



POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

High-Performance Computing for Fractional Rheology

TESI DI LAUREA MAGISTRALE IN
HIGH PERFORMANCE COMPUTING ENGINEERING - INGEGNE-
RIA DEL CALCOLO AD ELEVATE PRESTAZIONI

Author: **Lorenzo Fonnesu, Andrea Grassi**

Student ID: 249663, 252516

Advisor: Dr. Alessandra Bonfanti

Co-advisors: Prof. Alexandre Kabla, Dr. Jonathan Louis Kaplan

Academic Year: 2025-26

Abstract

Soft materials such as polymers, hydrogels, and biological tissues are often subjected to mechanical forces, and understanding how they respond is essential for engineering applications — a question studied by rheology. These materials typically dissipate forces gradually, often following a power-law behaviour in time. Fractional calculus, which generalizes differentiation and integration to non-integer order, provides a mathematical tool for capturing this behaviour with a minimal set of parameters. However, the high computational cost of fractional models limits their adoption in practical engineering workflows.

In this work, we address this bottleneck by reformulating fractional viscoelastic models in a differential, time-domain framework, replacing the costly convolution-based evaluation with a discretized approach better suited to high-performance computing. This removes the main computational obstacles and enables more demanding simulation and fitting tasks.

To achieve this, we developed `NumFracDiff`, an open-source Julia library for the evaluation of fractional derivatives via Finite Difference discretization, further accelerated through code parallelization, the Short Memory Principle, and the Fast Fourier Transform. Numerical validation confirms that all schemes are accurate, with multi-threading and FFT achieving speedups of $11 - 15\times$.

The differential approach was then applied to two practical tasks. For parameter identification, replacing the convolution-based cost function with the differential or FFT-based formulation yields over $20\times$ speedup in fitting fractional models. For stress and strain prediction under arbitrary loading histories, the two methods were validated against analytical solutions; FFT achieves speedups of up to $100\times$ over convolution, while the parallel finite-difference solver provides intermediate but consistent gains. Both extensions were integrated into the open-source rheology framework RHEOS.

Keywords: Viscoelasticity, Fractional Calculus, High-Performance Computing, Finite Differences, Fast Fourier Transform

Abstract in lingua italiana

I materiali morbidi, quali polimeri, idrogel e tessuti biologici, sono spesso sottoposti a forze meccaniche, e comprendere come reagiscono è fondamentale per le applicazioni ingegneristiche - un aspetto studiato dalla reologia. Questi materiali in genere dissipano le forze in modo graduale, seguendo spesso un andamento di legge di potenza nel tempo. Il calcolo frazionario, che generalizza la derivazione e l'integrazione a ordini non interi, fornisce uno strumento matematico per descrivere questo comportamento con un insieme minimo di parametri. Tuttavia, l'alto costo computazionale dei modelli frazionari ne limita l'adozione nei flussi di lavoro ingegneristici pratici.

In questo lavoro affrontiamo tale collo di bottiglia riformulando i modelli viscoelastici frazionari in un quadro differenziale nel dominio del tempo, sostituendo il costoso calcolo basato sulla convoluzione con un approccio discretizzato più adatto al calcolo ad alte prestazioni. Ciò elimina i principali ostacoli computazionali e consente di eseguire attività di simulazione e fitting più complesse.

A tal fine, abbiamo sviluppato `NumFracDiff`, una libreria open-source in Julia per il calcolo delle derivate frazionarie tramite discretizzazione a differenze finite, accelerata grazie alla parallelizzazione del codice, allo Short Memory Principle e alla Fast Fourier Transform. La validazione numerica conferma che tutti gli schemi sono accurati, con il multi-threading e la FFT che ottengono uno speedup di $11 - 15\times$.

L'approccio differenziale è stato quindi applicato a due compiti pratici. Per l'identificazione dei parametri, la sostituzione della funzione di costo basata sulla convoluzione con la formulazione differenziale o sulla FFT consente di ottenere uno speedup di oltre $20\times$ nel fitting dei modelli frazionari. Per la previsione di stress e strain in presenza di carichi arbitrari, i due metodi sono stati convalidati rispetto alle soluzioni analitiche; l'FFT raggiunge velocità fino a $100\times$ superiori rispetto alla convoluzione, mentre il risolutore a differenze finite parallelo fornisce guadagni intermedi ma costanti. Entrambe le estensioni sono state integrate nel framework reologico open-source RHEOS.

Parole chiave: Viscoelasticità, Calcolo Frazionale, Calcolo ad Elevate Prestazioni, Differenze Finite, Trasformata Veloce di Fourier

Contents

Abstract	i
Abstract in lingua italiana	iii
Contents	v
1 Introduction	1
1.1 Viscoelasticity	2
1.2 Fractional Viscoelastic Models	3
1.3 Classical modelling using the Boltzmann integral formulation	4
1.4 The computational challenge	5
1.5 The need for efficient computation and the novel differential approach proposed	5
1.6 Aim and objectives	6
1.7 Outline of the thesis	7
2 Theoretical Background and Methods	9
2.1 Fundamentals of Linear Viscoelasticity	9
2.1.1 Relaxation and Creep Moduli	10
2.1.2 Boltzmann Superposition Principle	12
2.2 Classical viscoelastic models	13
2.2.1 The two simplest viscoelastic models	14
2.3 Fractional viscoelastic models	19
2.3.1 Generalized fractional models	22
2.4 Computational Challenges	28
2.4.1 The Non-Locality Problem	28
2.4.2 The Mittag-Leffler Function Bottleneck	29
2.5 State of the Art and Proposed Approach	30
2.5.1 Available computational frameworks	30

2.5.2	The Differential Approach	31
2.6	Computational Environment and Implementation	31
2.6.1	The Julia Programming Language	31
2.6.2	Cluster	32
3	Fractional Derivatives Implementation	33
3.1	Introduction	33
3.2	Numerical Implementation via Finite Differences	36
3.2.1	Grünwald–Letnikov (GL) Discretization	37
3.2.2	Caputo Discretization	37
3.2.3	Riemann–Liouville Discretization	39
3.2.4	Parallelization	40
3.3	Short Memory Principle	41
3.3.1	Correction term	41
3.4	FFT-based acceleration	43
3.5	Results and Discussion	45
3.5.1	Numerical Validation and Error Analysis	46
3.5.2	Performance Analysis	48
3.5.3	Scalability Analysis	50
3.6	NumFracDiff Library	51
3.6.1	Architecture	53
3.7	Conclusions	54
4	Fitting	57
4.1	Introduction	57
4.2	Cost Function	59
4.3	Optimization	61
4.4	Implementation in RHEOS	62
4.5	Results and Discussion	63
4.5.1	Cost functions	64
4.5.2	Fitting	66
4.5.3	Carbon footprint	67
4.6	Conclusions	68
5	Prediction	71
5.1	Introduction	71
5.2	Finite Difference prediction method	72
5.3	FFT-based prediction method	75

5.3.1	Motivations and Computational Efficiency	76
5.3.2	Stress Prediction	76
5.3.3	Strain Prediction	77
5.4	Implementation in RHEOS	78
5.5	Results and Discussion	79
5.5.1	Accuracy Validation	79
5.5.2	Computational Performance	81
5.6	Conclusions	86
6	Conclusions and future developments	89
6.1	Summary of Contributions	89
6.2	Future Developments	90
	 Bibliography	 93
	 A Appendix A	 97
	B Appendix B	99
	C Appendix C	101
C.1	Maxwell Model	101
C.2	Springpot Model	102
	 List of Figures	 105
	 List of Tables	 109

1 | Introduction

Characterizing how materials respond to applied forces is critical for a wide range of engineering, industrial, and biomedical applications, since the accuracy with which this response is predicted directly impacts product safety, performance, and design efficiency. Across various fields, researchers and engineers routinely encounter complex materials, such as polymers, hydrogels, biological tissues, food products, and asphalt, whose mechanical behavior defies simple classification [1–5]. For instance, predicting how a hydrogel deforms under physiological loads is essential for designing tissue scaffolds and drug-delivery systems, understanding the long-term deformation of asphalt pavements is useful for planning infrastructure maintenance, and characterizing the texture of food products aids quality control in industrial production. *Rheology* is the study of the deformation and flow of materials, allowing us to understand how they react to mechanical stimuli [6].

To comprehend these behaviours, the primary goal of rheology is to construct mathematical constitutive laws that can describe them by accurately relating the applied force F to the observed elongation ΔL (or vice versa) for a given material [7]. Experimentally, the response of a material is measured directly as a force F and the resulting elongation ΔL . However, these quantities depend on the specimen's geometry: a thicker or longer sample would require a different force to produce the same elongation, even though it is made of the same material. To describe the material itself, independently of the shape of the tested specimen, F and ΔL are normalized by the specimen's cross-sectional area A , and original length L_0 , respectively, yielding the stress (σ) and, strain or deformation (ϵ)

$$\sigma = \frac{F}{A} \quad \epsilon = \frac{\Delta L}{L_0}. \quad (1.1)$$

The equations relating stress and strain serve as the foundational link between macroscopic experimental observations and the underlying material physics, transforming continuous physical deformations into predictable mathematical frameworks suitable for numerical simulation and parameter identification. Once validated, such models allow engineers and researchers to simulate material response under untested loading conditions,

reducing the need for extensive and costly experimental campaigns and providing a quantitative basis for material selection, structural design, and diagnostic assessment.

1.1. Viscoelasticity

The mechanical response of materials to applied deformations and forces is, in general, a remarkably complex phenomenon. A material's behaviour can depend on the magnitude of the applied deformation, the rate at which it is applied, the temperature, and even the entire history of past loading, with different materials exhibiting markedly different sensitivities to each of these factors. Here we focus specifically on time- and history-dependent material response. Within this scope, the mechanical response of materials is generally bounded by two different behaviours: purely elastic solids and purely viscous fluids. The former are ideal solids that store all the mechanical energy supplied during deformation and release it completely upon unloading, with no dependence on the rate at which the deformation is applied; on the other hand, the latter are ideal (Newtonian) fluids that dissipate all the input energy as heat, and their resistance depends solely on the instantaneous rate of deformation, with no memory of past states [1, 2].

Real-world materials, however, rarely conform to either idealization. The mechanical response of most engineering and biological materials depends not only on the current state of deformation but also on the entire history of loading. Such materials are called viscoelastic: they simultaneously store a portion of the energy supplied, as an elastic solid, and dissipate the remainder, as a viscous fluid [3, 8]. This duality implies that their response is inherently time-dependent, and this dependence leads to two direct consequences when different stimuli are applied [9]:

- if we apply a constant force (stress) to a viscoelastic material, its deformation will keep increasing in time even though the applied load remains unchanged. This phenomenon is called *creep*. Typical examples include the long-term sagging of concrete structural elements under sustained service loads or the progressive lengthening of tendons and ligaments held at constant stretch, which is the physiological basis of static stretching;
- vice versa, if we apply a constant deformation (strain) to a viscoelastic material, the force required to sustain that deformation decreases over time. This phenomenon is called *relaxation*. A representative example is the orthodontic elastic band stretched to a fixed position, which must be periodically re-tensioned since the force it transmits decays over time even though its length is held constant by the bracket.

It is worth noting that creep and relaxation are not two phases of the same loading cycle, but two independent experiments: the former probes the material response to a constant stress, the latter to a constant strain, and both are needed to fully characterize the time-dependent behaviour of a viscoelastic material.

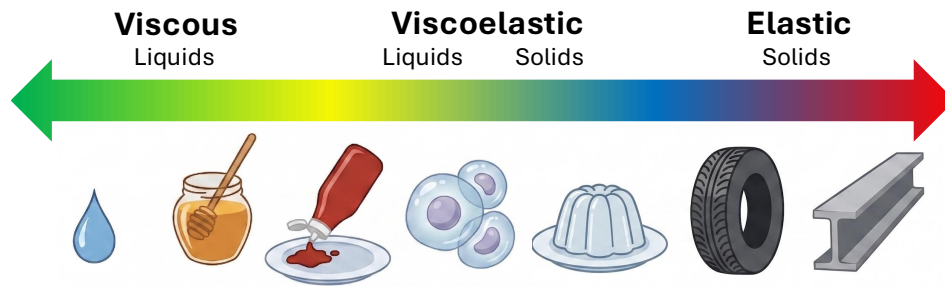


Figure 1.1: Qualitative spectrum of mechanical behaviour ranging from purely viscous liquids to purely elastic solids. Real materials populate the intermediate region, behaving as viscoelastic liquids (e.g., ketchup) or viscoelastic solids (e.g., biological cells, gelatin), depending on the relative balance of viscous and elastic contributions.

Viscoelastic behaviour is very common in both engineering and biology. In polymer science, which is the basis for some of the most important sectors, such as the textile and pharmaceutical industries, it governs energy dissipation and damping [10]. In biomechanics, soft tissues, such as muscles and arterial walls, or even cells exhibit pronounced viscoelastic characteristics [11, 12]. Similarly, food engineering leverages viscoelasticity to control the texture of different products, such as cheese and those requiring thickening agents [13, 14]. Finally, in civil engineering, the ease with which freshly mixed concrete can be mixed and laid is determined by its rheological properties, as it is a viscoelastic material [15].

1.2. Fractional Viscoelastic Models

The mathematical description of complex materials, such as polymers [1] and biological tissues [16], presents a long-standing challenge. Unlike ideal solids or fluids, these media

exhibit viscoelastic behavior, where the current state of stress mainly depends on the history of deformation.

This behavior has been modeled using networks of idealized simple components: *Springs* for elastic properties and *Dashpots* for viscous properties. Such components can be arranged in series or in parallel exactly like a digital circuit to try to reconstruct each material and its behaviour, and in doing so, they describe relaxation mechanisms governed by localized, discrete exponential decays. However, real-world complex materials consistently demonstrate multi-scale, hierarchical microstructures that do not adhere to single-scale relaxations; instead, they exhibit continuous power-law scaling laws over broad temporal windows. To resolve this continuous behavior using classical frameworks, one must resort to more complex, generalized configurations with an unfeasible or infinite number of parameters. This approach triggers severe over-parameterization, resulting in highly ill-conditioned optimization landscapes during material parameter identification.

To overcome these limitations, fractional calculus has been employed as a practical alternative. By replacing standard derivatives with non-local fractional-order operators where the order of integration is non-integer. Within this framework, the classical spring and dashpot can be generalized into a *springpot* [17, 18]. This single constitutive element naturally captures continuous power-law memory and multi-scale energy dissipation using a minimal set of parameters.

1.3. Classical modelling using the Boltzmann integral formulation

To systematically characterize this time-dependent behavior, researchers rely on fundamental material functions, specifically the relaxation modulus and the creep compliance, which quantify how a viscoelastic material stores and dissipates mechanical energy over time. While single-step tests such as ideal stress relaxation or creep compliance provide isolated insights into material properties, realistic engineering applications involve continuous, highly transient, and arbitrary loading histories. The Boltzmann Superposition Principle is what connects the ideal condition to the observed experiment results. This concept dictates that the total stress or strain response of a linear viscoelastic system is the cumulative sum of the effects of all preceding individual loading increments [8, 19, 20].

Mathematically, this history-dependent behavior is mapped via continuous convolution integrals over time. These mathematical formulations imply that if the characteristic material relaxation or compliance functions are known, the system's behavior can be fully

predicted under any arbitrary loading protocol. However, solving or evaluating these integrals for advanced, fractional-order models under realistic, noisy, or non-analytical inputs introduces severe mathematical and numerical barriers.

1.4. The computational challenge

As mentioned before, fractional models introduce two main complications in the viscoelastic models picture:

- **Non-Locality:** Classic integer-order derivatives need information only on direct neighborhood to be evaluated, but fractional derivatives are non-local operators; they are defined through a convolution integral across the entire history of the variable from the initial time. Computationally, this method presents linear scaling memory complexity and quadratic time complexity, which presents a huge bottleneck.
- **Mittag-Leffler Bottleneck:** The relaxation modulus and creep compliance for fractional models are expressed as a power series function, called Mittag-Leffler function [21–25]. While it captures the behaviour of complex materials, this function also introduces a severe bottleneck in the code, as it has slow convergence, it is numerically unstable and it is very computationally expensive.

The proposed fractional differential approach aims to circumvent these core limitations by opting for a complete discretization of the governing equations directly in the time domain. While the non-local nature of the fractional operator cannot be ignored, this architectural shift completely eliminates the need for the Mittag-Leffler function during execution, while simultaneously providing a highly optimized structure suitable for high-performance computing acceleration techniques [26, 27].

1.5. The need for efficient computation and the novel differential approach proposed

Despite the descriptive power of viscoelastic constitutive models, their practical application in engineering workflows faces a number of computational challenges. In particular, the main bottleneck occurs when the evaluation of the material’s constitutive response must be repeated a very large number of times, and each evaluation is intrinsically expensive when the material model retains memory of its entire loading history.

The first challenge is *parameter identification*: given a set of experimental measurements,

such as creep or relaxation curves, the objective is to find the set of material parameters that best reproduces the observed data. This is typically formulated as an optimization problem, in which the model response must be evaluated repeatedly as the optimizer iterates towards the best-fit solution. Even a moderate number of experimental datasets, each discretized over many time points, leads to a large number of model evaluations, making the identification procedure computationally demanding.

The second challenge arises in *Finite Element* (FE) simulations of structures composed of viscoelastic materials. In a spatially discretized FE model, the computation of the current stress or strain must be performed independently at every numerical integration point and at every time step of the simulation. For problems involving a large number of degrees of freedom or long time horizons, the cost of performing a history-dependent evaluation at each point and each instant becomes prohibitive.

In both contexts, a method that avoids the storage and processing of the full loading history, or that substantially reduces its cost, would translate directly into practical speedups, enabling parameter identifications and structural simulations that are currently intractable.

1.6. Aim and objectives

This thesis addresses precisely the final point of the previous section. Our aim is to develop and validate a computationally efficient differential approach for evaluating the viscoelastic response of materials under arbitrary loading histories, with particular focus on fractional viscoelastic models. By reformulating the problem within an optimized differential context, this work aims to mitigate the cumulative cost of the well-known slowest operations, thereby speeding up the simulation and model-fitting tasks by reducing the per-step computational cost.

To achieve this aim, the thesis sets out the following objectives:

- **Develop and analyze numerical methods for fractional calculus**, assessing their accuracy and computational cost across a range of schemes and discretization methods.
- **Apply the differential approach to parameter identification** from experimental rheological data, embedding it within an optimization framework, and evaluating the efficiency gains and speedups achieved against the Boltzmann integral evaluation to quantify its efficiency.

- **Predict and benchmark material responses**, for both stress and strain, of materials under arbitrary loading histories using the proposed differential formulation. The accuracy and execution time of our framework is then benchmarked against the traditional integral approach.
- **Deliver optimized open-source tools** implementing the proposed numerical frameworks to make the methods developed in this thesis accessible to the broader research community.

1.7. Outline of the thesis

Following this brief introduction to the main topics of the thesis, the rest of the document is structured as follows:

- Chapter 2 is devoted to an in-depth examination of the key theoretical concepts of viscoelasticity, taking into account the current state of the art regarding the libraries used in the analysis and fitting of rheological data, along with their limitations.
- Chapter 3 examines and compares various methods for calculating fractional derivatives based on finite differences, laying the foundations for the subsequent chapters.
- Chapter 4 is dedicated to an analysis of the parameter-fitting process for viscoelastic models. We start this analysis with an evaluation of the individual cost function, before moving on to an examination of the fitting as a whole, comparing it with that of previously validated approaches.
- Chapter 5, on the other hand, is devoted to predicting the relevant quantities (stress or strain) by using and comparing different approaches, again with the aim of developing accurate and rapid methods.
- Chapter 6 concludes the thesis by summarizing the various results obtained in previous chapters. Finally, we proceed to identify future research avenues that our work has opened up.

2 | Theoretical Background and Methods

This chapter provides the theoretical foundations of viscoelasticity, with particular emphasis on *linear viscoelasticity*. First, the fundamental concepts of stress relaxation and creep are introduced through the relaxation modulus and creep compliance, which describe the time-dependent response of viscoelastic materials. *The Boltzmann superposition principle* is then presented as a theoretical framework for predicting material responses under arbitrary loading histories.

Subsequently, the limitations of classical viscoelastic models are discussed, motivating the introduction of fractional viscoelastic models. The relationship between fractional constitutive models and the concepts introduced earlier in the chapter is examined, together with the mathematical and computational challenges associated with their implementation.

Finally, the chapter reviews the principal open-source software libraries currently available for the computation of fractional viscoelastic models, highlighting their strengths and limitations. This discussion provides the context and motivation for the computational approach proposed in this work and establishes the foundations for the study of more advanced nonlinear viscoelastic models.

2.1. Fundamentals of Linear Viscoelasticity

Linear viscoelasticity provides the fundamental mathematical framework for describing the response of materials subjected to external loads or deformations. In this framework, the relation among stress σ , strain ε and time t is specific to a given material. Such fundamental relationship is called *constitutive equation*.

However, for this to be valid, the materials in question must satisfy the assumptions of linearity, which consist of the following two properties:

- Scaling the applied strain by a constant factor results in an equivalent scaling of the corresponding stress response.

$$\sigma(A\varepsilon) = A\sigma(\varepsilon) \quad (2.1)$$

- The material's total response to a sequence of loads can be determined by linearly combining the responses to each individual load applied independently, which means the following:

$$\varepsilon(t) = \varepsilon_1(t) + \varepsilon_2(t) \quad (2.2)$$

$$\sigma(\varepsilon) = \sigma(\varepsilon_1) + \sigma(\varepsilon_2) \quad (2.3)$$

These assumptions are generally valid only for sufficiently small deformations. In practice, many materials exhibit linear or quasi-linear behaviour when subjected to strains below a material-dependent threshold [9, 19].

Small deformations The concept of small deformation is inherently material-dependent. The linearity assumption is considered valid as long as the applied strain remains below a critical threshold. For relatively stiff materials, such as metals and ceramics, linear behavior is typically observed for strains below approximately 0.1%. In contrast, soft materials such as hydrogels and biological tissues may remain within the linear regime for strains of several percent. For instance, epithelial monolayers have been shown to exhibit a viscoelastic stress relaxation response when subjected to stretch up to 30% strain [28]. Conversely, brain tissue, despite being an extremely soft material, displays a much narrower linear regime: strain sweep experiments on bovine brain tissues showed that the viscoelastic strain limit is of the order of 0.1% strain [29], highlighting how the extent of the linear regime can vary even between soft biological materials. Beyond these thresholds, one or both linearity assumptions may no longer hold, resulting in non-linear material behavior driven by additional physical mechanisms.

2.1.1. Relaxation and Creep Moduli

Two fundamental time-domain experiments are commonly used to characterize the viscoelastic behavior of materials: the stress relaxation test and the creep test.

In a stress relaxation experiment, an instantaneous strain ε_0 is applied at time $t = 0$ and subsequently maintained constant, while the resulting stress $\sigma(t)$ is monitored over time. For a viscoelastic material, the stress gradually decreases as the material undergoes

internal structural rearrangements that accommodate the imposed deformation. This behavior is characterized by the relaxation modulus $G(t)$, defined as the ratio of the time-varying stress to the applied constant strain:

$$G(t) = \frac{\sigma(t)}{\varepsilon_0} \quad (2.4)$$

For a step-strain input, the relaxation modulus is generally a monotonically decreasing function of time that characterizes the progressive decay of stress following deformation [30]. Physically, this behavior arises from stress relaxation mechanisms within the material. After the initial elastic response, the internal microstructure progressively reorganizes, reducing the stress required to maintain the imposed strain. Consequently, the stress decreases over time, leading to a corresponding decrease in the relaxation modulus.

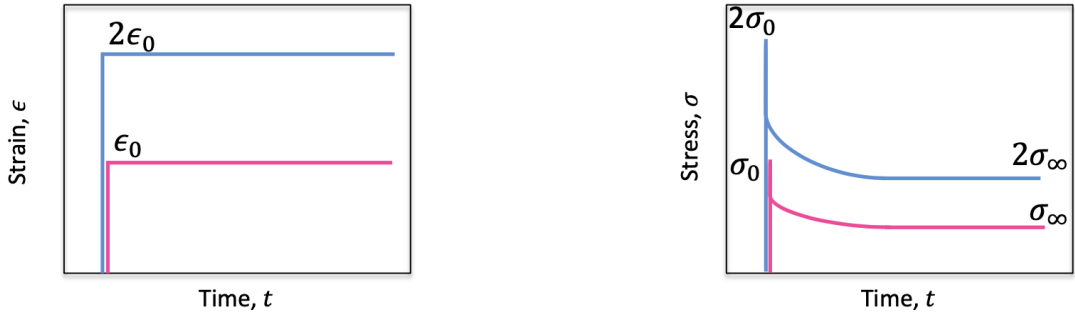


Figure 2.1: Relaxation test using a constant strain ε as input, adapted from [7]. The image also illustrates the first linearity assumption between stress and strain.

In contrast, a creep experiment consists of applying an instantaneous constant stress σ_0 at $t = 0$ and measuring the resulting time-dependent strain response $\varepsilon(t)$. The material typically exhibits an initial elastic deformation followed by a time-dependent increasing elongation. This response is described by the creep compliance $J(t)$, defined as:

$$J(t) = \frac{\varepsilon(t)}{\sigma_0} \quad (2.5)$$

Unlike the relaxation modulus, $J(t)$ is a monotonically increasing function over time.

It is important to note that $G(t)$ and $J(t)$ are material-specific properties. As will be discussed in Section 2.2, the exact analytical forms of these functions and, consequently, their ability to accurately capture the material's real-world behaviour depend entirely on the model we adopt.

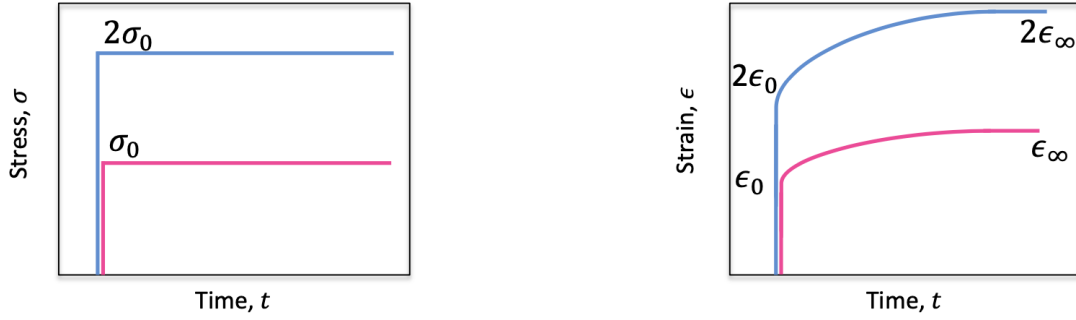


Figure 2.2: Creep test using a constant stress σ as input, adapted from [7]. The image also illustrates the first linearity assumption between stress and strain.

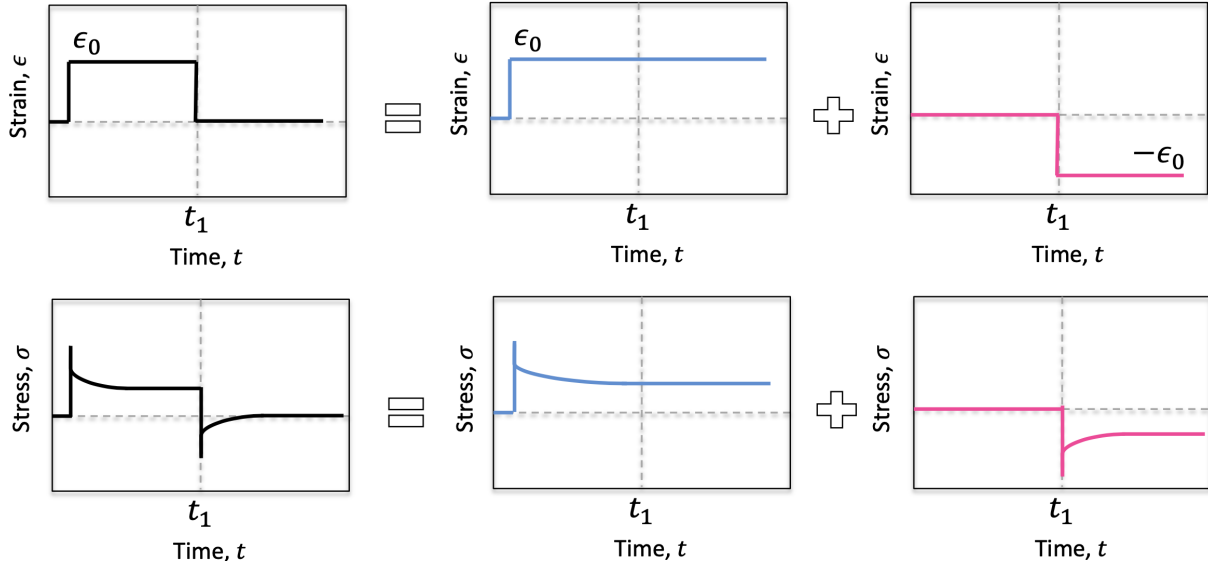


Figure 2.3: Application of the Boltzmann Superposition Effect on a constant strain ϵ input, adapted from [7]. As can be seen, the effect of the sum between two different strain functions applies in the same way to the corresponding stress.

