J Syst Sci Complex (2023) 36: 613–631

Distributed Solving Linear Algebraic Equations with Switched Fractional Order Dynamics*

YU Wenqiang · CHENG Songsong · HE Shuping

DOI: 10.1007/s11424-023-1350-6
Received: 15 September 2021 / Revised: 31 October 2021

Abstract This paper proposes a novel distributed optimization algorithm with fractional order dynamics to solve linear algebraic equations. Firstly, the authors proposed "Consensus + Projection" flow with fractional order dynamics, which has more design freedom and the potential to obtain a better convergent performance than that of conventional first order algorithms. Moreover, the authors prove that the proposed algorithm is convergent under certain iteration order and step-size. Furthermore, the authors develop iteration order switching scheme with initial condition design to improve the convergence performance of the proposed algorithm. Finally, the authors illustrate the effectiveness of the proposed method with several numerical examples.

Keywords Distributed optimization, fractional order dynamics, initial condition, iteration order, linear equations.

1 Introduction

Solving linear algebraic equations of the form $\boldsymbol{Hx} = \boldsymbol{z}$ is an important topic in many fields, such as the stability analysis[1, 2], sensor networks[3], and compressed sensing[4]. With the increasing complexity of scientific problems, the dimensions of linear algebraic equations are very large, which increase the computation complexity. Then the conventional centralized methods of solving linear algebraic equations, such as Jacobi[5, 6], Gauss-Seidel[7, 8], and Successive over-relaxation[9, 10], are ineffective to overcome this challenge.

YU Wenqiang · CHENG Songsong (Corresponding author) · HE Shuping
Anhui Engineering Laboratory of Human-Robot Integration System and Intelligent Equipment, School of Electrical Engineering and Automation, Anhui University, Hefei 230601*, China.*
Email: z20201017@stu.ahu.edu.cn; sscheng@amss.ac.cn; shuping.he@ahu.edu.cn.
*This work was supported by the National Natural Science Foundation of China under Grant Nos. 62103003, 62073001, and 61973002, the Anhui Provincial Key Research and Development Project under Grant 2022i01020013, the University Synergy Innovation Program of Anhui Province under Grant No. GXXT-2021-010, the Anhui Provincial Natural Science Foundation under Grant No. 2008085J32, the National Postdoctoral Program for Innovative Talents under Grant No. BX20180346, and the General Financial Grant from the China Postdoctoral Science Foundation under Grant No. 2019M660834.
◇ *This paper was recommended for publication by Editor HU Jiangping.*

In comparison with the conventional centralized methods, the distributed algorithms based on network systems have many advantages. On the one hand the parameter matrices of the linear algebraic equations can be stored in different geographic locations because of privacy protection issues. On the other hand the equation can be solved distributedly since it do not rely on any central node[11].

Inspired by distributed optimization[12–21], there are many distributed algorithms to solve linear algebraic equations. In [22], each agent can distributedly obtain the solution of $\boldsymbol{Ax} = \boldsymbol{b}$ by sharing its local decision varaible $\boldsymbol{x}_i$ and the kernal of parameters matrice $\boldsymbol{A}_i$. Moreover, by designing a specific initialization step for each agent, [23] proposed a distributed algorithm with an exponential convergence rate and [24] further improved the method of [23] with arbitrary initializations. Furthermore, [25, 26] proposed a "concensus+projection" method to achieve the optimal solution.

An efficient algorithm is the eternal pursuit of scholars. By making use of the history information, some excellent works were designed to improve the convergence performance, such as momentum gradient method[29] and Nesterov method[30]. For Nesterov method, [30] pointed out that its key stone is designing the related parameters for the algorithm with a underdamped dynamics. As a result, an intuitive and heuristic idea to accelerate convergence rate of optimization algorithm is indroducing underdamped dynamics to yield an appropriate oscillation[31]. Based on related researches, conventional gradient descent iteration can be converted into the form of first order difference dynamics. Moreover, one can generalize it as fractional order dynamics, which is underdamped and yields an appropriate oscillation to improve the convergence performance[32, 33].

