MASTER'S THESIS
ARTIFICIAL INTELLIGENCE

Fractional Order Distributed Optimization: Theory, Applications, and Empirical Validation

Author:
Andrei Lixandru
s1053555
andrei.lixandru@ru.nl

Daily Supervisor:
dr. Sérgio Pequito
sergio.pequito@tecnico.ulisboa.pt

Internal Supervisor:
prof. dr. Marcel van Gerven
marcel.vangerven@donders.ru.nl

Independent Second Reader:
Yuzhen Qin
yuzhen.qin@donders.ru.nl

February 6, 2025

Contents

Abstract

Distributed optimization (DO) is fundamental to modern machine learning applications like federated learning, but existing methods often struggle with ill-conditioned problems and face stability-versus-speed tradeoffs. In this thesis manuscript, we provide a theoretical background on DO and fractional calculus, and introduce fractional order distributed optimization (FrODO) - a theoretically-grounded framework that incorporates fractional-order memory terms to enhance convergence properties in challenging optimization landscapes. We provide a convergence proof for our algorithm and show through empirical validation that our algorithm achieves competitive performance with state of the art methodology.

Chapter 1

Distributed Optimization

1.1 Problem Statement

Advancements in communication, sensing, and digital technologies over recent years have given rise to networked systems, which consist of numerous interconnected subsystems (referred to as agents). These agents collaborate to achieve a common global objective. Networked systems find application in a wide range of domains, including federated learning [14], distributed multi-agent coordination [18], sensor networks [27], and resource allocation [25].

Many challenges in networked systems can be reformulated as convex optimization problems [6]. However, conventional centralized methods often fail to address these challenges effectively due to the distributed nature of the systems. Centralized optimization approaches suffer from inherent drawbacks, such as limited scalability, excessive computational requirements, high communication costs, and vulnerability to single points of failure. These limitations highlight the importance of distributed approaches for solving optimization problems more robustly and efficiently [6, 31].

In distributed optimization, a network of N agents is considered, where each agent i is associated with a private local convex objective function $f_i(x)$, with $x \in \mathbb{R}^n$ denoting the optimization variable. The aim of a distributed optimization algorithm is for all agents in the system to converge on the same x^*:

$$x^* = \arg\min_{x \in \mathbb{R}^n} \sum_{i=1}^{N} f_i(x)\,, \tag{1.1}$$

to be attained by the exchange of local estimates of the agent i optimum $\hat{x}_i \in \mathbb{R}^n$. The exchange occurs with the (out-)neighbors of agent i denoted by $\mathcal{N}_i^+$ and then an agent $j \in \mathcal{N}_i^+$ will update its own estimate $\hat{x}_j$ as a function $\mathcal{F}$ of its previous estimates, the optimization performed locally, and the incoming local optimum estimates of its (in-)neighbors $\mathcal{N}_j^-$.

Distributed optimization algorithms are primarily categorized into two types: discrete-time and continuous-time approaches. Both approaches involve agents maintaining a dynamic state, which represents their estimate of the optimization variable. The focus of this work is on discrete optimzation methodology.

1.2 Existing Work

Distributed optimization has emerged as a crucial area of research, particularly in the context of large-scale problems characterized by big data and high dimensionality [30]. The primary motivation for distributed optimization lies in its ability to leverage multiple computing agents that collaboratively solve optimization problems while each agent has access only to local data. This decentralized approach not only enhances computational efficiency but also addresses pressing concerns related to data privacy and communication overhead.

Historically, optimization problems were predominantly addressed through centralized methods, where a single entity would gather all relevant data and perform the necessary computations. However, as the complexity and size of datasets grew, these centralized approaches began to show significant limitations, particularly in terms of scalability and robustness. The emergence of distributed optimization was driven by the need to handle problems that are too large or complex for a single computational unit [15].

Recent advancements in the field have led to the development of various distributed algorithms that operate over networks of interconnected agents. These algorithms can be categorized into several classes. First-order methods, for instance, utilize gradient information to guide the optimization process and the algorithm proposed in Chapter 3. Another prominent framework is the alternating direction method of multipliers [6], which decomposes complex optimization problems into smaller subproblems that can be solved independently by different agents, thereby enabling parallel computation. Additionally, consensus-based methods aim to achieve agreement among agents on the optimal solution by iteratively sharing information with neighboring nodes.

Despite significant progress, distributed optimization still faces several key challenges. One such challenge is asynchrony, as many existing algorithms assume synchronous updates among agents. This assumption is often impractical in real-world scenarios, where communication delays and agent failures can occur [4]. Also, ensuring the robustness of distributed algorithms in the presence of network disruptions or adversarial conditions is critical for their practical deployment [32].

There is a plethora of DO algorithms which vary across a number dimensions such as the objective function they are suited for optimizing or their convergence rate. We included a number of DO algorithms for a comparative analysis in Table 1.1.

Approach	**Step-size**	**Objective Function**	**Communication Graph**	**Convergence**
distributed composite optimization [39]	fixed	convex but nonsmooth	connected, undirected	sublinear
distributed (sub)gradient method [25]	diminishing	convex but nonsmooth	uniformly jointly connected	$O(ln(k)/\sqrt{k})$
distributed dual averaging [1]	diminishing	convex but nonsmooth	connected undirected	$O(1/k^2)$
distributed stochastic subgradient method [33]	diminishing	convex but nonsmooth	undirected graphs with random link failures	almost sure
distributed PI [3]	fixed	convex and smooth	connected undirected	$O(1/k)$
push–pull based method [26]	fixed	strongly convex and smooth	strongly connected	linear

Table 1.1: Comparison of distributed optimization methods.

Distributed optimization still encounters substantial challenges, particularly when addressing optimization problems with ill-conditioned objective functions characterized by adverse curvature properties. These issues can significantly degrade convergence rates and create a tradeoff between achieving stability and maintaining fast convergence [40]. The methodology proposed in Chapter 3 aims to resolve these challenges effectively.

1.3 Applications

1.3.1 Application Domains

Distributed optimization plays an essential role in a broad array of modern technological systems, particularly in scenarios where multiple agents must work collaboratively to solve complex optimization tasks. Its application spans several critical domains, where the decentralized nature of the optimization process offers substantial benefits in terms of scalability, efficiency, and robustness.

One of the key domains in which distributed optimization is applied is power systems [20]. In the context of smart grids, distributed optimization is used to facilitate energy management and load balancing. This enables efficient resource allocation, which is crucial for minimizing energy waste and enhancing the reliability and resilience of the grid, thereby reducing the risk of system failures.

In the area of sensor networks [27], distributed optimization plays a crucial role in applications such as environmental monitoring and autonomous vehicle coordination. Here, the ability of agents to collaboratively process data allows for a more efficient use of resources, minimizing communication overhead and enabling real-time decision-making. This is especially valuable in scenarios where timely responses are necessary, and the network is constrained by bandwidth limitations.

Another significant application is in federated learning [17], where distributed optimization facilitates the collaborative training of machine learning models across decentralized devices. This paradigm preserves data privacy by ensuring that raw data never leaves the local devices. Federated learning allows for efficient model training on large-scale datasets while ensuring compliance with privacy regulations, as noted by Konečný *et al.* [14].

In smart manufacturing [37], distributed optimization enhances the coordination between machines and systems within a factory environment. This optimization is essential for tasks such as production scheduling, resource allocation, and energy management, ultimately improving operational efficiency and reducing energy consumption in industrial settings.

Finally, in the domain of multi-agent coordination [12], distributed optimization aids in the coordination of autonomous systems, such as robots or drones. It is used for solving complex coordination problems, including path planning and task allocation, ensuring that multiple agents can achieve their shared objectives while minimizing conflicts and maximizing efficiency.

These diverse applications underscore the versatility of distributed optimization. By enabling efficient, scalable, and robust solutions across a variety of domains, distributed optimization remains a foundational technology in the development of modern networked systems.