2.1.2. Boltzmann Superposition Principle

While $G(t)$ and $J(t)$ describe the material response to simple, single-step loading conditions, real-world applications involve continuous, time-varying loading histories. To relate stress and strain under arbitrary conditions, linear viscoelasticity relies on the Boltzmann Superposition Principle, which provides the fundamental framework for extending elementary responses to general loadings. Note that the Boltzmann Superposition Principle is a direct consequence of the assumptions of linearity: the material response at a given time t can be expressed as the cumulative effect of all infinitesimal perturbations applied at earlier times. Figure 2.3 illustrates the application of the Boltzmann Superposition Principle to the simple case of loading and unloading.

Mathematically, this principle is expressed through the Boltzmann integral [8, 31]. For an arbitrary strain history $\varepsilon(t)$, the resulting stress $\sigma(t)$ is computed by convolving the relaxation modulus over the entire history:

$$\sigma(t) = \int_0^t G(t - \tau) \frac{d\varepsilon(\tau)}{d\tau} d\tau \quad (2.6)$$

Similarly, if the stress history $\sigma(t)$ is known and the objective is to predict the resulting strain, the Boltzmann superposition integral has the following form:

$$\varepsilon(t) = \int_0^t J(t - \tau) \frac{d\sigma(\tau)}{d\tau} d\tau \quad (2.7)$$

It is important to emphasize that, in the linear viscoelastic regime, knowledge of either the relaxation modulus or the creep compliance is sufficient to compute the material response to any loading history. However, as will be discussed in the subsequent sections, evaluating these convolution integrals analytically for advanced constitutive models, or discretizing them efficiently for numerical implementation, introduces significant mathematical and computational challenges.

2.2. Classical viscoelastic models

Although the Boltzmann superposition principle defines the general form of the stress–strain relationship in linear viscoelasticity, it does not directly provide a usable constitutive model, since the relaxation modulus $G(t)$ and creep compliance $J(t)$ must still be specified. In order to obtain explicit expressions for the relaxation and creep moduli, different modeling strategies can be employed. A widely adopted approach is based on the construction of mechanical analogues using fundamental viscoelastic elements arranged in series and parallel configurations. This element-based formulation is conceptually similar to the representation of electrical circuits through basic circuit components.

These networks are not only convenient representations; each configuration is associated to a mathematical relation that links stress and strain and fully characterizes the mechanical response of a viscoelastic material. This relationship is called *constitutive equation*. Once the constitutive equation is specified, the relaxation modulus $G(t)$ and creep compliance $J(t)$ can be derived analytically by applying appropriate loading conditions, thus boundary conditions - respectively constant strain and constant stress.

The two fundamental building blocks of classical linear viscoelasticity are the Hookean

spring and the Newtonian dashpot.

The Hookean spring represents ideal elastic behavior, in which stress is proportional to strain:

$$\sigma(t) = k\varepsilon(t),$$

where k denotes the elastic modulus.

The Newtonian dashpot represents ideal viscous behavior, where stress is proportional to the strain rate:

$$\sigma(t) = \eta \frac{d\varepsilon(t)}{dt},$$

where η is the viscosity coefficient.

The spring and dashpot are shown in Figure 2.4. By combining these elementary components in series and parallel arrangements, more complex viscoelastic responses can be obtained.

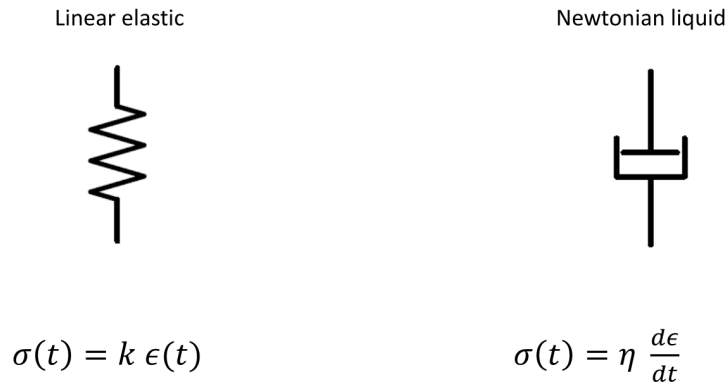


Figure 2.4: Visual representation of basic elastic and viscous components. They can be arranged together the same way as digital circuits.

2.2.1. The two simplest viscoelastic models

The simplest models are obtained by combining the two fundamental rheological elements in series and in parallel. In both cases, the mechanical configuration of the elements directly determines the constitutive equation of the material, which represents the fundamental relationship between stress and strain and fully defines the viscoelastic response of the model.

The Maxwell model

The Maxwell model consists of a spring and a dashpot connected in series, as shown in Figure 2.5.

Deriving Constitutive equation In this configuration, both elements experience the same stress, while the total strain is given by the sum of the strains of the individual components. Thus, the total strain $\varepsilon(t)$ is the sum of the elastic strain $\varepsilon_e(t)$ and the viscous strain $\varepsilon_v(t)$:

$$\varepsilon(t) = \varepsilon_e(t) + \varepsilon_v(t)$$

while the stress remains the same in both the elements:

$$\sigma(t) = \sigma_e(t) = \sigma_v(t).$$

Taking the first time derivative of the total strain yields the strain rate relation:

$$\dot{\varepsilon}(t) = \dot{\varepsilon}_e(t) + \dot{\varepsilon}_v(t)$$

The individual mechanical responses of the spring and dashpot are governed by their respective linear ideal equations:

$$\sigma(t) = k\varepsilon_e(t) \implies \dot{\varepsilon}_e(t) = \frac{\dot{\sigma}(t)}{k}$$

$$\sigma(t) = \eta\dot{\varepsilon}_v(t) \implies \dot{\varepsilon}_v(t) = \frac{\sigma(t)}{\eta}$$

Substituting these individual strain rates back into the total strain rate equation provides the classical first-order ordinary differential equation (ODE) that defines the Maxwell constitutive model:

$$\dot{\varepsilon}(t) = \frac{\dot{\sigma}(t)}{k} + \frac{\sigma(t)}{\eta} \quad (2.8)$$

This relation demonstrates that the total rate of deformation is a linear combination of the elastic stress rate and the viscous stress state.

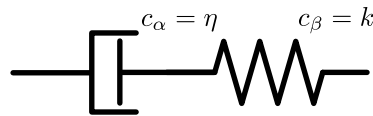


Figure 2.5: Schematic representation of the Maxwell model.

Deriving relaxation modulus The relaxation modulus $G(t)$ is defined as the stress response to a constant step strain applied at $t = 0$:

$$\varepsilon(t) = \varepsilon_0 H(t)$$

(where $H(t)$ is the Heaviside step function, meaning $\varepsilon(t) = \varepsilon_0$ and $\dot{\varepsilon}(t) = 0$ for $t > 0$). For $t > 0$, setting $\dot{\varepsilon}(t) = 0$ transforms the governing equation into a separable, first-order linear ODE:

$$0 = \frac{1}{k} \frac{d\sigma}{dt} + \frac{\sigma}{\eta} \implies \frac{d\sigma}{\sigma} = -\frac{k}{\eta} dt$$

Integrating both sides from $t = 0$ to t gives:

$$\ln \left(\frac{\sigma(t)}{\sigma(0)} \right) = -\frac{k}{\eta} t \implies \sigma(t) = \sigma(0) e^{-\frac{k}{\eta} t}$$

At the exact instant of loading ($t = 0$), the dashpot acts as an infinitely rigid wall because it cannot deform instantaneously. Therefore, the initial stress is absorbed purely by the spring: $\sigma(0) = k\varepsilon_0$. Substituting this back yields:

$$\sigma(t) = k\varepsilon_0 e^{-\frac{k}{\eta} t}$$

Since $G(t) = \frac{\sigma(t)}{\varepsilon_0}$, we define the relaxation modulus:

$$G(t) = k e^{-\frac{k}{\eta} t} \tag{2.9}$$

Stress decays exponentially as energy is dissipated through the dashpot.

Deriving creep modulus The Explicit Creep Compliance $J(t)$ measures the material's progressive deformation under a constant step stress:

$$\sigma(t) = \sigma_0 H(t)$$

(meaning $\sigma(t) = \sigma_0$ and $\dot{\sigma}(t) = 0$ for $t > 0$). Substituting $\dot{\sigma}(t) = 0$ into the core Maxwell equation for $t > 0$ yields:

$$\dot{\varepsilon}(t) = \frac{\sigma_0}{\eta} \implies \frac{d\varepsilon}{dt} = \frac{\sigma_0}{\eta}$$

Integrating this relation with respect to time gives:

$$\varepsilon(t) = \varepsilon(0) + \frac{\sigma_0}{\eta} t$$

The initial instantaneous deformation $\varepsilon(0)$ occurs entirely within the spring at $t = 0$ before the dashpot begins to flow, meaning $\varepsilon(0) = \frac{\sigma_0}{k}$. Combining these terms results in:

$$\varepsilon(t) = \frac{\sigma_0}{k} + \frac{\sigma_0}{\eta}t = \sigma_0 \left(\frac{1}{k} + \frac{t}{\eta} \right)$$

Since $J(t) = \frac{\varepsilon(t)}{\sigma_0}$, the creep compliance is explicitly resolved as:

$$J(t) = \frac{1}{k} + \frac{t}{\eta} \quad (2.10)$$

Displays an instantaneous elastic jump followed by a permanent, linear viscous flow over time.

The Kelvin-Voigt model

In contrast, the Kelvin-Voigt model combines a spring and a dashpot in parallel, as shown in 2.6.

Deriving Constitutive equation Here, both elements undergo the same strain,

$$\varepsilon(t) = \varepsilon_e(t) = \varepsilon_v(t)$$

while the total stress is obtained by summing the contributions of the elastic and viscous components

$$\sigma(t) = \sigma_e(t) + \sigma_v(t).$$

We substitute the ideal, linear equations for each elementary component into the stress summation rule, the spring governs the ideal elastic stress response, which is directly proportional to strain

$$\sigma_e(t) = k\varepsilon_e(t),$$

The dashpot governs the ideal viscous stress response, which is directly proportional to the strain rate

$$\sigma_v(t) = \eta\dot{\varepsilon}_v(t) = \eta\frac{d\varepsilon_v(t)}{dt}.$$

Using the strain equivalence, we substitute the individual stress expressions back into the total stress balance equation:

$$\sigma(t) = k\varepsilon(t) + \eta\dot{\varepsilon}(t)$$

Rearranging this into standard differential form yields the classical first-order ordinary differential equation that defines the Kelvin–Voigt model:

$$\sigma(t) = k\varepsilon(t) + \eta \frac{d\varepsilon(t)}{dt}. \quad (2.11)$$

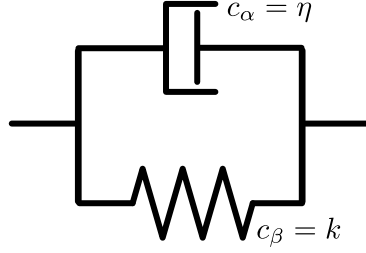


Figure 2.6: Schematic representation of the Kelvin Voigt model.

Deriving relaxation modulus The relaxation modulus measures the stress response to an instantaneous, constant step strain applied at $t = 0$:

$$\varepsilon(t) = \varepsilon_0 H(t)$$

Because the elements are in parallel and deform independently under a forced kinematic history, we can substitute the input directly into the original governing ODE without integrating. The mathematical derivative of the Heaviside step function is the Dirac delta function ($\delta(t)$), which represents an infinitely tall, infinitely thin spike at the origin:

$$\frac{dH(t)}{dt} = \delta(t)$$

Substituting $\varepsilon(t) = \varepsilon_0 H(t)$ into the constitutive equation gives:

$$\sigma(t) = k\varepsilon_0 H(t) + \eta\varepsilon_0 \delta(t)$$

For $t > 0$, the delta spike vanishes ($\delta(t) = 0$) and the step function stabilizes ($H(t) = 1$). Since $G(t) = \frac{\sigma(t)}{\varepsilon_0}$, the relaxation modulus settles directly to a constant value for all positive times:

$$G(t) = k \quad (\text{for } t > 0) \quad (2.12)$$

Deriving creep modulus The creep compliance measures the progressive deformation over time when a constant step stress is applied at $t = 0$:

$$\sigma(t) = \sigma_0 H(t).$$

For $t > 0$, the governing equation becomes a non-homogeneous, first-order linear ODE:

$$\sigma_0 = k\varepsilon(t) + \eta \frac{d\varepsilon}{dt} \implies \frac{d\varepsilon}{dt} + \frac{k}{\eta} \varepsilon(t) = \frac{\sigma_0}{\eta}$$

To solve this, we introduce the characteristic relaxation time constant, $\tau = \frac{\eta}{k}$. Rearranging the terms to integrate yields:

$$\frac{d\varepsilon}{\frac{\sigma_0}{k} - \varepsilon} = \frac{k}{\eta} dt \implies \int \frac{d\varepsilon}{\varepsilon - \frac{\sigma_0}{k}} = - \int \frac{1}{\tau} dt$$

Integrating both sides from the initial state ($\varepsilon(0) = 0$, as the dashpot prevents instantaneous jumping) to time t gives:

$$\ln \left(\frac{\varepsilon(t) - \frac{\sigma_0}{k}}{-\frac{\sigma_0}{k}} \right) = -\frac{t}{\tau} \implies \varepsilon(t) = \frac{\sigma_0}{k} \left(1 - e^{-\frac{t}{\tau}} \right)$$

Dividing the total strain by the constant stress input ($J(t) = \frac{\varepsilon(t)}{\sigma_0}$) yields the explicit creep compliance:

$$J(t) = \frac{1}{k} \left(1 - e^{-\frac{t}{\tau}} \right) \quad (2.13)$$

This mechanical arrangement provides a direct representation of characteristic creep behavior. When a constant step stress is applied, the model predicts a gradual, delayed strain increase that asymptotically approaches an equilibrium value. Unlike the Maxwell model, the Kelvin–Voigt configuration cannot exhibit instantaneous deformation under a finite applied stress; the viscous dashpot would require an infinite force to deform at an infinite rate. Conversely, if an ideal instantaneous step strain is imposed, the strain rate becomes singular, producing a Dirac delta contribution in the stress response at $t = 0$. Following this initial instantaneous impulse, the stress immediately settles to its long-term elastic equilibrium value ($k\varepsilon_0$) and exhibits no further relaxation over time.

2.3. Fractional viscoelastic models

Real-world viscoelastic materials often exhibit relaxation mechanisms that span multiple time scales. Within the classical framework, this behavior can be approximated by extending the single Maxwell element to a Generalized Maxwell model, consisting of N Maxwell arms arranged in parallel. Under a step-strain condition, the total stress is given by the sum of the contributions of each arm, resulting in a relaxation modulus expressed as a finite sum of exponentials.

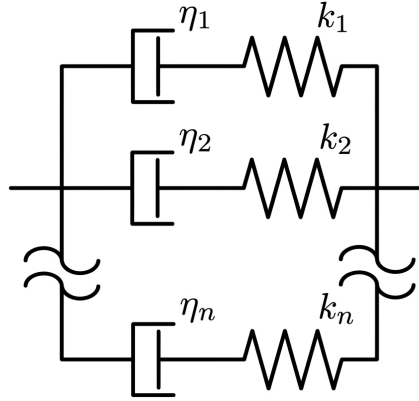


Figure 2.7: Schematic representation of the Generalized Maxwell model

This representation corresponds to a discrete approximation of the underlying relaxation spectrum, where each Maxwell arm contributes a characteristic relaxation time. While increasing the number of elements improves the flexibility of the model and allows accurate fitting over a wide time range, this approach remains fundamentally discrete.

Experimental observations on complex materials, however, often reveal a fundamentally different behavior: the relaxation modulus follows a power-law scaling rather than a sum of exponentials,

$$G(t) = Ct^{-\alpha}, \quad 0 < \alpha < 1, \quad (2.14)$$

where α governs the strength of viscoelastic memory effects.

A power-law response can be interpreted as the limiting case of a continuous distribution of relaxation times. In this view, the Generalized Maxwell model, shown in Figure 2.7, can reproduce such behavior only in the limit $N \rightarrow \infty$, with relaxation times distributed over a continuous spectrum, typically on a logarithmic scale. This continuum limit highlights the limitation of integer-order differential models, which rely on a finite number of characteristic time scales.

This mismatch between discrete-time representations and observed continuous-scale behavior provides the motivation for fractional calculus. Fractional-order operators naturally encode power-law memory effects in a compact form, allowing the description of a continuous relaxation spectrum with a minimal number of parameters.

To overcome the limitations of classical spring-dashpot networks, fractional calculus introduces a new constitutive element: the springpot. Unlike ideal springs or dashpots, which represent purely elastic or purely viscous limits, the springpot models an intermediate, interpolative material state where the stress is proportional to a fractional-order time derivative of the strain. The relaxation modulus for power-law takes the form $G(t) = At^{-\beta}$.

Substituting this into the stress equation, the relationship between stress and strain for a power-law material becomes:

$$\sigma(t) = A \int_0^t (t - \tau)^{-\beta} \frac{d\varepsilon(\tau)}{d\tau} d\tau \quad (2.15)$$

The definition of the generalization of the differentiation operation to non-integer order valid for a function $f(t) = 0$ for $t < 0$ let us define $A = \frac{c_\beta}{\Gamma(1-\beta)}$ to get the equation:

$$\sigma(t) = \frac{c_\beta}{\Gamma(1-\beta)} \int_0^t (t - \tau)^{-\beta} \frac{d\varepsilon(\tau)}{d\tau} d\tau = c_\beta \frac{d^\beta \varepsilon(t)}{dt^\beta}, \quad (2.16)$$

which provides a simple constitutive equation for power-law materials [9].

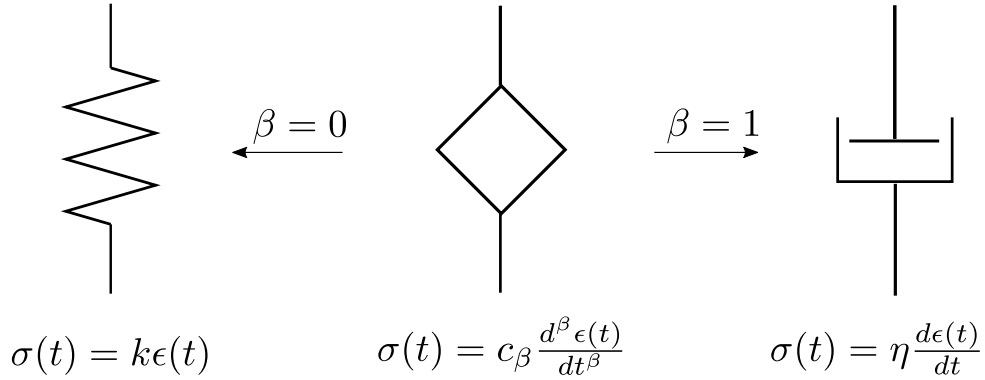


Figure 2.8: Visual representation of springpot component and its relation to spring and dashpot.

From Repeated Integration to Fractional Operators To better understand fractional operators, it is useful to examine the mathematical structure that motivates their definition. Consider the n -fold repeated integration of a function $f(t)$, with $n \in \mathbb{N}$:

$$I^n f(t) = \int_0^t \int_0^{t_1} \cdots \int_0^{t_{n-1}} f(\tau) d\tau dt_{n-1} \cdots dt_1. \quad (2.17)$$

The Cauchy formula for repeated integration collapses it into a single convolution integral:

$$I^n f(t) = \frac{1}{(n-1)!} \int_0^t (t - \tau)^{n-1} f(\tau) d\tau = \frac{1}{\Gamma(n)} \int_0^t (t - \tau)^{n-1} f(\tau) d\tau, \quad (2.18)$$

where the factorial $(n-1)!$ has been rewritten using the Gamma function via the identity $\Gamma(n) = (n-1)!$, valid for $n \in \mathbb{N}$. This particular integral forms the basis for the calculation of fractional derivatives, which we will examine in Chapter 3.

2.3.1. Generalized fractional models

While a single springpot accurately models a continuous power-law regime, real-world materials often exhibit power-law dynamics that are bounded or transition between different states across temporal scales. To capture this diversity of behaviors without falling into the over-parameterization trap, springpots can be combined with traditional elements or with other springpots, forming generalized fractional models.

A major advantage of representing these constitutive equations through network schematics is that it allows for an immediate visual and physical intuition of their asymptotic behavior. By analyzing the interaction of two fractional elements with different exponents α and β (assuming $\alpha > \beta$), a universal heuristic rule can be established for short-time ($t \rightarrow 0$) and long-time ($t \rightarrow \infty$) responses:

- Fractional Maxwell shown in 2.9, when two springpots are arranged in series, the total compliance is the sum of individual compliances. Consequently, the stiffer, more elastic element (lower exponent β) dominates the high-frequency or short time-scale response ($t \rightarrow 0$). Conversely, the more fluid-like element (higher exponent α) dictates the long time-scale behavior ($t \rightarrow \infty$). Mathematically, the asymptotic limits for the relaxation modulus $G(t)$ and creep compliance $J(t)$ satisfy:

$$G(t) \sim \begin{cases} t^{-\beta} & t \rightarrow 0 \\ t^{-\alpha} & t \rightarrow \infty \end{cases} \quad \text{and} \quad J(t) \sim \begin{cases} t^{\beta} & t \rightarrow 0 \\ t^{\alpha} & t \rightarrow \infty \end{cases}$$

Deriving constitutive equation To formally derive the governing fractional differential equation, we follow a mechanical splitting approach analogous to the classical integer-order framework. Let the two springpots be characterized by their respective fractional parameters: (c_{α}, α) and (c_{β}, β) , with $0 \leq \beta < \alpha \leq 1$. Because the elements are arranged in a series configuration, the total strain $\varepsilon(t)$ is partitioned as the sum of the individual strains of each springpot, whereas the stress $\sigma(t)$ remains uniform across the entire network:

$$\varepsilon(t) = \varepsilon_{\alpha}(t) + \varepsilon_{\beta}(t)$$

$$\sigma(t) = \sigma_{\alpha}(t) = \sigma_{\beta}(t)$$

The constitutive relations for each individual springpot, defined via the Caputo fractional derivative, are given by:

$$\sigma(t) = c_\alpha \frac{d^\alpha \varepsilon_\alpha(t)}{dt^\alpha}$$

$$\sigma(t) = c_\beta \frac{d^\beta \varepsilon_\beta(t)}{dt^\beta}$$

To isolate the individual strains, we apply the fractional integral operators of order α and β respectively (where I^γ is the Riemann-Liouville fractional integral, acting as the inverse operator to the Caputo derivative under zero initial conditions):

$$\varepsilon_\alpha(t) = \frac{1}{c_\alpha} I^\alpha \sigma(t)$$

$$\varepsilon_\beta(t) = \frac{1}{c_\beta} I^\beta \sigma(t)$$

Substituting these expressions back into the total strain decomposition equation yields:

$$\varepsilon(t) = \frac{1}{c_\alpha} I^\alpha \sigma(t) + \frac{1}{c_\beta} I^\beta \sigma(t)$$

To eliminate the fractional integral operators and obtain the final differential form, we apply the highest-order fractional derivative present in the system, which is $\frac{d^\alpha}{dt^\alpha}$, to the entire equation. Utilizing the composition property of fractional operators ($\frac{d^\alpha}{dt^\alpha} I^\gamma = \frac{d^{\alpha-\gamma}}{dt^{\alpha-\gamma}}$), we arrive at the constitutive relation for the fractional Maxwell model:

$$\frac{d^\alpha \varepsilon(t)}{dt^\alpha} = \frac{\sigma(t)}{c_\alpha} + \frac{1}{c_\beta} \frac{d^{\alpha-\beta} \sigma(t)}{dt^{\alpha-\beta}}$$

Rearranging the terms to match the standard format of fractional differential equations yields:

$$\sigma(t) + \frac{c_\alpha}{c_\beta} \frac{d^{\alpha-\beta} \sigma(t)}{dt^{\alpha-\beta}} = c_\alpha \frac{d^\alpha \varepsilon(t)}{dt^\alpha} \quad (2.19)$$

This fundamental equation demonstrates that the stress state and its fractional rate of change are coupled directly to the fractional strain rate of order α . In the limiting case where $\beta = 0$ (pure spring, $c_0 = k$) and $\alpha = 1$ (pure dashpot, $c_1 = \eta$), the operator $\frac{d^{\alpha-\beta}}{dt^{\alpha-\beta}}$ reduces to a first-order derivative, recovering precisely the classical Maxwell ordinary differential equation.

Deriving Relaxation modulus To determine the stress response under a constant step strain $\varepsilon(t) = \varepsilon_0 H(t)$, we transform the system into the Laplace domain

($\mathcal{L}\{\cdot\}$). Under quiescent initial conditions, the serial coupling produces a rational fractional polynomial for the transformed modulus $\bar{G}(s)$:

$$\bar{G}(s) = \frac{\bar{\sigma}(s)}{s\bar{\varepsilon}(s)} = \frac{1}{s \left(\frac{1}{c_\alpha s^\alpha} + \frac{1}{c_\beta s^\beta} \right)} = \frac{c_\alpha c_\beta s^{\alpha+\beta}}{s(c_\alpha s^\alpha + c_\beta s^\beta)}.$$

Inverting this expression back to the time domain is mathematically intensive due to the cross-coupling of the fractional exponents in the denominator. Following the operational calculus derived by [18](specifically Equations 27 to 31), this rational polynomial division is resolved via an infinite series expansion. The final analytical solution for the relaxation modulus forces the emergence of the two-parameter Mittag-Leffler function:

$$G(t) = c_\beta t^{-\beta} E_{\alpha-\beta, 1-\beta} \left(-\frac{c_\beta}{c_\alpha} t^{\alpha-\beta} \right), \quad (2.20)$$

where $E_{a,b}(z)$ represents the two-parameter Mittag-Leffler function defined as:

$$E_{a,b}(z) = \sum_{k=0}^{\infty} \frac{z^k}{\Gamma(ak + b)}. \quad (2.21)$$

Deriving Creep modulus We begin from the complex modulus $G^*(\omega)$ of the serial Maxwell configuration, which relates the transformed stress and strain profiles

$$G^*(\omega) = \frac{\bar{\sigma}(\omega)}{\bar{\varepsilon}(\omega)} = \frac{1}{\frac{1}{c_\alpha (i\omega)^\alpha} + \frac{1}{c_\beta (i\omega)^\beta}}.$$

The complex compliance function, $J^*(\omega)$, is defined as the reciprocal of the complex modulus. Invertibility in the frequency domain allows the serial operators to uncouple into a direct algebraic sum

$$J^*(\omega) = \frac{1}{G^*(\omega)} = \frac{1}{c_\alpha (i\omega)^\alpha} + \frac{1}{c_\beta (i\omega)^\beta}.$$

To map this frequency response back to the time-domain creep compliance $J(t)$, we utilize the inverse Fourier cosine transform relation

$$J(t) = \frac{2}{\pi} \int_0^\infty \operatorname{Re} \frac{J^*(\omega)}{i\omega} \cos(\omega t), d\omega.$$

Substituting the decoupled components of $J^*(\omega)$ into the inversion integral yields two independent improper integrals. Because the operators are uncoupled, these integrals evaluate directly to standard Euler Gamma functions without cross-exponent terms. Resolving the integration yields the final time-domain expression for the compliance profile:

$$J(t) = \frac{t^\alpha}{c_\alpha \Gamma(1 + \alpha)} + \frac{t^\beta}{c_\beta \Gamma(1 + \beta)} \quad (2.22)$$

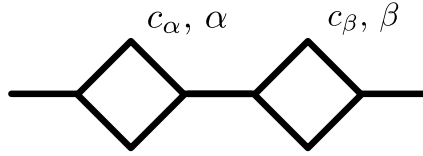


Figure 2.9: Schematic representation of the General Fractional Maxwell model.

