethelm, K., J. M. Ford, N. J. Ford, and M. Weilbeer. "A comparison of
backward differentiation approaches for ordinary and partial differential
equations of fractional order." In Proceedings of the 1st IFAC Workshop on
Fractional Differentiation and its Applications, Bourdeaux, France. 2004.
"""

class Diffusion_Wave:
    def __init__(self, params):
        self.alpha = params["alpha"]
        xmin = params["xmin"]; xmax = params["xmax"]
        tmin = params["tmin"]; tmax = params["tmax"]
        dx = params["dx"]; dt = params["dt"]
        x = np.arange(xmin, xmax, dx); N = len(x)
        t = np.arange(tmin, tmax, dt); M = len(t)

        self.u = np.zeros((N, M))
        assert 0. <= self.alpha <= 2., "Please input 0 <= alpha <= 2"
        assert len(params["ICs"]) == int(np.ceil(self.alpha)), "Please input
            the right number of initial conditions"
        if int(np.ceil(self.alpha)) == 1:
            self.u0 = params["ICs"][0]
            self.u[:, 0] = self.u0(x)
        if int(np.ceil(self.alpha)) == 2:
            self.u0, self.u1 = params["ICs"]
            self.u[:, 0] = self.u0(x)
            self.u[:, 1] = self.u1(x)*dt + self.u[:, 0]

        self.g1, self.g2 = params["g"]
        self.h1, self.h2 = params["h"]
        self.r1, self.r2 = params["r"]
        self.phi = params["phi"]
        self.f = params["f"]

        self.x = x; self.t = t
        self.N = N; self.M = M
        self.dx = dx; self.dt = dt

    # Calculate the
    def omega(self, j, k):
        # Use the Diethelm coefficients for 0 < alpha < 1
        if 0. < self.alpha < 1.:
            omg = 1.
            if 1 <= j <= k-1:
                omg = (j-1)**(1-self.alpha) - 2*j**(1-self.alpha) + (j+1)**(1-
                    self.alpha)
            if j == k and j >= 1:
                omg = (k-1)**(1-self.alpha) + (1-self.alpha-k)*k**-self.alpha
            if j == 0:
                omg = 1.
```

```

        omg *= 1/gamma(2-self.alpha)
        # Use the modified GL coefficients of  $1 \leq \alpha \leq 2$  or  $\alpha = 0$ 
elif 1. <= self.alpha <= 2. or self.alpha == 0.:
        omg = (-1)**j * binom(self.alpha, j)
        if j > 0 and k-j == 0:
            omg += j**(-self.alpha) / gamma(1-self.alpha)
            omg -= (-1)**j * gamma(self.alpha) / (gamma(self.alpha-j) * gamma
                (j+1))
        return omg

def dcoeff(self, k):
    g1term = self.g1(self.t[k])/self.dx
    g2term = self.g2(self.t[k])/self.dx
    h1term = self.h1(self.t[k])*(-1)**1
    h2term = self.h2(self.t[k])*(-1)**2
    dlcoeff = g1term + h1term \
        * ((1/self.dx**2) - self.omega(k, k) / (2*self.phi(min(self.x), k)
            *self.dt**self.alpha))
    d2coeff = g2term + h2term \
        * ((1/self.dx**2) - self.omega(k, k) / (2*self.phi(max(self.x), k)
            *self.dt**self.alpha))
    return dlcoeff, d2coeff

def ocoeff(self, k):
    olcoeff = (-1)**(1+1)*self.h1(self.t[k])/self.dx**2
    o2coeff = (-1)**(2+1)*self.h2(self.t[k])/self.dx**2
    return olcoeff, o2coeff