1.3.2 Federated Learning in Healthcare

Federated learning (FL) has emerged as a transformative approach to machine learning, particularly in the healthcare sector, where privacy and data security are of utmost importance [35]. This method allows multiple healthcare institutions, such as hospitals and research centers, to collaboratively train machine learning models without sharing raw patient data. Instead, each institution retains control over its local data and only shares model parameters or derived insights. This decentralization of data not only addresses privacy concerns but also enables the aggregation of diverse datasets, which can significantly enhance the performance and generalizability of predictive models [14].

FL has found significant applications in clinical research, particularly in fields where data is often siloed across multiple institutions. In radiology and medical imaging, for example, FL has been utilized to develop models for diagnosing conditions such as brain tumors [24] and predicting COVID-19 outcomes [10] from chest X-ray images. These applications benefit from the ability to aggregate knowledge across institutions while ensuring patient confidentiality. This key feature of FL is illustrated in 1.1. Similarly, FL has proven useful in studies involving electronic health records. By enabling multicenter collaboration without compromising data security, FL has supported the development of predictive models for clinical outcomes such as hospitalization risks [7].

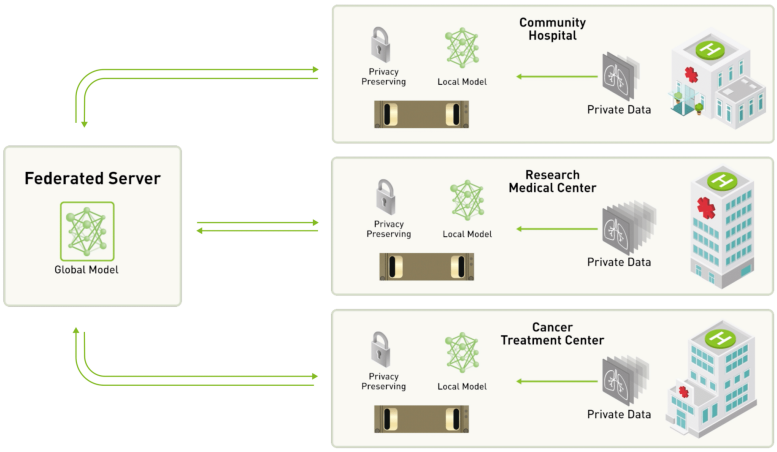

Figure 1.1: Possible workflow of FL in healthcare (figure taken from [28]).

Another notable application of FL in healthcare is within the realm of mobile health. As the use of mobile devices for health monitoring continues to grow, FL facilitates remote monitoring [38] and diagnostic support by allowing health data from mobile applications to be used collaboratively. This enables real-time health status tracking and provides diagnostic insights across diverse populations, all while maintaining the privacy of patient data.

Despite its promise, the deployment of FL in healthcare faces several challenges. One key issue is the technical barrier posed by communication costs, statistical heterogeneity of datasets, and system differences among institutions. Addressing these concerns requires ongoing advancements in model personalization techniques and efficient communication protocols. Moreover, regulatory challenges, such as compliance with data privacy laws like HIPAA in the United States or GDPR in Europe, complicate the integration of FL into clinical practice. Furthermore, while many studies have demonstrated the feasibility of FL, only a small fraction have transitioned to real-world applications, underscoring

the need for further research to bridge the gap between theoretical models and practical implementations.

In conclusion, federated learning holds great potential to advance healthcare research by enabling collaborative model training while preserving patient privacy. As technical and regulatory hurdles are overcome, FL is poised to become an integral tool in enhancing the efficiency and accuracy of healthcare applications, leading to improved patient outcomes and more robust predictive models. This thesis contributes to this effort by proposing in Chapter 3 an efficient and stable DO algorithm to be used in this setting.

Chapter 2

Fractional Calculus

2.1 Theoretical Background

Fractional calculus is a branch of mathematical analysis that extends the concepts of differentiation and integration to non-integer (fractional) orders [8]. This generalization allows for the definition of fractional derivatives and integrals, which can be applied to a wide range of problems in various scientific fields, including physics, engineering, and finance.

The origins of fractional calculus can be traced back to the correspondence between mathematicians such as L'Hôpital and Leibniz in the late 17th century, where the notion of fractional derivatives was first contemplated. Over the centuries, significant contributions have been made by various mathematicians, notably Riemann and Liouville, who formalized definitions for fractional integrals and derivatives [11]. The Riemann-Liouville definition is one of the most commonly used frameworks, although it has been complemented by other definitions such as the Caputo and Hadamard derivatives, each with its unique properties and applications [2].

In fractional calculus, the order of differentiation or integration can be any real or complex number. This flexibility enables fractional calculus to model phenomena that exhibit memory effects or long-range dependence, characteristics often observed in natural systems. The mathematical formulation involves operators that are non-local in nature, meaning that the value of a fractional derivative at a point depends not only on values at that point but also on values over an interval.

The Riemann-Liouville variant of fractional derivatives is defined as:

$$I^{\lambda} f(x) = \frac{1}{\Gamma(\lambda)} \int_{a}^{x} (x-t)^{n-1} f(t) dt, \tag{2.1}$$

where $x \in \mathbb{R}^n$, Γ is the well-known gamma function, $\lambda \in \mathbb{R}_0^+$ is the fractional order and a an arbitrary but fixed base point. The operation I^{λ} takes a function $f(x)$ defined on $[a, x]$ and produces another function $I^{\lambda} f(x)$, which represents the fractional integral of order λ of $f(x)$ up to x. The expression can be adapted for the discrete case as:

$$I^{\lambda} f(x) \approx \frac{1}{\Gamma(\lambda)} \sum_{t=a}^{x} (x-t)^{\lambda-1} f(t) \cdot \Delta t. \tag{2.2}$$

where Δt is the step size. As we see from the expression, the fractional derivative of a function at a particular point can be thought of as a weighted sum of the derivatives of

the points between a base point a and our point of interest x. In Figure 2.1 we illustrate the magnitude of the fractional weights of the elements of discrete fractional derivatives of different fractional orders, all with a base point $a = 9$ and a step size $\Delta t = 1$.

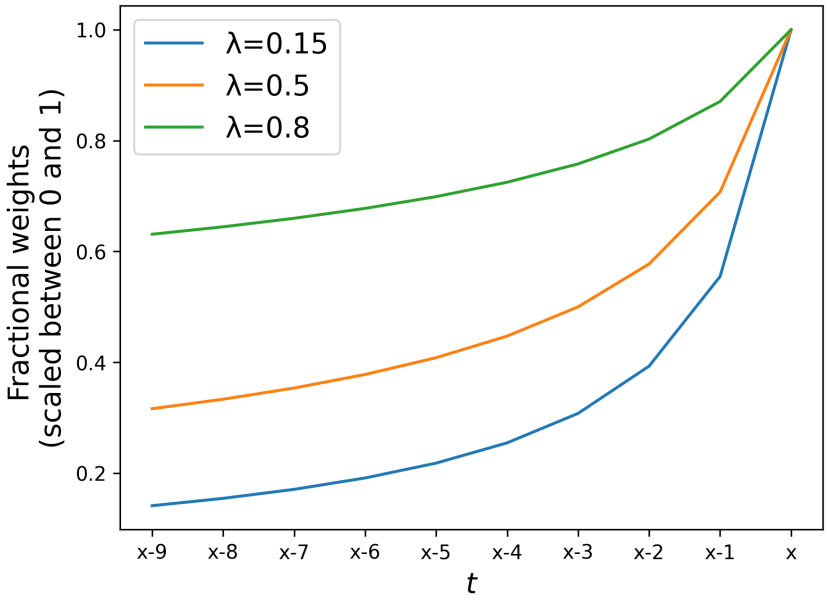

Figure 2.1: Fractional weights.

2.2 Applications

Fractional calculus has found applications across numerous disciplines due to its ability to model complex systems more accurately than traditional integer-order models. Some notable applications include physics, control theory and machine learning optimization.