- Fractional Kelvin-Voigt shown in Figure 2.10, when two springpots are placed in parallel, the total stress is the sum of individual stresses. In this case, the response is reversed: the highly viscous, higher-order element (α) dominates the short time-scale deformation ($t \rightarrow 0$) due to the high rate of shear. At long time-scales ($t \rightarrow \infty$), the stress relaxes through the lower-order, more solid-like element (β). The asymptotic scaling laws invert to:

$$G(t) \sim \begin{cases} t^{-\alpha} & t \rightarrow 0 \\ t^{-\beta} & t \rightarrow \infty \end{cases} \quad \text{and} \quad J(t) \sim \begin{cases} t^\alpha & t \rightarrow 0 \\ t^\beta & t \rightarrow \infty \end{cases}$$

Deriving Constitutive equation The same properties of parallel components we used for classical kelvin voigt are still valid,

$$\varepsilon(t) = \varepsilon_\alpha(t) = \varepsilon_\beta(t)$$

$$\sigma(t) = \sigma_\alpha(t) + \sigma_\beta(t).$$

Each branch contains a distinct springpot characterized by its own fractional order ($0 \leq \beta < \alpha \leq 1$) and material parameter (c_α, c_β):

$$\sigma_\alpha(t) = c_\alpha \frac{d^\alpha \varepsilon_\alpha(t)}{dt^\alpha}$$

$$\sigma_\beta(t) = c_\beta \frac{d^\beta \varepsilon_\beta(t)}{dt^\beta}.$$

Substituting these individual constitutive expressions into the stress balance equation and enforcing strain equivalence yields the governing Fractional Differential Equation for the model:

$$\sigma(t) = c_\alpha \frac{d^\alpha \varepsilon(t)}{dt^\alpha} + c_\beta \frac{d^\beta \varepsilon(t)}{dt^\beta} \quad (2.23)$$

Deriving Relaxation modulus Starting from the constitutive equation we can apply the continuous Fourier transform ($\mathcal{F}\{\cdot\}$) to transition the equation from the time domain to the angular frequency domain (ω). Utilizing the fractional derivative property $\mathcal{F}\left\{\frac{d^\gamma \varepsilon(t)}{dt^\gamma}\right\} = (i\omega)^\gamma \bar{\varepsilon}(\omega)$, we obtain:

$$\bar{\sigma}(\omega) = E\tau^\alpha (i\omega)^\alpha \bar{\varepsilon}(\omega) + E\tau^\beta (i\omega)^\beta \bar{\varepsilon}(\omega)$$

Factoring out the transformed strain $\bar{\varepsilon}(\omega)$ isolates the complex modulus $G^*(\omega)$

$$G^*(\omega) = \frac{\bar{\sigma}(\omega)}{\bar{\varepsilon}(\omega)} = E(i\omega\tau)^\alpha + E(i\omega\tau)^\beta.$$

To separate the complex modulus into real and imaginary parts, we apply Euler's formula to expand the imaginary unit identity $i^\gamma = \cos\left(\frac{\gamma\pi}{2}\right) + i \sin\left(\frac{\gamma\pi}{2}\right)$:

$$G^*(\omega) = E(\omega\tau)^\alpha \left[\cos\left(\frac{\alpha\pi}{2}\right) + i \sin\left(\frac{\alpha\pi}{2}\right) \right] + E(\omega\tau)^\beta \left[\cos\left(\frac{\beta\pi}{2}\right) + i \sin\left(\frac{\beta\pi}{2}\right) \right]$$

Grouping the real terms yields the Storage Modulus $G'(\omega)$, and grouping the imaginary terms yields the Loss Modulus $G''(\omega)$:

$$G'(\omega) = E(\omega\tau)^\alpha \cos\left(\frac{\alpha\pi}{2}\right) + E(\omega\tau)^\beta \cos\left(\frac{\beta\pi}{2}\right)$$

$$G''(\omega) = E(\omega\tau)^\alpha \sin\left(\frac{\alpha\pi}{2}\right) + E(\omega\tau)^\beta \sin\left(\frac{\beta\pi}{2}\right)$$

$$G^*(\omega) = G'(\omega) + iG''(\omega)$$

To return to the relaxation modulus $G(t)$ from the frequency components we utilize the Fourier sine transform integral:

$$G(t) = \frac{2}{\pi} \int_0^\infty \frac{G'(\omega)}{\omega} \sin(\omega t) d\omega$$

Substituting the derived expression for $G'(\omega)$ into this relation and evaluating the improper integrals via standard gamma function definitions yields the explicit, non-singular time-domain relaxation modulus:

$$G(t) = E\tau^\alpha \frac{t^{-\alpha}}{\Gamma(1-\alpha)} + E\tau^\beta \frac{t^{-\beta}}{\Gamma(1-\beta)}. \quad (2.24)$$

Deriving creep modulus To fully characterize the material behavior under a constant stress history $\sigma(t) = \sigma_0 H(t)$, the governing fractional differential equation must be inverted to solve for the strain response $\varepsilon(t)$. Moving from the Fourier domain back to the time-domain creep compliance $J(t)$ requires a rigorous mathematical inversion of coupled fractional operators. We begin from the fractional constitutive, the analytical inversion of this coupled operator structure is intensely complex, as it demands the resolution of rational fractional polynomials within the Laplace domain. A detailed step-by-step transformation of this framework, stepping through generalized operational calculus is provided by [18] (specifically within Equations 43 to 48). By adopting their operator inversion methodology, the coupled fractional polynomials are expanded into an infinite series that maps directly to the generalized transcendental framework. The final time-domain solution for the creep compliance $J(t)$ emerges not as a simple power law, but as a function governed by the two-parameter Mittag-Leffler (ML) function:

$$J(t) = \frac{\varepsilon(t)}{\sigma_0} = \frac{1}{E\tau^\alpha} t^\alpha E_{\alpha-\beta, \alpha+1} \left(-\frac{1}{\tau^{\alpha-\beta}} t^{\alpha-\beta} \right) \quad (2.25)$$

where $E_{a,b}(z)$ represents the two-parameter Mittag-Leffler defined in 2.21

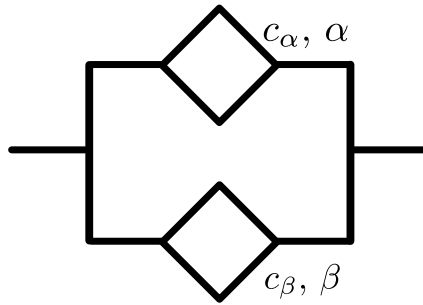


Figure 2.10: Schematic representation of the General Fractional Kelvin-Voigt model.

2.4. Computational Challenges

While fractional viscoelastic models provide a powerful framework for capturing the behavior of complex materials, their analytical and numerical evaluation introduces significant computational challenges. In contrast to classical integer-order models, whose differential equations are local in time, i.e., the system response at any instant depends solely on its current state and its previous state, fractional models are governed by non-local operators and complex special functions. The two primary challenges that arise in this context are the non-locality of fractional operators and the numerical evaluation of the Mittag-Leffler function.

2.4.1. The Non-Locality Problem

The fundamental mathematical distinction between integer-order and fractional-order derivatives lies in their locality. An integer-order derivative is a local operator: evaluating the derivative of a function at a specific time t requires information solely from the immediate infinitesimal neighborhood of t . Consequently, classical viscoelastic models can be formulated as systems of ordinary differential equations (ODEs), which can be solved step-by-step by updating only the current state variables. On the contrary, fractional derivatives are inherently non-local operators: they are defined through convolution integrals that span the entire history of the variable, from the initial time $t = 0$ up to the current time t , as seen before in Equations (2.6) and (2.7).

In a computational setting, this non-locality translates into the following: if the time domain is discretized into N steps, computing the system's state at the n -th step requires retrieving and processing all the $n - 1$ previous states. This leads to two major computational inefficiencies when solving these equations directly:

- **Memory complexity:** The storage requirements scale linearly with time, with complexity $\mathcal{O}(N)$.
- **Computational complexity:** The number of operations required to compute the convolution integral at step n is proportional to n . Therefore, the total computational cost to simulate N time steps scales quadratically, $\mathcal{O}(N^2)$.

For High-Performance Computing applications involving long-term simulations, this $\mathcal{O}(N^2)$ time complexity becomes a massive bottleneck, rendering naive numerical integration approaches practically unfeasible.

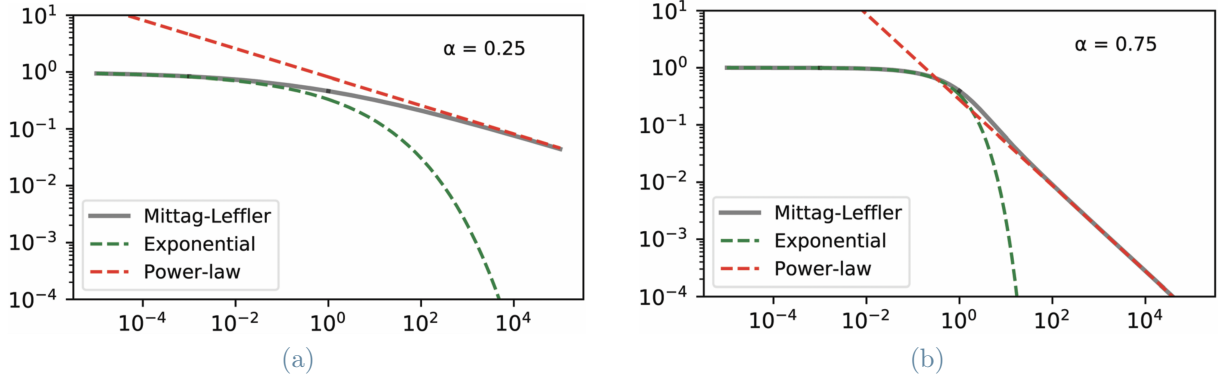


Figure 2.11: Log-log comparison of the Mittag-Leffler function (solid grey) with exponential (dashed green) and power-law (dashed red) decays for fractional parameters $\alpha = 0.25$, $\alpha = 0.5$ and $\beta = 1.0$. The graphs highlight how, for $\beta = 1.0$, the function interpolates between an exponential-like relaxation at short times and a power-law tail at long times.

2.4.2. The Mittag-Leffler Function Bottleneck

The second major challenge arises when we want to compute the relaxation modulus $G(t)$ and the creep compliance $J(t)$ for more complex models. Indeed, in fractional viscoelasticity, these two functions are typically expressed using a power series function, which represents the main bottleneck of the problem we want to solve, i.e., the Mittag-Leffler function E [21–24], also called the Queen Function of the Fractional Calculus [25]. This function is defined in its two-parameter form by the following power series:

$$E_{\alpha,\beta}(z) = \sum_{k=0}^{\infty} \frac{z^k}{\Gamma(\alpha k + \beta)} \quad (2.26)$$

where Γ is the Gamma function, $\alpha > 0$, $\beta \in \mathbb{R}$ and $z \in \mathbb{C}$.

While capturing the power-law decay and heavy-tailed relaxation typical of real viscoelastic materials, the Mittag-Leffler function introduces a massive bottleneck in computational routines, particularly during parameter identification and data fitting. Evaluating $E_{\alpha,\beta}(z)$ is notoriously difficult for several numerical reasons:

- **Slow convergence:** For large magnitudes of the argument z , the function converges extremely slowly, requiring an impractical number of terms to achieve acceptable accuracy.
- **Numerical instability:** In viscoelastic relaxation, the argument z is typically negative and large. Evaluating the series directly for large negative arguments leads

to high approximation errors due to alternating signs.

- **High computational overhead:** Typically, computing the Mittag-Leffler function may involve the analysis of different cases and the calculation of very complex integrals or series, which can take a considerable amount of time. In [32], the authors address this limitation by proposing advanced numerical routines based on the integral representation of the function, partitioning the complex plane into distinct regions to apply optimal quadrature formulas or asymptotic expansions. However, despite these algorithmic optimizations, the evaluation of the Mittag-Leffler function remains extremely expensive.

2.5. State of the Art and Proposed Approach

Currently, the analysis and fitting of rheological data, with the aim of studying the behaviour of viscoelastic materials, is primarily carried out using two open-source libraries: RHEOS.jl and pyRheo. While these tools provide valuable functionality for rheological modeling, they also present limitations that restrict their applicability in large-scale or high-performance computational settings. In the following, both frameworks are briefly discussed, highlighting their main strengths and drawbacks, which motivates the approach proposed in this work.

2.5.1. Available computational frameworks

RHEOS.jl RHEOS (Rheology Open Source), developed in Julia, is a flexible library designed for the simulation of general viscoelastic materials under arbitrary loading histories [33, 34]. Its computational core is based on the numerical evaluation of the Boltzmann superposition principle, where the stress or strain response is obtained through time-domain convolution integrals. Although this approach ensures high generality, it directly inherits the computational limitations associated with fractional models discussed in Section 2.4: in many cases, RHEOS must continuously evaluate the Mittag-Leffler function and retain the full memory of the system. As a consequence, the computational cost increases significantly when dealing with large datasets or high-frequency sampling, making the library unsuitable for High-Performance Computing (HPC) applications.

pyRheo pyRheo is a Python-based library developed with a stronger focus on accelerating the parameter identification process and facilitating model fitting [35]. However, it achieves this computational speedup by relying almost exclusively on known analytical closed-form solutions. In fact, the software is strictly limited to canonical inputs, being

able to efficiently process ideal step strains for relaxation tests or constant stress for creep tests. While this strategy allows for efficient parameter identification under controlled experimental conditions, it restricts the applicability of the library to idealized input scenarios. As a result, its use is less suited for problems involving arbitrary or noisy loading histories, where a more general time-evolution framework is required.

2.5.2. The Differential Approach

The aim of this work is to bridge the gap between generality and computational efficiency by developing a numerical framework that retains the flexibility of convolution-based formulations while avoiding their computational cost. Instead of directly evaluating the Boltzmann superposition integral and its associated special functions, the proposed approach is based on a direct discretization of the governing fractional differential equations (see Chapter 3). By discretizing the fractional differential equations directly, what we are doing is replacing the convolution with a faster loop that calculates the solution step by step, whilst also introducing techniques to further speed up this process.

This approach not only improves computational efficiency but also provides a flexible framework for extending fractional viscoelastic models to more complex, including non-linear, constitutive behaviors.

2.6. Computational Environment and Implementation

All numerical simulations presented in this work were carried out on a high-performance computing (HPC) cluster managed by the *PBS (Portable Batch System)* scheduler. Each job was allocated a single compute node equipped with up to 56 logical CPUs, enabling shared-memory parallelism within a single node without the overhead of inter-node communication protocols such as MPI.

2.6.1. The Julia Programming Language

The algorithms developed in this work were implemented in the Julia programming language [36], a high-level, high-performance language designed specifically for scientific and numerical computing. Julia addresses the so-called two-language problem: the need to prototype in a dynamic, expressive language (such as Python or MATLAB) and then rewrite performance-critical routines in a low-level language (such as C or Fortran). Julia achieves both goals simultaneously, offering the productivity of a high-level syntax while

delivering performance comparable to that of statically compiled languages.

This is made possible by Julia’s Just-In-Time (JIT) compilation model, built on the LLVM (Low Level Virtual Machine) compiler infrastructure. Rather than interpreting code at runtime or relying on static ahead-of-time compilation, Julia compiles each function to native machine code upon its first invocation, exploiting type specialization through its *multiple dispatch* system. As a result, the output generated is typically as efficient as the equivalent C code.

For the computationally intensive routines required by fractional viscoelastic models, such as those already discussed in Section 2.4, Julia’s performance characteristics are particularly advantageous. Moreover, Julia provides native support for multi-threading through its **Threads** standard library. Parallel execution was enabled at runtime via the `-t` flag, which specifies the number of threads allocated to the Julia process, allowing the workload to be distributed across all available logical CPUs on the compute node. This shared-memory parallelism was particularly beneficial in reducing wall-clock times for parameter identification routines, which require a large number of independent function evaluations.

2.6.2. Cluster

All the benchmarks and results obtained in this work were executed on a high-performance computing (HPC) cluster managed by the OpenPBS (Portable Batch System) workload manager - lab.c, the High Performance Computing Laboratory of the Civil and Environmental Engineering Department [37].

Batch computational jobs were submitted to the cluster environment via specialized shell scripts, specifying the required hardware resources, including dedicated compute nodes, up to 56 CPU cores, memory allocation, and maximum walltime constraints. OpenPBS is a software for workload management on HPC environment, its key features are speed, scalability, security, resilience and support for all modern infrastructure [38].

3 | Fractional Derivatives Implementation

This chapter explains why fractional derivatives play a fundamental role in rheology and examines the main algorithms used for their computation. First, the most widely adopted definitions of fractional derivatives are introduced, together with their numerical implementations and finite-difference discretizations. Next, the *Short Memory Principle* is presented as an effective strategy for significantly reducing computational cost while preserving an acceptable level of accuracy through suitable correction terms. Finally, the *Fast Fourier Transform* (FFT) is discussed as an alternative and highly efficient approach for accelerating the evaluation of fractional convolution operators.

3.1. Introduction

The accurate modeling of complex rheological behaviors, particularly in viscoelastic materials, often requires mathematical tools that go beyond classical integer-order calculus. As explained before in 2.2, it is well-established in the literature that the mechanical response of such materials is naturally described by Fractional Calculus [20, 26]. Unlike classical integer-order models, fractional-order derivatives are capable of capturing the hereditary properties and the long-term memory effects inherent in these physical systems.

However, the numerical solution of fractional operators within a High Performance Computing (HPC) context presents significant challenges. The fundamental characteristic of fractional derivatives is their *non-locality*. While the evaluation of a standard derivative depends only on the local behavior of a function, a fractional derivative of order α requires the integration of the function's entire history, from the initial time up to the current time step.

From a computational perspective, this implies that the calculation of the fractional derivative at the n -th time step requires the storage and processing of the solution at all previous $n - 1$ steps. Consequently, standard numerical implementations suffer from

a computational complexity and memory requirement that scale as $O(N^2)$ (where N is the number of time steps) [27, 39], making long-time simulations of large-scale rheological problems prohibitively expensive. This bottleneck gives rise to the need for specific strategies aimed at ensuring scalability and performance. To this end, we employ a combination of code parallelization, efficient finite difference discretization schemes and different techniques to reduce complexity and to minimize execution time.

Types of Fractional Derivatives

In this study, we consider three fundamental definitions of the fractional derivative: the Grünwald–Letnikov (GL), the Riemann–Liouville (RL), and the Caputo formulation [26, 39, 40]. Let $f(t)$ be the time-dependent function of interest (e.g., stress or strain) defined on the interval $[0, T]$, and let $\alpha \in (0, 1)$ denote the fractional order of the derivative.

Grünwald–Letnikov derivative. The Grünwald–Letnikov derivative is naturally connected to finite-difference limits and can be written as

$${}_{t_0}^{\text{GL}}D_t^\alpha f(t) = \lim_{h \rightarrow 0} \frac{1}{h^\alpha} \sum_{j=0}^{\lfloor \frac{t-t_0}{h} \rfloor} (-1)^j \binom{\alpha}{j} f(t - jh), \quad t \geq t_0. \quad (3.1)$$

where $\binom{\alpha}{j} = \frac{\Gamma(\alpha + 1)}{\Gamma(j + 1)\Gamma(\alpha - j + 1)}$ denotes the generalized binomial coefficient.

Caputo derivative. The Caputo derivative is defined as

$${}_{t_0}^{\text{C}}D_t^\alpha f(t) = \frac{1}{\Gamma(1 - \alpha)} \int_{t_0}^t \frac{f'(\tau)}{(t - \tau)^\alpha} d\tau, \quad t \geq t_0. \quad (3.2)$$

Here, the derivative acts on the function inside the integral, requiring $f(\tau)$ to be differentiable.

Riemann–Liouville derivative. The Riemann–Liouville definition is typically introduced via the fractional integral operator

$$({}_{t_0}I_t^\alpha f)(t) = \frac{1}{\Gamma(\alpha)} \int_{t_0}^t \frac{f(\tau)}{(t - \tau)^{1-\alpha}} d\tau, \quad \alpha > 0, \quad (3.3)$$

directly derived from Equation (2.18) and where $\Gamma(\cdot)$ is the Gamma function. The corresponding RL fractional derivative of order α is then defined as

$$\begin{aligned} {}^{\text{RL}}D_t^\alpha f(t) &= \frac{d}{dt} ({}_t I_t^{1-\alpha} f)(t) = \\ &= \frac{1}{\Gamma(1-\alpha)} \frac{d}{dt} \int_{t_0}^t \frac{f(\tau)}{(t-\tau)^\alpha} d\tau, \quad t \geq t_0. \end{aligned} \quad (3.4)$$

Relationships and equivalences. The existence of multiple definitions of the fractional derivative is due to the fact that the classical notion of differentiation admits different generalizations to non-integer orders, each prioritizing different analytical or computational properties. The Riemann–Liouville definition follows naturally from the theory of fractional integration and is often regarded as the canonical analytical formulation. The Grünwald–Letnikov definition generalizes the finite-difference approach used for integer-order derivatives and is therefore particularly attractive for numerical implementation. The Caputo derivative was introduced later by Caputo [41] to facilitate the treatment of initial-value problems arising in physical applications.

Moreover, provided that the Grünwald–Letnikov derivative exists and f satisfies the regularity assumptions required for both operators to be well defined, the Grünwald–Letnikov and Riemann–Liouville derivatives coincide [27]:

$${}^{\text{RL}}D_t^\alpha f(t) = {}^{\text{C}}D_t^\alpha f(t) + \sum_{k=0}^{m-1} \frac{f^{(k)}(t_0)(t-t_0)^{k-\alpha}}{\Gamma(k+1-\alpha)}, \quad (3.5)$$

where $m-1 < \alpha < m$, m is a positive integer, $f \in C^{m-1}[t_0, t]$ and $f^{(m)}$ is integrable on $[t_0, t]$.

Furthermore, if f is sufficiently smooth on $[t_0, t]$, the Grünwald–Letnikov and Riemann–Liouville derivatives are equivalent [27]:

$${}^{\text{GL}}D_t^\alpha f(t) = {}^{\text{RL}}D_t^\alpha f(t), \quad f \in C^m([t_0, T]). \quad (3.6)$$

Finally, the Caputo formulation is often preferred in physical and engineering applications due to its compatibility with standard integer-order initial conditions, which are the type of conditions typically available from experimental measurements. In contrast, the Riemann–Liouville and Grünwald–Letnikov formulations generally require fractional initial conditions, which lack a direct physical interpretation and are difficult to impose experimentally. Nevertheless, in the present work the GL formulation is adopted for numerical

computations because of its natural finite-difference structure, which makes it particularly suitable for efficient discretization and high-performance implementation.

Common properties and HPC implications. Despite their analytical differences, all three formulations share a non-local character, requiring the integration (or summation) of the function's entire history up to the current time t . However, since the historical values $f(\tau)$ for $\tau < t$ are already computed and immutable at the current time step, this data dependency structure poses no race conditions for past values. This feature renders the calculation inherently suitable for parallelization strategies, as the heavy contribution of the "history tail" can be distributed across multiple processors while the current step is being evaluated.

3.2. Numerical Implementation via Finite Differences

In our work, all fractional operators are discretized in time by means of Finite Difference Method (FDM) on a uniform grid. Let Δt be the constant time step, $t_n = t_0 + n\Delta t$ (for $n = 0, 1, \dots, N$) be the time at step n , and let $f_n \approx f(t_n)$ denote the approximated value of the function at step n . The FDM allows us to transform the integral and derivative operators defined in the previous section into algebraic summations involving the history of the known function f .

Regardless of the specific formulation, all the discretization methods described below share a common structural pattern characterized by two main phases:

- **Precomputation of Convolution Weights:** Before entering the time-stepping simulation, we compute a set of coefficients, known as convolution weights ω_j , which depend on the method used; consequently, the strategy employed to calculate them has a significant impact on performance.
- **Nested Loop Structure:** The actual evaluation of the fractional derivative relies on two nested loops. The former, the outer loop, iterates through each discrete time step to reconstruct the time evolution of the system. Inside it, an inner loop, the latter one, computes the algebraic summation over the historical states of the function f , weighting past contributions through the precomputed ω_j .

3.2.1. Grünwald–Letnikov (GL) Discretization

$${}_{t_0}^{\text{GL}} D_t^\alpha f_n \approx \frac{1}{\Delta t^\alpha} \sum_{j=0}^n \omega_j^{(\text{GL})} f_{n-j}, \quad (3.7)$$

where $f_{n-j} = f(t_n - j\Delta t)$ and the weights $\omega_j^{(\text{GL})}$ are the generalized binomial coefficients $(-1)^j \binom{\alpha}{j}$. Although these coefficients can be calculated using the Gamma function $\Gamma(\cdot)$, this approach is inefficient and highly computationally demanding for large j . In our implementation, we substitute it with the well-known recurrence relationship to efficiently precompute the weights [26, 42]:

$$\omega_0^{(\text{GL})} = 1, \quad \omega_j^{(\text{GL})} = \left(1 - \frac{\alpha + 1}{j}\right) \omega_{j-1}^{(\text{GL})}, \quad j \geq 1. \quad (3.8)$$

This recursive formulation reduces the complexity of calculating weights to $O(1)$ per step, ensuring high efficiency during the simulation.

Algorithm 3.1 Grünwald–Letnikov Fractional Derivative Implementation

Require: $f, w, \alpha, \Delta t$

Ensure: $D^\alpha f$

```

1:  $N \leftarrow \text{length}(f)$ 
2:  $C \leftarrow 1.0/(\Delta t^\alpha)$  ▷ Precomputed scaling factor
3: for  $i \leftarrow 0$  to  $N$  do ▷ Loop over time steps
4:    $acc \leftarrow 0.0$ 
5:   for  $j \leftarrow 0$  to  $i$  do ▷ Convolution loop (History)
6:      $acc \leftarrow acc + w[j] \cdot f[i - j]$ 
7:   end for
8:    $D^\alpha f[i] \leftarrow acc \cdot C$ 
9: end for
```

3.2.2. Caputo Discretization

For the Caputo derivative, a widely used FD approach is based on approximating the integral formulation by dividing it into intervals $[t_k, t_{k+1}]$. Within each interval, we approximate the derivative $f'(\tau)$ using a forward difference approximation (assuming f is piecewise linear) [27]:

$$f'(\tau) \approx \frac{f_{k+1} - f_k}{\Delta t}, \quad \tau \in [t_k, t_{k+1}]. \quad (3.9)$$

Substituting this into Equation (3.2) yields the following form:

$${}^C D_t^\alpha f_n \approx \frac{1}{\Gamma(1-\alpha)} \sum_{k=0}^{n-1} \frac{f_{k+1} - f_k}{\Delta t} \int_{t_k}^{t_{k+1}} (t_n - \tau)^{-\alpha} d\tau. \quad (3.10)$$

Finally, by evaluating the exact integral of the kernel $(t_n - \tau)^{-\alpha}$:

$$\begin{aligned} \int_{t_k}^{t_{k+1}} (t_n - \tau)^{-\alpha} d\tau &= \left[-\frac{(t_n - \tau)^{1-\alpha}}{1-\alpha} \right]_{t_k}^{t_{k+1}} = \\ &= \frac{\Delta t^{1-\alpha}}{1-\alpha} [(n-k)^{1-\alpha} - (n-k-1)^{1-\alpha}], \end{aligned} \quad (3.11)$$

and replacing for simplicity $j = n - k - 1$, we arrive at the discrete form:

$${}^C D_t^\alpha f_n \approx \frac{\Delta t^{-\alpha}}{\Gamma(2-\alpha)} \sum_{j=0}^{n-1} \omega_j^{(C)} (f_{n-j} - f_{n-j-1}), \quad (3.12)$$

where the coefficients $\omega_j^{(C)}$ are given by:

$$\omega_j^{(C)} = (j+1)^{1-\alpha} - j^{1-\alpha}. \quad (3.13)$$

Algorithm 3.2 Caputo Fractional Derivative Implementation

Require: $f, w, \alpha, \Delta t$

Ensure: $D^\alpha f$

```

1:  $N \leftarrow \text{length}(f)$ 
2:  $C \leftarrow (\Delta t^{-\alpha})/\Gamma(2-\alpha)$ 
3:  $D^\alpha f[0] \leftarrow C \cdot w[0] \cdot f[0]$  ▷ Initial condition handling
4: for  $i \leftarrow 1$  to  $N$  do
5:    $acc \leftarrow 0.0$ 
6:   for  $j \leftarrow 0$  to  $i-1$  do ▷ Summing differences
7:      $acc \leftarrow acc + w[j] \cdot (f[i-j] - f[i-j-1])$ 
8:   end for
9:    $acc \leftarrow acc + w[i] \cdot f[0]$  ▷ Boundary term correction
10:   $D^\alpha f[i] \leftarrow C \cdot acc$ 
11: end for
```

3.2.3. Riemann–Liouville Discretization

Unlike the GL definition, the Riemann–Liouville derivative involves a two-step process: an integration followed by differentiation. To derive its discrete form, we first approximate the inner fractional integral and subsequently the outer derivative $\frac{d}{dt}$.