The fractional order calculus was proposed three hundred years ago and has many perfect applications in system modeling and identification[34–36], stability analysis[37, 38], and signal processing[39, 40]. Existing results show that fractional order dynamics has more design freedom and the potential to obtain better performance than that of conventional first order counterparts. [41] solved the phase retrieval problem if only two close fractional power spectra were known with fractional Fourier transform moments. [42] proposed a fractional order proportional and derivative motion controller for a class of second-order plants. [43] used the fractional order dynamics to design an LMS algorithm with a better convergence performance. What is more, [43] proposed an iteration order hybrid switching to further increase the speed of convergence and response of modified LMS algorithm. Inspired by this idea, this paper proposes a distributed algorithm with fractional order dynamics to solve linear algebraic equations and achieves better convergence performance than that of conventional distributed algorithms with integer order dynamics.

In this paper, we propose a distributed algorithm with fractional oder dynamics to slove linear algebraic equations, analyze the convergence performance, and improve the convergence rate with iteration order switching scheme. The main contributions of this paper are listed as follows:

1) We propose a novel distributed algorithm with fractional order dynamics, which has more design freedom and has the potential to achieve better convergence performance than that of

conventional integer order algorithms in [25];

2) Inspired by [32], we transform the proposed algorithm in the form of fractional order difference system and analyze its convergence speed and response speed;

3) According to the convergent performances of the proposed algorithm, we design a switching law of iteration orders with restarting initial condition to further improve the convergence performance.

The rest of this paper is organized as follows. Section 2 introduces some preliminaries on graph theory as well as fractional calculus and formulates the problem. Section 3 designs a distributed optimization algorithm and analyzes the convergence of the proposed algorithm. Then Section 4 improves the algorithm with iteration order switching law. Section 5 develops several examples to illustrate the effectiveness of the proposed algorithm and Section 6 concludes this paper.

Notations Let $a \in \mathbb{R}$, $\boldsymbol{a} \in \mathbb{R}^n$, and $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ be a real scalar, n-dimensional column vectors, and $n \times n$-dimensional matrix, respectively. A positive integer set is denoted by $\mathbb{N}_+$. $\nabla \boldsymbol{d}(k) = \boldsymbol{d}(k)-\boldsymbol{d}(k-1)$ denotes backward difference. We denote the Euclidean norm of a vector by $\|*\|$. Let $|*|$ be the cardinality of a set. Define the Kronecker product of two matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ as $\boldsymbol{A}\otimes\boldsymbol{B}$. $\boldsymbol{I}$ denotes an identity matrix with a proper dimension. blkdiag$\{\boldsymbol{A}_1, \boldsymbol{A}_2, \cdots, \boldsymbol{A}_N\}$ (diag$\{t_1, t_2, \cdots, t_N\}$) is a diagonal block matrix with $\boldsymbol{A}_1, \boldsymbol{A}_2, \cdots, \boldsymbol{A}_N$ (t_1, t_2, $\cdots$, t_N) in the diagonal.

2 Preliminaries

2.1 Graph Theory

We describe a graph with $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, where $\mathcal{V} = \{1, 2, \cdots, N\}$ denotes a finite set of nodes and $\mathcal{E}$ is composed of an unordered pair of two different nodes in $\mathcal{V}$. A path in a graph is a sequence of two different nodes, a_1, a_2, $\cdots$, a_N, such as $(a_i, a_j) \in \mathcal{E}$, $i \in \mathcal{V}, j \in \mathcal{N}_i$. A graph is undirected and connected if there exists a path between any two different nodes $i, j \in \mathcal{V}$ and any two different nodes can communicate mutually. We denote $\mathcal{N}_i = \{j \in \mathcal{V} : (i, j) \in \mathcal{E}\}$ as the neighbor set of node i. Define the degree matrix $\boldsymbol{D} = \text{diag}\{|\mathcal{N}_1|, |\mathcal{N}_2|, \cdots, |\mathcal{N}_\text{N}|\}$. $\boldsymbol{A}$ is the adjacency matrix of graph $\mathcal{G}$, where $[\boldsymbol{A}_{ij}] = 1$ if $j \in \mathcal{N}_i$ and $[\boldsymbol{A}_{ij}] = 0$ otherwise. $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{A}$ is the Laplacian matrix of graph $\mathcal{G}$.

For the convenience of the analysis hereinafter, we provide the following mild assumption.

Assumption 1 ([25]) The graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ is undirected and connected.