Fractional calculus has found significant applications in various areas of physics, particularly in modeling complex phenomena that exhibit non-local behavior and memory effects. One notable application is in Newtonian mechanics, where fractional derivatives can generalize Newton's second law, allowing for the analysis of motion under constant forces and the exploration of fractional gravitational potentials. This approach aids in addressing challenges such as galactic rotation curves, which are often attributed to dark matter issues [36].

Additionally, fractional calculus is instrumental in describing anomalous diffusion, where particles do not follow classical diffusion patterns, and in the study of electrical impedance in complex fluids, which provides insights into the scaling properties of signals [5]. Overall, the versatility of fractional calculus makes it a powerful tool for advancing theoretical and applied physics across multiple disciplines.

Fractional calculus has emerged as a powerful tool in control theory, particularly in the design and implementation of fractional-order controllers that enhance system performance and robustness. Unlike traditional integer-order controllers, fractional-order controllers, such as fractional PID (Proportional-Integral-Derivative) controllers [19], incorporate non-integer derivatives, allowing for greater flexibility in tuning and optimization. This capability enables better handling of complex dynamic systems characterized by long-term memory effects and non-linear behaviors. Applications of fractional calculus in control theory span various fields, including robotics, aerospace [21], and process

control, smart beams [23] and refrigeration systems [22].

Finally, fractional calculus has also been used in the field of machine learning optimization. A recent study has shown that fractional calculus methods, which account for long-term memory, are effective in stabilizing optimization trajectories for complex objective functions [9].

Chapter 3

Fractional Order Distributed Optimization

3.1 Algorithm

We introduce fractional order distributed optimization (FrODO), a new algorithm that generalizes conventional proportional-integrative controllers to distributed frameworks by incorporating fractional-order memory terms. As described in Algorithm 1, FrODO operates through three primary steps: (1) calculation of descent directions using gradient and memory components, (2) updating of local states guided by these directions and (3) aligning of states among neighboring agents to achieve consensus.

In Step 1, one computes the gradient term and the memory feedback term of each agent. The term $g_i : \mathbb{R}^n \to \mathbb{R}^n$ shortly denotes the gradient $\nabla f_i(x_i)$ of a differentiable objective $f_i : \mathbb{R}^n \to \mathbb{R}$. The memory feedback term is computed as

$$M_i^{(k)} = \sum_{n=1}^{T} \mu(n;\lambda) \cdot g_i^{(k-n)},$$

and used to enhance the optimization process by the use of long-term memory. Specifically, it is a weighted sum of past gradient terms, weighted by $\mu(n;\lambda)$, which is defined as

$$\mu_0(n;\lambda) = \frac{1}{\Gamma(\lambda)} \cdot \frac{1}{n^{1-\lambda}} \quad \text{and} \quad \mu(n;\lambda) = \frac{\mu_0(n;\lambda)}{\max_{n\in\{1,\dots,T\}} \mu_0(n;\lambda)}\,, \tag{3.1}$$

with $\mu_0(n;\lambda) \in \mathbb{R}$ encapsulating a power-law decaying function with respect to $\lambda \in \mathbb{R}_0^+$ – the fractional-order exponent – that weighs the past gradient terms based on their distance in time from the current iteration, and the 'normalized´ version of it is given by $\mu(n;\lambda) \in [0,1]$.

where k is the current optimization step, T the memory length and g the gradient of the agent state with respect to its optimization function.

We have the weighting function defined as $\mu(n;\lambda) = \mu_0(n;\lambda)/\max_n \mu_0(n;\lambda)$, with

$$\mu_0(n;\lambda) = \frac{1}{\Gamma(\lambda)} \cdot \frac{1}{n^{1-\lambda}}$$

.

$\mu_0(n;\lambda)$ is a power-law decay function parameterized by $\lambda \in \mathbb{R}_0^+$, the fractional-order exponent, to reflect the temporal proximity of previous gradient terms to the current iteration. After some derivations, our memory component looks like the discretized variation

of a fractional derivative from (2.2) in which we have a step size of 1 and instead of $f'(x)$ we have g.

In Step 2, we update the local states of the agents with

$$x_i = x_i - \alpha g_i - \beta M_i,$$

where $\alpha \in \mathbb{R}^n$ and $\beta \in \mathbb{R}^n$ are hyperparameters that adjust the impact of the gradient and memory feedback terms respectively.

In Step 3, the agent states are aligned by performing an average as follows:

$$x_i \leftarrow \frac{1}{|\mathcal{N}_i^-|} \sum_{j \in \mathcal{N}_i^-} x_j \,. \tag{3.2}$$

Besides these three steps, the algorithm also includes an alignment on the first iteration using (3.2), to adjust for cases in which agents start from different states, which would have problematic effects for the gradient memory feedback if left unaddressed.

Algorithm 1 FrODO: Fractional order distributed optimization

Require:

Number of communication rounds K; In-neighbors of each agent $\mathcal{N}_i^-$ for $1 \leq i \leq N$; Initial states $\mathbf{x} = \{x_1, \ldots, x_N\}$; Private objective functions $\mathbf{f} = \{f_1, \ldots, f_N\} : \{\mathbb{R}^n \to \mathbb{R}\}^N$; Gradient term magnitude$\alpha \in \mathbb{R}^+$; Memory feedback magnitude $\beta \in \mathbb{R}^+$; Memory length of memory feedback $T \in \mathbb{N}^+$; Fractional order exponent $\lambda \in (0, 1)$.

for k in $\{1, \ldots, K\}$ **do**

  if $k > 1$ **then**

    $g_i \leftarrow \nabla f_i(x_i),\ i \in \{1, \ldots, N\}$ ▷ Gradient computation

    $M_i \leftarrow \sum_{n=1}^{T} \mu(n; \lambda) \cdot g_i^{(k-n)},\ i \in \{1, \ldots, N\}$ ▷ Memory feedback

    $x_i \leftarrow x_i - \alpha g_i - \beta M_i,\ i \in \{1, \ldots, N\}$ ▷ Gradient descent with memory

  end if

  $x_i \leftarrow \frac{1}{|\mathcal{N}_i^-|} \sum_{j \in \mathcal{N}_i^-} x_j \ ,\ i \in \{1, \ldots, N\}$ ▷ Alignment of network agents' states

end for

3.2 Discrete Time Dynamics

To illustrate the dynamics of FrODO, we examine its centralized version with a single agent containing only one parameter. We frame our system as the state-space model:

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k \tag{3.3}$$

where $\mathbf{B}$ represents the transformations applied on $g^{(k)}$ to apply Step 2 of Algorithm 1. Specifically, we have $\mathbf{B} = \begin{bmatrix} -\alpha & -\beta M_i^{(k)} \end{bmatrix}$, where $\mathbf{u}_k = \begin{bmatrix} g^{(k)} & 1 \end{bmatrix}$, and $\mathbf{A}$ is a 1×1 identity matrix.

We can see the dynamics of FrODO illustrated in Figure 3.2, where it is applied in controlling different types of noise sequences. Here, the noise sequences replace the gradient signal $g^{(k)}$ used in the context of optimization.

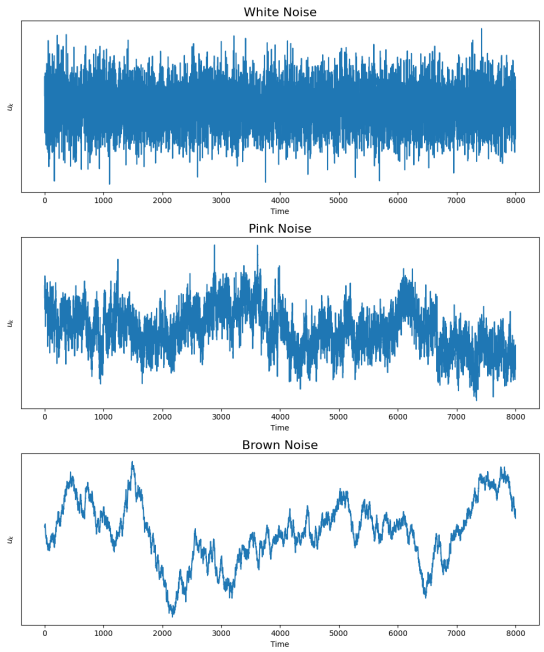

Figure 3.1: Types of inputs.