For the first point, we need to evaluate it for any time point t_k . To do this, the integral is split into a summation of integrals over each time interval $[t_k, t_{k+1}]$:

$$\int_{t_0}^t \frac{f(\tau)}{(t - \tau)^\alpha} d\tau \approx \sum_{k=0}^{n-1} \int_{t_k}^{t_{k+1}} (t_n - \tau)^{-\alpha} f(\tau) d\tau. \quad (3.14)$$

Assuming that $f(\tau)$ is piecewise constant on each interval (i.e., $f(\tau) \approx f(t_{k+1})$ for $\tau \in [t_k, t_{k+1}]$), we can extract f and evaluate the integral as seen in Equation (3.11). Then, by applying a backward finite difference scheme to the time derivative and performing algebraic reordering, considering for simplicity $n - k = j$, the RL derivative takes the following form[27]:

$${}_{t_0}^{\text{RL}} D_t^\alpha f_n \approx \frac{\Delta t^{-\alpha}}{\Gamma(2 - \alpha)} \left(f(t_n) + \sum_{j=1}^n \omega_j^{(\text{RL})} f_{n-j} \right). \quad (3.15)$$

In this specific formulation, the weights $\omega_j^{(\text{RL})}$ are given by:

$$\omega_0^{(\text{RL})} = 1, \quad \omega_j^{(\text{RL})} = (j + 1)^{1-\alpha} - 2j^{1-\alpha} + (j - 1)^{1-\alpha}. \quad (3.16)$$

Algorithm 3.3 Riemann–Liouville Fractional Derivative Implementation

Require: $f, w, \alpha, \Delta t$

Ensure: $D^\alpha f$

```

1:  $N \leftarrow \text{length}(f)$ 
2:  $C \leftarrow (\Delta t^{-\alpha}) / \Gamma(2.0 - \alpha)$ 
3:  $D^\alpha f[0] \leftarrow C \cdot w[0] \cdot f[0]$ 
4: for  $i \leftarrow 1$  to  $N$  do
5:    $acc \leftarrow 0.0$ 
6:   for  $j \leftarrow 0$  to  $i$  do ▷ Full history convolution
7:      $acc \leftarrow acc + w[j] \cdot f[i - j]$ 
8:   end for
9:    $D^\alpha f[i] \leftarrow C \cdot acc$ 
10: end for
```

3.2.4. Parallelization

The computation of the fractional derivative is well suited for shared-memory parallelization, since when the entire signal is known a priori, the evaluation of the derivative at each discrete time index i is independent from the others, once the input signal and the convolution weights are fixed. This property allows the outer loop over the time indices to be parallelized without introducing data races or synchronization overhead. The implementation exploits the native parallel programming model provided by Julia, combining multi-threading, SIMD vectorization, and careful memory management strategies:

- **Multi-Threading(@thread :dynamic)**: The outer loop over the time index i is parallelized using Julia's `@threads` macro. Each iteration computes a single entry of the fractional derivative vector and writes to a distinct memory location, ensuring thread safety. The `:dynamic` scheduling policy is employed to improve load balancing, as the computational cost of each iteration grows linearly with i . Threads that complete their assigned work earlier are dynamically reassigned new iterations, minimizing idle time and improving overall scalability.
- **Vectorization(@simd)**: Inside each thread, the inner loop performing the convolution sum is annotated with the `@simd` (Single Instruction Multiple Data) macro. This enables the compiler to generate vectorized instructions, allowing multiple floating-point operations to be executed per clock cycle. Since the loop has no loop-carried dependencies and performs a reduction on a local accumulator, it is well suited for SIMD execution.
- **Bounds-check elimination (@inbounds)**: The `@inbounds` macro is used to disable array bounds checking in performance-critical sections. As the loop limits are statically determined and verified to be safe, this optimization removes unnecessary runtime checks and reduces instruction overhead.
- **Reuse of memory**: The output array storing the fractional derivative is pre-allocated and reused across function calls, avoiding repeated memory allocations inside the parallel loops. This approach reduces pressure on the memory allocator and garbage collector, leading to more predictable performance and improved cache efficiency.

Despite the differences in their mathematical definitions, all three operators share a common algorithmic structure, characterized by a history-dependent accumulation process whose computational cost grows with the length of the time domain. Algorithm 3.4 presents a generalized formulation of this approach, expressing them in a parallel form

and exploiting the independence of the derivative evaluations at distinct time indices.

Algorithm 3.4 Parallel Fractional Derivative Implementation (GL, Caputo, RL)

Require: $f, w, \alpha, \Delta t$

Ensure: $D^\alpha f$

```

1: // Compute  $N, C$  and  $D^\alpha f[0]$ 
2: for all  $i \leftarrow 0$  to  $N$  in parallel do
3:    $acc \leftarrow 0.0$ 
4:   for  $j \leftarrow 0$  to  $i$  (SIMD) do
5:     // Compute  $acc$ 
6:   end for
7:    $D^\alpha f[i] \leftarrow C \cdot acc$ 
8: end for

```

3.3. Short Memory Principle

As we addressed earlier, the non-locality property of fractional derivatives drastically increases the computational cost of the problem in relation to its size. To address this issue, we applied the Short Memory Principle [27, 43]. This method is based on the decreasing contribution in the earlier steps, so we calculate an L parameter defined as:

$$L \geq \left(\left| \frac{M}{\epsilon_0 \Gamma(1 - \alpha)} \right| \right)^{\frac{1}{\alpha}}, \quad (3.17)$$

where α is the order of the derivative, ϵ_0 is the tolerance, that is, the value such that the error we want to achieve is less than it, and $M > 0$ is a value greater than $|f(t)|$ in the entire domain. The calculations used to derive Equation (3.17) are shown in the Appendix A.

This parameter defines how many of the previous steps we need to take into account before their contribution to the final result goes below the set tolerance. The objective is to reduce computational overhead, accepting a marginal loss in precision, so we respect the non-locality property only up to a certain L parameter.

3.3.1. Correction term

While the Short Memory (SM) principle significantly boosts performance, simply truncating the history introduces a "step" error. To mitigate this without increasing the $O(L)$ complexity, we implement a first-order correction term that approximates the tail of the GL summation, that is, the "forgotten" interval $[0, i - L]$.

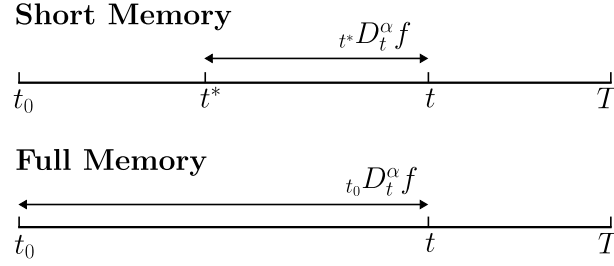


Figure 3.1: Visual comparison between Short (top) and Full (bottom) Memory approaches. In the Short Memory principle, the lower bound of the fractional derivative is shifted from the initial time t_0 to a moving truncated bound $t^* = t - L$, accounting only for the recent history of length L . In the Full Memory approach, the entire history from t_0 to t is considered.

The idea is to calculate this approximation using the value at the edge of the memory window:

1. At first, we compute two constants, k_0 and k_1 , which represent the coefficients of the fractional integral of a constant and a linear function, respectively:

$$k_0 = \frac{1}{\Gamma(1 - \alpha)}, \quad k_1 = \frac{-\alpha}{\Gamma(2 - \alpha)},$$

where α is the fractional order.

2. To find the contribution of the specific "forgotten" period, we calculate the difference between the integral from the start of the signal (i) to the current point and the integral over the active window (L). In Algorithm 3.5, this is represented by:

- m_0 : Approximates the integral of the kernel over the truncated past.
- m_1 : Accounts for the linear trend (slope) of the signal in that same period.

3. At each step $i > L$, the algorithm identifies the signal value y_0 at the memory boundary and then calculates a local slope y_1 using a backward difference:

$$y_0 = \text{data}[i - L], \quad y_1 = \text{data}[i - L] - \text{data}[i - L - 1],$$

with the second one representing the "velocity" of the signal as it exits the memory window.

4. Finally, the correction is synthesized by combining these terms:

$$\text{Correction} = y_0 \cdot m_0 + y_1 \cdot (L \cdot m_0 - m_1).$$

Adding this correction to the accumulated sum (acc) effectively reconstructs the forgotten history with a linear approximation, reducing truncation artifacts at no extra computational cost.

Algorithm 3.5 GL Fractional Derivative with First-Order SM Correction

Require: data, α , Δt , weights

Ensure: deriv

```

1:  $n \leftarrow \text{length}(\text{data})$ 
2: // Compute L
3:  $C \leftarrow 1/\Delta t^\alpha$ 
4:  $k_0 \leftarrow 1/\Gamma(1-\alpha)$ ,  $k_1 \leftarrow -\alpha/\Gamma(2-\alpha)$ 
5:  $\text{off}_0 \leftarrow k_0 \cdot L^{-\alpha}$ 
6:  $\text{off}_1 \leftarrow k_1 \cdot L^{1-\alpha}$ 
7:  $\text{limit} \leftarrow \min(n, L)$ 
8: for  $i \leftarrow 0$  to  $\text{limit}$  do
9:    $\text{acc} \leftarrow \sum_{j=0}^i \text{weights}[j] \cdot \text{data}[i-j]$ 
10:   $\text{deriv}[i] \leftarrow \text{acc} \cdot C$ 
11: end for
12: if  $L < n$  then
13:   for  $i \leftarrow L+1$  to  $n$  do
14:    for  $j \leftarrow 0$  to  $L$  do
15:       $\text{acc} \leftarrow \sum_{j=0}^L \text{weights}[j] \cdot \text{data}[i-j]$ 
16:    end for
17:     $m_0 \leftarrow k_0 \cdot i^{-\alpha} - \text{off}_0$ 
18:     $m_1 \leftarrow k_1 \cdot i^{1-\alpha} - \text{off}_1$ 
19:     $y_0 \leftarrow \text{data}[i-L]$ 
20:    if  $i = L+1$  then
21:       $y_1 \leftarrow 0.0$ 
22:    else
23:       $y_1 \leftarrow y_0 - \text{data}[i-L-2]$  ▷ Linear slope at memory edge
24:    end if
25:     $\text{correction} \leftarrow y_0 \cdot m_0 + y_1 \cdot (L \cdot m_0 - m_1)$ 
26:     $\text{deriv}[i] \leftarrow (\text{acc} + \text{correction}) \cdot C$ 
27:  end for
28: end if

```

3.4. FFT-based acceleration

The direct evaluation of the three fractional derivatives we saw relies on a causal convolution between the input signal and a sequence of precomputed weights. When implemented explicitly, this approach exhibits a quadratic computational complexity $\mathcal{O}(N^2)$, which quickly becomes prohibitive for large time horizons, regardless of the degree of parallelization applied.

To address this limitation, the convolution can be reformulated using the convolution theorem and computed efficiently in the frequency domain via the Fast Fourier Transform (FFT) [26, 27]. By exploiting the equivalence between convolution in the time domain and pointwise multiplication in the frequency domain, the overall computational complexity is reduced to $\mathcal{O}(N \log N)$.

In this work, the derivative is computed using an FFT-based filtering approach provided by the `fftfilt` function from `DSP.jl`[44], where the discrete convolution between the signal and the fractional weights is evaluated through FFTs. This method not only improves scalability for large problem sizes, but also yields significant performance gains in the serial case, making it particularly effective when the number of available threads is limited or when memory bandwidth becomes the dominant bottleneck.

After computing the convolution in the frequency domain, the result is scaled by the appropriate power of the time step in order to obtain the final fractional derivative approximation. The FFT-based formulation preserves the direct convolution method while substantially reducing the computational cost.

Algorithm 3.6 FFT-based Fractional Derivative

Require: $f, w, \alpha, \Delta t$

Ensure: $D^\alpha f$

- 1: $N \leftarrow \text{length}(f)$
 - 2: *//* Compute discrete convolution $c \leftarrow f * w$ using FFT-based filtering
 - 3: $C \leftarrow (\Delta t)^{-\alpha}$
 - 4: **for** $i \leftarrow 0$ to N **do**
 - 5: $D^\alpha f[i] \leftarrow C \cdot c[i]$
 - 6: **end for**
-

The Overlap-Save Algorithm To efficiently compute the convolution, `fftfilt` applies a FIR (Finite Impulse Response) filter using an FFT-based overlap-save algorithm, which consists of the following steps [45]:

1. The input is split into overlapping blocks, the length of which depends on the size of the signal and the filter.
2. Discrete Fourier Transforms of each block are multiplied with the FIR filter coefficients (i.e., the Grünwald–Letnikov weight coefficients in the frequency domain) and then converted back to the time domain (this step is also called circular convolution).
3. the first part of each output block is discarded to prevent artifacts, while the remaining elements are concatenated to form the true output.

3.5. Results and Discussion

To verify the accuracy and computational efficiency of the implemented algorithms, we selected different input functions $f(t)$ to cover distinct dynamic behaviors typically encountered in rheology. For each case, the exact analytical fractional derivative is known [26], allowing for precise error quantification.

- **Unit Step Function (Heaviside):** Representing an instantaneous loading condition:

$$f(t) = 1, \quad t \geq 0. \quad (3.18)$$

Its Riemann-Liouville fractional derivative is given by:

$$D^\alpha f(t) = \frac{t^{-\alpha}}{\Gamma(1-\alpha)}. \quad (3.19)$$

This function is particularly challenging due to the singularity at $t \rightarrow 0$, serving as a stress-test for the singularity handling of the discretization schemes.

- **Power-Law Function:** Representing a smooth polynomial growth:

$$f(t) = t^k, \quad t \geq 0. \quad (3.20)$$

The analytical derivative is:

$$D^\alpha f(t) = \frac{\Gamma(k+1)}{\Gamma(k+1-\alpha)} t^{k-\alpha}. \quad (3.21)$$

- **Ramp Function:** Representing a linear growth condition starting from the origin:

$$f(t) = m \cdot t, \quad t \geq 0, \quad (3.22)$$

where m is the gradient of the ramp. Its analytical fractional derivative is expressed as:

$$D^\alpha f(t) = \frac{m}{\Gamma(2-\alpha)} t^{1-\alpha}. \quad (3.23)$$

Unlike the step function, this formulation provides a continuous profile at $t = 0$, reducing the severity of the initial singularity for $0 < \alpha < 1$.

3.5.1. Numerical Validation and Error Analysis

In this section, we validate the accuracy of the implemented solvers by analyzing the relative error evolution over the simulation time T , computed at each point as:

$$\epsilon_r = \frac{|\bar{y} - y|}{|y|}, \quad (3.24)$$

with y and $\bar{y}$ being respectively the exact value and the computed one at each time step.

Baseline Accuracy. First, we establish the baseline accuracy using the baseline implementation. Figure 3.2a illustrates the squared errors for the Grünwald-Letnikov, Caputo and Riemann-Liouville definitions applied to the Power-Law test case ($f(t) = t^k$). As

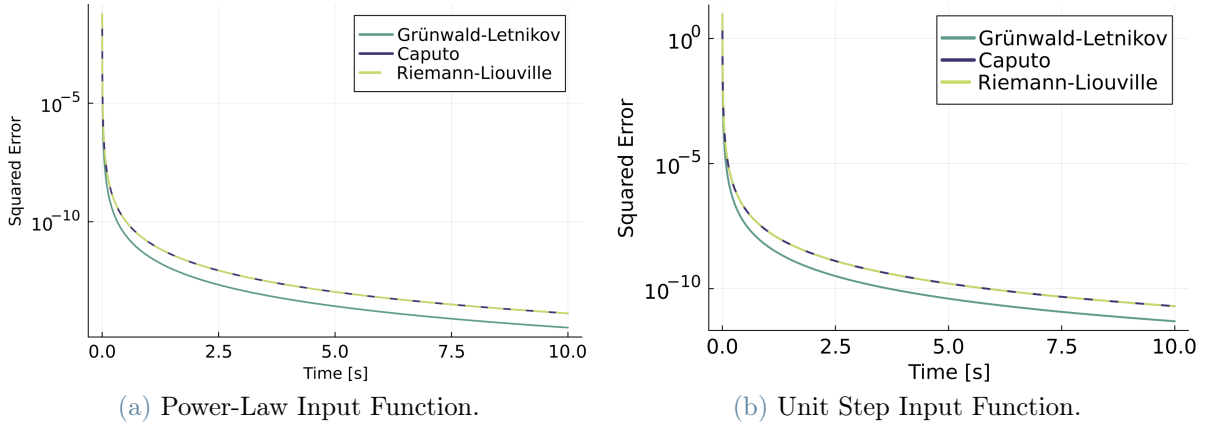


Figure 3.2: Plot of the squared error over time for both Power-Law and Unit Step input functions using the baseline implementations, a constant time step $\Delta t = 0.001$, order $\alpha = 0.5$ and $k = 0.5$ for the Power-Law.

observed, all three methods exhibit consistent behavior, with the GL being slightly more accurate than the others. In each case, the error is generally higher in the initial time steps due to the transient nature of the fractional operator and then settles to a stable value as the solution progresses. This confirms that the finite difference discretizations are correctly implemented and stable for continuous input functions.

Regarding discontinuous inputs, Figure 3.2b compares the analytical solution of the Unit Step function with its numerical approximation. As the previous case, the algorithm rapidly recovers and the numerical solution aligns perfectly with the analytical solution for $t > 0$.

Short Memory Principle Accuracy As previously discussed, the application of the Short Memory Principle introduces an approximation error in exchange for significant computational gains. To quantify this error, Figure 3.3 illustrates how the error starts to grow once the simulation time exceeds L , computed using Equation (3.17). The plot also demonstrates the efficacy of the correction term described in Section 3.3.1 (hereinafter referred to as High Precision SM). For this case, we consider only the Grünwald-Letnikov fractional derivative, since preliminary results proved this method to offer better performance than Caputo and Riemann-Liouville, as can be seen in Figure 3.4.

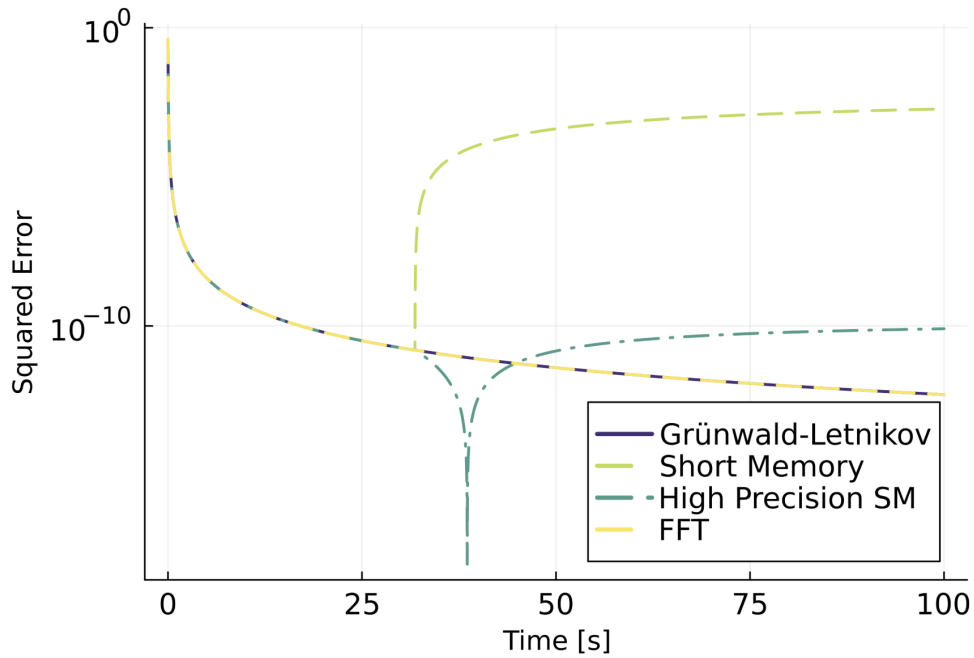


Figure 3.3: Plot of the squared error over time of the Grünwald-Letnikov fractional derivative using each implementation of it and Unit Step input function, a constant time step $\Delta t = 0.01$, order $\alpha = 0.5$ and $\epsilon_0 = 0.1$.

Fast Fourier Transform Accuracy Unlike the Short Memory Principle, the FFT-based acceleration preserves the exact mathematical formulation of the direct causal convolution. Consequently, the accuracy of the FFT implementation is fundamentally identical to the baseline approach discussed previously. The squared errors for the fractional derivatives computed via the frequency domain perfectly mirror those presented in Figure 3.2, as can be seen in Figure 3.3. This confirms that the $\mathcal{O}(N \log N)$ FFT-based algorithm successfully achieves significant computational scaling without sacrificing the numerical fidelity of the fractional operator approximation.

3.5.2. Performance Analysis

Once the accuracy of our method has been validated, we now turn our attention to the computational efficiency of the proposed solvers. Here we analyze the performance gains achieved through two distinct strategies: *hardware parallelization* and *algorithmic optimization* (applying the Short Memory Principle and the FFT).

Baseline Implementations First, we evaluate the impact of multi-threading on the standard algorithms, which exhibit $\mathcal{O}(N^2)$ complexity. Figure 3.4 compares the execution times of the serial implementations (1 thread) with the parallel implementation utilizing 32 threads. From this point onwards, all simulations were carried out using the specifications already described in Section 2.6.

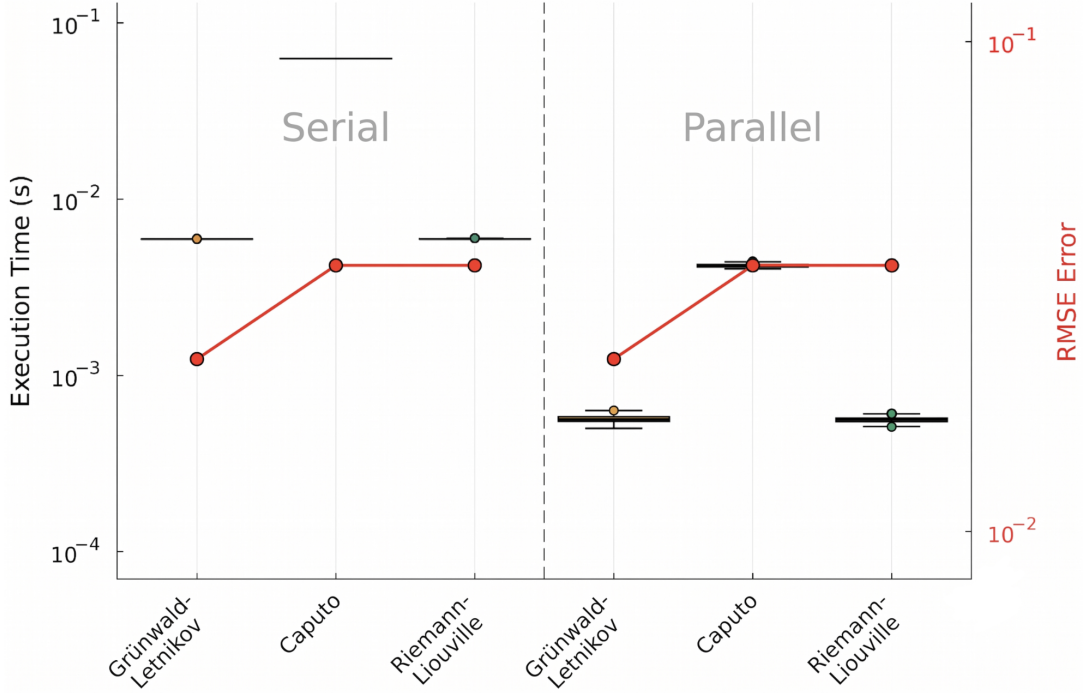


Figure 3.4: Comparison between baseline implementations (left: serial version; right: parallel version with 32 threads) using Unit Step input function, a constant time step $\Delta t = 0.001$ ($N = 10001$ steps, $t \in [0, 10]$) and order $\alpha = 0.5$. Execution time is handled by BenchmarkTools.jl [46].

The results reveal a significant difference between the Caputo definition and the GL/RL approaches, due to the higher arithmetic intensity of the first one. Specifically, in the serial execution, the Caputo method requires approximately $6 \cdot 10^{-2}$ s, which is roughly one order of magnitude slower than the GL and RL definitions (both around $6 \cdot 10^{-3}$ s). Overall, all methods show a substantial speedup, due to the absence of race conditions.

By exploiting 32 threads, the execution times drop significantly: GL and RL scale down to approximately $5.5 \cdot 10^{-4}$ s, while Caputo drops to around $4 \cdot 10^{-3}$ s. These results are better discussed in Subsection 3.5.3.

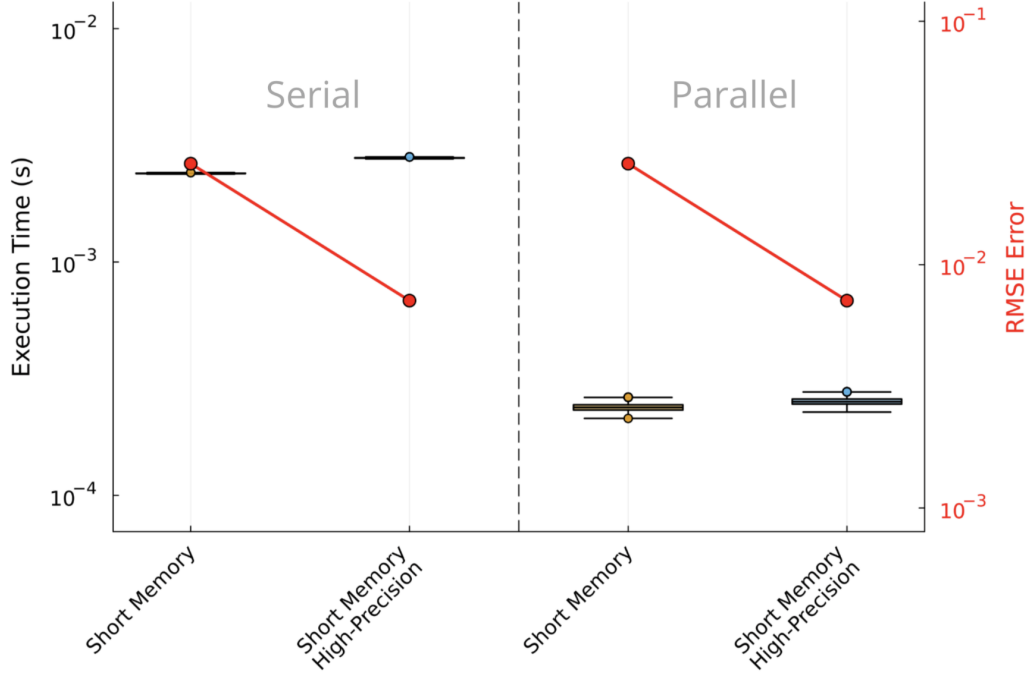


Figure 3.5: Comparison between Grünwald-Letnikov Short Memory implementations (left: serial version; right: parallel version with 32 thread) using Unit Step input function, a constant time step $\Delta t = 0.01$ ($N = 10001$ steps, $t \in [0, 100]$), order $\alpha = 0.5$ and $\epsilon_0 = 0.1$. Execution time is handled by BenchmarkTools.jl [46].

Short Memory and FFT Implementations We subsequently applied the same parallelization strategy to the Short Memory implementation of the GL definition, since its baseline implementation proved to be faster with respect to the Caputo one and more accurate than both Caputo and RL, as shown in Figure 3.4. Figure 3.5 shows once again how parallelization significantly reduces computational time, while Figure 3.6 provides a direct comparison between each approach (also including the one implementing the Fast Fourier Transform).

Quantitatively, the parallelized full-memory baseline is approximately $5.5 \cdot 10^{-4}$ s. The Short Memory and Short Memory High-Precision algorithms reduce this time to $\sim 2.0 \cdot 10^{-4}$ s, cutting the computational effort by roughly half. As expected, the FFT implementation provides the most efficient solution, since it achieves an execution time of approximately $2.0 \cdot 10^{-4}$ s.