def rcoeff(self, k):
    a = min(self.x); b = max(self.x)
    r1term = self.r1(self.t[k])/self.dx
    r2term = self.r2(self.t[k])/self.dx
    h1term = ((-1)**(1+1) * self.h1(self.t[k]) / (2*self.phi(a, self.t[k])))
    h2term = ((-1)**(2+1) * self.h2(self.t[k]) / (2*self.phi(b, self.t[k])))
    sum1 = (1/self.t[k]**self.alpha)*np.sum([self.omega(mu, 0)*self.u[0, k-
        mu] for mu in range(k+1)])
    sum2 = (1/self.t[k]**self.alpha)*np.sum([self.omega(mu, self.M)*self.u
        [-1, k-mu] for mu in range(k+1)])
    Dtalpha1 = ((1-self.alpha)/gamma(2-self.alpha))*self.t[k]*self.u0(self
        .x[0])
    Dtalpha2 = ((1-self.alpha)/gamma(2-self.alpha))*self.t[k]*self.u0(self
        .x[-1])
    if 1 < self.alpha <= 2 or self.alpha == 0:
        Dtalpha1 += (1./gamma(2-self.alpha))*self.t[k]**(1-self.alpha) *
            self.u1(self.x[0])
        Dtalpha2 += (1./gamma(2-self.alpha))*self.t[k]**(1-self.alpha) *
            self.u1(self.x[-1])
    r1coeff = r1term + h1term*(self.f(a, self.t[k]) - sum1 + Dtalpha1)
    r2coeff = r2term + h2term*(self.f(b, self.t[k]) - sum2 + Dtalpha2)
    return r1coeff, r2coeff

def numerical(self):
    from scipy.linalg import solve
    self.get_params()

```

```

print("Numerical simulation size: %.2d x %.2d points" %(self.x.shape
    [0],self.t.shape[0]))
print("Evaluating numerical solution...")
# Generate central difference matrix
Cdiff = np.zeros((self.N,self.N))
main_diag = np.diag_indices(self.N)
lower_diag = (main_diag[0][1:],(main_diag[1]-1)[1:])
upper_diag = (main_diag[0][: -1],(main_diag[1]+1)[: -1])
Cdiff[main_diag] = -2.
Cdiff[lower_diag] = 1.
Cdiff[upper_diag] = 1.
Cdiff *= 1./self.dx**2

for k in range(int(np.ceil(self.alpha)),self.M):
    # Calculate and set d,o,r-coefficients for the central difference
    matrix
    dlcoeff,d2coeff = self.dcoeff(k)
    olcoeff,o2coeff = self.ocoeff(k)
    rlcoeff,r2coeff = self.rcoeff(k)
    Cdiff[0,0] = dlcoeff/self.dx**2; Cdiff[-1,-1] = d2coeff/self.dx**2
    Cdiff[0,1] = olcoeff/self.dx**2; Cdiff[-1,-2] = o2coeff/self.dx**2

    # Calculate the terms for the RHS vector
    RHS1 = self.f(self.x,self.t[k])
    RHS1[0] = rlcoeff; RHS1[-1] = r2coeff
    RHS2 = self.u0(self.x)*(1-self.alpha) / gamma(2-self.alpha) / self
        .t[k]**self.alpha
    RHS2[0] = 0.; RHS2[-1] = 0.
    RHS3 = np.array([0.]+[np.sum([self.omega(mu,k)*self.u[j,k-mu] for
        mu in range(1,k+1)]) for j in range(1,self.N-1)]+[0.])
    RHS3 *= 1./self.dt**self.alpha
    RHS = RHS1 + RHS2 - RHS3

    # Calculate the terms for the LHS matrix
    LHS = np.multiply(self.phi(self.x,self.t[k]),Cdiff)
    LHS[main_diag] += self.omega(0,k) / self.dt**self.alpha

    # Solve for the numerical solution
    self.u[:,k] = solve(LHS,RHS)
print("DONE!")
return self.u

def plot(self):
    import matplotlib.pyplot as plt
    from mpl_toolkits.mplot3d import Axes3D
    fig = plt.figure(figsize=(10,7))
    ax = fig.add_subplot(111,projection="3d")
    X,T = np.meshgrid(self.x,self.t)
    ax.plot_wireframe(X,T,self.u.T,alpha=0.5)
    ax.set_title(r"Fractional Diffusion-Wave Equation ( $\alpha$  = %.1f)" %
        self.alpha)
    ax.set_xlabel("x")
    ax.set_ylabel("t")
    ax.set_zlabel("u(x,t)")

```