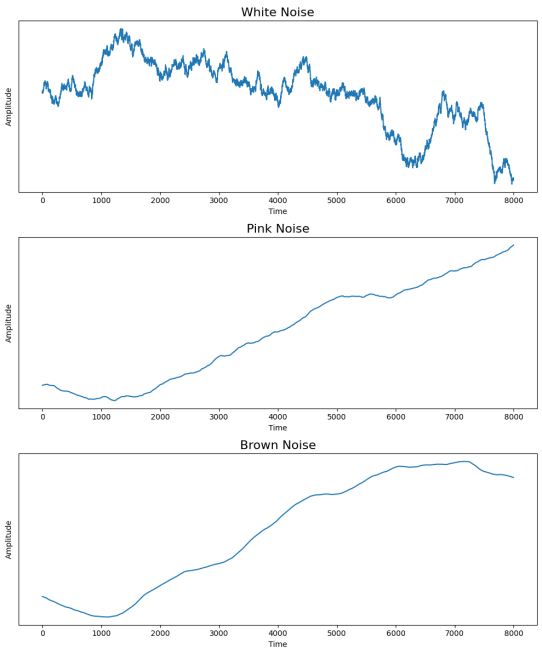

Figure 3.2: Controlled inputs.

Figure 3.3: Qualitative analysis of FrODO dynamics.

Figure 3.2 helps to illustrate qualitatively the nature of a controller with fractional derivatives. Here, we observe a smoothing effect of the controller on the input signal, which might be due to the long term memory aspect of fractional derivatives. That is, the control of a current time point staying close to the values of previous controls.

3.3 Experiments

In the following subsections, we examine three distinct experiments, assuming a fully connected network of agents for simplicity in describing the setup. We start with a pedagogical example using a quadratic function with an ill-defined Hessian, a benchmark analysis with the well-known Rosenbrock function [29], and an application-driven experiment in federated learning, where we aim to train an artificial neural network across multiple agents.

3.3.1 Experiment 1: Objective with an ill-defined Hessian

We start by exploring a pedagogical example that illustrates the effectiveness of our algorithm in a DO setting with four agents, each having the following objective functions with ill-conditioned Hessian matrices described as follows:

$$f_1(x_1, x_2) = 0.5(2 - x_1)^2 + 0.005x_2^2, \quad f_2(x_1, x_2) = 0.5(2 + x_1)^2 + 0.005x_2^2,$$
$$f_3(x_1, x_2) = 0.5x_1^2 + 0.005(2 - x_2^2), \text{ and } f_4(x_1, x_2) = 0.5x_1^2 + 0.005(2 + x_2^2)\,.$$

The global objective as in (1.1) amounts to $f_{global}(x_1, x_2) = x_1^2 + 0.02x_2^2 + 4.04$, which has an ill-conditioned Hessian matrix and a global minimum at $(0, 0)$. The simulation setup is illustrated in Figure 3.4.

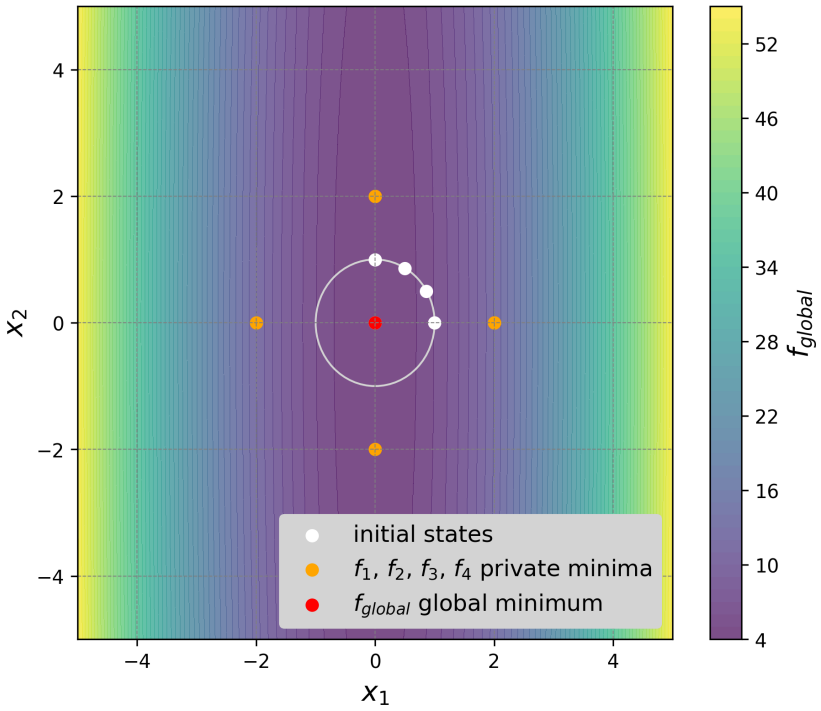

Figure 3.4: Heatmap of the global objective in Experiment 1

At each iteration, the agents start from the same state on one of the points on the unit circle, with coordinates (1, 0), (0.86, 0.5), (0.5, 0.86), and (0, 1). The coordinate (1, 0) corresponds to the initial condition in which the agents have the steepest initial gradient, for which the fastest convergence is expected. Conversely, (0, 1) is the initial condition with the flattest initial gradient, for which the slowest convergence is expected.

A hundred sets of hyperparameters have been sampled, which our 3 algorithms shared across 100 runs across the 4 initial conditions on our setup. The Fractional algorithm represents our long-term memory approach, the No Memory algorithm is our proposed algorithm with $\beta = 0$ and Heavy Ball is our algorithm with $T = 1$. Hyperparameter ranges are $\alpha \in [0.6, 1]$, $\beta \in [\alpha/1.5, \alpha/2.5]$, $\lambda \in [0.1, 0.2]$ and $T \in [80, 100]$. The barplots in Figure 3.5 represent the mean, range and interquartile range of iterations needed to converge across 100 runs for a given condition. Using two-sided Kolmogorov-Smirnov (KS) tests, we determined that for the fractional condition there is no significant difference in performance between starting on the steepest and flattest initial gradient ($p = 1$). The other two types of memory show a significant difference in performance across these two initial conditions ($p < 0.00001$).

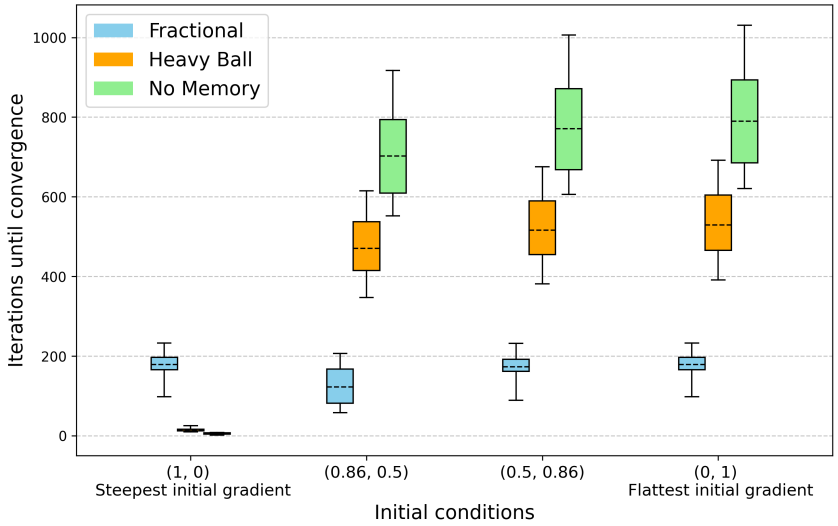

Figure 3.5: Performance plot Experiment 1

A further test was conducted, in which the initial states of individual agents were sampled from a uniform distribution of coordinates on the unit circle. The iteration counts until convergence are 427±145 for Fractional, 1,538±400 for Heavy Ball and 1,864±312 for No Memory. The results show the advantage fractional memory has in optimization conditions similar to those in Experiment 1. Using one-sided KS significance tests we concluded that our proposed method for long-term memory needs significantly fewer iterations to converge than the other types of memory ($p < 0.00001$).