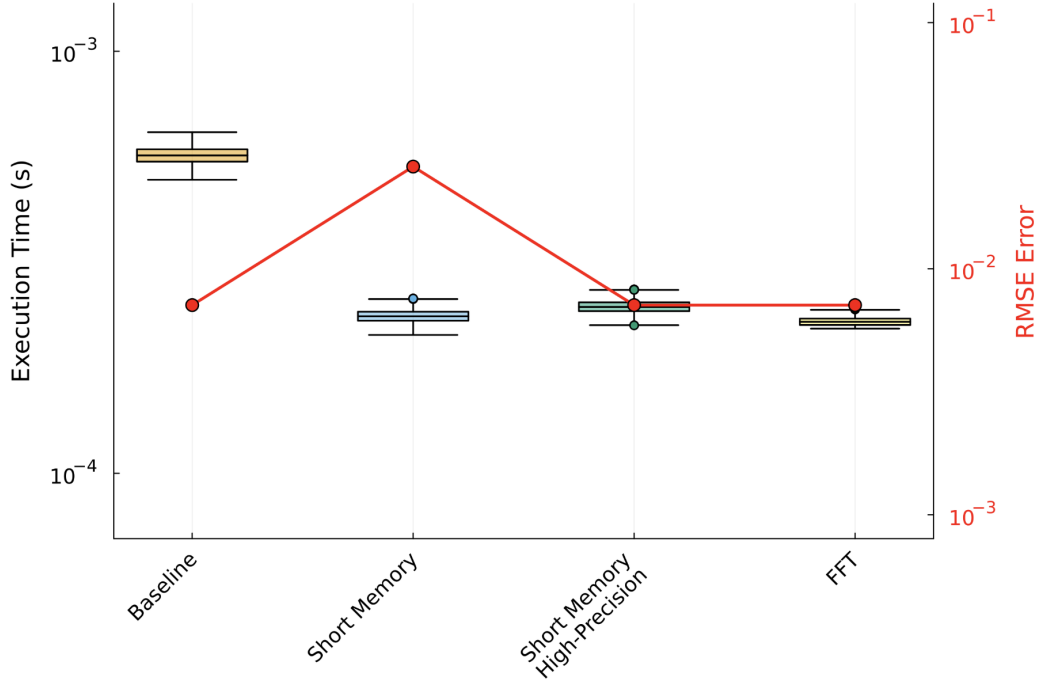


Figure 3.6: Comparison between each Grünwald-Letnikov implementation (32 threads) using Step input function, a constant time step $\Delta t = 0.01$ ($N = 10001$ steps, $t \in [0, 100]$), order $\alpha = 0.5$ and $\epsilon_0 = 0.1$ for the Short Memory approaches. Execution time is handled by BenchmarkTools.jl [46].

3.5.3. Scalability Analysis

To fully understand the computational dynamics of the proposed solvers, it is important to analyze their scaling behavior with respect to both the problem size (algorithmic scaling) and the number of processing units (hardware scaling). Performance is traditionally quantified through two complementary metrics: strong scaling and weak scaling.

Amdahl's Law and Scaling Models Strong scaling analyzes how the execution time varies with an increasing number of processing units for a fixed total problem size. This behavior is governed by Amdahl's law, which states that the speedup S achieved by parallelizing a program is limited by the fraction of its serial time. Considering s the serial component of the algorithm (where $0 \leq s \leq 1$) and $p = 1 - s$ denoting the parallelizable fraction, the theoretical speedup using N threads is formulated as:

$$S(N) = \frac{T_1}{T_N} = \frac{1}{s + \frac{1-s}{N}} \quad (3.25)$$

where T_1 is the sequential execution time and T_N is the parallel execution time. As N approaches infinity, the speedup asymptotically converges to $1/s$. Consequently, even a minimal serial fraction (e.g., due to memory allocation, I/O operations, or synchronization barriers) establishes an immutable upper bound on hardware efficiency, eventually leading to a performance plateau.

On the other hand, weak scaling evaluates the system's ability to handle larger workloads by proportionally increasing both the problem size and the thread count, ideally maintaining execution time as constant.

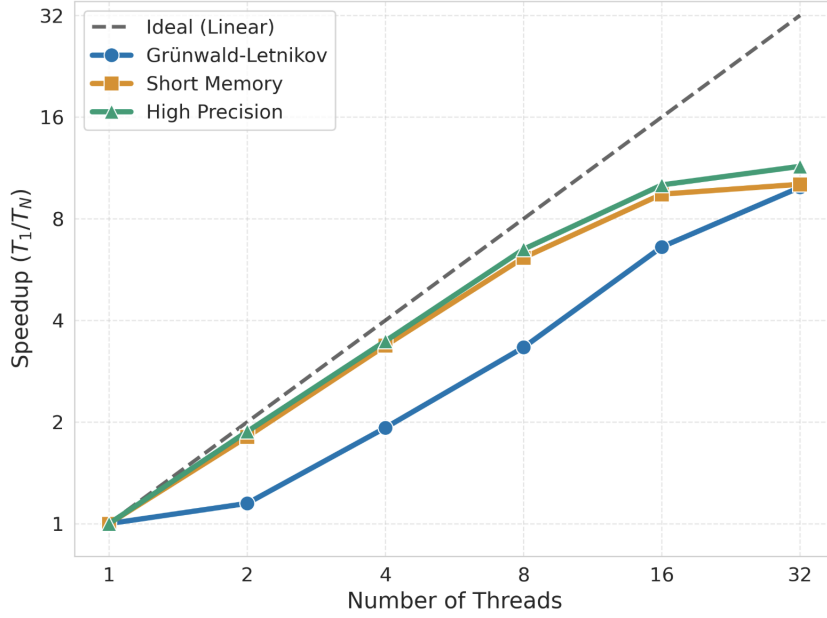
Experimental Scalability Results From a hardware perspective, we evaluated both the strong and weak scaling behaviors of the parallel implementations by gradually increasing the number of threads up to 32. Note that the FFT-based approach is excluded from this specific analysis, as its current implementation relies on a strictly serial execution.

For strong scaling (Figure 3.7a), the Short Memory implementations demonstrate robust performance gains up to 16 threads, closely following the ideal linear trend. Beyond this point, the speedup begins to plateau, reflecting the impact of the Amdahl's law. The standard Grünwald-Letnikov implementation, in contrast, exhibits a noticeably lower parallel efficiency from the outset.

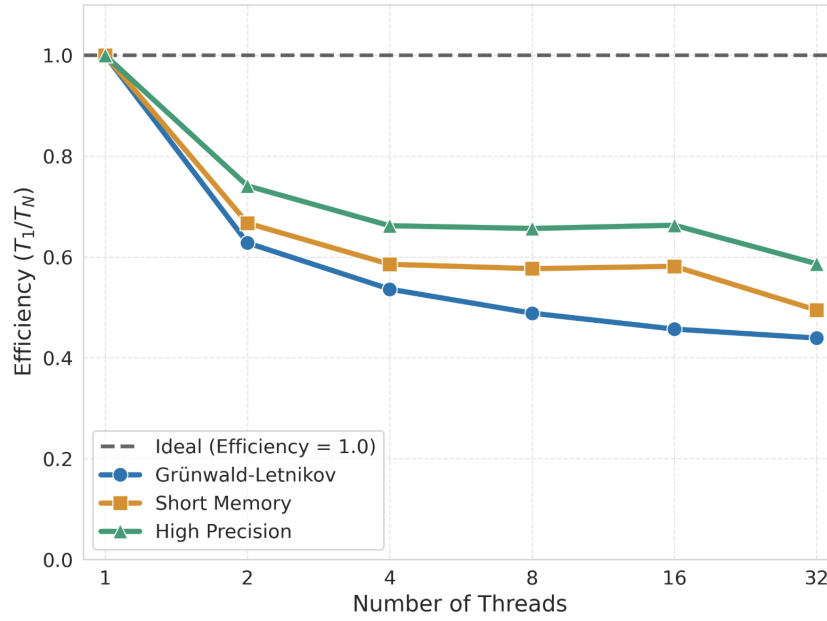
Regarding the weak scaling results, Figure 3.7b illustrates the parallel efficiency, i.e. the ratio of the execution time for a single thread T_1 to that for N threads T_N . While an ideal weak scaling scenario would maintain a constant efficiency of 1.0, the results show an initial drop that gradually stabilizes, with the High Precision SM method retaining the best efficiency.

3.6. NumFracDiff Library

To consolidate the numerical methods discussed throughout this chapter, we developed **NumFracDiff** [47], an open-source library dedicated to the efficient evaluation of fractional derivatives. The primary objective of this package is to provide a highly optimized and scalable tool that extends beyond the specific scope of this thesis. To this end, the library is designed to be easily integrated into different computational frameworks and external software ecosystems, such as RHEOS.



(a) Strong Scaling.



(b) Weak Scaling.

Figure 3.7: Scalings of the Grünwald-Letnikov implementations using $\Delta t = 0.01$, order $\alpha = 0.5$ and $\epsilon_0 = 0.1$ for the Short Memory approaches. Execution time is handled by BenchmarkTools.jl [46].



Figure 3.8: Logo of the NumFracDiff library

3.6.1. Architecture

The architecture of the library is designed around the principles of High-Performance Computing and memory management in Julia. To minimize Garbage Collection (GC) overhead and ensure type stability, the core logic is decoupled into two fundamental structures: the `Problem` and the `Workspace`.

The Problem Structure. The `NumDiffProblem` is a mutable data structure that encapsulates the mathematical and numerical definition of the differentiation task, storing all the necessary parameters required to configure the fractional operator.

The structure contains the following key parameters:

- `dt`: The constant sampling interval (Δt).
- `order`: The fractional order (α).
- `n`: The expected length of the input data array.
- `method`: An enumerable type (`DiffMethod`) specifying the chosen discretization scheme (e.g., Grünwald-Letnikov, Caputo, RL, or FFT-based).
- `_L`: An internal integer parameter defining the memory window length when the Short Memory principle is applied.

The Workspace Structure. The `NumDiffWorkspace` is an immutable data structure responsible for holding pre-allocated arrays. Because the structure itself does not need to change its internal references, we defined it as immutable, allowing the Julia compiler to optimize its usage significantly. It consists of:

- `weights`: A vector of the convolution coefficients (ω_j).

- **deriv:** The pre-allocated vector that will store the final output of the fractional derivative.

Execution. Once the numerical problem is defined and the workspace is allocated, the actual calculation is triggered by invoking the `compute` function, which populates the `deriv` array by executing the algorithm defined by the chosen method `method`. The different methods are implemented in the same way as described before in this chapter.

Usage example The standard execution pipeline consists of three main stages:

- **Problem Configuration:** The simulation parameters, such as the continuous discrete time step Δt , the fractional order of differentiation α , and the array length, are bound to a central execution structure called *NumDiffProblem*. At this stage, the numerical approximation method is explicitly chosen, for example the Grünwald-Letnikov (*GL()*) method.
- **Workspace Allocation:** To avoid runtime performance degradation caused by garbage collection or frequent heap allocations, the library relies on an explicit memory management structure called *NumDiffWorkspace*. This step pre-allocates the necessary internal auxiliary vectors and caches required by the underlying algorithms.
- **In-Place Computation:** The numerical derivative is evaluated by invoking the mutating *compute!* function. This routine takes the target input signal array and updates the pre-allocated workspace in-place without generating temporary arrays.

3.7. Conclusions

The aim of this chapter was to find different suitable fractional derivative methods for the differential approach. The numerical validation confirms that the baseline Grünwald-Letnikov, Caputo, and Riemann-Liouville solvers accurately reproduce the analytical fractional derivatives for both continuous (Power-Law) and discontinuous (Unit Step) inputs, with the error stabilizing after the initial transient. From a computational standpoint, GL and RL consistently outperform Caputo, and all three benefit substantially from multi-threaded parallelization: the 32-thread implementation yields a speedup of approximately $11\times$ for GL/RL and $15\times$ for Caputo.

The Short Memory Principle further reduces the computational cost by truncating the convolution history, at the price of a controlled accuracy loss beyond the truncation

horizon L ; the High-Precision correction term effectively mitigates this error without sacrificing the performance gain.

Finally, the FFT-based implementation offers the best overall trade-off, matching the numerical accuracy of the full-memory baseline while achieving the lowest execution times among the tested methods. Table 3.1 summarizes the execution times and accuracy behavior of all tested implementations.

Method	Serial Time [s]	Parallel Time [s]	RMSE
Grünwald-Letnikov	$\sim 6 \times 10^{-3}$	$\sim 5.5 \times 10^{-4}$	$\sim 7 \times 10^{-3}$
Caputo	$\sim 6 \times 10^{-2}$	$\sim 4 \times 10^{-3}$	$\sim 1.5 \times 10^{-2}$
Riemann-Liouville	$\sim 6 \times 10^{-3}$	$\sim 5.5 \times 10^{-4}$	$\sim 1.5 \times 10^{-2}$
GL + Short Memory	$\sim 2.5 \times 10^{-3}$	$\sim 2.0 \times 10^{-4}$	$\sim 1.8 \times 10^{-2}$
GL + SM High-Precision	$\sim 2.5 \times 10^{-3}$	$\sim 2.0 \times 10^{-4}$	$\sim 7 \times 10^{-3}$
GL + FFT	$\sim 2.0 \times 10^{-4}$	N/A (serial-only)	$\sim 7 \times 10^{-3}$

Table 3.1: Summary of execution times (serial and 32-thread parallel) and qualitative error behaviour for the tested fractional derivative implementations, evaluated on the Unit Step input function with $\alpha = 0.5$ and a constant time step $\Delta t = 0.01$ ($N = 10001$ steps, $t \in [0, 100]$)

4 | Fitting

This chapter presents the fitting framework developed to estimate the parameters of fractional viscoelastic models from experimental data. Building on the mathematical foundations introduced in the previous chapter, this section analyzes the importance of parameter identification in rheological modeling and then outlines the computational approach adopted in this study. The chapter also summarizes the data structures used in the implementation and the main results obtained with the proposed framework.

4.1. Introduction

The objective of the fitting procedure considered in this work is the estimation of the parameters associated with a selected viscoelastic model, specifically the material parameters. These parameters may vary depending on the constitutive formulation associated with the chosen model and determine the dynamic response predicted by the model, as discussed in 2.2. Their identification is formulated as a non-linear optimization problem of the form:

$$\min_{\mathbf{x} \in \mathbb{R}^n} f(\mathbf{x}),$$

where f is the cost function, which will be discussed later, and $\mathbf{x}$ are the n optimization parameters to estimate.

The input used for this purpose consists of discrete time series of stress σ and strain ε sampled from measurements using a constant time step Δt , while the optimizer we adopted is provided by *NLOpt* [48]. This library in particular allowed us to select between different optimization strategies, and each algorithm can be classified following two independent criteria: the scope of search and the use of derivative information.

The former criterion represents what type of minimum we want to find by resolving our optimization problem, and can be divided in:

- **global algorithms**, which aims at finding a solution that minimize the cost function over the entire domain;

- **local algorithms**, that only search for a local minimum of the cost function.

As to the latter criterion, we can divide algorithms in the following way:

- **gradient based algorithms** use the derivative of the cost function to help finding the optimal solution;
- **derivative-free algorithms**, on the contrary, do not rely on the gradient of the cost function, since in some cases this can be difficult to compute.

In our specific case, we decided to exploit a local derivative-free optimizer for two main reasons:

1. by searching only a local minimum, and not a global one, we reduce dramatically the complexity of the problem, since local algorithms are able to easily locate local minimum even in higher dimensions.
2. although gradient-based methods can be highly efficient, especially in high-dimensional settings, their practical use requires an explicit analytical derivation of ∇f within the codebase. Since this would add complexity to the problem, we decided to take advantage of the derivative-free approach proposed by the library.

The two main algorithms used in our code are the SUBPLEX [49] algorithm and the COBYLA [50], for evaluation with constraints, as the former does not accept any constraint.

From a computational perspective, the evaluation of fractional derivatives is the dominant cost in the fitting loop, as it must be repeated many times during the optimization process. The techniques introduced in the previous chapter for the efficient computation of fractional derivatives in serial and in parallel are therefore directly relevant to the performance of the fitting framework. By reducing the cost of model evaluation and leveraging the most suitable optimizers, these methods enable a fast optimization procedure and improve the overall scalability of the parameter estimation process.

The fitting procedure is applied to constitutive models describing viscoelastic behavior in order to identify the corresponding material parameters. Viscoelastic models can be represented using mechanical analogues made of springs, which describe the elastic response, and dashpots, which describe the viscous response. Different configurations of these elements lead to different constitutive laws, as illustrated in 2.2.

As an example, the classical Maxwell model consists of a spring and a dashpot connected

in series. Its behavior is governed by the differential equation (4.1):

$$\sigma + \frac{\eta}{E}\dot{\sigma} = \eta\dot{\varepsilon}, \quad (4.1)$$

where η denotes the viscosity of the dashpot and E the elastic modulus of the spring. In the fitting procedure, these two parameters constitute the unknowns to be estimated through the optimization process, while the stress history $\sigma(t)$ and $\varepsilon(t)$ are provided experimentally.

4.2. Cost Function

One of the primary objectives of RHEOS is to fit a physical model parameters to a given couple of strain and stress experimental data. This requires defining a quantitative measure of how well a given set of parameters reproduces the observed material response. By defining a cost function where the returned result is the error, we can calculate the penalty for inaccuracies in the model and set the goal of the optimization process as a minimization problem, effectively tuning the parameters. The cost function we used, shown in (4.2), takes as input the strain and stress vectors, their derivatives, a set of parameters and a constitutive equation, i.e. the correlation between stress and strain, that represents the model used.

Any constitutive equation can be written in the form of

$$\sum_i a_i \frac{d^{\alpha_i} \sigma}{dt^{\alpha_i}} = \sum_j b_j \frac{d^{\beta_j} \varepsilon}{dt^{\beta_j}}, \quad (4.2)$$

where a_i and b_j are the coefficients for each term, α_i and β_j are the orders of derivation of each term (which can be fractional thus a real value between 0 and 1), and σ and ε are the stress and strain vectors, respectively. Since every variable we need is known, we can compute the vector values for the left and right hand-side of the equation by computing the derivatives of strain and stress values, multiply them by the coefficient for each term in the equation and then add each contribution together. The final cost (4.3) is then calculated as the summation over the squared difference of the lhs and rhs vectors:

$$Cost = \sum_{i=1}^n (LHS_i - RHS_i)^2. \quad (4.3)$$

A pseudo code for the practical implementation of the whole cost function is shown below.

Algorithm 4.1 Objective Function for Fractional Constitutive Models

Require: params, equation, $\Delta t, \varepsilon, \dot{\varepsilon}, \sigma, \dot{\sigma}$, data_struct, SM_flag

Ensure: cost

```

1: coeffs  $\leftarrow$  get_coefficients(equation, params)
2: LHS  $\leftarrow$  0, RHS  $\leftarrow$  0 ▷ Initialize accumulation vectors
3:
4: for each  $(\alpha, c) \in$  coeffs.strain do ▷ Compute Right-Hand Side (Strain)
5:   if  $\alpha = 0$  then
6:     RHS  $\leftarrow$  RHS +  $c \cdot \varepsilon$ 
7:   else if  $\alpha = 1$  then
8:     RHS  $\leftarrow$  RHS +  $c \cdot \dot{\varepsilon}$ 
9:   else ▷ Fractional Order Case
10:     $L \leftarrow$  length( $\varepsilon$ )
11:     $w \leftarrow$  generate_GL_weights( $\alpha, L$ )
12:    frac_deriv  $\leftarrow$  compute_GL_frac_deriv( $\varepsilon, w, \alpha, \Delta t, L$ )
13:    RHS  $\leftarrow$  RHS +  $c \cdot$  frac_deriv
14:   end if
15: end for
16:
17: for each  $(\beta, d) \in$  coeffs.stress do ▷ Compute Left-Hand Side (Stress)
18:   if  $\beta = 0$  then
19:     LHS  $\leftarrow$  LHS +  $d \cdot \sigma$ 
20:   else if  $\beta = 1$  then
21:     LHS  $\leftarrow$  LHS +  $d \cdot \dot{\sigma}$ 
22:   else
23:     $L \leftarrow$  length( $\sigma$ )
24:     $w \leftarrow$  generate_GL_weights( $\beta, L$ )
25:    frac_deriv  $\leftarrow$  compute_GL_frac_deriv( $\sigma, w, \beta, \Delta t, L$ )
26:    LHS  $\leftarrow$  LHS +  $d \cdot$  frac_deriv
27:   end if
28: end for
29:
30: cost  $\leftarrow \sum (\text{LHS} - \text{RHS})^2$ 
31: return cost

```

4.3. Optimization

Once the cost function has been defined, the next step is to solve the associated optimization problem in order to identify the model parameters that best fit the experimental stress–strain data. This task is performed using the *NLopt* [48] library, a high-performance optimization framework that provides a wide range of derivative-free and gradient-based algorithms. In this work, the default algorithm used is the Subplex algorithm [49], an extension of the Nelder–Mead method designed for high-dimensional optimization problems, making it suitable for derivative-free parameter estimation. However when constraints on the parameters are required, the optimisation is handled using the COBYLA (Constrained Optimization BY Linear Approximations) algorithm [50]. To facilitate the use of *NLopt*, a wrapper function to handle the optimization parameters has been implemented, allowing the user to:

- **Set parameter boundaries:** The framework enforces *box constraints* by defining lower and upper boundaries for the parameter vector. This allows to maintain thermodynamic consistency, such as ensuring that elastic moduli E_i and viscosities η_i remain between 0 and 1, without needing to resort to *NLopt* constraints that would limit the algorithms possibilities.
- **Set initial parameters:** Given that the error surfaces of fractional viscoelastic models are often non-convex, the final solution can be sensitive to the starting point. The wrapper allows for the injection of a custom initial guess $\mathbf{p}_{init}$, enabling the user to try different initial configurations.
- **Convergence Criteria (Tolerances):** To allow for deeper research in case of highly variable cost function or to prevent unnecessary computation once a solution has stabilized, two relative tolerance thresholds are implemented:
 1. *Parameter Tolerance* ($xtol_rel$): Stops the search when the relative change in the parameters between iterations falls below a threshold ϵ_x .
 2. *Function Tolerance* ($ftol_rel$): Terminates the process when the relative improvement in the cost function value is less than ϵ_f .
- **Computational Constraints:** To handle large-scale datasets or computationally expensive fractional derivatives, the function implements hard limits on resources. A *wall-clock timeout* prevents the system from hanging in non-convergent scenarios, while a *maximum evaluation ceiling* limits the maximum number of iterations to a fixed number.

4.4. Implementation in RHEOS

This section describes the modifications introduced in RHEOS to integrate the new fitting procedures, focusing on the integration of the fractional and FFT-based methods into the existing codebase.

RHEOS is built around a core structure representing the model, which stores its parameters together with the relaxation and creep moduli, represented as symbolic expressions.

Since the fractional approach operates directly on the constitutive equation, it was necessary to extend the model structure in a way that includes such equation and allows easy access to each of its components. To achieve this, each term was reduced to a unified pattern consisting of a coefficient multiplying the derivative of one of the unknown variables, namely stress or strain, and a second floating-point value representing the order of the derivative of said unknown variable.

```

struct DTerm{T}
    coef::T
    order::T
end

struct DiffEqu{T}
    leftvar::Symbol    #\epsilon
    rightvar::Symbol   #\sigma
    leftde::Set{DTerm{T}}
    rightde::Set{DTerm{T}}
end

```

In this formulation, each term can be represented using two floating-point values. These are stored in a dedicated structure called `DTerm`. A full constitutive equation is then expressed as a collection of such terms, separated into a set representing the left-hand side and another representing the right-hand side.

The fitting function calls in the updated version of RHEOS remain unchanged, leveraging Julia's multiple dispatch capabilities. The only modification is the introduction of an additional parameter, `method`, which can take the values "Convolutional", "Differential" or "FFT" (the original RHEOS approach and our own implementations, respectively), and will determine which part of the code to run, to have the various solving methods all grouped under one function call.

This parameter acts as a dispatcher, selecting the appropriate underlying implementation while keeping the high-level user interface unchanged. In this way, the complexity of the different numerical approaches is fully encapsulated within the internal code, allowing users to interact with the library through a consistent function call as in the original version.

This design improves extensibility by allowing new numerical approaches to be incorporated without modifying the user-facing API or introducing additional function variants. It exploits Julia’s multiple dispatch mechanism to resolve the appropriate backend implementation based on the selected method, ensuring a clean separation between the high-level interface and the underlying computational details. This approach also facilitates future extensions of the library, as additional methods can be integrated with minimal changes to the existing structure.

The fitting procedure can be executed using the following function calls:

```
modelfit(data,model,fittype=Convolution())
modelfit(data,model,fittype=Differential,method=GLThreads())
```

Where *data* contains the time series and either the stress or strain, while *model* is one of the instantiated structs in RHEOS representing a viscoelastic model. To ensure a unified syntax across all solving methods, algorithm-specific configurations are handled via keyword arguments with default values. As a result, certain parameters remain inactive if they are not required by the selected methodology. This consistent interface allows the user to switch seamlessly between different computational backends without altering the core structure of the function call.

4.5. Results and Discussion

All numerical simulations were performed on a high-performance compute node managed by the *PBS (Portable Batch System)* scheduler. All jobs requested a single compute node with a maximum of 56 logical CPUs, allowing for shared-memory parallelism. The algorithms were implemented in the Julia programming language. Parallel execution was managed via Julia’s native multi-threading interface, enabled at runtime using the `-t` flag to allocate the required computational resources. All numerical simulations were performed using the computational framework detailed in Section 2.6.1.

4.5.1. Cost functions

To assess the computational efficiency of the different implementations, the execution time of a single cost-function evaluation was measured for all models. Since parameter estimation requires multiple repeated cost-function evaluations, the runtime of a single iteration provides a representative measure of the computational burden associated with each formulation.

In this section we will show the results for a Maxwell and a Fractional Maxwell model, to show the difference between a model with only integer-ordered derivatives and one with fractional-ordered derivatives that involves calculating the Mittag-Leffler function on the convolutional approach. Starting with the classical Maxwell model, which serves as an ideal baseline since it does not involve the computationally expensive Mittag-Leffler, all three implementations exhibited comparable performance shown in image 4.1, with execution times on the order of 10^{-4} s. The Convolution implementation achieved the lowest median runtime, while the parallel differential method and FFT-based approaches were marginally slower, remaining within the same order of magnitude. Although the differences were statistically consistent across repeated runs, they remained within the same order of magnitude. This behavior aligns with expectations, as the classical Maxwell model relies exclusively on local integer-order derivatives and therefore does not suffer from the non-local memory tracking or complex function overhead inherent to fractional formulations.

In contrast, the Fractional Maxwell model produced markedly different results as shown in 4.2. While the fractional approach and FFT-based implementations maintained execution times below 10^{-3} s, the Convolution implementation required approximately 2×10^{-1} s per cost-function evaluation. This corresponds to a performance difference of roughly two to three orders of magnitude, with the fractional and FFT methods achieving speedups exceeding 10^2 relative to the Convolution evaluation.

As already mentioned in Section 2.4.2, fractional models like this introduce the Mittag-Leffler, as shown in Section 2.3.1, function and it bottleneck issues for the convolution approach. Consequently, while direct time-domain evaluation leads to a substantial increase in computational cost, FFT-based and specialized fractional approaches leverage the problem's structure to manage this complexity without a significant impact on execution time.

These results demonstrate that the benefits of advanced numerical techniques become increasingly significant as model complexity increases. For constitutive models that do not involve fractional terms, all implementations exhibit comparable performance. How-

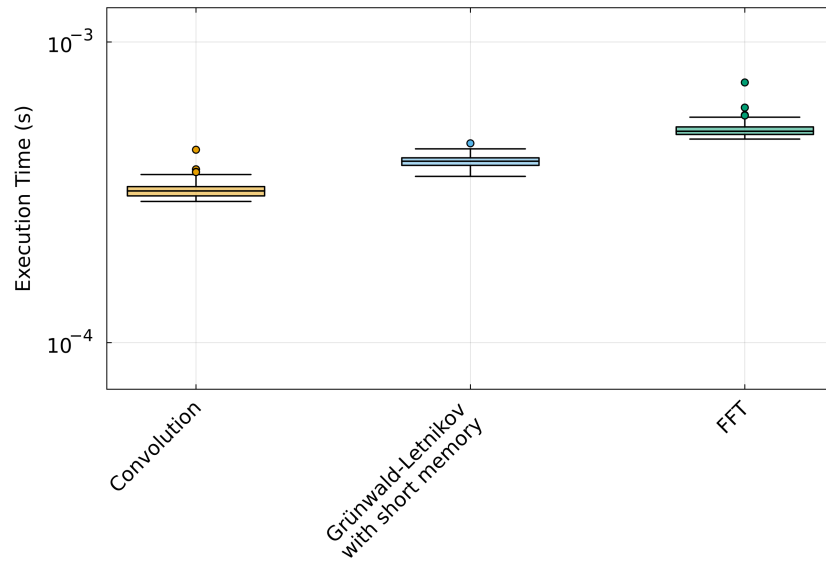


Figure 4.1: Average time of a single cost function execution for a Maxwell model on a ramp input. Comparison between Convolution, Derivative with short memory and FFT. Fractional method is executed parallel with 32 thread, $\Delta t = 0.001$ and vector length is 10001. Execution time is handled by BenchmarkTools.jl [46].