```

plt.show()

def get_params(self):
    print("-----")
    print("TIME-FRACTIONAL DIFFUSION-WAVE EQUATION")
    print("-----")
    print("D_t^alpha y(x,t) = phi(x,t) (d^2/dx^2) y(x,t)+f(x,t) ")
    print("-----")
    print("PARAMS:")
    print("alpha = %.2f" %self.alpha)
    print("x range = [%.2f,%.2f]" %(min(self.x),max(self.x)))
    print("t range = [%.2f,%.2f]" %(min(self.t),max(self.t)))
    print("dx = %.2f, dt = %.2f" %(self.dx, self.dt))

def output(self):
    return self.x, self.t, self.u

if __name__ == "__main__":
    params = {
        "alpha" : 1.5,
        "tmin" : 0.,
        "tmax" : 10.,
        "dt" : 0.1,
        "xmin" : 0.01,
        "xmax" : np.pi,
        "dx" : 0.01,
        "ICs" : [lambda x : np.sin(x),lambda x : 0.], # Define ICs with list
                of lambda functions
        "g" : [lambda t : 1., lambda t : 1.], # Define g,h,r with
            list of lambda functions
        "h" : [lambda t : t*0., lambda t : t*0.],
        "r" : [lambda t : t*0., lambda t : t*0.],
        "f" : lambda x,t : 0.*x*t, # Define f, phi with
            lambda functions
        "phi" : lambda x,t : -1.
    }
    DW = Diffusion_Wave(params)
    DW.numerical()
    DW.plot()

```

B.6 Four-Term Fractional Wave Equation

```
import numpy as np
from scipy.special import gamma, binom
import matplotlib.pyplot as plt

class FTWE:
    def __init__(self, params):
        self.alpha = params["alpha"]
        xmin = params["xmin"]; xmax = params["xmax"]
        tmin = params["tmin"]; tmax = params["tmax"]
        dx = params["dx"]; dt = params["dt"]
        x = np.arange(xmin, xmax, dx); N = len(x)
        t = np.arange(tmin, tmax, dt); M = len(t)

        self.u = np.zeros((N, M))
        self.nsrc = int(params["nsrc"] * N)
        self.src = params["src"]
        self.u[self.nsrc, 0] = self.src(t[0])
        self.u[self.nsrc, 1] = self.src(t[1])

        self.x = x; self.t = t
        self.N = N; self.M = M
        self.dx = dx; self.dt = dt
        self.A0, self.A1, self.A2 = params["coeffs"]
        self.mu0 = params["mu0"]
        self.mur = params["mur"]
        self.eps0 = params["eps0"]

        self.wj1 = np.array([(-1)**j * binom(self.alpha, j) for j in range(M)])
        self.wj2 = np.array([(-1)**j * binom(self.alpha+2, j) for j in range(M)])

    def numerical(self):
        from scipy.sparse import csr_matrix
        self.get_params()
        print("Evaluating numerical solution...")
        Cdiff = np.zeros((self.N, self.N))
        main_diag = np.diag_indices(self.N)
        lower_diag = (main_diag[0][1:], (main_diag[1]-1)[1:])
        upper_diag = (main_diag[0][: -1], (main_diag[1]+1)[: -1])
        Cdiff[upper_diag] = 1.
        Cdiff[main_diag] = -2.
        Cdiff[lower_diag] = 1.
        Cdiff = csr_matrix(Cdiff)
        for k in range(1, self.M-1):
            term1 = Cdiff.dot(self.u[:, k])
            term1 *= self.A1 * (self.dt**2 / self.dx**2) / (self.mu0*self.mur*
                self.eps0)

            term2 = Cdiff.dot(self.u[:, k-1:: -1])
            term2 = term2.dot(self.wj1[:k])
            term2 *= self.A2 * (self.dt** -self.alpha / self.dx**2) / (self.mu0
                *self.mur*self.eps0)
```

```

        term3 = np.matmul(self.wj2[:,k],self.u[:,k-1::-1].T)
        term3 *= self.A0 / self.dt**self.alpha

        self.u[self.nsrc,k] = self.src(self.t[k])
        self.u[:,k+1] = term1 + term2 - term3 + 2*self.u[:,k] - self.u[:,k-1]
    print("DONE!")
    return self.u