3.3.2 Experiment 2: Non-convex objective

We tested the effect our algorithm in a DO setting with two agents, each having the following objective functions:

$$f_1(x_1, x_2) = (1 - x_1)^2 \quad \text{and} \quad f_2(x_1, x_2) = 100(x_2 - x_1^2)^2 \,. \tag{3.4}$$

The global objective amounts to $f_{\text{global}}(x_1, x_2) = (1-x_1)^2 + 100(x_2 - x_1^2)^2$, which is the Rosenbrock function with a global minimum at $(1, 1)$. The simulation setup is depicted in Figure 3.6. Across 100 runs, the agents start from different initial states sampled from $[-1.5, 1.5] \times [-1.5, 1.5]$. The algorithms shared both the sampled hyperparameters and the sampled initial states. Hyperparameter ranges are $\alpha \in [0.001, 0.003]$, $\beta \in [\alpha/1.5, \alpha/2.5]$, $\lambda \in [0.1, 0.2]$, and $T \in [80, 100]$.

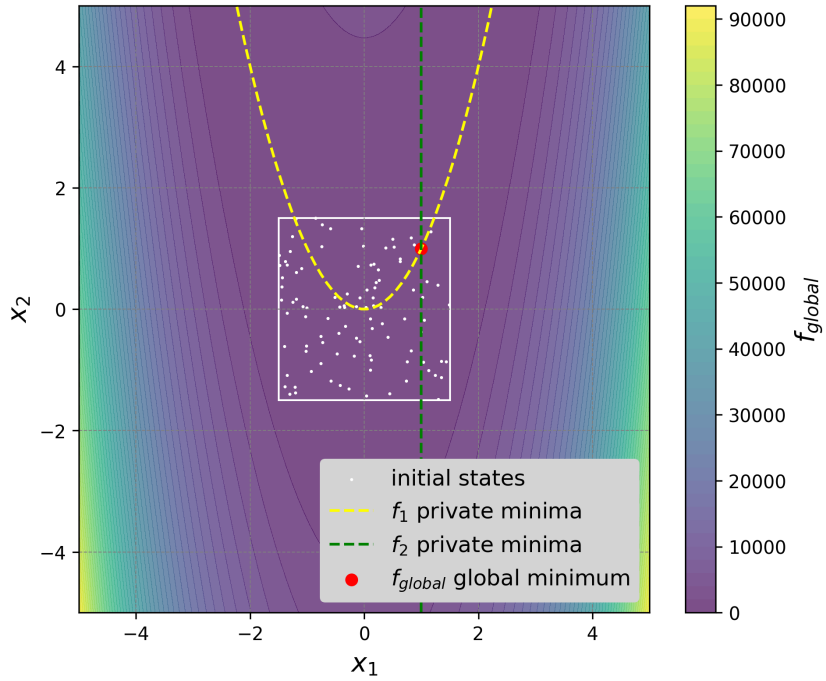

Figure 3.6: Heatmap of the global objective in Experiment 2

In the non-convex setting, our method brings significant improvements in convergence rate and stability. The iteration counts are $3,056 \pm 1,125$ for Fractional, $9,993 \pm 3,116$ for Heavy Ball and $15,057 \pm 4,663$ for No Memory. Using one-sided KS tests we determined that the fractional memory profile brings a significant reduction in the number of iterations needed to converge ($p < 0.00001$). The barplots in Figure 3.7 represent the mean, range and interquartile range of iterations needed to converge across 100 runs for a given condition.

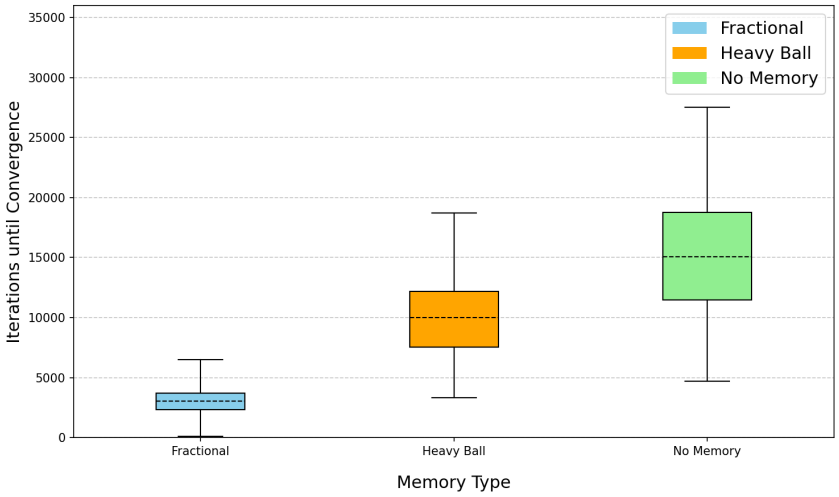

Figure 3.7: Performance plot Experiment 2

3.3.3 Experiment 3: Artificial neural networks

We tested the performance of our algorithm in a federated learning setting that employs two artificial neural networks (ANNs) consisting of 918,192 parameters each. We consider

an image classification task in a federated learning setup in which each agents receives a distinct balanced set of images from the MNIST dataset [16], with grayscale images of dimension 28×28. These two distinct datasets correspond to their private objective functions. The loss over a batch of 64 samples corresponds to the error signal used when optimizing the network weights.

As baselines, we consider variations of Algorithm 1, in which the private descent terms from Step 2 are replaced with the optimization methods of other optimization algorithms. These include gradient descent – which only contains the gradient terms, with no memory feedback term, gradient descent with Nesterov momentum – in which an agent's memory feedback term is replaced by the momentum term, gradient descent with heavy ball – where the memory feedback term has a memory of $T = 1$, and Adam – in which the updates from Step 2 are replaced by the Adam update rule.

Five runs were performed per algorithm, in which the agents started with randomly sampled state initializations and data partitions, all algorithms sharing the same learning rate and memory weighting where applicable. In Figure 3.8 and Figure 3.9 the FrODO algorithm shows promising results, achieving faster convergence than most of the baselines, and better convergence, albeit slower, than Adam.