2.2 Fractional Order Sum and Difference

The α-th order sum of a discrete sequence $x(k)$ is

$$\nabla^{-\alpha} x(k) = \sum_{j=0}^{k} (-1)^j \binom{-\alpha}{j} x(k-j), \tag{1}$$

where $\alpha > 0$, $\binom{p}{q} = \frac{\boldsymbol{\Gamma}(p+1)}{\boldsymbol{\Gamma}(1+p-q)\boldsymbol{\Gamma}(1+q)}$, and $\boldsymbol{\Gamma}(p) = \int_0^\infty \mathrm{e}^{-t} t^{p-1} dt$ is Gamma function.

The conventional t-th order backward difference of a discrete sequence $x(n)$ is given by

$$\nabla^t x(k) = \sum_{\kappa=0}^{t} (-1)^\kappa \binom{t}{\kappa} x(k+t-\kappa), \quad t \in \mathbb{N}_+. \tag{2}$$

According to (1) and (2), we provide the following definition of Caputo fractional order difference

Definition 2.1 (see [32]) The Caputo fractional order difference is

$$\nabla^\alpha x(k) = \nabla^{\alpha-t} \nabla^t x(k), \tag{3}$$

where $t-1 < \alpha \leq t$.

In the following, we introduce the discrete Mittag-Leffler function, which is essential for the analysis of distributed algorithm hereinafter.

Definition 2.2 (see [43]) The discrete Mittag-Leffler function is

$$\mathcal{M}_{\alpha,\beta}(\lambda, k) = \sum_{i=0}^{+\infty} \lambda^i \frac{\boldsymbol{\Gamma}(i\alpha+\beta+k-1)}{\boldsymbol{\Gamma}(i\alpha+\beta)\boldsymbol{\Gamma}(k)}, \tag{4}$$

where $|\lambda| < 1$.

Based on Definitions 2.1 and 2.2, we give the following basic results.

Lemma 2.3 (see [32]) *For a given discrete sequence $x(n)$, its discrete Laplace transformation is*

$$\mathcal{N}\nabla^\alpha x(k) = z^\alpha X(z) - \sum_{\kappa=0}^{t-1} z^{\alpha-\kappa-1} \nabla^\kappa x(k)|_{k=0}, \tag{5}$$

where $X(z) = \mathcal{N}\{x(k)\} = \sum_{k=0}^{+\infty} (1-z)^{k-1} x(k)$ is the nabla transform of $x(n)$.

Lemma 2.4 (see [32]) *For the following fractional order difference equation*

$$\nabla^\alpha x(k) = \lambda x(k) + u(k), \quad t-1 < \alpha \leq t, \quad \lambda \neq 1, \tag{6}$$

with initial conditions $\nabla^\alpha x(0) = b_j, j = 0, 1, \cdots, t-1$, its solution is given by

$$x(k) = \sum_{j=0}^{t-1} b_j \mathcal{M}_{\alpha,j+1}(\lambda, k) + \sum_{j=1}^{k} \mathcal{M}_{\alpha,\alpha}(\lambda, j) u(k-j+1). \tag{7}$$

Lemma 2.5 (see [32]) *If $0 < \alpha < 2$ and $\lambda < 0$, then*

$$\mathcal{M}_{\alpha,\beta}(\lambda, k) \approx -\sum_{j=1}^{+\infty} \frac{\lambda^{-j}(k-1)^{-j\alpha}}{\boldsymbol{\Gamma}(-j\alpha+\beta)}. \tag{8}$$

2.3 Problem Formulation

Consider the following linear algebraic equation

$$\boldsymbol{z} = \boldsymbol{H}\boldsymbol{x}_o, \tag{9}$$

where $\boldsymbol{x}_o \in \mathbb{R}^m$, $\boldsymbol{H} = [\boldsymbol{H}_1^{\mathrm{T}}, \boldsymbol{H}_2^{\mathrm{T}}, \cdots, \boldsymbol{H}_n^{\mathrm{T}}]^{\mathrm{T}} \in \mathbb{R}^{N\times m}$ with $\boldsymbol{H}_i \in \mathbb{R}^{r_i\times m}$, $\boldsymbol{z} = [\boldsymbol{z}_1^{\mathrm{T}}, \boldsymbol{z}_2^{\mathrm{T}}, \cdots, \boldsymbol{z}_n^{\mathrm{T}}]^{\mathrm{T}} \in \mathbb{R}^N$ with $\boldsymbol{z}_i \in \mathbb{R}^{r_i}$, and $N = \sum_{i=1}^n r_i$. For (9),