def plot(self):
    import matplotlib.pyplot as plt
    plt.figure(1,figsize=(10,7))
    plt.xlabel("x",fontsize=16)
    plt.ylabel(r"$E_y$",fontsize=16)
    plt.title("Four-Term 1D Fractional Wave Equation",fontsize=20)
    plt.plot(self.x,self.u[:,0],label="t=%.3e" %self.t[0])
    plt.plot(self.x,self.u[:,int(self.N/4)],label="t=%.3e" %self.t[int(
        self.N/4)])
    plt.plot(self.x,self.u[:,int(self.N/2)],label="t=%.3e" %self.t[int(
        self.N/2)])
    plt.plot(self.x,self.u[:,int(3*self.N/4)],label="t=%.3e" %self.t[int(
        3*self.N/4)])
    plt.plot(self.x,self.u[:,-1],label="t=%.3e" %self.t[-1])
    plt.legend(loc=1,prop={"size":14})
    plt.show()

def animate(self):
    from matplotlib import rcParams
    import matplotlib.animation as animation
    print("Generating animation...")
    rcParams['animation.convert_path'] = r'/usr/bin/convert'
    fig = plt.figure(2,figsize=(10,7))
    ax = plt.axis([np.min(self.x),np.max(self.x),np.min(self.u),np.max(
        self.u)])
    curve, = plt.plot(self.x,self.u[:,0])
    def animate(time):
        curve.set_data(self.x,self.u[:,time])
        plt.title(r"Four-Term 1D Fractional Wave Equation, $\alpha$=%.2f$ (
            timestep=%.1d)" %(self.alpha,time),fontsize=18)
        return curve,
    myAnimation = animation.FuncAnimation(fig,animate,frames=np.arange(0,
        len(self.t),int(len(self.t)/200)), \
        interval=10, blit=True, repeat=
        False)
    myAnimation.save("fractional_wave-"+str(self.alpha)+".gif", writer="
       imagemagick",fps=30)
    print("DONE!")

def get_params(self):
    print("-----")
    print("FOUR-TERM FRACTIONAL WAVE EQUATION")
    print("-----")
    print("PARAMS:")

```

```

        print("alpha    = %.2f" %self.alpha)
        print("x range = [%.2f,%.2f]" %(min(self.x),max(self.x)))
        print("t range = [%.2f,%.2f]" %(min(self.t),max(self.t)))
        print("dx = %.2f, dt = %.2f" %(self.dx, self.dt))
        print("A0 = %.3e, A1 = %.3e, A2 = %.3e" %(self.A0,self.A1,self.A2))
        print("mu0 = %.3e, mur = %.3e, eps0 = %.3e" %(self.mu0,self.mur,self.
            eps0))