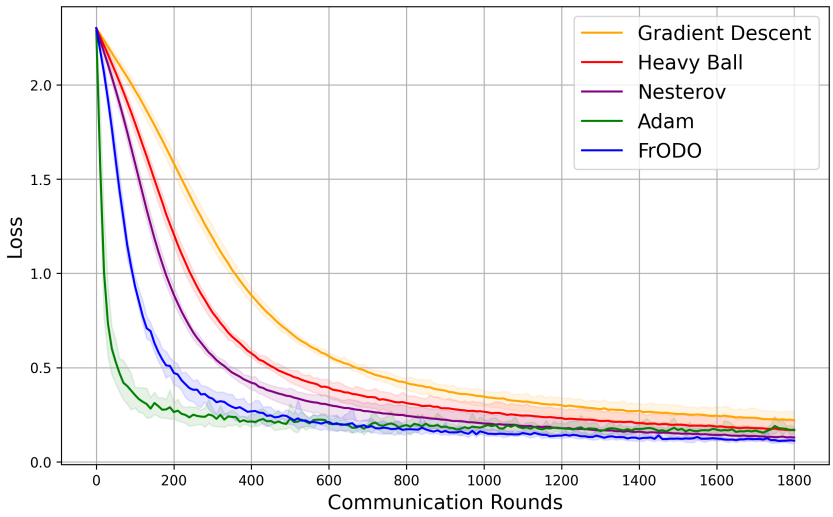

Figure 3.8: Loss performance plot Experiment 3

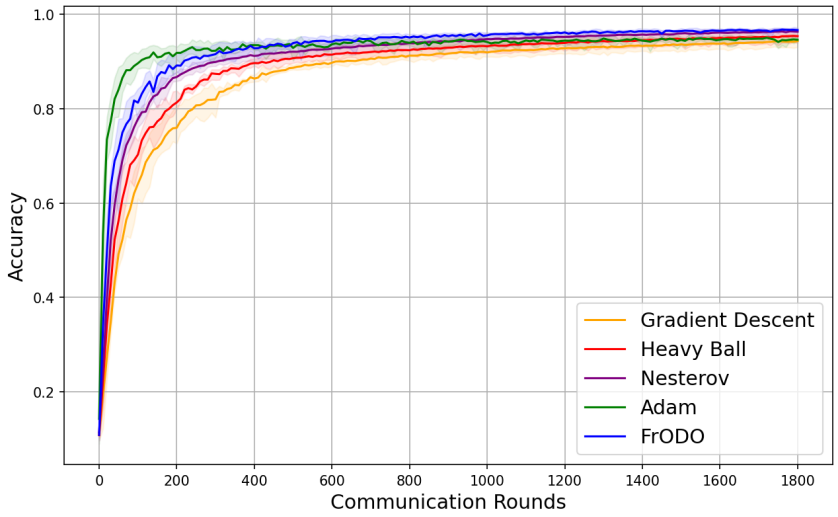

Figure 3.9: Accuracy performance plot Experiment 3

3.4 Convergence Analysis

3.4.1 Introduction

The findings in [13] state that one can prove DO algorithm convergence by showing that (i) it uses consensus and (ii) its centralized version converges. As seen in 1, our algorithm uses consensus, so we satisfy (i). In what follows, we will prove (ii) is satisfied by showing that there are ranges of hyperparameters for which the centralized version of FrODO converges.

We define our centralized implementation of FrODO as

$$x^{t+1} = x^t - \alpha\nabla f(x^t) - \beta\mu_1\nabla f(x^{t-1}) - \beta\mu_2\nabla f(x^{t-2}) - \cdots - \beta\mu_T\nabla f(x^{t-T}), \tag{3.5}$$

where $\mu_n = \mu(n;\lambda)$, for a given λ.

3.4.2 Existence of stable hyperparameters

According to the main result of [34], we can prove our algorithm's convergence by showing that there are hyperparameter values for which $\sum_{j=1}^{T}\beta\mu_j = 1$ holds.

By solving this equality for β, we get $\beta = \frac{1}{\sum_{j=1}^{T}\mu_j}$. The ranges we used to constrain each hyperparameter ($\beta \in \mathbb{R}^+$, $T \in \mathbb{Z}^+$, $\lambda \in \mathbb{R}^+$) allow for the existence of sets that satisfy the equality, which can be derived with numerical methods or analytical approximations.

Thus, there exist hyperparameters for which the base algorithm of FrODO converges to an optimal solution.

3.4.3 Defining the range of stable hyperparameters

To constrain the hyperparameter range further, we examine the ranges which make the algorithm satisfy the Jury criteria for stability when considering a fixed $T = 3$.

We consider objective functions which are continuously differentiable and strongly convex. If our objective f is continuously differentiable and strongly convex, f behaves

like a strongly convex quadratic function in a neighborhood of x^*, where x^* is a solution to f. We consider the strongly convex quadratic function

$$f(x) = \frac{1}{2}x^T Q x - b^T x + c, \tag{3.6}$$

with positive definite Hessian Q and eigenvalues

$$0 < m = \lambda_n \leq \lambda_{n-1} \leq \cdots \leq \lambda_2 \leq \lambda_1 = L$$

Note that $x^* = Q^{-1}b$ is the minimizer of f, and that $\nabla f(x) = Qx - b = Q\left(x - x^*\right)$. If we specialize (3.5) to (3.6), we obtain

$$x^{t+1} - x^* = x^t - x^* - \alpha Q(x^k - x^*) - \beta\mu_1 Q(x^{t-1} - x^*) - \beta\mu_2(x^{t-2} - x^*) - \cdots - \beta\mu_T Q(x^{t-T} - x^*) \tag{3.7}$$

In the following, we prove that our algorithm converges to the solution of a continuously differentiable and strongly convex objective function by proving it converges given a strongly convex quadratic function in a neighborhood of x^*.

We put our algorithm in the form of a linear time invariant system for its discrete implementation.

$$\begin{bmatrix} x^{(t+1)} - x^* \\ x^{(t)} - x^* \\ \vdots \\ x^{(t-T-1)} - x^* \end{bmatrix} = \mathrm{A} \cdot \begin{bmatrix} x^{(t)} - x^* \\ x^{(t-1)} - x^* \\ \vdots \\ x^{(t-T)} - x^* \end{bmatrix}$$

$$\mathrm{A} = \begin{bmatrix} 1 - \alpha Q & -\beta\mu_1 Q & -\beta\mu_2 Q & \cdots & -\beta\mu_{T-1} Q & -\beta\mu_T Q \\ 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 \end{bmatrix}$$

According to Jury's stability criterion, a linear discrete system is stable - which in our case means the optimization algorithm - converges if its characteristic polynomial satisfies a number of conditions. In the following, we will show that for a given $T = 3$, there exist ranges of hyperparameters α and β for which our system is stable.

The characteristic polynomial of our system for $T = 3$ is

$$p_A(x) = -1 + \alpha Q + \beta\mu_1 Q x + \beta\mu_2 Q x^2 + \beta\mu_3 Q x^3. \tag{3.8}$$

According to the table in Jury's stability criterion, we have

$$\begin{aligned} a_0 &= -1 + \alpha Q \\ a_1 &= \beta\mu_1 Q \\ a_2 &= \beta\mu_2 Q \\ b_0 &= 1 - 2\alpha Q + \alpha^2 Q^2 - \beta^2\mu_3^2 Q^2 \\ b_2 &= -\beta\mu_2 Q + \alpha\beta\mu_2 Q^2 - \beta^2 Q^2 \mu_1\mu_3. \end{aligned} \tag{3.9}$$

Condition 1: $p(1) > 0$

$$p(1) > 0 \iff -1 + \alpha Q + \beta\mu_1 Q + \beta\mu_2 Q + \beta\mu_3 Q > 0 \iff \begin{cases} \alpha > \frac{1}{m} - \beta(\mu_1 + \mu_2 + \mu_3) \\ \beta > \frac{\frac{1}{m} - \alpha}{\mu_1 + \mu_2 + \mu_3} \end{cases}$$

for $0 < m = \lambda_n \leq \lambda_{n-1} \leq \cdots \leq \lambda_2 \leq \lambda_1 = L$, where $\lambda_i, \forall i \in n$ are the eigenvalues of the Hessian Q.

The above can be extended to the case where $T \in \mathbb{Z}^+$ with

$$\begin{cases} \alpha > \frac{1}{m} - \beta(\mu_1 + \mu_2 + \cdots + \mu_T) \\ \beta > \frac{\frac{1}{m} - \alpha}{\mu_1 + \mu_2 + \cdots + \mu_T}. \end{cases}$$

Thus, we proved that there exist hyperparameters α and β such that the characteristic polynomial of our system is greater than 0, for a given T.

Condition 2: $(-1)^T p(-1) > 0$

We derive the hyperparameter ranges for the general case where $T \in \mathbb{Z}^+$:

$$(-1)^T p(-1) > 0 \iff \begin{cases} \frac{1}{m} - \beta(-\mu_1 + \mu_2 - \cdots + \mu_T) < \alpha < \frac{1}{L} - \beta(-\mu_1 + \cdots - \mu_T) \\ \frac{\frac{1}{m} - \alpha}{-\mu_1 + \mu_2 - \cdots + \mu_T} < \beta < \frac{\frac{1}{L} - \alpha}{-\mu_1 + \mu_2 - \cdots - \mu_T} \end{cases}$$

Now we also showed that there are acceptable ranges of α and β for which Condition 2 holds.