1) if $\mathrm{rank}(\boldsymbol{H}) = m$ and $\boldsymbol{z} \in \mathrm{span}(\boldsymbol{H})$, there exists an exact solution;

2) if $\mathrm{rank}(\boldsymbol{H}) < m$ and $\boldsymbol{z} \in \mathrm{span}(\boldsymbol{H})$, there exists an infinite set of solutions;

3) if $\boldsymbol{z} \notin \mathrm{span}(\boldsymbol{H})$, there exist no exact solutions. A least-squares solution of (9) is defined as follows

$$\boldsymbol{x}_l = \mathrm{argmin}_{\boldsymbol{x}_o\in\mathbb{R}^m}\|\boldsymbol{z} - \boldsymbol{H}\boldsymbol{x}_o\|^2. \tag{10}$$

Here, we make the following assumptions for the linear algebraic equation in (9).

Assumption 2 There exists an exact solution $\boldsymbol{x}^*$.

For distributedly solving (10), we consider a network system with n agents indexed by $\mathcal{V} \triangleq \{1, 2, \cdots, n\}$, where agent i has only access to $\boldsymbol{H}_i$ and $\boldsymbol{z}_i$. Then we formulate it as the following distributed optimization problem

$$\begin{aligned} \min_{\boldsymbol{x}} \ & \frac{1}{2}\sum_{i=1}^n \|\boldsymbol{z}_i - \boldsymbol{H}_i\boldsymbol{x}_i\|^2 \\ \text{s.t.} \quad & \boldsymbol{x}_i = \boldsymbol{x}_j, \quad \forall i, j \in \mathcal{V}, \end{aligned} \tag{11}$$

where $\boldsymbol{x} = [\boldsymbol{x}_1^{\mathrm{T}}, \boldsymbol{x}_2^{\mathrm{T}}, \cdots, \boldsymbol{x}_n^{\mathrm{T}}]^{\mathrm{T}} \in \mathbb{R}^{nm}$ with $\boldsymbol{x}_i \in \mathbb{R}^m, \forall i \in \mathcal{V}$.

3 Main Result

3.1 Algorithm Design

For solving (12), we take a "Consensus+Projection" to flow with discrete fractional order dynamics in Algorithm 1.

For Algorithm 1, the fractional order dynamics can be unwrapped as follows

$$\begin{aligned} \nabla^\alpha \boldsymbol{x}_i(k) &= \sum_{j=0}^k (-1)^j \binom{\alpha - t}{j} \nabla^t \boldsymbol{x}_i(k-j) \\ &= \nabla^t \boldsymbol{x}_i(k) + \sum_{j=0}^{k-1} (-1)^{j+1} \binom{\alpha - t}{j+1} \nabla^t \boldsymbol{x}_i(k-j-1) \\ &= \nabla^t \boldsymbol{x}_i(k) + \boldsymbol{\Phi}_{1i}(k-1), \end{aligned} \tag{12}$$

where $t = \lceil \alpha \rceil$, $\boldsymbol{g}_i(k) = (-1)^{k+1}\binom{\alpha - t}{k+1}$, and $\boldsymbol{\Phi}_{1i}(k) = \boldsymbol{g}(k) * \nabla^t \boldsymbol{x}_i(k)$.

According to the update flows of $\nabla^\alpha \boldsymbol{x}_i(k)$ and (12), we have

1) for $0 < \alpha < 1$,

$$\boldsymbol{x}_i(k) = \boldsymbol{x}_i(k-1) + h\left[\sum_{j\in\mathcal{N}_i}(\boldsymbol{x}_j(k-1) - \boldsymbol{x}_i(k-1)) - \boldsymbol{H}_i^{\mathrm{T}}(\boldsymbol{H}_i\boldsymbol{x}_i(k-1) - \boldsymbol{z}_i)\right] - \boldsymbol{\Phi}_{1i}(k-1); \tag{13}$$

Algorithm 1 Distributed algorithm

Initialization: For each $i \in \mathcal{V}$,

step-size h, $\boldsymbol{H}_i \in \mathbb{R}^{r_i \times m}$, $\boldsymbol{z}_i \in \mathbb{R}^{r_i}$, $\boldsymbol{x}_i(0) \in \mathbb{R}^m$, and $\boldsymbol{x}_j(0) \in \mathbb{R}^m, j \in \mathcal{N}_i$.