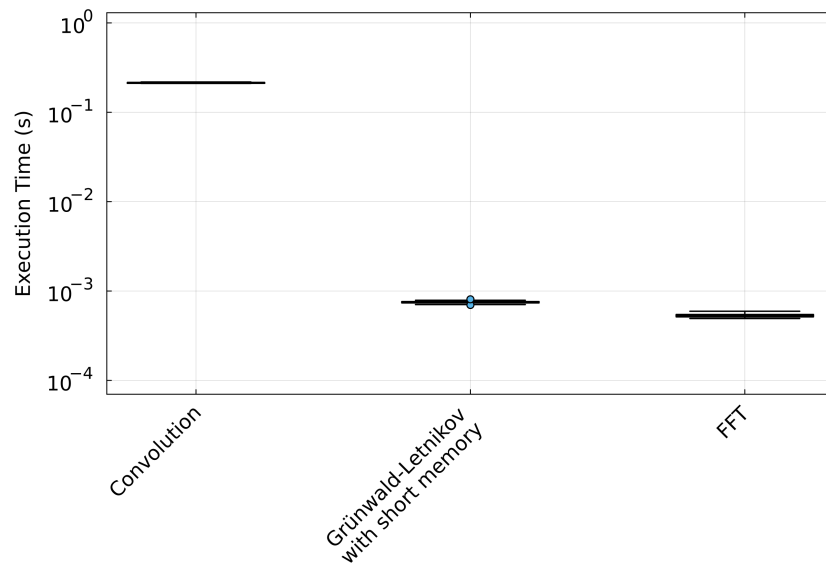


Figure 4.2: Average time of a single cost function execution for a Fractional Maxwell model on a ramp input. Comparison between Convolution, Derivative with short memory and FFT. Fractional method is executed parallel with 32 thread, $\Delta t = 0.001$ and vector length is 10001. Time is handled by BenchmarkTools.jl [46]

ever, for fractional viscoelastic models, where the response is governed by memory effects expressed through Mittag–Leffler functions, the FFT-based and parallel implementations significantly reduce the computational cost of each optimization step.

4.5.2. Fitting

To validate whether the computational trends observed at the level of single cost-function evaluations translate to full fitting problems, the same implementations were evaluated in the context of complete parameter fitting procedures.

For the classical Maxwell model, shown in 4.3, all implementations converge to the same solution, although with measurable differences in both fitting time and accuracy. The Convolution implementation achieves the shortest fitting time among all tested methods, while the three differential implementations are essentially indistinguishable from one another, with a fitting time roughly four times longer than Convolution. This reflects the relatively low computational complexity of the model, where the overhead introduced by FFT-based or parallel evaluation strategies overshadows potential time gains. Despite its faster execution, however, the Convolution implementation converges to a substantially higher RMSE error, several orders of magnitude above that of the differential approaches.

In contrast, the Fractional Maxwell model exhibits a substantially different behaviour, shown in 4.4. The Convolution implementation results in significantly longer fitting times, consistent with the high per-iteration cost previously observed in the single evaluation benchmark. Conversely, both the parallel and FFT-based implementations achieve a 20× speedup, reducing total fitting times by 95%.

Additionally, the error report by optimizer is one order of magnitude higher for convolution, showing that the differential approach also gives more precise results.

Overall, these results confirm that the relative performance hierarchy identified in the cost-function benchmark is preserved in a full problem setting. While all implementations perform similarly for simple constitutive laws, the advantage of FFT-based and parallelized formulations becomes increasingly pronounced as model complexity increases. In particular, for fractional viscoelastic models, the computational savings observed at the single-evaluation level propagate directly to the fitting stage, making these approaches more suitable for large-scale parameter estimation problems.

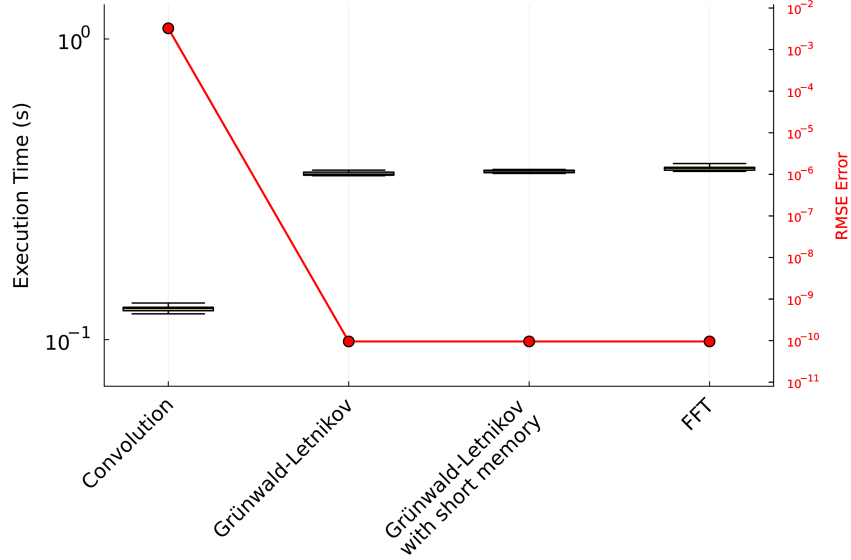


Figure 4.3: Maxwell model comparison between RHEOS, fractional model with and without Short Memory and FFT. Input data is a ramp with a 0.8 slope, on a 10 second window with $\Delta t = 0.001$, resulting vector length 10001, for the strain, while the stress is calculated with RHEOS’ modelpredict. Time is handled by BenchmarkTools.jl [46], while error reported is returned by NLOpt optimization process.

4.5.3. Carbon footprint

A preliminary assessment of the computational energy consumption was performed using the LIKWID performance monitoring suite. LIKWID is an open-source toolkit for performance analysis on x86 processors, providing access to hardware performance counters and energy measurements exposed through processor interfaces such as Intel RAPL (Running Average Power Limit). In addition to execution time and processor utilization metrics, LIKWID can estimate the energy consumed by different components of the system during program execution.

The experimental results in Table 4.1 reveal a clear performance trade-off between integer ordered and fractional frameworks. For classical formulations like the Maxwell and SLS Zener models, the standard Convolution approach is the most efficient, outperforming alternative methods in both execution time and carbon footprint.

However, this trend is sharply reversed for fractional configurations. Due to the historical dependence of fractional derivatives, the computational cost of Convolution scales severely, particularly for the Fractional Zener model. In these scenarios, the Differential and FFT-accelerated methods reduce both energy demands and carbon emissions by over 91%.

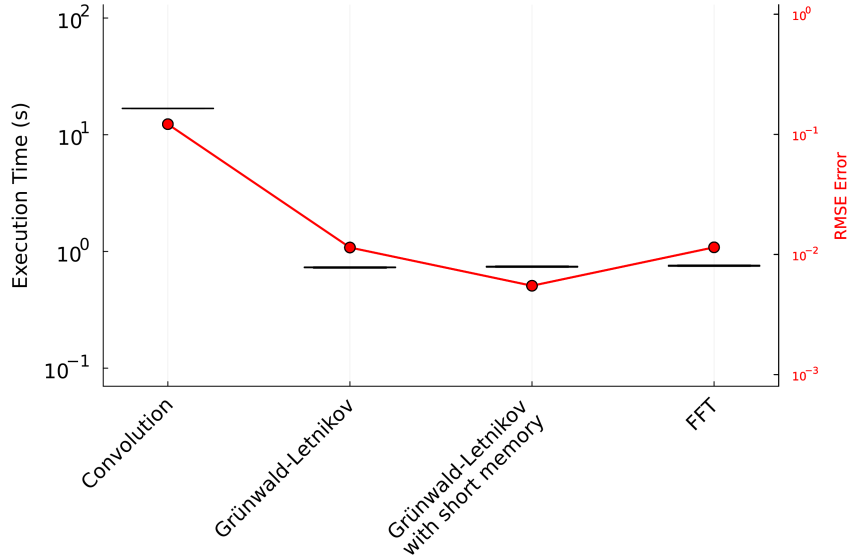


Figure 4.4: Fractional Maxwell model comparison between RHEOS, fractional model with and without Short Memory and FFT. Input data is a ramp with a 0.8 slope, on a 10 second window with $\Delta t = 0.001$, resulting in vector length 10001, for the strain, while the stress is calculated with RHEOS’ modelpredict. Time is handled by BenchmarkTools.jl [46], while error reported is returned by NLOpt optimization process. Note that Convolution error has been calculated by taking output parameters and using them in the cost function used for the differential methods, to have a coherent validation score.

These benchmarks highlight that the optimal solver depends strictly on the chosen constitutive equation. Architectural enhancements like FFT acceleration are critical to keeping simulations computationally and environmentally viable when transitioning to fractional rheological modeling.

Although the primary objective of this work was not explicit green-computing optimization, these benchmarks highlight how structural improvements in algorithmic design inherently minimize the resource footprint of rheological simulations, a factor that becomes critical when scaling to high-throughput analysis pipelines.

4.6. Conclusions

The fitting process was evaluated primarily on two models: the Maxwell and the Fractional Maxwell. For the former, the convolution implementation is the fastest of the four approaches ($\sim 0.13s$), while the other methods are essentially indistinguishable from one another in terms of fitting time (~ 0.40 s). This is due to the fact that Maxwell does not contain fractional terms and therefore does not benefit from the new methods. Neverthe-

Table 4.1: Efficiency and environmental impact of numerical methods for fitting. Simulations were run on a ramp input with slope 0.8, on a 10 second window with $\Delta t = 0.001$, resulting in vector length 10001, for the strain, while the stress is calculated with RHEOS' modelpredict.

Model	Method	Energy (J)	Footprint (gCO ₂ e)
Maxwell	Convolution	2.22	0.00016
	Differential	7.741	0.00057
	FFT	7.7	0.00056
Fractional Maxwell	Convolution	272.74	0.020
	Differential	21.14	0.0015
	FFT	21.34	0.0015
SLS Zener	Convolution	3.41	0.00025
	Differential	84.07	0.0062
	FFT	76.44	0.0056
Fractional Zener	Convolution	2829	0.21
	Differential	241	0.0177
	FFT	251	0.0185
Kelvin-Voigt	Convolution	4.9	0.00036
	Differential	4.86	0.00035
	FFT	4.97	0.00036

less, all three differential implementations converge to a RMSE error roughly eight orders of magnitude lower than Convolution ($\sim 10^{-10}$ versus $\sim 10^{-2}$), showing that, even for the simplest constitutive law, the differential formulation is markedly more numerically robust. On the other hand, for the latter, the convolution approach is the worst one, requiring approximately 15s to converge, whereas the other implementations complete the same fitting procedure with a $\sim 20\times$ speedup, consistent with what was already observed at the level of single cost-function evaluations. In terms of accuracy, the differential implementations also yield an error roughly one order of magnitude lower than convolution ($\sim 10^{-2}$ versus $\sim 10^{-1}$), confirming that the differential approach provides both a computational and a numerical accuracy advantage as model complexity increases. Results are shown in Table 4.2.

Model	Method	Time (s)	RMSE
Maxwell	Convolution	~ 0.13	$\sim 10^{-2}$
	Grünwald-Letnikov	~ 0.4	$\sim 10^{-8}$
	Grünwald-Letnikov with SM	~ 0.4	$\sim 10^{-8}$
	FFT	~ 0.4	$\sim 10^{-8}$
Fractional Maxwell	Convolution	~ 15.0	$\sim 10^{-1}$
	Grünwald-Letnikov	~ 0.8	$\sim 10^{-2}$
	Grünwald-Letnikov with SM	~ 0.8	$\sim 7 \times 10^{-3}$
	FFT	~ 0.8	$\sim 10^{-2}$

Table 4.2: Summary of execution time and RMSE relative to the four methods for the two benchmarked models, evaluated with Ramp input function (slope equal to 0.8), on a 10 second window with $\Delta t = 0.001$, resulting in vector length 10001, for the strain, while the stress is calculated with RHEOS' modelpredict.

5 | Prediction

This chapter describes the prediction process and the mathematical methods developed to perform it. In particular, two different approaches are considered: the *finite difference method* and the *fast Fourier transform-based convolution* method. Their accuracy and computational performance are then compared with those of the convolutional integral approach already implemented in RHEOS, considering both serial and parallel configurations.

The work in this section has been performed during the period abroad at University of Cambridge in the group of Prof. Alexandre Kabla.

5.1. Introduction

Once a fractional viscoelastic model has been identified through the fitting procedure, the resulting model can be used to predict the mechanical response of the system to other mechanical stimuli. In particular, the prediction problem consists in computing the unknown variable of the constitutive relation, either the stress σ or the strain ε , given the corresponding input signal and the model parameters. The model parameters used throughout this chapter are reported in Table 5.1.

Model	Parameters
Springpot	$c_\beta = 0.6, \beta = 0.3$
Dashpot	$\eta = 0.4$
Maxwell	$\eta = 0.3, k = 0.6$
Fractional Maxwell	$c_\alpha = 0.2, \alpha = 0.9, c_\beta = 0.3, \beta = 0.4$
Kelvin–Voigt	$\eta = 0.1, k = 0.35$
Fractional Kelvin–Voigt	$c_\alpha = 0.6, \alpha = 0.7, c_\beta = 0.2, \beta = 0.3$
SLS Zener	$\eta = 0.2, k_\beta = 0.7, k_\gamma = 0.3$
Fractional Zener	$c_\alpha = 0.1, \alpha = 0.7, c_\beta = 0.3, \beta = 0.3, c_\gamma = 0.5, \gamma = 0.6$

Table 5.1: Parameters used for the different viscoelastic models in the prediction tests.

Prediction problem formulation. As stated before, the viscoelastic models considered in this work can be expressed as a linear constitutive relation [17] involving fractional derivatives of stress and strain:

$$\sum_i a_i D^{\alpha_i} \sigma(t) = \sum_j b_j D^{\beta_j} \varepsilon(t), \quad (5.1)$$

where D^α denotes a fractional derivative of order α , and the coefficients a_i and b_j are model-dependent constants. The known signal, $\sigma(t)$ or $\varepsilon(t)$, is available as discrete time series sampled on a uniform temporal grid

$$t_k = k\Delta t, \quad k = 0, 1, \dots, n, \quad (5.2)$$

where Δt denotes the time step. As in the previous task, the prediction requires the numerical evaluation of the constitutive equation, also using the numerical methods for fractional derivatives discussed in Chapter 3.

In this work, two different numerical strategies are considered to solve the prediction problem. The first approach directly discretizes the fractional derivatives in (5.1), leading to a recursive formulation for the unknown signal: each new value is computed using only the previously obtained ones, one step at a time. This property makes the method particularly suitable for situations in which the input variable is not known in advance, but becomes available incrementally as the process unfolds. The second approach exploits the convolution structure of fractional operators and accelerates their evaluation using the fast Fourier transform (FFT). Since the FFT operates on the entire input vector at once, this approach is useful when the entire input signal is known and we want to obtain the complete output vector in a single computation. These methods are described in the following sections.

5.2. Finite Difference prediction method

The first approach considered to solve the prediction problem is based on the direct discretization of the different terms in the constitutive equation. Once a signal is defined on a discrete temporal grid, each derivative can be approximated using finite difference schemes for integer orders or the Grünwald-Letnikov (GL) formulation for fractional orders. For first-order derivatives, we adopted standard finite difference approximations [51]. In particular, two schemes are considered: the backward difference (BD) and the

central difference (CD). The former approximates the derivative at time t_k as:

$$\frac{df(t_k)}{dt} \approx \frac{f(t_k) - f(t_{k-1})}{\Delta t}. \quad (5.3)$$

This scheme only depends on the current and previous values of the signal and is therefore well suited for recursive computations. Alternatively, the central difference scheme can be used, which provides a higher-order approximation of the derivative. In this case, the derivative at time t_k is approximated as:

$$\frac{df(t_k)}{dt} \approx \frac{f(t_{k+1}) - f(t_{k-1}))}{2\Delta t}. \quad (5.4)$$

This approximation generally improves accuracy but requires information from both neighboring time steps. In the implementation, the chosen scheme is applied to the input signal before evaluating the constitutive equation.

Fractional derivatives are instead computed using the Grünwald-Letnikov definition and weights defined in Equations (3.7) and (3.8). This formulation naturally introduces a history dependence, since the derivative at time t_k depends on all previous values of the signal.

Given these approximations, the constitutive equation can be evaluated step by step in time, as also shown in Appendix B. The terms that depend only on the known signal are first collected into a right-hand side (RHS) vector. This includes contributions from integer derivatives and fractional derivatives of the input signal, which are computed using the previously described discretization schemes.

The terms involving the unknown quantity, which we denote by f , are treated separately and contribute to the left-hand side of the discretized equation. Since the solution is computed sequentially in time, at step t_k the approximated values of f are known up to t_{k-1} , and each term involving it can therefore be decomposed into a contribution depending on the current value $f(t_k)$ and a contribution depending on previously computed values. Let c and γ denote the coefficient and the order of a generic term involving f . The discretization is performed as follows:

- for *non-derivative terms* of the form $c f(t_k)$, no discretization is required, since the term already involves the unknown value to be computed at the current time step;
- *first-order derivative terms* are discretized using the backward difference scheme

seen in Equation (5.3):

$$\frac{df(t_k)}{dt} \approx \frac{f(t_k) - f(t_{k-1})}{\Delta t}, \quad (5.5)$$

- *fractional derivative terms* of the form $c D^\gamma f(t_k)$ are approximated using the Grünwald-Letnikov discretization as seen in Equation (3.7). Then, the sum can be separated into the contribution associated with the current value and the historical terms as:

$$c D^\gamma f(t_k) \approx \frac{c}{\Delta t^\gamma} f(t_k) + \frac{c}{\Delta t^\gamma} \sum_{j=1}^k \omega_j f(t_{k-j}), \quad (5.6)$$

where the weights $\omega_j^{(\gamma)}$ are computed through the recurrence relation given in Equation (3.8). As described above, the calculation of the derivative can be carried out in parallel.

After discretizing all the terms as described above, the constitutive equation can be reorganized by separating the contributions associated with the current value $f(t_k)$ from those depending only on previously computed values. The latter contributions are collected into the right-hand side (RHS) of the equation, together with the terms involving the known input signal. The coefficients multiplying the current value $f(t_k)$ are instead grouped into a denominator term.

Finally, the prediction is obtained through a recursive procedure in which, at each time step t_k , the RHS is updated with the contributions coming from previously computed values of the unknown variable, which arise from the history terms of the Grünwald-Letnikov discretization. The new value of the signal is finally obtained by dividing the resulting expression by the denominator.

This iterative formulation allows the solution to be computed sequentially along the temporal grid. However, due to the history dependence introduced by the Grünwald-Letnikov derivatives, the computational cost grows with the length of the time series. In particular, the evaluation of the fractional derivative at time t_k requires a summation over all previous samples of the signal. As a consequence, the overall computational complexity of the method scales quadratically with the number of time steps, resulting in a cost of $\mathcal{O}(N^2)$ for a signal of length N [27, 39].

Example To better understand the procedure just explained, in this paragraph we will outline all the mathematical steps. For simplicity, we will calculate only the stress σ using the Grünwald-Letnikov derivative and the fractional Maxwell model, which has the

following constitutive equation:

$$\sigma(t) + \frac{c_\alpha}{c_\beta} \frac{d^{\alpha-\beta}\sigma(t)}{dt^{\alpha-\beta}} = c_\alpha \frac{d^\alpha \varepsilon(t)}{dt^\alpha}.$$

Since the strain vector ε is known in this case, we first compute the derivative of order α . Having done so, from now on we will consider the right-hand side of the equation as:

$$rhs = c_\alpha \frac{d^\alpha \varepsilon(t)}{dt^\alpha}.$$

Moreover, for simplicity, consider:

$$\gamma = \alpha - \beta, \quad k = \frac{c_\alpha}{c_\beta},$$

and substitute them into the constitutive equation, obtaining:

$$\sigma(t) + k \frac{d^\gamma \sigma(t)}{dt^\gamma} = rhs.$$

From Equation (3.7), and given that we are following a step-by-step process, the vector ε is known up to step $n - 1$, we can state that:

$$\frac{d^\gamma \sigma(t)}{dt^\gamma} = \frac{1}{\Delta t^\gamma} \sigma_n + \frac{1}{\Delta t^\gamma} \sum_{j=1}^n \omega_j^{(\gamma)} \sigma_{n-j}, \quad \sigma(t_n) = \sigma_n,$$

which leads us to the following formula:

$$\sigma_n + \frac{k}{\Delta t^\gamma} \sigma_n = rhs - \frac{k}{\Delta t^\gamma} \sum_{j=1}^n \omega_j^{(\gamma)} \sigma_{n-j}.$$

Finally, by reordering this last equation, we obtain that the stress σ at each step can be computed as:

$$\sigma(t) = \frac{rhs - \frac{k}{\Delta t^\gamma} \sum_{j=1}^n \omega_j^{(\gamma)} \sigma_{n-j}}{1 + \frac{k}{\Delta t^\gamma}}.$$

5.3. FFT-based prediction method

The second numerical strategy implemented to solve the prediction problem leverages the Fast Fourier Transform (FFT). While the Finite Difference approach provides an iterative

solution in the time domain, its computational cost becomes prohibitive for long time series. The FFT-based method addresses this limitation by transforming the constitutive relations into the frequency domain.

5.3.1. Motivations and Computational Efficiency

The main reason why the FFT was implemented is to reduce computation time, especially in scenarios where only a limited number of threads is available. To obtain this result, the FFT-based method exploits the Convolution Theorem. By transforming the discrete temporal operators into the frequency domain, the costly time-domain convolution is replaced by point-wise multiplications and, through the use of the Fast Fourier Transform (FFT) algorithm provided by the FFTW library [52], the computational complexity is drastically reduced to $\mathcal{O}(N \log N)$, enabling the efficient processing of extensive time series.

In order to maximize the efficiency of this approach, since the physical signals and weights are strictly real-valued, FFT routines are handled by using real-to-complex transforms (`rfft`) and their inverses (`irfft`) [52]; this optimization halves both the memory footprint and the number of floating-point operations required, computing only $L/2 + 1$ complex coefficients. Furthermore, to prevent the aliasing artifacts characteristic of circular convolutions (inherent to the Discrete Fourier Transform), the time-domain arrays must be zero-padded. In our implementation, the padded length L is dynamically calculated as the next power of two greater than or equal to $2N - 1$ (i.e., $L = 2^{\lceil \log_2(2N-1) \rceil}$). Finally, to minimize garbage collection overhead and memory fragmentation, all necessary buffers are pre-allocated once at the beginning of the routine, while element-wise multiplications in the frequency domain are accelerated using Single Instruction, Multiple Data (SIMD) vectorization.

5.3.2. Stress Prediction

In the stress prediction phase, the strain history $\varepsilon(t)$ is provided as the known input data, and the goal is to compute the resulting stress $\sigma(t)$. The numerical procedure relies on transforming both the known input signal and the fractional differential operators into the frequency domain.

Given the generic fractional viscoelastic equation, the terms can be split into dependencies (acting on the known strain) and unknowns (acting on the stress). The algorithm proceeds through the following steps:

1. The known strain signal is zero-padded to length L and transformed into the fre-

quency domain to obtain $\hat{\varepsilon}(\omega)$.

2. For each dependency term $b_j D^{\beta_j}$ acting on the strain, which belongs to the Right-Hand Side, the Grünwald-Letnikov discrete weights are computed for the specific order β_j . These temporal weights are then transformed into the frequency domain via FFT. The contribution of the term is obtained by multiplying the transformed weights, the input signal $\hat{\varepsilon}(\omega)$, and the corresponding model coefficients. The results are accumulated into a single RHS buffer.
3. Similarly, for each unknown term $a_i D^{\alpha_i}$ acting on the stress, the specific Grünwald-Letnikov weights are generated and transformed. They are multiplied by the model coefficients and accumulated into an LHS buffer, which represents the frequency-domain operator acting on the unknown stress.
4. The predicted stress in the frequency domain is computed element-wise as:

$$\hat{\sigma}(\omega) = \frac{\text{RHS}(\omega)}{\text{LHS}(\omega)} \quad (5.7)$$

5. Finally, the inverse real-to-complex FFT (`irfft`) is applied to $\hat{\sigma}(\omega)$, and the result is truncated back to the original length N to yield the predicted stress time series $\sigma(t)$.

5.3.3. Strain Prediction

In this case, the objective is to find the strain $\varepsilon(t)$ resulting from an applied stress $\sigma(t)$. While the underlying sequence of operations (padding, weight generation, transformation, division, and inversion) is the same as in the stress prediction algorithm, the numerical inversion of the constitutive equation creates a severe stability issue.

In the frequency domain, solving for the strain often requires dividing by terms associated with fractional derivatives. For low frequencies ($\omega \rightarrow 0$), this division can lead to numerical singularities or unstable unbounded growths, particularly as the fractional order α of one or more terms gets close to 1, corresponding to a more viscous fluid-like behaviour.

To overcome this issue, the governing fractional differential equation is mathematically preconditioned before the FFT evaluation by applying a fractional integration operator. Let α_{max} be the maximum fractional order among all the terms acting on the unknown strain. By applying the fractional integral operator $D^{-\alpha_{max}}$ to both sides of the constitu-

tive Equation (5.1)

$$D^{-\alpha_{max}} \sum_i a_i D^{\alpha_i} \sigma(t) = D^{-\alpha_{max}} \sum_j b_j D^{\beta_j} \varepsilon(t), \quad (5.8)$$

we obtain:

$$\sum_i a_i D^{\alpha_i - \alpha_{max}} \sigma(t) = \sum_j b_j D^{\beta_j - \alpha_{max}} \varepsilon(t), \quad (5.9)$$

By the composition properties of fractional operators, applying an integral of order α_{max} to a derivative of order α yields a new operator of order $\alpha - \alpha_{max}$. In the temporal implementation, this translates to shifting the order of all derivative terms:

$$\text{new_order} = \text{original_order} - \alpha_{max}. \quad (5.10)$$

By factoring out the maximum order, the algorithm computes the corresponding Grünwald-Letnikov weights for these newly shifted orders. In this way, the LHS denominator in the frequency domain is regularized, preventing division-by-zero errors at zero frequency while preserving the mathematical equivalence of the equation. Once the stabilized division $\text{RHS}(\omega)/\text{LHS}(\omega)$ is performed, the result is safely inverse-transformed to yield the strain prediction in the time domain.

5.4. Implementation in RHEOS

Mirroring the design principles implemented in the fitting section, the prediction functions in the updated version of RHEOS preserve complete API continuity. Rather than introducing distinct functions for each computational backend, the system architecture relies on the same dispatch-driven paradigm to encapsulate numerical complexity.

The high-level user interface remains uniform across all simulation types. The selection of the underlying solver is controlled via optional arguments, hiding the algorithmic differences between convolutional, differential, and FFT methods away from the user.

The standardized syntax for executing prediction is structured as follows:

```
modelpredict(data, model, predtype=Convolution())
modelpredict(data, model, predtype=Differential, deriv_method=GLThreads())
```

Here, `data` represents the input driving protocol, i.e., the prescribed kinematic or dynamic history (e.g., a strain or stress time series), while `model` is a fully parameterized RHEOS

model data structure. By preserving full compatibility with the fitting interface, this approach ensures a clear separation of concerns: the user specifies the physical system to be simulated, while the package internally handles the numerical evaluation of the underlying fractional operators.

5.5. Results and Discussion

In this section, we present the results obtained from the numerical prediction methods described in this chapter. The evaluation is divided into two main parts: at first, we validate the accuracy of our implementations by comparing the numerical results against analytical solutions; then, we analyze the computational performance, benchmarking our Finite Difference (both serial and parallel) and FFT-based methods against the original convolution of RHEOS. All numerical simulations were performed using the computational framework detailed in Section 3.5.