Condition 3: $|a_0| < a_n$

$$|a_0| < a_n \iff \begin{cases} \alpha < \frac{1}{L} + \beta\mu_T \\ \\ \beta > \frac{\alpha - \frac{1}{L}}{\mu_T} \end{cases}$$

Condition 4: $|b_0| > |b_2|$

$$|b_0| > |b_2| \iff |1 - 2\alpha Q + \alpha^2 Q^2 - \beta^2 \mu_3^2 Q^2| > |-\beta\mu_2 Q + \alpha\beta\mu_2 Q^2 - \beta^2 Q^2 \mu_1 \mu_3| \iff$$

$$\alpha_{\text{lower}} = \max\left(\frac{Q(-Q\beta\mu_2 + 2) - \sqrt{-Q^3\beta(4Q\beta\mu_1\mu_3 - Q\beta\mu_2^2 + 4Q\beta\mu_3^2 + 8\mu_2)}}{2Q^2}, \right.$$
$$\left. -\frac{\beta\mu_2}{2} - \frac{\beta\sqrt{4\mu_1\mu_3 + \mu_2^2 + 4\mu_3^2}}{2} + \frac{1}{Q} \right)$$

$$\alpha_{\text{upper}} = \min\left(\frac{Q(-Q\beta\mu_2 + 2) + \sqrt{-Q^3\beta(4Q\beta\mu_1\mu_3 - Q\beta\mu_2^2 + 4Q\beta\mu_3^2 + 8\mu_2)}}{2Q^2}, \right.$$
$$\left. -\frac{\beta\mu_2}{2} + \frac{\beta\sqrt{4\mu_1\mu_3 + \mu_2^2 + 4\mu_3^2}}{2} + \frac{1}{Q} \right)$$

The acceptable range for α is:

$$\alpha_{\text{lower}} < \alpha < \alpha_{\text{upper}}.$$

We found a range for α satisfying Condition 4 and now we proceed to do the same for β.

$$\begin{aligned}\beta_{\text{lower}} =& \frac{-\mu_2(Q\alpha+1)+\sqrt{-4Q^2\alpha^2\mu_1\mu_3+Q^2\alpha^2\mu_2^2-4Q^2\alpha^2\mu_3^2}}{2Q\mu_3(\mu_1+\mu_3)} \\ &+\frac{\sqrt{8Q\alpha\mu_1\mu_3+2Q\alpha\mu_2^2+8Q\alpha\mu_3^2+4\mu_1\mu_3+\mu_2^2+4\mu_3^2}}{2Q\mu_3(\mu_1+\mu_3)}\end{aligned}$$

$$\begin{aligned}\beta_{\text{upper}} =& \frac{-\mu_2(Q\alpha+1)-\sqrt{-4Q^2\alpha^2\mu_1\mu_3+Q^2\alpha^2\mu_2^2-4Q^2\alpha^2\mu_3^2}}{2Q\mu_3(\mu_1+\mu_3)} \\ &-\frac{\sqrt{8Q\alpha\mu_1\mu_3+2Q\alpha\mu_2^2+8Q\alpha\mu_3^2+4\mu_1\mu_3+\mu_2^2+4\mu_3^2}}{2Q\mu_3(\mu_1+\mu_3)}\end{aligned}$$

The acceptable range for β is:

$$\beta_{\text{lower}} < \beta < \beta_{\text{upper}}.$$

For each of the four conditions of Jury's stability criterion, we find ranges of hyperparameters α and β which satisfy the conditions, so we proved that our algorithm converges for the case of a given T.

In conclusion, by expressing FrODO as a linear time invariant system and using the Jury criterion to show its stability for a given T, we proved that our optimization algorithm is guaranteed to converge for a given T.

Chapter 4

Conclusion and Future Work

In this thesis manuscript, we introduce the concepts of distributed optimization and fractional calculus, providing a theoretical background and examples of societal impact that stemmed from both of these areas separately. We then demonstrated, through our fractional-order distributed optimization algorithm, that fractional calculus can significantly improve distributed optimization problems by increasing stability when optimizing functions with ill-defined Hessians. Our results show up to a 4× faster convergence compared to baseline methods on ill-conditioned problems and a 2-3× speedup in federated neural network training, all while maintaining stability and theoretical guarantees.

4.1 General DO opportunities

Beyond the above contributions, we also found a critical gap in our research in the field: the lack of standardization in distributed optimization benchmarks and implementation practices. During the reimplementation of existing DO methods, we observed that, unlike in centralized optimization—where standard machine learning optimizers can be easily incorporated into existing frameworks—the distributed setting introduces additional layers of complexity that can significantly impact optimization outcomes. One crucial example is agent alignment in the computational graph of DO algorithms, which is rarely explicitly addressed in existing literature. Depending on whether this alignment step is performed before, after, or concurrently with the independent optimization steps of the agents, convergence speeds can differ by orders of magnitude. Our empirical findings indicate that executing the alignment step *after* independent optimization leads to substantially faster convergence, a result that underscores the importance of explicitly defining computational graph structures in DO research.

Another limitation we identified is the nature of the most commonly used benchmarks in the DO and federated learning communities. At the time of conducting our literature review, many studies in the field relied on overly simplistic optimization problems, often involving only a single parameter. The most sophisticated benchmark encountered was the MNIST handwritten digits dataset, which, while useful, is far removed from the high-resolution medical imaging data encountered in real-world federated learning applications. This gap suggests a need for more realistic benchmarking datasets that better reflect practical challenges, such as those found in the healthcare sector, where federated models are applied to super-resolution medical scans. The development of such datasets could provide a significant contribution to the field by ensuring that DO methods are rigorously tested under conditions that align with real-world use cases.

4.2 Extensions of FrODO

Future research can build upon our findings in several key directions. First, further investigation is needed to delineate the specific elements of fractional optimization methods that enable superior performance compared to state-of-the-art techniques. A promising avenue is the development of machine learning optimizers that selectively integrate the most beneficial aspects of existing methods—for instance, leveraging the fast convergence speed of distributed ADAM while maintaining the convergence performance and stability of FrODO.

Another meaningful direction for future work is modifying the current FrODO framework to enhance its versatility across different application domains. Presently, our methodology requires storing T times more memory than stochastic gradient descent due to the history-dependent nature of fractional-order derivatives. This memory overhead poses a limitation when applying FrODO to high-capacity models, such as large language models. One potential solution is to replace the current truncation-based memory management with more efficient techniques, such as convolutional approaches, which can retain long-term dependencies while significantly reducing memory requirements.

In conclusion, the advancements presented in this thesis not only demonstrate the potential of fractional calculus in distributed optimization but also highlight key areas for improvement in benchmarking standards, computational graph design, and memory-efficient implementations. By addressing these challenges, future research can further refine and extend these methods, ultimately enabling more robust and efficient distributed optimization techniques for real-world applications across diverse domains.

Bibliography

[1] Alekh Agarwal, Martin J Wainwright, and John C Duchi. Distributed dual averaging in networks. *Advances in Neural Information Processing Systems*, 23, 2010.

[2] Ricardo Almeida, Małgorzata Guzowska, and Tatiana Odzijewicz. A remark on local fractional calculus and ordinary derivatives. *Open Mathematics*, 14(1):1122–1124, 2016.

[3] Martin Andreasson, Dimos V Dimarogonas, Henrik Sandberg, and Karl H Johansson. Distributed pi-control with applications to power systems frequency control. In *2014 American control conference*, pages 3183–3188. IEEE, 2014.

[4] Mahmoud Assran, Arda Aytekin, Hamid Reza Feyzmahdavian, Mikael Johansson, and Michael G Rabbat. Advances in asynchronous parallel and distributed optimization. *Proceedings of the IEEE*, 108(11):2013–2031, 2020.

[5] Giovanni Barbero, Luiz R Evangelista, Rafael S Zola, Ervin K Lenzi, and Antonio M Scarfone. A brief review of fractional calculus as a tool for applications in physics: Adsorption phenomena and electrical impedance in complex fluids. *Fractal and Fractional*, 8(7):369, 2024.

[6] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, Jonathan Eckstein, et al. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning*, 3(1):1–122, 2011.