Update flows: For each $i \in \mathcal{V}$,

For $k = 1$ to K **do**

$$\nabla^\alpha \boldsymbol{x}_i(k) = h\Big[\sum_{j\in\mathcal{N}_i}(\boldsymbol{x}_j(k-1) - \boldsymbol{x}_i(k-1)) - \boldsymbol{H}_i^{\mathrm{T}}(\boldsymbol{H}_i\boldsymbol{x}_i(k-1) - \boldsymbol{z}_i)\Big].$$

End for

2) for $1 < \alpha < 2$,

$$\begin{aligned}\boldsymbol{x}_i(k) =& 2\boldsymbol{x}_i(k-1) - \boldsymbol{x}_i(k-2) + h\Big[\sum_{j\in\mathcal{N}_i}(\boldsymbol{x}_j(k-1) - \boldsymbol{x}_i(k-1)) - \boldsymbol{H}_i^{\mathrm{T}}(\boldsymbol{H}_i\boldsymbol{x}_i(k-1) - \boldsymbol{z}_i)\Big]\\ &- \boldsymbol{\Phi}_{1i}(k-1).\end{aligned} \tag{14}$$

Remark 3.1 For $\alpha = 1$, Algorithm 1 is degenerated into the conventional integer "Consensus+Projection" algorithm[25]. Therefore, compared with [25], our method has more design freedom and the potential to obtain better convergent performance.

3.2 Convergence Analysis

We rewrite Algorithm 1 in the following compact form

$$\nabla^\alpha \boldsymbol{x}(k) = -h\boldsymbol{F_d}\boldsymbol{x}(k-1) + h\boldsymbol{z_H}, \tag{15}$$

where $\boldsymbol{x}(k) = [\boldsymbol{x}_1^{\mathrm{T}}(k), \boldsymbol{x}_2^{\mathrm{T}}(k), \cdots, \boldsymbol{x}_n^{\mathrm{T}}(k)]^{\mathrm{T}}$, $\boldsymbol{x}_i(k) \in \mathbb{R}^m$, $\boldsymbol{z_H} = [\boldsymbol{H}_1^{\mathrm{T}}\boldsymbol{z}_1, \boldsymbol{H}_2^{\mathrm{T}}\boldsymbol{z}_2 \cdots, \boldsymbol{H}_n^{\mathrm{T}}\boldsymbol{z}_n]^{\mathrm{T}}$, $\boldsymbol{L} \in \mathbb{R}^{n\times n}$, $\boldsymbol{H_d} = \mathrm{blkdiag}\{\boldsymbol{H}_1^{\mathrm{T}}\boldsymbol{H}_1, \boldsymbol{H}_2^{\mathrm{T}}\boldsymbol{H}_2, \cdots, \boldsymbol{H}_n^{\mathrm{T}}\boldsymbol{H}_n\}$, and $\boldsymbol{F}_d = \boldsymbol{L} \otimes \boldsymbol{I}_m + \boldsymbol{H_d}$.

According to $H_i^{\mathrm{T}}\boldsymbol{z}_i = H_i^{\mathrm{T}}H_i\boldsymbol{x}_i^*$ and $\nabla^\alpha \boldsymbol{x}_i^* = \boldsymbol{0}$, we rewrite (15) as follows

$$\nabla^\alpha[\boldsymbol{x}(k) - \boldsymbol{x}^*] = -h\boldsymbol{F_d}[\boldsymbol{x}(k-1) - \boldsymbol{x}^*]. \tag{16}$$

Since $\boldsymbol{F_d}$ is a symmetrical positive definite matrix[25], there exists a unitary matrix $\boldsymbol{U}$, which satisfies $\boldsymbol{U}^{\mathrm{T}}\boldsymbol{F_d}\boldsymbol{U} = \boldsymbol{P}$. Then

$$\nabla^\alpha[\boldsymbol{x}(k) - \boldsymbol{x}^*] = -h\boldsymbol{U}\boldsymbol{P}\boldsymbol{U}^{\mathrm{T}}[\boldsymbol{x}(k-1) - \boldsymbol{x}^*], \tag{17}$$

where $\boldsymbol{U} = [\boldsymbol{u}_1, \boldsymbol{u}_2, \cdots, \boldsymbol{u}_{nm}] \in \mathbb{R}^{nm\times nm}$ with $\boldsymbol{u}_i \in \mathbb{R}^{nm}$ and $\boldsymbol{P} = \mathrm{diag}\{p_1, p_2, \cdots, p_{nm}\}$.