5.5.1. Accuracy Validation

To verify the accuracy and reliability of the implemented prediction algorithms, we performed an error analysis by evaluating the accuracy of the numerical stress prediction given a known strain input.

This validation phase was conducted on a specific subset of viscoelastic models, while the input strain histories were restricted to step, ramp and ramp & hold functions. This choice is motivated by the fact that, for these specific strain inputs, exact analytical solutions for the stress $\sigma(t)$ can be derived from the theoretical relaxation modulus $G(t)$ of each model using the following expressions (see Appendix C for further details):

- **Unit Step Function (Heaviside):**

$$\sigma(t) = \varepsilon_0 G(t)$$

where ε_0 is the amplitude of the step (it will always be equal to 1.0, since we are considering only the unitary step);

- **Ramp Function:**

$$\sigma(t) = \dot{\varepsilon}_r \int_0^t G(t - \tau) d\tau$$

where $\dot{\varepsilon}_r$ is the constant strain rate (gradient);

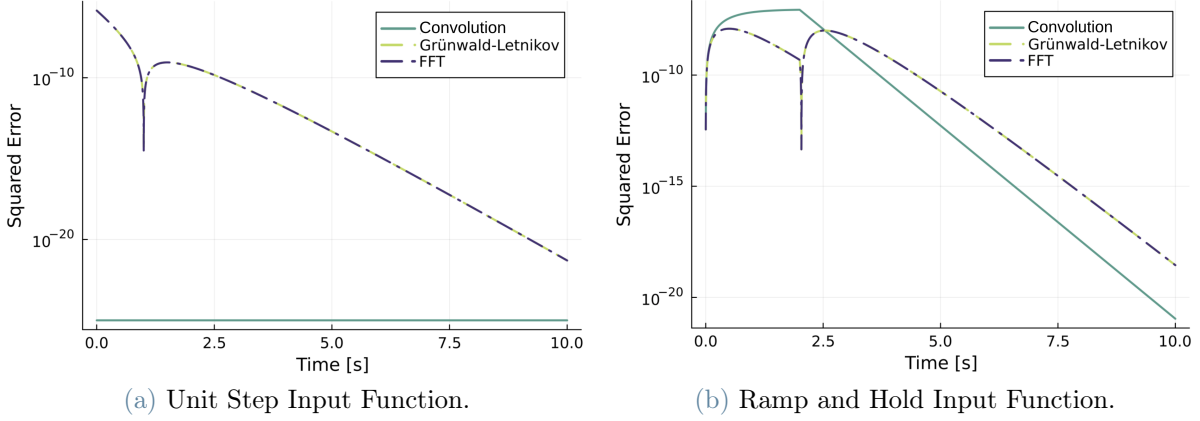


Figure 5.1: Plot of the squared error over time for both Unit Step and Ramp & Hold input functions using the Maxwell model and the discussed implementations, a constant time step $\Delta t = 0.001$ and, for the second one, a slope $\varepsilon_r = 0.1$ and plateau equal to 2.0.

- **Ramp and Hold Function:**

$$\sigma(t) = \varepsilon_r \int_0^{t_1} G(t - \tau) d\tau$$

where t_1 is the time at which the holding phase begins and $H(t)$ is the Heaviside step function.

To quantify the discrepancy between the numerical prediction and the exact solution, we compute the squared error of the convolution and our implementations for each model and input function. As shown in Figures 5.1 and 5.2, the numerical results validate both the correctness and the stability of the implemented algorithms across different loading conditions. Moreover, in all analyzed scenarios, the error curves for the Grünwald-Letnikov finite difference approach and the FFT-based method perfectly overlap, proving that the frequency-domain transformation accurately evaluates the fractional operators.

Finally, Figure 5.2b show how our numerical implementations become more precise under more complex transient conditions and models. In particular, both the GL and FFT methods maintain a squared error that is approximately two orders of magnitude lower than that of the original RHEOS approach throughout the entire duration of the test, dropping sharply at the plateau and steadily decaying.

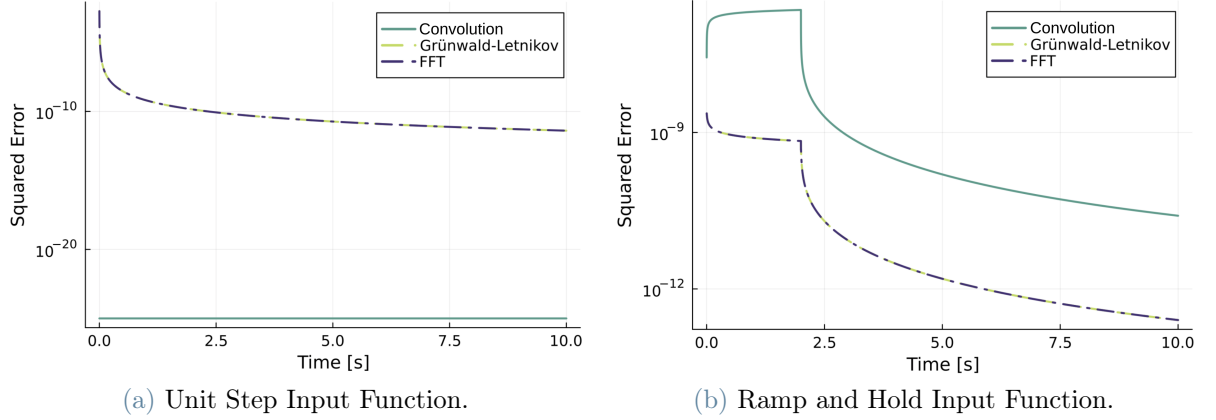


Figure 5.2: Plot of the squared error over time for both Unit Step and Ramp & Hold input functions using the Springpot model and the discussed implementations, a constant time step $\Delta t = 0.001$ and, for the second one, a slope $\varepsilon_r = 0.1$ and plateau equal to 2.0.

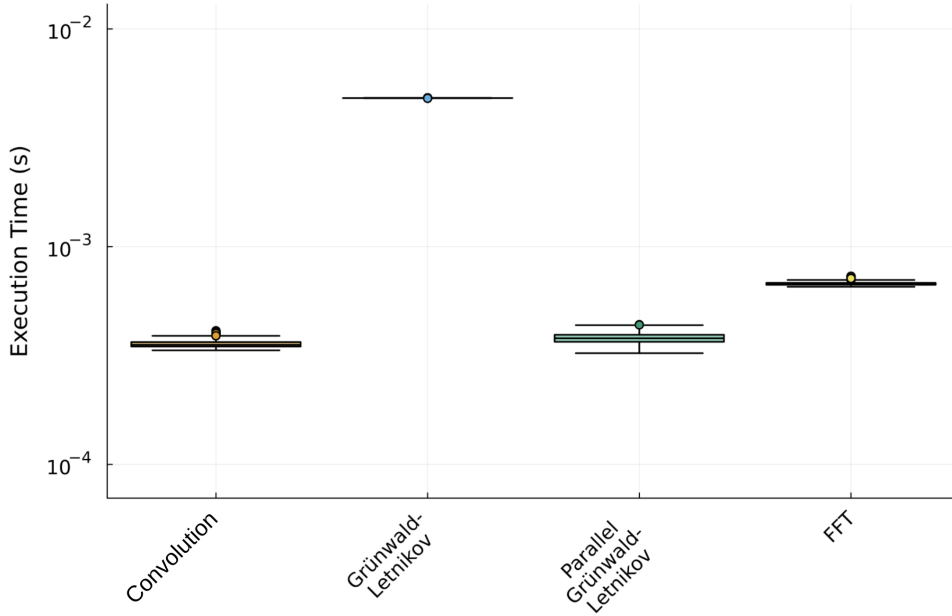
5.5.2. Computational Performance

Having validated the accuracy of the numerical solvers, the second phase of our analysis focuses on evaluating their computational efficiency and to demonstrate that the Finite Difference approach (both in serial and parallel configurations) and the FFT-based method outperform the original convolution of RHEOS, particularly for more complex models, namely those involving several parameters.

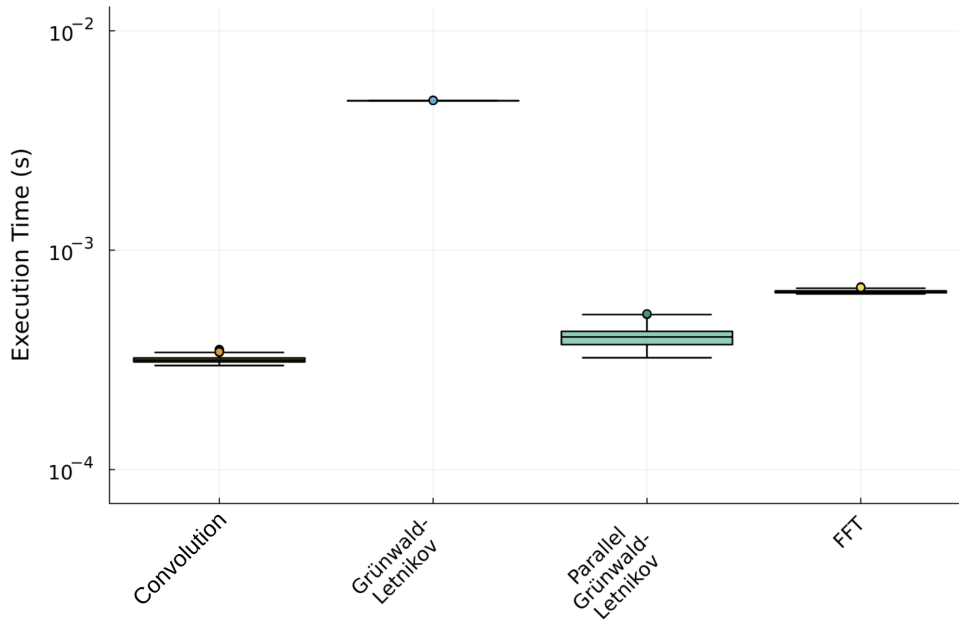
The benchmark was performed using the models already cited in Table 5.1 for both stress σ and strain ε . To clearly illustrate the performance differences, we compared the execution times for a classical integer-order model (Maxwell) and two highly complex fractional models (Fractional Kelvin-Voigt and Fractional Zener). The results are visualized using logarithmic scale boxplots in Figures 5.3, 5.5 and 5.6.

As shown in Figure 5.3, when evaluating the classical Maxwell model (Figure 2.5), the convolutional approach proves to be highly optimized. Because this model involves only standard first-order integer derivatives, the generalized fractional approaches carry a slight computational overhead. The serial Grünwald-Letnikov (GL) method is the slowest, while the parallel GL implementation successfully reduces the execution time to match the convolutional approach closely. The FFT-based method remains extremely fast (under 1 millisecond), but does not provide a definitive advantage for purely integer-order equations.

When moving to fractional models, such as the Fractional Kelvin-Voigt shown in Figure 2.10, the performance begins to shift (Figure 5.5): for stress prediction, all approaches



(a) Stress Prediction (Maxwell).



(b) Strain Prediction (Maxwell).

Figure 5.3: Execution times (log scale) for stress and strain prediction using the standard integer-order Maxwell model and $\Delta t = 0.01$ ($N = 10001$ steps, $t \in [0, 100]$). Execution time is handled by BenchmarkTools.jl [46]. The parallel implementation was executed with 32 threads.

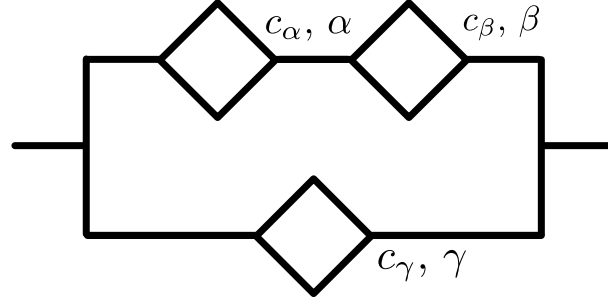
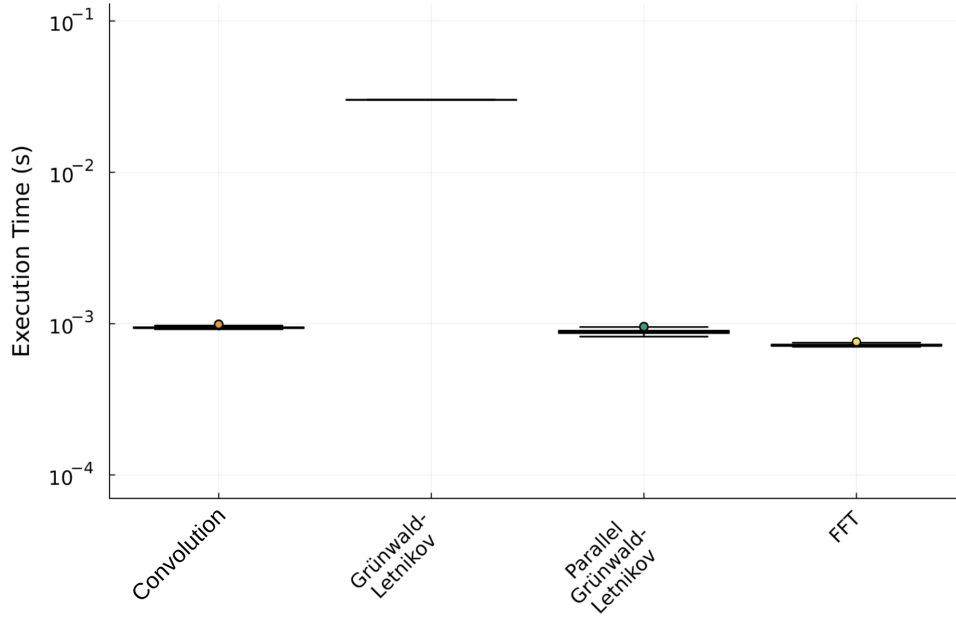


Figure 5.4: Schematic representation of the Fractional Kelvin Voigt (left) and Fractional Zener (right).

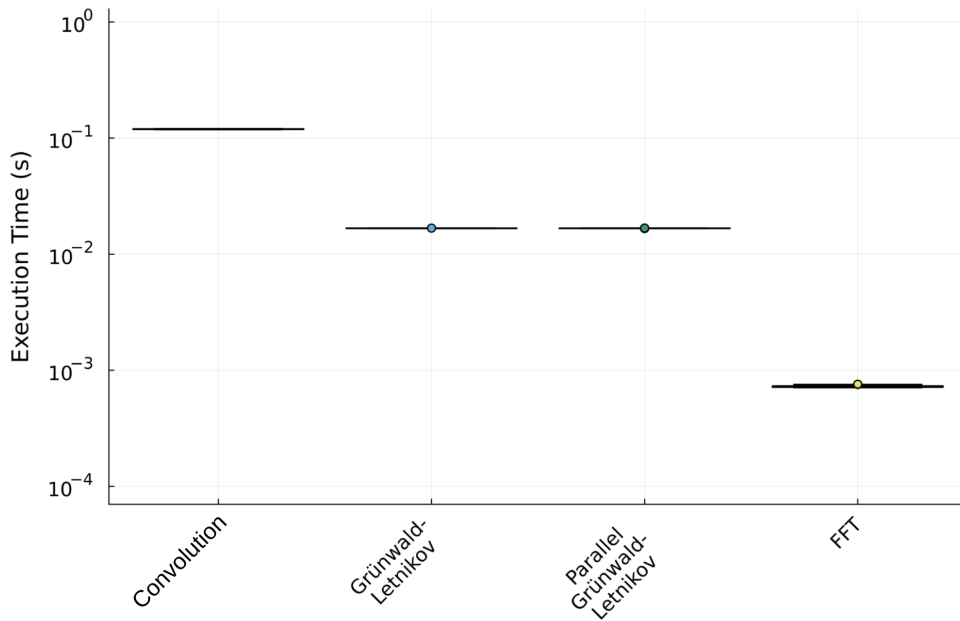
perform similarly. For strain prediction, however, the methods differ significantly. In this case, the convolution approach is the slowest, with an execution time of approximately 0.1 seconds. On the other hand, the GL method achieves a speedup of roughly $10\times$ over it, while the FFT method reaches up to $100\times$, maintaining a sub-millisecond efficiency. Notably, the serial and parallel GL implementations exhibit nearly identical execution times: the parallelization targets exclusively the fractional derivative computations on the right-hand side, which vanish in this case since the known stress input is represented by a single integer-order term.

The true computational advantage of our implementations becomes evident when analyzing highly complex fractional models, such as the Fractional Zener model (Figure 5.4), which involves multiple fractional derivatives. In this scenario, shown in Figure 5.6, the performance of the original RHEOS library approach significantly degrades for both stress and strain predictions, requiring roughly 0.1 seconds to compute the solution.

The serial GL method handles the fractional history summation more efficiently, and distributing this historical summation across multiple threads (Parallel GL) reduces the execution time even further. However, the best result is achieved by the FFT-based method, which computes both stress and strain predictions in approximately 0.001 seconds, representing a huge performance increase of a $100\times$ speedup compared to the original one.

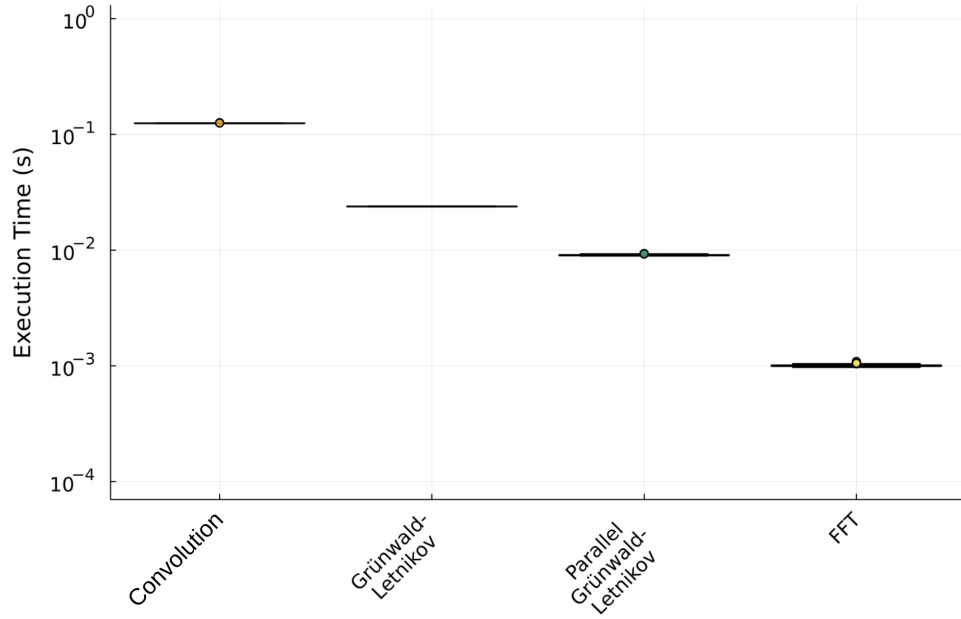


(a) Stress Prediction (Fractional Kelvin-Voigt).

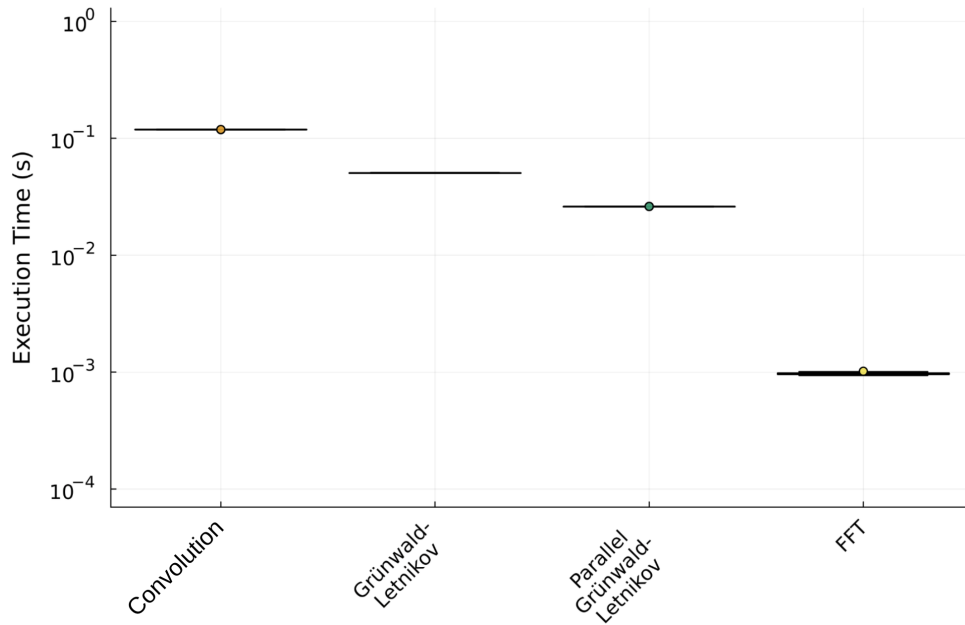


(b) Strain Prediction (Fractional Kelvin-Voigt).

Figure 5.5: Execution times (log scale) for stress and strain prediction using the Fractional Kelvin-Voigt model and $\Delta t = 0.01$ ($N = 10001$ steps, $t \in [0, 100]$). Execution time is handled by BenchmarkTools.jl [46]. The parallel implementation was executed with 32 threads.



(a) Stress Prediction (Fractional Zener).



(b) Strain Prediction (Fractional Zener).

Figure 5.6: Execution times (log scale) for stress and strain prediction using the Fractional Zener model and $\Delta t = 0.01$ ($N = 10001$ steps, $t \in [0, 100]$). Execution time is handled by BenchmarkTools.jl [46]. The parallel implementation was executed with 32 threads.

Carbon Footprint

The experimental results for the forward prediction tasks, presented in Table 5.2, display different dynamics compared to the fitting phase. Because the time horizons evaluated

for predictions are comparatively short, the absolute differences in execution time, energy consumption, and environmental impact across the numerical methods are significantly less pronounced.

For classical formulations like the Maxwell and SLS Zener models, the low computational cost makes the Convolution and Differential approaches highly efficient, whereas the FFT-accelerated method exhibits a slight overhead, doubling the execution time and resource footprint due to the initialization costs of the transforms on small data sizes. However, when transitioning to fractional configurations, the algorithmic scaling advantages of the alternative methods re-emerge, even within short time frames.

For a representative strain prediction on the Fractional Zener model, the Differential and FFT-based methods bypass the history-dependent bottleneck, reducing both energy demand and carbon emissions by up to 31% compared to the standard Convolution approach. While these savings are minor for individual short-term predictions, they confirm that differential and transform-based formulations maintain a consistent efficiency advantage for fractional models across both inverse fitting and forward prediction workflows.

Table 5.2: Efficiency and environmental impact of numerical methods for predicting. Simulations were run on a ramp input data with slope 0.8, on a time period going from 0 to 10 seconds with a $\Delta t = 0.001$, resulting in a vector length of 10001. The input data is stress imposed and its calculating strain.

Model	Method	Energy (J)	Footprint (gCO ₂ e)
Maxwell	Convolution	2.23	0.00016
	Differential	2.25	0.00016
	FFT	4.6	0.00033
Fractional Maxwell	Convolution	5.5	0.0004
	Differential	3.8	0.00028
	FFT	4.44	0.00032
SLS Zener	Convolution	1.95	0.00014
	Differential	1.88	0.0013
	FFT	4.5	0.00033
Fractional Zener	Convolution	6.6	0.00048
	Differential	4.5	0.00033
	FFT	5.2	0.00038

5.6. Conclusions

The accuracy validation confirms that both the Grünwald-Letnikov finite difference and the FFT-based methods accurately reproduce the analytical stress predictions across dif-

ferent loading conditions, with their error curves perfectly overlapping in every tested scenario. From a computational perspective, the advantage of the proposed methods scales with model complexity. For the classical, integer-order Maxwell model, the convolution-based approach remains highly competitive, as expected for a model lacking a genuine fractional history term. However, as the model becomes more complex, the benefit becomes significant: for the Fractional Kelvin-Voigt model, the GL method achieves roughly a $10\times$ speedup over the convolution approach for strain prediction, while the FFT-based method reaches up to $100\times$. Similar results are also obtained for the Fractional Zener model, where both stress and strain involve multiple fractional derivatives. Table 5.3 summarizes the execution times and relative speedups for the three benchmarked models.

Model	Convolution [s]	GL Speedup	FFT Speedup
Maxwell	$\sim 2.5 \times 10^{-4}$	no advantage	no advantage
Fractional KV (stress)	$\sim 10^{-3}$	$\sim 1.1\times^*$	$\sim 1.3\times$
Fractional KV (strain)	~ 0.1	$\sim 10\times$	$\sim 100\times$
Fractional Zener (stress)	~ 0.1	$\sim 5\times^{**}, \sim 10\times$	$\sim 100\times$
Fractional Zener (strain)	~ 0.1	$\sim 2.5\times^{**}, \sim 5\times$	$\sim 100\times$

Table 5.3: Summary of execution time and speedup relative to the original RHEOS convolution for the three benchmarked models, evaluated with Unit Step input function and a constant time step $\Delta t = 0.01$ ($N = 10001$ steps, $t \in [0, 100]$). Parallel executions are evaluated using 32 threads. ^{*}Parallel GL only; serial GL shows no advantage. ^{**}Speedup for Serial GL.

6 | Conclusions and future developments

This thesis addresses the computational challenges inherent in the simulation and parameter identification of fractional viscoelastic models, proposing and validating a differential approach as an efficient alternative to the classical Boltzmann integral formulation.

The entire work was motivated by a clear limitation: while fractional models offer a powerful mathematical description of the time-dependent mechanical response of a wide range of materials, such as polymers and biological tissues, their adoption in engineering workflows has been limited by the high computational cost of computing convolution integrals, particularly when model evaluations must be repeated thousands of times during parameter fitting or at every point of a finite element simulation.

6.1. Summary of Contributions

The first objective was the development and analysis of numerical methods for fractional calculus. Chapter 3 presented a systematic comparison of finite difference discretizations of fractional derivatives (Grünwald-Letnikov, Caputo and Riemann-Liouville) together with the Short Memory Principle and the Fast Fourier Transform. The numerical validation confirmed that all implementations achieve the expected accuracy, and scalability experiments demonstrated robust strong and weak scaling up to 16 threads, consistent with the theoretical limits imposed by Amdahl's law. The methods were consolidated into NumFracDiff, an open-source Julia library designed for high performance evaluation of fractional derivatives and designed to integrate seamlessly with external rheological frameworks such as RHEOS.

The second objective was the application of the differential approach to parameter identification. Chapter 4 proved the great potential of the fractional approach for fitting procedures. The Mittag-Leffler bottleneck problem was completely avoided by not using the function in the first place, while the impact on computational costs imposed by the

non-locality property was lowered thanks to the Short Memory variation of the code. This approach also enables us to use parallelization techniques to accelerate computation that were previously not available. This work amounted to lower the execution time by one order of magnitude on the experimental data tested, while being very promising to scale a lot better on problems with bigger input datas.

The third objective was the prediction of mechanical responses under arbitrary loading histories. Chapter 5 introduced two complementary implementations, a finite difference time domain solver and an FFT-based method, and validated both against exact analytical solutions for step, ramp, and ramp and hold inputs. For complex fractional models such as the Fractional Zener model, the FFT-based predictor achieved approximately a $100\times$ speedup over the convolution integral approach for both stress and strain prediction, while the parallel finite-difference implementation provided intermediate but consistent gains. Moreover, the accuracy results confirmed that the GL finite difference and FFT methods produce overlapping error curves, establishing the mathematical equivalence of the two approaches and their correctness.

The last objective was the delivery of open-source tools. Both the NumFracDiff library and the extensions to RHEOS are publicly available, providing ready-to-use implementations of the methods developed in this thesis.

6.2. Future Developments

Several directions emerge from this work for future research and development:

1. **GPU Acceleration:** To handle significantly larger datasets or high-throughput computations, the core algorithms would benefit from acceleration via GPU architectures. Porting the intensive numerical routines to CUDA would reduce execution times.
2. **Custom Optimization Algorithms:** Focus on replacing the general external `NLopt` dependency with a custom-written, ad-hoc optimization solver. The high variability in performance among `NLopt`'s diverse algorithms indicates that a tailored optimization routine can be engineered to specifically fit fractional rheological problems, eliminating black-box constraints while enhancing code integration and performance.
3. **Integration into Finite Elements Frameworks:** Current Finite Element (FEM) solvers involving fractional viscoelasticity are severely bottlenecked by the high computational cost of evaluating convolution integrals at every single integration point,

making large-scale spatial simulations impractically slow. Since the differential and FFT-based methods developed in this work have demonstrated speedups of up to two orders of magnitude within the RHEOS framework, their integration into FEM solvers would offer a concrete pathway to accelerate the entire structural analysis pipeline.