[7] Theodora S Brisimi, Ruidi Chen, Theofanie Mela, Alex Olshevsky, Ioannis Ch Paschalidis, and Wei Shi. Federated learning of predictive models from federated electronic health records. *International journal of medical informatics*, 112:59–67, 2018.

[8] Paul L Butzer and Ursula Westphal. An introduction to fractional calculus. In *Applications of fractional calculus in physics*, pages 1–85. World Scientific, 2000.

[9] Sarthak Chatterjee, Subhro Das, and Sérgio Pequito. Neo: Neuro-inspired optimization—a fractional time series approach. *Frontiers in Physiology*, 12:724044, 2021.

[10] Ittai Dayan, Holger R Roth, Aoxiao Zhong, Ahmed Harouni, Amilcare Gentili, Anas Z Abidin, Andrew Liu, Anthony Beardsworth Costa, Bradford J Wood, Chien-Sung Tsai, et al. Federated learning for predicting clinical outcomes in patients with covid-19. *Nature medicine*, 27(10):1735–1743, 2021.

[11] Paulo M de Carvalho-Neto and Renato Fehlberg Junior. On the fractional version of leibniz rule. *Mathematische Nachrichten*, 293(4):670–700, 2020.

[12] Trevor Halsted, Ola Shorinwa, Javier Yu, and Mac Schwager. A survey of distributed optimization methods for multi-robot systems. *arXiv preprint arXiv:2103.12840*, 2021.

[13] Shuo Han. Systematic design of decentralized algorithms for consensus optimization. *IEEE Control Systems Letters*, 3(4):966–971, 2019.

[14] Jakub Konečnỳ, Brendan McMahan, and Daniel Ramage. Federated optimization: Distributed optimization beyond the datacenter. *arXiv preprint arXiv:1511.03575*, 2015.

[15] Benjamin Kroposki, Andrey Bernstein, Jennifer King, Deepthi Vaidhynathan, Xinyang Zhou, Chin-Yao Chang, and Emiliano Dall'Anese. Autonomous energy grids: Controlling the future grid with large amounts of distributed energy resources. *IEEE Power and Energy Magazine*, 18(6):37–46, 2020.

[16] Yann LeCun. The MNIST database of handwritten digits. *http://yann.lecun.com/exdb/mnist/*, 1998.

[17] Li Li, Yuxi Fan, Mike Tse, and Kuo-Yi Lin. A review of applications in federated learning. *Computers & Industrial Engineering*, 149:106854, 2020.

[18] Ilan Lobel and Asuman Ozdaglar. Distributed subgradient methods over random networks. In *Proc. Allerton Conf. Commun., Control, Comput*, pages 1–27. Citeseer, 2008.

[19] Farshad Merrikh-Bayat, Nafiseh Mirebrahimi, and Mohammad Reza Khalili. Discrete-time fractional-order PID controller: Definition, tuning, digital realization and some applications. *International Journal of Control, Automation and Systems*, 13:81–90, 2015.

[20] Daniel K Molzahn, Florian Dörfler, Henrik Sandberg, Steven H Low, Sambuddha Chakrabarti, Ross Baldick, and Javad Lavaei. A survey of distributed optimization and control algorithms for electric power systems. *IEEE Transactions on Smart Grid*, 8(6):2941–2962, 2017.

[21] Cristina I Muresan, Cosmin Copot, Isabela Birs, Robin De Keyser, Steve Vanlanduit, and Clara M Ionescu. Experimental validation of a novel auto-tuning method for a fractional order pi controller on an ur10 robot. *Algorithms*, 11(7):95, 2018.

[22] Cristina I Muresan, Robin De Keyser, Isabela Birs, Dana Copot, and Clara Ionescu. Benchmark challenge: A robust fractional order control autotuner for the refrigeration systems based on vapor compression. *IFAC-PapersOnLine*, 51(4):31–36, 2018.

[23] Cristina I Muresan, Robin De Keyser, Isabela R Birs, Silviu Folea, and Ovidiu Prodan. An autotuning method for a fractional order pd controller for vibration suppression. *Mathematical Methods in Engineering: Applications in Dynamics of Complex Systems*, pages 245–256, 2019.

[24] Ahmad Naeem, Tayyaba Anees, Rizwan Ali Naqvi, and Woong-Kee Loh. A comprehensive analysis of recent deep and federated-learning-based methodologies for brain tumor diagnosis. *Journal of Personalized Medicine*, 12(2):275, 2022.

[25] Angelia Nedic and Asuman Ozdaglar. Distributed subgradient methods for multi-agent optimization. *IEEE Transactions on Automatic Control*, 54(1):48–61, 2009.

[26] Shi Pu, Wei Shi, Jinming Xu, and Angelia Nedić. Push–pull gradient methods for distributed optimization in networks. *IEEE Transactions on Automatic Control*, 66(1):1–16, 2020.

[27] Michael Rabbat and Robert Nowak. Distributed optimization in sensor networks. In *Proceedings of the 3rd international symposium on Information processing in sensor networks*, pages 20–27, 2004.

[28] Nicola Rieke. What is federated learning?, October 13 2019.

[29] Howard Rosenbrock. An automatic method for finding the greatest or least value of a function. *The computer journal*, 3(3):175–184, 1960.

[30] Augustinos D Saravanos, Hunter Kuperman, Alex Oshin, Arshiya Taj Abdul, Vincent Pacelli, and Evangelos A Theodorou. Deep distributed optimization for large-scale quadratic programming. *arXiv preprint arXiv:2412.12156*, 2024.

[31] Ali H Sayed et al. Adaptation, learning, and optimization over networks. *Foundations and Trends® in Machine Learning*, 7(4-5):311–801, 2014.

[32] Shreyas Sundaram and Bahman Gharesifard. Distributed optimization under adversarial nodes. *IEEE Transactions on Automatic Control*, 64(3):1063–1076, 2018.

[33] S Sundhar Ram, Angelia Nedić, and Venugopal V Veeravalli. Distributed stochastic subgradient projection algorithms for convex optimization. *Journal of optimization theory and applications*, 147:516–545, 2010.

[34] Adrien Taylor, Bryan Van Scoy, and Laurent Lessard. Lyapunov functions for first-order methods: Tight automated convergence guarantees. In *International Conference on Machine Learning*, pages 4897–4906. PMLR, 2018.

[35] Zhen Ling Teo, Liyuan Jin, Nan Liu, Siqi Li, Di Miao, Xiaoman Zhang, Wei Yan Ng, Ting Fang Tan, Deborah Meixuan Lee, Kai Jie Chua, et al. Federated machine learning in healthcare: A systematic review on clinical applications and technical architecture. *Cell Reports Medicine*, 5(2), 2024.

[36] Gabriele U Varieschi. Applications of fractional calculus to newtonian mechanics. *arXiv preprint arXiv:1712.03473*, 2017.

[37] Jesús M Velásquez-Bermúdez, Marzieh Khakifirooz, and Mahdi Fathi. *Large Scale Optimization in Supply Chains and Smart Manufacturing*. Springer, 2019.

[38] Tongnian Wang, Yan Du, Yanmin Gong, Kim-Kwang Raymond Choo, and Yuanxiong Guo. Applications of federated learning in mobile health: scoping review. *Journal of Medical Internet Research*, 25:e43006, 2023.

[39] Jinming Xu, Ye Tian, Ying Sun, and Gesualdo Scutari. Distributed algorithms for composite optimization: Unified framework and convergence analysis. *IEEE Transactions on Signal Processing*, 69:3555–3570, 2021.

[40] Tao Yang, Xinlei Yi, Junfeng Wu, Ye Yuan, Di Wu, Ziyang Meng, Yiguang Hong, Hong Wang, Zongli Lin, and Karl H Johansson. A survey of distributed optimization. *Annual Reviews in Control*, 47:278–305, 2019.

Appendix

The Python code used for implementing the FrODO algorithm, running the experiments and visualizing the results can be found at `https://github.com/AndreiLix/FrODO`, with the FrODO algorithm implementation at `https://github.com/AndreiLix/FrODO/blob/main/FOLDER_code/comdo/utils_proper.py`.