Taking $\overline{\boldsymbol{x}}(k) = \boldsymbol{U}^{\mathrm{T}}[\boldsymbol{x}(k) - \boldsymbol{x}^*]$,

$$\nabla^\alpha\overline{\boldsymbol{x}}(k) = -h\boldsymbol{P}\overline{\boldsymbol{x}}(k-1) \tag{18}$$

which implies

$$\nabla^\alpha\overline{x}_i(k) = -hp_i\overline{x}_i(k-1). \tag{19}$$

Then we provide the convergent performance of Algorithm 1 as follows.

Theorem 3.2 *Under Assumptions* 1 *and* 2, *let* $\boldsymbol{x}_i(k)$ *be generated by Algorithm* 1.
1) *If* $0 < \alpha \leq 1$, *then* $\overline{\boldsymbol{x}}_i(k)$ *monotonously converges to* $\mathbf{0}$ *at large iteration steps.*
2) *If* $1 < \alpha \leq 2$ *and* $\nabla\overline{\boldsymbol{x}}_i(0) = \mathbf{0}$, *then* $\overline{\boldsymbol{x}}_i(k)$ *asymptotically converges to* $\mathbf{0}$ *with oscillation.*

Proof Based on Lemma 2.4,

$$\overline{x}_i(k) = \sum_{j=0}^{t-1} \overline{x}_i(0)\mathcal{M}_{\alpha,j+1}(-hp_i, k-1). \tag{20}$$

1) For $0 < \alpha < 1$ and $t-1 < \alpha \leq t$, we obtain $t = 1$ and $j = 0$ with (20). Then

$$\overline{x}_i(k) = \overline{x}_i(0)\mathcal{M}_{\alpha,1}(-hp_i, k-1). \tag{21}$$

Combining Lemma 2.5 and $\boldsymbol{\Gamma}(1-a)\boldsymbol{\Gamma}(a) = \frac{\pi}{\sin(\pi a)}$, we modify (21) as

$$\overline{x}_i(k) = \overline{x}_i(0)\mathcal{M}_{\alpha,1}(-hp_i, k-1) \approx \overline{x}_i(0)\frac{-(-hp_i)^{-1}(k-2)^{-\alpha}}{\boldsymbol{\Gamma}(-\alpha+1)} = \overline{x}_i(0)\frac{\sin(\alpha\pi)}{\pi}\frac{\boldsymbol{\Gamma}(\alpha)}{(hp_i)(k-2)^{\alpha}}. \tag{22}$$

For $\alpha = 1$, combining $\sin(a + k\pi) = (-1)^k \sin a$ and $\boldsymbol{\Gamma}(1-a)\boldsymbol{\Gamma}(a) = \frac{\pi}{\sin(\pi a)}$,

$$\begin{aligned}\overline{x}_i(k) &= \overline{x}_i(0)\mathcal{M}_{1,1}(-hp_i, k-1) = \overline{x}_i(0)\sum_{j=0}^{+\infty}(-hp_i)^j\frac{\boldsymbol{\Gamma}(j+k-1)}{\boldsymbol{\Gamma}(j+1)\boldsymbol{\Gamma}(k-1)}\\ &= \overline{x}_i(0)\sum_{j=0}^{+\infty}(hp_i)^j\frac{\boldsymbol{\Gamma}(2-k)}{\boldsymbol{\Gamma}(2-j-k)\boldsymbol{\Gamma}(1+j)} = \overline{x}_i(0)\sum_{j=0}^{+\infty}(hp_i)^j\binom{1-k}{j}\\ &= \frac{\overline{x}_i(0)}{(1+hp_i)^{k-1}}.\end{aligned} \tag{23}$$

According to (23) and (22), $\overline{\boldsymbol{x}}_i(k)$ asymptotically converges to $\mathbf{0}$.