    def output(self):
        return self.x,self.t,self.u

if __name__ == "__main__":
    freq = 1.          # Frequency of the sine wave
    tau = 5/freq       # Duration of initial ramp function (5 periods)
    params = {
        "alpha" : 0.5,
        "tmin"  : 0.,
        "tmax"  : 20.,
        "dt"    : 0.01,
        "xmin"  : 0.,
        "xmax"  : 20.,
        "dx"    : 0.01,
        "coeffs": [10e-4,0.1,10e-8],
        "mu0"   : 1.,
        "mur"   : 1.,
        "eps0"  : 1.,
        "nsrc"  : .5,
        "src"   : lambda t : 0.5*(1+np.sin(np.pi/2/tau*(2*t-tau)))*np.sin(2*np
            .pi*freq*t) if t < tau else np.sin(2*np.pi*t)
    }
    FT = FTWE(params)
    FT.numerical()
    FT.plot()
    FT.animate()

```

Attachment 1 - Data Files for Fractional Lumped Capacitance Code

Table 1-1. Laser heating/cooling experimental data

Time (s)	Temp (°C)	Time (s)	Temp (°C)	Time (s)	Temp (°C)
0	29.87911	0.02778	29.88132	0.05555	55.56258
0.00069	29.88135	0.02847	29.87688	0.05625	55.27757
0.00139	29.87018	0.02916	29.88134	0.05694	55.00961
0.00208	29.86569	0.02986	29.86794	0.05764	54.73193
0.00278	29.8992	0.03055	29.86123	0.05833	54.5064
0.00347	29.83445	0.03125	29.84334	0.05902	54.24924
0.00417	29.91034	0.03194	29.85669	0.05972	54.00803
0.00486	29.86794	0.03264	29.83667	0.06041	53.79441
0.00556	29.86794	0.03333	29.86793	0.06111	53.5739
0.00625	29.89692	0.03403	29.87228	0.0618	53.36676
0.00694	29.85901	0.03472	29.86794	0.0625	53.16981
0.00764	29.87234	0.03541	29.84779	0.06319	52.96629
0.00833	29.87465	0.03611	29.88799	0.06388	52.78691
0.00903	29.85675	0.0368	29.82994	0.06458	52.60622
0.00972	29.87242	0.0375	29.86345	0.06527	52.44293
0.01042	29.86125	0.03819	29.86349	0.06597	52.27832
0.01111	29.85902	0.03889	29.90142	0.06666	52.11151
0.0118	29.86796	0.03958	34.4598	0.06736	51.95207
0.0125	29.87687	0.04028	36.00667	0.06805	51.80588
0.01319	29.88579	0.04097	37.77582	0.06875	51.65825
0.01389	29.84562	0.04166	39.9261	0.06944	51.93619
0.01458	29.87688	0.04236	42.07068	0.07013	51.76829
0.01528	29.88582	0.04305	44.15475	0.07083	51.24504
0.01597	29.86574	0.04375	46.26234	0.07152	51.10888
0.01667	29.8568	0.04444	48.21935	0.07222	50.96265
0.01736	29.86793	0.04514	50.20858	0.07291	51.22715
0.01805	29.87235	0.04583	52.90122	0.07361	50.70775
0.01875	29.84563	0.04652	54.10017	0.0743	50.57562
0.01944	29.86125	0.04722	56.70048	0.075	50.85312
0.02014	29.85229	0.04791	58.52755	0.07569	50.73627
0.02083	29.83887	0.04861	60.37032	0.07638	50.22646
0.02153	29.82549	0.0493	59.48719	0.07708	50.49547
0.02222	29.85899	0.05	58.23593	0.07777	50.38563
0.02292	29.86795	0.05069	57.82335	0.07847	50.28003
0.02361	29.87012	0.05139	57.4548	0.07916	49.76892
0.0243	29.85455	0.05208	57.10311	0.07986	50.04644
0.025	29.87465	0.05277	56.77205	0.08055	49.95673
0.02569	29.87913	0.05347	56.45823	0.08124	49.86964
0.02639	29.83445	0.05416	56.15033	0.08194	49.77506
0.02708	29.91259	0.05486	55.84005	0.08263	49.67042

Continued on next page

Attachment 1 - Data Files for Fractional Lumped Capacitance Code

Table 1-1 – Continued from previous page

Time (s)	Temp (°C)	Time (s)	Temp (°C)	Time (s)	Temp (°C)