4. **Machine Learning for Model Selection:** Integrating machine learning techniques into the workflow could automate the selection process. By training a model on raw experimental data, the framework could autonomously identify the most suitable model prior to fitting parameters, based on data behaviour.
5. **Extension to Non-Linear Models:** The framework can be extended to accommodate non-linear viscoelastic models, for example those governed by the Casson equation, to capture complex strain-dependent behaviors. In these non-linear regimes, the FFT-based approach becomes mathematically unsuitable due to the breakdown of the superposition principle required by the convolution theorem. Conversely, the differential approach remains inherently viable since it discretizes operators directly in the time domain. Focus of the work could switch to adapting the current codebase to handle non-linear differential solvers and state-dependent coefficient updates.

Bibliography

- [1] John D Ferry and Henry S Myers. Viscoelastic properties of polymers. *Journal of The Electrochemical Society*, 108(7):142C–143C, 1961.
- [2] Alexander Ya Malkin, Svetlana R Derkach, and Valery G Kulichikhin. Rheology of gels and yielding liquids. *Gels*, 9(9):715, 2023.
- [3] Robert Byron Bird, Robert Calvin Armstrong, and Ole Hassager. Dynamics of polymeric liquids. vol. 1: Fluid mechanics. 1986.
- [4] Viridiana Tejada-Ortigoza and Enrique Cuan-Urquizo. Towards the development of 3d-printed food: A rheological and mechanical approach. *Foods*, 11(9):1191, 2022.
- [5] Marco Cattaneo, Giulia Guerriero, Gazendra Shakya, Lisa A Krattiger, Lorenza G. Paganella, Maria L Narciso, and Outi Supponen. Cyclic jetting enables microbubble-mediated drug delivery. *Nature Physics*, 21(4):590–598, 2025.
- [6] Howard A Barnes, John Fletcher Hutton, and Kenneth Walters. *An introduction to rheology*, volume 3. Elsevier, 1989.
- [7] Alessandra Bonfanti. Applications of fractional calculus to viscoelasticity. Short course at the Annual European Rheology Conference, April 2025. 13th–17th April.
- [8] Roderic S Lakes. *Viscoelastic materials*. Cambridge university press, 2009.
- [9] Alessandra Bonfanti, Jonathan Louis Kaplan, Guillaume Charras, and Alexandre Kabla. Fractional viscoelastic models for power-law materials. *Soft matter*, 16(26): 6002–6020, 2020.
- [10] Hal F Brinson and L Catherine Brinson. *Polymer engineering science and viscoelasticity: an introduction*. Springer, 2008.
- [11] PingGen Cai, Yusuke Mizutani, Masahiro Tsuchiya, John M Maloney, Ben Fabry, Krystyn J Van Vliet, and Takaharu Okajima. Quantifying cell-to-cell variation in power-law rheology. *Biophysical journal*, 105(5):1093–1102, 2013.
- [12] Ben Fabry, Geoffrey N Maksym, James P Butler, Michael Glogauer, Daniel Navajas,

- and Jeffrey J Fredberg. Scaling the microrheology of living cells. *Physical review letters*, 87(14):148102, 2001.
- [13] Peter Fischer and Erich J Windhab. Rheology of food materials. *Current Opinion in Colloid & Interface Science*, 16(1):36–40, 2011.
- [14] JG Davis. 162. the rheology of cheese, butter and other milk products.(the measurement of “body” and “texture”). *Journal of Dairy Research*, 8(2):245–264, 1937.
- [15] Fabrizio Barpi and Silvio Valente. Creep and fracture in concrete: a fractional order rate approach. *Engineering Fracture Mechanics*, 70(5):611–623, 2002.
- [16] Alessandra Bonfanti, Jonathan Fouchard, Nargess Khalilgharibi, Guillaume Charras, and Alexandre Kabla. A unified rheological model for cells and cellularised materials. *Royal Society open science*, 7(1), 2020.
- [17] Francesco Mainardi and Giorgio Spada. Creep, relaxation and viscosity properties for basic fractional models in rheology. *The European Physical Journal Special Topics*, 193(1):133–160, 2011.
- [18] H Schiessel, R Metzler, A Blumen, and TF0921 Nonnenmacher. Generalized viscoelastic models: their fractional equations with solutions. *Journal of physics A: Mathematical and General*, 28(23):6567–6584, 1995.
- [19] William N Findley and Francis A Davis. *Creep and relaxation of nonlinear viscoelastic materials*. Courier corporation, 2013.
- [20] Francesco Mainardi. *Fractional calculus and waves in linear viscoelasticity: an introduction to mathematical models*. World Scientific, 2022.
- [21] Rudolf Gorenflo, Anatoly A Kilbas, Francesco Mainardi, Sergei V Rogosin, et al. *Mittag-Leffler functions, related topics and applications*, volume 1451. Springer, 2020.
- [22] Cleota Gage Fry. *Asymptotic developments of certain integral functions*. Purdue university, 1939.
- [23] Pierre Humbert and Ravi P Agarwal. Sur la fonction de mittag-leffler et quelques-unes de ses généralisations. *Bull. Sci. Math*, 77(2):180–185, 1953.
- [24] Pierre Humbert. Quelques résultats relatifs à la fonction de mittag-leffler. *Comptes rendus hebdomadaires des seances de l academie des sciences*, 236(15):1467–1468, 1953.
- [25] Francesco Mainardi. Why the mittag-leffler function can be considered the queen function of the fractional calculus? *Entropy*, 22(12):1359, 2020.

- [26] Igor Podlubny. *Fractional differential equations: an introduction to fractional derivatives, fractional differential equations, to methods of their solution and some of their applications*, volume 198. elsevier, 1998.
- [27] Changpin Li and Fanhai Zeng. *Numerical methods for fractional calculus*. CRC Press, 2015.
- [28] Nargess Khalilgharibi, Jonathan Fouchard, Nina Asadipour, Ricardo Barrientos, Maria Duda, Alessandra Bonfanti, Amina Yonis, Andrew Harris, Payman Mosaffa, Yasuyuki Fujita, et al. Stress relaxation in epithelial monolayers is controlled by the actomyosin cortex. *Nature physics*, 15(8):839–847, 2019.
- [29] Lynne E Bilston, Zizhen Liu, and Nhan Phan-Thien. Linear viscoelastic properties of bovine brain tissue in shear. *Biorheology*, 34(6):377–385, 1997.
- [30] Bernard D Coleman and Walter Noll. Foundations of linear viscoelasticity. *Reviews of modern physics*, 33(2):239, 1961.
- [31] Danton Gutierrez-Lemini. *Engineering viscoelasticity*, volume 910. Springer, 2014.
- [32] Rudolf Gorenflo. Computation of the mittag-leffler function $e\alpha, \beta(z)$ and its derivative. *Fractional Calculus and Applied Analysis*, 2002.
- [33] Jonathan Louis Kaplan, Alessandra Bonfanti, and Alexandre Kabla. Rheos. jl-a julia package for rheology data analysis. *arXiv preprint arXiv:2005.02538*, 2020.
- [34] Jonathan Louis Kaplan. *Rheological Methods and Their Application to Biological Systems*. PhD thesis, 2020.
- [35] Isaac Y Miranda-Valdez, Aaro Niinistö, Tero Mäkinen, Juha Lejon, Juha Koivisto, and Mikko J Alava. pyrheo: an open-source python package for complex rheology. *Digital Discovery*, 4(4):1075–1082, 2025.
- [36] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. Julia: A fresh approach to numerical computing. *SIAM review*, 59(1):65–98, 2017.
- [37] Dipartimento di Ingegneria Civile e Ambientale (DICA), Politecnico di Milano. Dica, 2026. URL <https://www.dica.polimi.it/it/hpc-at-dica/>.
- [38] Altair Engineering. Openpbs open source workload manager. <https://www.openpbs.org/>. Accessed: 2026-06-13.
- [39] Kai Diethelm and NJ Ford. The analysis of fractional differential equations. *Lecture notes in mathematics*, 2004, 2010.

- [40] Keith Oldham and Jerome Spanier. *The fractional calculus theory and applications of differentiation and integration to arbitrary order*, volume 111. Elsevier, 1974.
- [41] Michele Caputo. Linear models of dissipation whose q is almost frequency independent—ii. *Geophysical journal international*, 13(5):529–539, 1967.
- [42] Mariusz Matusiak. Optimization for software implementation of fractional calculus numerical methods in an embedded system. *Entropy*, 22(5):566, 2020.
- [43] Yufeng Xu and Zhimin He. The short memory principle for solving abel differential equation of fractional order. *Computers & Mathematics with Applications*, 62(12):4796–4805, 2011.
- [44] Simon Kornblith, Martin Holters, Galen Lynch, Viral B Shah, João Felipe Santos, Spencer Russell, Jay Kickliter, Jeff Bezanson, Gudmundur Adalsteinsson, Alex Arslan, et al. Juliadsp/dsp. jl: v0. 8.4. *Zenodo*, 2025.
- [45] Alan V. Oppenheim and Ronald W. Schaffer. *Discrete-Time Signal Processing*. Prentice-Hall, Englewood Cliffs, NJ, 1989.
- [46] Jiahao Chen and Jarrett Revels. Robust benchmarking in noisy environments. *arXiv preprint arXiv:1608.04295*, 2016. URL <https://github.com/JuliaCI/BenchmarkTools.jl>. Julia package version 1.8.
- [47] Lorenzo Fonnesu, Andrea Grassi, Alexandre Kabla, and Alessandra Bonfanti. Numfracdiff. <https://github.com/JuliaRheology/NumFracDiff>, 2026.
- [48] Steven G. Johnson. The NLOpt nonlinear-optimization package. <https://github.com/stevengj/nlopt>, 2007.
- [49] Thomas Harvey Rowan. *Functional stability analysis of numerical algorithms*. PhD thesis, Department of Computer Science, University of Texas at Austin, Austin, TX, 1990.
- [50] M. J. D. Powell. A direct search optimization method that models the objective and constraint functions by linear interpolation. In S. Gomez and J.-P. Hennart, editors, *Advances in Optimization and Numerical Analysis*, volume 275 of *Mathematics and Its Applications*, pages 51–67. Springer, 1994. doi: 10.1007/978-94-015-8330-5_4.
- [51] Alfio Quarteroni, Riccardo Sacco, and Fausto Saleri. *Numerical mathematics*, volume 37. Springer Science & Business Media, 2006.
- [52] Matteo Frigo and Steven G Johnson. The design and implementation of fftw3. *Proceedings of the IEEE*, 93(2):216–231, 2005.

A | Appendix A

In this brief appendix, we explain in detail how to calculate the memory length L , which was discussed in Section 3.3 and is fundamental to the application of the Short Memory Principle, following in particular the approach outlined in [27].

Given a function $f(t)$ defined in the interval $[t_0, T]$, let L be a positive number such that:

$$R(t, L) = {}_{RL}D_{t_0, t}^\alpha f(t) - {}_{RL}D_{t-L, t}^\alpha f(t). \quad (\text{A.1})$$

From [27], we know that the following formula holds:

$$|R(t, L)| = |{}_{RL}D_{t_0, t}^\alpha f(t) - {}_{RL}D_{t-L, t}^\alpha f(t)| \leq \frac{ML^{-\alpha}}{|\Gamma(1-\alpha)|}, \quad t_0 + L \leq t \leq T, \quad (\text{A.2})$$

where $|f(t)| \leq M$ and $M > 0$. Now, we can introduce the tolerance ϵ_0 as a positive value such that

$$|R(t, L)| \leq \epsilon_0.$$

Given these relationships, we can finally manipulate Equation (A.2) to obtain

$$L \geq \left(\left| \frac{M}{\epsilon_0 \Gamma(1-\alpha)} \right| \right)^{\frac{1}{\alpha}}.$$

B | Appendix B

In this appendix, we show the pseudocode related to the prediction of the unknown signal using finite differences, already explained in Section 5.2. This method takes as input the vector **data**, containing the input signal (stress or strain), the constitutive equation of the model and the derivative method we want to use.

Algorithm B.1 Model Prediction via Fractional Differential Equation

Require: **data** (time-series with σ, ϵ, t), **equation** (left/right DE terms), **deriv_method**

Ensure: **computed** (predicted output), **input**, **type**

```

1:  $N \leftarrow \text{length}(\mathbf{t})$ 
2:  $\Delta t \leftarrow t[1] - t[0]$ 
3:
4: if data contains strain then
5:   input  $\leftarrow \epsilon$ ,   unknown  $\leftarrow$  leftDE,   dependency  $\leftarrow$  rightDE
6: else if data contains stress then
7:   input  $\leftarrow \sigma$ ,   unknown  $\leftarrow$  rightDE,   dependency  $\leftarrow$  leftDE
8: end if
9:
10: rhs  $\leftarrow \mathbf{0}_N$ 
11: computed  $\leftarrow \mathbf{0}_N$ 
12: denom  $\leftarrow 0.0$ 
     $\triangleright$  — Phase 1: Build right-hand side from known (dependency) terms —
13: for each term  $c \in$  dependency do
14:   if  $c.order = 0$  then
15:     rhs  $+= c.coef \cdot \mathbf{input}$ 
16:   else
17:     Compute  $D^{c.order}$  input using deriv_method    $\triangleright$  deriv_method: it could be
    any different implementation of the GL method.
18:     rhs  $+= c.coef \cdot D^{c.order} \mathbf{input}$ 
19:   end if
20: end for

```

Algorithm B.1 Model Prediction via Fractional Differential Equation (cont.)

▷ — Phase 2: Build denominator and store GL weights for unknown terms —

```

21:  $\mathcal{W} \leftarrow \{\}$ 
22: for each term  $c \in \text{unknown}$  do
23:   if  $c.order = 0$  then
24:      $denom \mathrel{+}= c.coef$ 
25:   else if  $c.order = 1$  then
26:      $denom \mathrel{+}= c.coef/\Delta t$ 
27:   else if  $c.order \notin \mathbb{Z}$  and  $c.order \notin \mathcal{W}$  then
28:      $denom \mathrel{+}= c.coef/\Delta t^{c.order}$ 
29:     Generate GL weights  $\mathbf{w}^{(c.order)}$  of length  $N$ 
30:      $\mathcal{W}[c.order] \leftarrow \mathbf{w}^{(c.order)}$ 
31:   else
32:      $denom \mathrel{+}= c.coef/\Delta t^{c.order}$ 
33:   end if
34: end for

```

▷ — Phase 3: Solve iteratively for computed output —

```

35:  $computed[0] \leftarrow rhs[0]/denom$ 
36: for  $i \leftarrow 1$  to  $N$  do
37:    $val \leftarrow rhs[i]$ 
38:   for each term  $c \in \text{unknown}$  do
39:     if  $c.order = 1$  then
40:        $val \mathrel{+}= c.coef \cdot computed[i-1]/\Delta t$ 
41:     else if  $c.order \notin \mathbb{Z}$  then
42:        $\mathbf{w} \leftarrow \mathcal{W}[c.order]$ 
43:        $S \leftarrow \sum_{k=0}^{i-1} \mathbf{w}[k+1] \cdot computed[i-k-1]$ 
44:        $val \mathrel{-}= (c.coef/\Delta t^{c.order}) \cdot S$ 
45:     end if
46:   end for
47:    $computed[i] \leftarrow val/denom$ 
48: end for
49: return  $computed$ ,  $input$ ,  $type$ 

```

C | Appendix C

In this appendix, we derive the exact analytical solutions for the stress $\sigma(t)$ for the Maxwell and Springpot viscoelastic models, subjected to the three deformation histories used to validate the algorithms: unit step, ramp and ramp & hold.

The solutions are based on Boltzmann's superposition principle. Given the expressions for the relaxation modulus $G(t)$ for each model, the stress is calculated using the formulas introduced in 5.5.1.

C.1. Maxwell Model

The relaxation modulus for the Maxwell model is defined as:

$$G(t) = ke^{-\frac{k}{\eta}t},$$

where k and η are the model parameters.

Unit Step Function For a unit step input function, the strain is defined by an amplitude ϵ_0 . The stress response is simply proportional to the relaxation modulus:

$$\sigma(t) = \epsilon_0 G(t) = \epsilon_0 ke^{-\frac{k}{\eta}t}. \quad (\text{C.1})$$

Since we are considering a unitary step, the amplitude ϵ_0 is equal to 1.0, which means that the stress is:

$$\sigma(t) = ke^{-\frac{k}{\eta}t}. \quad (\text{C.2})$$

Ramp Function For a ramp deformation with a constant strain rate ϵ_r , the stress is calculated using the convolution integral:

$$\sigma(t) = \epsilon_r \int_0^t G(t - \tau) d\tau = \epsilon_r \int_0^t ke^{-\frac{k}{\eta}(t-\tau)} d\tau \quad (\text{C.3})$$

Extracting the independent terms, we get:

$$\sigma(t) = \epsilon_r k e^{-\frac{k}{\eta}t} \int_0^t e^{\frac{k}{\eta}\tau} d\tau \quad (\text{C.4})$$

Solving the integral yields the exact analytical solution:

$$\sigma(t) = \epsilon_r k e^{-\frac{k}{\eta}t} \left[\frac{\eta}{k} e^{\frac{k}{\eta}\tau} \right]_0^t = \epsilon_r \eta \left(1 - e^{-\frac{k}{\eta}t} \right) \quad (\text{C.5})$$

Ramp and Hold Function This deformation history is divided into two transient phases. For $t \leq t_1$ (ramp phase), the solution is identical to the standard ramp response derived above. For the holding phase ($t > t_1$), the strain remains constant, meaning the upper limit of the convolution integral becomes t_1 :

$$\sigma(t) = \epsilon_r \int_0^{t_1} k e^{-\frac{k}{\eta}(t-\tau)} d\tau = \epsilon_r k e^{-\frac{k}{\eta}t} \int_0^{t_1} e^{\frac{k}{\eta}\tau} d\tau \quad (\text{C.6})$$

Evaluating the integral gives:

$$\sigma(t) = \epsilon_r k e^{-\frac{k}{\eta}t} \left[\frac{\eta}{k} e^{\frac{k}{\eta}\tau} \right]_0^{t_1} = \epsilon_r \eta \left(e^{\frac{k}{\eta}(t_1-t)} - e^{-\frac{k}{\eta}t} \right) \quad (\text{C.7})$$

Alternatively, applying Boltzmann's superposition principle (which directly reflects the numerical implementation strategy), the stress during the holding phase can be expressed as the difference between two delayed ramp responses:

$$\sigma(t) = \sigma_{\text{ramp}}(t) - \sigma_{\text{ramp}}(t - t_1) = \epsilon_r \eta \left(1 - e^{-\frac{k}{\eta}t} \right) - \epsilon_r \eta \left(1 - e^{-\frac{k}{\eta}(t-t_1)} \right) \quad (\text{C.8})$$

which algebraically simplifies to the exact same closed-form result.

C.2. Springpot Model

The relaxation modulus for the Springpot model is defined as:

$$G(t) = \frac{c_\beta}{\Gamma(1-\beta)} t^{-\beta},$$

where c_β and β are the model parameters.

Unit Step Function For a unit step input function, the strain is defined, as in the previous case, by an amplitude ϵ_0 , which we consider equal to 1.0. In this way, the stress

response is:

$$\sigma(t) = G(t) = \frac{c_\beta}{\Gamma(1-\beta)} t^{-\beta} \quad (\text{C.9})$$

Ramp Function Knowing that for a ramp deformation with a constant strain rate ϵ_r , the stress is computed as:

$$\sigma(t) = \epsilon_r \int_0^t G(t-\tau) d\tau, \quad (\text{C.10})$$

by replacing the Springpot relaxation modulus we obtain:

$$\sigma(t) = \epsilon_r \int_0^t \frac{c_\beta}{\Gamma(1-\beta)} (t-\tau)^{-\beta} d\tau = \frac{\epsilon_r c_\beta}{\Gamma(1-\beta)} \int_0^t (t-\tau)^{-\beta} d\tau \quad (\text{C.11})$$

Evaluating the integral, we obtain:

$$\sigma(t) = \frac{\epsilon_r c_\beta}{\Gamma(1-\beta)} \left[-\frac{(t-\tau)^{1-\beta}}{1-\beta} \right]_0^t = \frac{\epsilon_r c_\beta}{\Gamma(2-\beta)} t^{1-\beta} \quad (\text{C.12})$$

Ramp and Hold Function As with the Maxwell model, the stress response is divided in two phases: the ramp phase ($t \leq t_1$), in which the stress is equal to the standard ramp, and the holding phase ($t > t_1$), where the stress becomes as follows:

$$\sigma(t) = \epsilon_r \int_0^{t_1} \frac{c_\beta}{\Gamma(1-\beta)} (t-\tau)^{-\beta} d\tau = \frac{\epsilon_r c_\beta}{\Gamma(1-\beta)} \int_0^{t_1} (t-\tau)^{-\beta} d\tau \quad (\text{C.13})$$

Evaluating it, let us obtain:

$$\sigma(t) = \frac{\epsilon_r c_\beta}{\Gamma(1-\beta)} \left[-\frac{(t-\tau)^{1-\beta}}{1-\beta} \right]_0^{t_1} = \frac{\epsilon_r c_\beta}{\Gamma(2-\beta)} (t^{1-\beta} - (t-t_1)^{1-\beta}). \quad (\text{C.14})$$

List of Figures

1.1	Qualitative spectrum of mechanical behaviour ranging from purely viscous liquids to purely elastic solids. Real materials populate the intermediate region, behaving as viscoelastic liquids (e.g., ketchup) or viscoelastic solids (e.g., biological cells, gelatin), depending on the relative balance of viscous and elastic contributions.	3
2.1	Relaxation test using a constant strain ε as input, adapted from [7]. The image also illustrates the first linearity assumption between stress and strain.	11
2.2	Creep test using a constant stress σ as input, adapted from [7]. The image also illustrates the first linearity assumption between stress and strain.	12
2.3	Application of the Boltzmann Superposition Effect on a constant strain ε input, adapted from [7]. As can be seen, the effect of the sum between two different strain functions applies in the same way to the corresponding stress.	12
2.4	Visual representation of basic elastic and viscous components. They can be arranged together the same way as digital circuits.	14
2.5	Schematic representation of the Maxwell model.	15
2.6	Schematic representation of the Kelvin Voigt model.	18
2.7	Schematic representation of the Generalized Maxwell model	20
2.8	Visual representation of springpot component and its relation to spring and dashpot.	21
2.9	Schematic representation of the General Fractional Maxwell model.	25
2.10	Schematic representation of the General Fractional Kelvin-Voigt model.	27
2.11	Log-log comparison of the Mittag-Leffler function (solid grey) with exponential (dashed green) and power-law (dashed red) decays for fractional parameters $\alpha = 0.25$, $\alpha = 0.5$ and $\beta = 1.0$. The graphs highlight how, for $\beta = 1.0$, the function interpolates between an exponential-like relaxation at short times and a power-law tail at long times.	29

3.1	Visual comparison between Short (top) and Full (bottom) Memory approaches. In the Short Memory principle, the lower bound of the fractional derivative is shifted from the initial time t_0 to a moving truncated bound $t^* = t - L$, accounting only for the recent history of length L . In the Full Memory approach, the entire history from t_0 to t is considered.	42
3.2	Baseline	46
3.3	StepOverTimeSM	47
3.4	Comparison between baseline implementations (left: serial version; right: parallel version with 32 threads) using Unit Step input function, a constant time step $\Delta t = 0.001$ ($N = 10001$ steps, $t \in [0, 10]$) and order $\alpha = 0.5$. Execution time is handled by BenchmarkTools.jl [46].	48
3.5	Comparison between Grünwald-Letnikov Short Memory implementations (left: serial version; right: parallel version with 32 thread) using Unit Step input function, a constant time step $\Delta t = 0.01$ ($N = 10001$ steps, $t \in [0, 100]$), order $\alpha = 0.5$ and $\epsilon_0 = 0.1$. Execution time is handled by BenchmarkTools.jl [46].	49
3.6	Comparison between each Grünwald-Letnikov implementation (32 threads) using Step input function, a constant time step $\Delta t = 0.01$ ($N = 10001$ steps, $t \in [0, 100]$), order $\alpha = 0.5$ and $\epsilon_0 = 0.1$ for the Short Memory approaches. Execution time is handled by BenchmarkTools.jl [46].	50
3.7	Scaling	52
3.8	Logo of the NumFracDiff library	53
4.1	Average time of a single cost function execution for a Maxwell model on a ramp input. Comparison between Convolution, Derivative with short memory and FFT. Fractional method is executed parallel with 32 thread, $\Delta t = 0.001$ and vector length is 10001. Execution time is handled by BenchmarkTools.jl [46].	65
4.2	Average time of a single cost function execution for a Fractional Maxwell model on a ramp input. Comparison between Convolution, Derivative with short memory and FFT. Fractional method is executed parallel with 32 thread, $\Delta t = 0.001$ and vector length is 10001. Time is handled by BenchmarkTools.jl [46]	65

4.3	Maxwell model comparison between RHEOS, fractional model with and without Short Memory and FFT. Input data is a ramp with a 0.8 slope, on a 10 second window with $\Delta t = 0.001$, resulting vector length 10001, for the strain, while the stress is calculated with RHEOS' modelpredict. Time is handled by BenchmarkTools.jl [46], while error reported is returned by NLOpt optimization process.	67
4.4	Fractional Maxwell model comparison between RHEOS, fractional model with and without Short Memory and FFT. Input data is a ramp with a 0.8 slope, on a 10 second window with $\Delta t = 0.001$, resulting in vector length 10001, for the strain, while the stress is calculated with RHEOS' modelpredict. Time is handled by BenchmarkTools.jl [46], while error reported is returned by NLOpt optimization process. Note that Convolution error has been calculated by taking output parameters and using them in the cost function used for the differential methods, to have a coherent validation score.	68
5.1	Maxwell	80
5.2	Springpot	81
5.3	Execution times (log scale) for stress and strain prediction using the standard integer-order Maxwell model and $\Delta t = 0.01$ ($N = 10001$ steps, $t \in [0, 100]$). Execution time is handled by BenchmarkTools.jl [46]. The parallel implementation was executed with 32 threads.	82
5.4	Schematic representation of the Fractional Kelvin Voigt (left) and Fractional Zener (right).	83
5.5	Execution times (log scale) for stress and strain prediction using the Fractional Kelvin-Voigt model and $\Delta t = 0.01$ ($N = 10001$ steps, $t \in [0, 100]$). Execution time is handled by BenchmarkTools.jl [46]. The parallel implementation was executed with 32 threads.	84
5.6	Execution times (log scale) for stress and strain prediction using the Fractional Zener model and $\Delta t = 0.01$ ($N = 10001$ steps, $t \in [0, 100]$). Execution time is handled by BenchmarkTools.jl [46]. The parallel implementation was executed with 32 threads.	85

List of Tables

3.1	Summary of execution times (serial and 32-thread parallel) and qualitative error behaviour for the tested fractional derivative implementations, evaluated on the Unit Step input function with $\alpha = 0.5$ and a constant time step $\Delta t = 0.01$ ($N = 10001$ steps, $t \in [0, 100]$)	55
4.1	Efficiency and environmental impact of numerical methods for fitting. Simulations were run on a ramp input with slope 0.8, on a 10 second window with $\Delta t = 0.001$, resulting in vector length 10001, for the strain, while the stress is calculated with RHEOS' modelpredict.	69
4.2	Summary of execution time and RMSE relative to the four methods for the two benchmarked models, evaluated with Ramp input function (slope equal to 0.8), on a 10 second window with $\Delta t = 0.001$, resulting in vector length 10001, for the strain, while the stress is calculated with RHEOS' modelpredict.	70
5.1	Parameters used for the different viscoelastic models in the prediction tests.	71
5.2	Efficiency and environmental impact of numerical methods for predicting. Simulations were run on a ramp input data with slope 0.8, on a time period going from 0 to 10 seconds with a $\Delta t = 0.001$, resulting in a vector length of 10001. The input data is stress imposed and its calculating strain. . . .	86
5.3	Summary of execution time and speedup relative to the original RHEOS convolution for the three benchmarked models, evaluated with Unit Step input function and a constant time step $\Delta t = 0.01$ ($N = 10001$ steps, $t \in [0, 100]$). Parallel executions are evaluated using 32 threads. *Parallel GL only; serial GL shows no advantage. **Speedup for Serial GL.	87