0.08333	49.59237	0.11249	46.70263	0.14582	44.82157
0.08402	49.47563	0.11319	46.64986	0.14652	44.77872
0.08472	49.39325	0.11388	46.60018	0.14721	44.7643
0.08541	49.30652	0.11458	46.5775	0.14791	44.72876
0.08611	49.22679	0.11527	46.24434	0.1486	44.71204
0.0868	49.13079	0.11596	46.47463	0.1493	44.68658
0.08749	49.04964	0.11666	46.44303	0.14999	44.6641
0.08819	48.95823	0.11735	46.37212	0.15069	44.6243
0.08888	48.89956	0.11805	46.09216	0.15138	44.60371
0.08958	48.77525	0.11874	46.02227	0.15207	44.5457
0.09027	48.72093	0.11944	46.24745	0.15277	44.54437
0.09097	48.658	0.12013	46.19129	0.15346	44.51604
0.09166	48.56407	0.12083	46.14393	0.15416	44.46659
0.09236	48.14584	0.12152	45.8556	0.15485	44.23645
0.09305	48.42068	0.12221	46.06136	0.15555	44.42777
0.09374	48.33471	0.12291	46.01452	0.15624	44.41012
0.09444	48.27144	0.1236	45.70702	0.15693	44.37387
0.09513	48.21092	0.1243	45.93781	0.15763	44.32283
0.09583	48.12304	0.12499	45.91468	0.15832	44.3124
0.09652	48.05035	0.12569	45.85607	0.15902	44.27417
0.09722	47.97596	0.12638	45.82094	0.15971	44.24035
0.09791	47.92676	0.12708	45.48799	0.16041	44.23253
0.0986	47.84879	0.12777	45.73885	0.1611	44.2115
0.0993	47.45978	0.12846	45.70247	0.1618	44.17331
0.09999	47.72849	0.12916	45.65474	0.16249	44.15971
0.10069	47.66489	0.12985	45.62424	0.16318	44.12339
0.10138	47.28976	0.13055	45.30668	0.16388	44.10009
0.10208	47.52549	0.13124	45.52729	0.16457	44.06663
0.10277	47.18343	0.13194	45.50333	0.16527	44.07738
0.10347	47.1184	0.13263	45.46434	0.16596	44.02614
0.10416	47.37091	0.13333	45.42473	0.16666	44.00354
0.10485	47.31119	0.13402	45.40076	0.16735	43.97161
0.10555	46.94992	0.13471	45.08981	0.16805	43.9439
0.10624	47.19632	0.13957	45.12687	0.16874	43.92628
0.10694	47.13944	0.14027	45.08111	0.16943	43.91584
0.10763	47.07699	0.14096	45.05913	0.17013	43.88327
0.10833	47.03756	0.14166	45.0091	0.17082	43.85365
0.10902	46.97411	0.14235	44.97222	0.17152	43.65189
0.10972	46.90713	0.14305	44.96108	0.17221	43.81452
0.11041	46.86906	0.14374	44.90614	0.17291	43.80931
0.1111	46.8099	0.14444	44.8925	0.1736	43.76917
0.1118	46.75834	0.14513	44.85019	0.17429	43.74649

Attachment 1 - Data Files for Fractional Lumped Capacitance Code

Table 1-2. Pennes bioheat model fitting results

Time (s)	Temp (°C)	Time (s)	Temp (°C)	Time (s)	Temp (°C)
0.00E+00	2.99E+01	7.00E-03	5.96E+01	3.11E-02	5.88E+01
1.25E-04	3.05E+01	7.15E-03	6.01E+01	3.26E-02	5.83E+01
2.46E-04	3.11E+01	7.45E-03	6.13E+01	3.52E-02	5.75E+01
4.12E-04	3.18E+01	7.63E-03	6.20E+01	3.65E-02	5.71E+01
5.90E-04	3.27E+01	7.78E-03	6.25E+01	3.92E-02	5.63E+01
8.25E-04	3.38E+01	7.95E-03	6.31E+01	4.07E-02	5.59E+01
1.00E-03	3.46E+01	8.00E-03	6.33E+01	4.38E-02	5.52E+01
1.28E-03	3.59E+01	8.22E-03	6.42E+01	4.55E-02	5.48E+01
1.44E-03	3.66E+01	8.41E-03	6.49E+01	4.87E-02	5.42E+01
1.61E-03	3.74E+01	8.71E-03	6.58E+01	5.07E-02	5.38E+01
1.83E-03	3.81E+01	8.87E-03	6.62E+01	5.37E-02	5.33E+01
1.98E-03	3.85E+01	9.00E-03	6.66E+01	5.63E-02	5.28E+01
2.00E-03	3.86E+01	9.31E-03	6.75E+01	5.94E-02	5.24E+01
2.28E-03	3.97E+01	9.50E-03	6.82E+01	6.25E-02	5.20E+01
2.47E-03	4.07E+01	9.66E-03	6.87E+01	6.63E-02	5.15E+01
2.64E-03	4.15E+01	9.93E-03	6.94E+01	7.02E-02	5.10E+01
2.77E-03	4.20E+01	1.00E-02	6.96E+01	7.45E-02	5.06E+01
2.91E-03	4.25E+01	1.01E-02	6.95E+01	7.87E-02	5.02E+01
3.00E-03	4.30E+01	1.03E-02	6.94E+01	8.36E-02	4.98E+01
3.17E-03	4.39E+01	1.05E-02	6.93E+01	8.82E-02	4.94E+01
3.37E-03	4.50E+01	1.07E-02	6.92E+01	9.39E-02	4.90E+01
3.62E-03	4.60E+01	1.10E-02	6.91E+01	9.89E-02	4.87E+01
3.80E-03	4.67E+01	1.13E-02	6.89E+01	1.06E-01	4.83E+01
4.00E-03	4.76E+01	1.17E-02	6.86E+01	1.11E-01	4.80E+01
4.30E-03	4.88E+01	1.22E-02	6.84E+01	1.18E-01	4.77E+01
4.51E-03	4.96E+01	1.27E-02	6.81E+01	1.25E-01	4.74E+01
4.68E-03	5.03E+01	1.33E-02	6.77E+01	1.32E-01	4.71E+01
4.95E-03	5.17E+01	1.40E-02	6.73E+01	1.39E-01	4.69E+01
5.00E-03	5.19E+01	1.48E-02	6.68E+01	1.48E-01	4.66E+01
5.17E-03	5.28E+01	1.57E-02	6.62E+01	1.56E-01	4.64E+01
5.48E-03	5.40E+01	1.69E-02	6.55E+01	1.63E-01	4.62E+01
5.64E-03	5.46E+01	1.81E-02	6.48E+01	1.70E-01	4.60E+01
5.95E-03	5.56E+01	1.96E-02	6.40E+01	1.78E-01	4.58E+01
6.00E-03	5.59E+01	2.11E-02	6.31E+01	1.86E-01	4.56E+01
6.16E-03	5.65E+01	2.27E-02	6.23E+01	1.94E-01	4.55E+01
6.32E-03	5.71E+01	2.43E-02	6.16E+01	2.01E-01	4.53E+01
6.55E-03	5.79E+01	2.59E-02	6.08E+01	2.10E-01	4.52E+01
6.68E-03	5.84E+01	2.76E-02	6.01E+01	2.19E-01	4.50E+01
6.93E-03	5.93E+01	2.93E-02	5.95E+01	2.28E-01	4.49E+01

Continued on next page

Attachment 1 - Data Files for Fractional Lumped Capacitance Code

Table 1-2 – Continued from previous page

Time (s)	Temp (°C)	Time (s)	Temp (°C)	Time (s)	Temp (°C)
2.35E-01	4.48E+01	6.01E-01	4.22E+01	1.05E+00	4.10E+01
2.44E-01	4.47E+01	6.11E-01	4.22E+01	1.06E+00	4.09E+01
2.54E-01	4.45E+01	6.20E-01	4.21E+01	1.08E+00	4.09E+01
2.63E-01	4.44E+01	6.30E-01	4.21E+01	1.09E+00	4.09E+01
2.73E-01	4.43E+01	6.40E-01	4.20E+01	1.10E+00	4.09E+01
2.81E-01	4.42E+01	6.50E-01	4.20E+01	1.11E+00	4.09E+01
2.89E-01	4.41E+01	6.61E-01	4.20E+01	1.13E+00	4.08E+01
2.97E-01	4.40E+01	6.71E-01	4.19E+01	1.14E+00	4.08E+01
3.05E-01	4.40E+01	6.81E-01	4.19E+01	1.15E+00	4.08E+01
3.14E-01	4.39E+01	6.92E-01	4.19E+01	1.16E+00	4.08E+01
3.22E-01	4.38E+01	7.02E-01	4.18E+01	1.17E+00	4.07E+01
3.30E-01	4.37E+01	7.12E-01	4.18E+01	1.19E+00	4.07E+01
3.39E-01	4.37E+01	7.23E-01	4.18E+01	1.20E+00	4.07E+01
3.48E-01	4.36E+01	7.33E-01	4.17E+01	1.21E+00	4.07E+01
3.57E-01	4.35E+01	7.44E-01	4.17E+01	1.23E+00	4.06E+01
3.66E-01	4.34E+01	7.55E-01	4.17E+01	1.24E+00	4.06E+01
3.75E-01	4.34E+01	7.65E-01	4.16E+01	1.25E+00	4.06E+01
3.84E-01	4.33E+01	7.76E-01	4.16E+01	1.27E+00	4.06E+01
3.94E-01	4.32E+01	7.78E-01	4.16E+01	1.28E+00	4.06E+01
3.94E-01	4.32E+01	7.89E-01	4.16E+01	1.30E+00	4.05E+01
4.04E-01	4.32E+01	7.99E-01	4.16E+01	1.31E+00	4.05E+01
4.13E-01	4.31E+01	8.10E-01	4.15E+01	1.32E+00	4.05E+01
4.21E-01	4.31E+01	8.21E-01	4.15E+01	1.34E+00	4.05E+01
4.30E-01	4.30E+01	8.32E-01	4.15E+01	1.35E+00	4.05E+01
4.39E-01	4.30E+01	8.43E-01	4.14E+01	1.36E+00	4.04E+01
4.48E-01	4.29E+01	8.54E-01	4.14E+01	1.37E+00	4.04E+01
4.57E-01	4.29E+01	8.65E-01	4.14E+01	1.38E+00	4.04E+01
4.66E-01	4.28E+01	8.76E-01	4.14E+01	1.39E+00	4.04E+01
4.75E-01	4.28E+01	8.87E-01	4.13E+01	1.41E+00	4.04E+01
4.85E-01	4.27E+01	8.99E-01	4.13E+01	1.42E+00	4.04E+01
4.95E-01	4.27E+01	9.10E-01	4.13E+01	1.43E+00	4.03E+01
5.04E-01	4.26E+01	9.21E-01	4.12E+01	1.44E+00	4.03E+01
5.12E-01	4.26E+01	9.33E-01	4.12E+01	1.46E+00	4.03E+01
5.21E-01	4.25E+01	9.44E-01	4.12E+01	1.47E+00	4.03E+01
5.29E-01	4.25E+01	9.56E-01	4.12E+01	1.48E+00	4.03E+01
5.38E-01	4.25E+01	9.67E-01	4.11E+01	1.49E+00	4.02E+01
5.46E-01	4.24E+01	9.79E-01	4.11E+01	1.51E+00	4.02E+01
5.55E-01	4.24E+01	9.91E-01	4.11E+01	1.52E+00	4.02E+01
5.64E-01	4.23E+01	1.00E+00	4.11E+01	1.53E+00	4.02E+01
5.73E-01	4.23E+01	1.01E+00	4.10E+01	1.55E+00	4.02E+01
5.82E-01	4.23E+01	1.03E+00	4.10E+01	1.56E+00	4.02E+01
5.92E-01	4.22E+01	1.04E+00	4.10E+01	1.57E+00	4.01E+01

LIST OF SYMBOLS, ABBREVIATIONS, AND ACRONYMS

α	The order of a fractional derivative or fractional integral
Γ	Gamma function defined in (A-1)
$\tau, \tau_1, \dots$	Dummy variable for convolution
$\omega_{j,k}^\alpha$	Diethelm weights for evaluating fractional derivatives of order α for $0 < \alpha < 1$
ω_j^α	Grünwald-Letnikov weights for evaluating fractional derivatives or fractional integrals of order α
B_N^α	The backward difference matrix for an α -order derivative.
F_N	The matrix method notation for the source function vector defined for all N points of the numerical solution.
$G_2(t)$	The fractional Green's function for the two-term fractional-order differential equation with constant coefficients.
$G_3(t)$	The fractional Green's function for the three-term fractional-order differential equation with constant coefficients.
N	The total number of points where the numerical solution to the fractional differential equation will be solved. Determined by taking the difference between the upper and lower terminals of the fractional operator and dividing by the step size, h .
$P_N^{(l)}$	The matrix method notation for the diagonalized $p_l(t)$ function.
$S_{0,1,\dots,n-1}$	The matrix method notation for the eliminator matrix where the subscript list denotes which rows of an identity matrix are removed for the resulting eliminator matrix.
Y_N	The matrix method notation for the solution vector for the N points of the numerical solution.
${}_a^C D_t^\alpha$	Caputo fractional operator between a and t of order α
${}_a^{RL} D_t^\alpha$	Riemann-Liouville fractional operator between a and t of order α
${}_a D_t^\alpha$	General fractional operator between a and t of order α
h	change in variable t in difference quotient

k	Denotes the current time step or space step of interest, the desired point in which to evaluate the solution to a differential equation.
m	The number of terms for a linear fractional differential equation.
$p_l(t)$	The matrix method notation for the coefficient function for the l -th term of the m -term linear fractional differential equation
w_j^α	Starting weights applied to the GL weights specifically for the diffusion-wave problem.
BFD	backward finite difference
BT	Bagley-Torvik
FDE	fractional differential equation
FODE	fractional ordinary differential equation
FPDE	fractional partial differential equation
GL	Grünwald-Letnikov
ODE	ordinary differential equation
PDE	partial differential equation
RL	Riemann-Liouville