2) Firstly, we prove that the solution of (19) is convergent in case $1 < \alpha < 2$. Since $1 < \alpha < 2$, the solution of (19) is

$$\overline{x}_i(k) = \overline{x}_i(0)\mathcal{M}_{\alpha,1}(-hp_i, k-1) + \nabla^1\overline{x}_i(0)\mathcal{M}_{\alpha,2}(-hp_i, k-1). \tag{24}$$

Recalling Lemma 2.5, we obtain

$$|\mathcal{M}_{\alpha,j+1}(\lambda, k-1)| \approx \sum_{i=1}^{+\infty}\left|\frac{\lambda^{-i}(k-2)^{-i\alpha}}{\boldsymbol{\Gamma}(-i\alpha+j+1)}\right| \leq M\sum_{i=1}^{+\infty}|\lambda^{-i}(k-2)^{-i\alpha}|, \tag{25}$$

where M is the maximum of $|\frac{1}{\boldsymbol{\Gamma}(-i\alpha+j+1)}|$ for $j = 0, 1$ and $i \in [0, \infty)$.

Due to $-hp < 0$ and $n \to +\infty$,

$$\begin{aligned}|\mathcal{M}_{\alpha,j+1}(-hp, k-1)| &\approx \sum_{i=1}^{+\infty}\left|\frac{(-hp)^{-i}(n-1)^{-i\alpha}}{\boldsymbol{\Gamma}(-i\alpha+j+1)}\right| \leq M\sum_{i=1}^{+\infty}|(-hp)^{-i}(k-2)^{-i\alpha}|\\ &= M\frac{1}{|-hp(k-2)^{\alpha}|-1} \to 0.\end{aligned} \tag{26}$$

Therefore, $\mathcal{M}_{\alpha,j+1}(\lambda,k-1)$ is convergent.

Next, we prove $\overline{x}_i(k)$ converges to $\mathbf{0}$ with oscillation in $1<\alpha<2$ case.

Considering the related nabla transform of $\mathcal{M}_{\alpha,2+\sigma}(\lambda,k)$, we get

$$z\mathcal{N}\{\mathcal{M}_{\alpha,2+\sigma}(\lambda,k-1)\}=\mathcal{N}\{\mathcal{M}_{\alpha,1+\sigma}(\lambda,k-1)\}=\mathcal{N}\{\nabla^{-\sigma}\mathcal{M}_{\alpha,1}(\lambda,k-1)\}=\frac{z^{\alpha-\sigma-1}(1-z)}{z^{\alpha}(1-z)-\lambda},\tag{27}$$

with $0<\sigma<\alpha-1$.

By introducing the final value theorem[32], we have

$$\lim_{k\to+\infty}\mathcal{M}_{\alpha,2+\sigma}(\lambda,k-1)=\lim_{z\to 0}\frac{z^{\alpha-\sigma-1}(1-z)}{z^{\alpha}(1-z)-\lambda}=0.\tag{28}$$

With the help of the nabla transform of $\nabla^{-\sigma}\mathcal{M}_{\alpha,1}(\lambda,k-1)$, we obtain

$$\begin{aligned}\lim_{z\to 0}\mathcal{N}\{\nabla^{-\sigma}\mathcal{M}_{\alpha,1}(\lambda,k-1)\}&=\lim_{z\to 0}\sum_{k=1}^{+\infty}(1-z)^{k-1}\nabla^{-\sigma}\mathcal{M}_{\alpha,1}(\lambda,k-1)\\&=\sum_{k=1}^{+\infty}\lim_{z\to 0}(1-z)^{k-1}\nabla^{-\sigma}\mathcal{M}_{\alpha,1}(\lambda,k-1)\\&=\sum_{k=1}^{+\infty}\nabla^{-\sigma}\mathcal{M}_{\alpha,1}(\lambda,k-1)\\&=0.\end{aligned}\tag{29}$$

Therefore, the sign of $\nabla^{-\sigma}\mathcal{M}_{\alpha,1}(\lambda,k-1)$ is converted between positive and negative which implies that the sign of $\mathcal{M}_{\alpha,1}(\lambda,k)$ must be changed. According to initial condition $\nabla\overline{\boldsymbol{x}}_i(0)=\mathbf{0}$ of Theorem 3.2, we obtain $\overline{x}_i(k)=\overline{x}_i(0)\mathcal{M}_{\alpha,1}(-hp_i,k-1)$. From the above discussions, $\overline{x}_i(k)$ converges to 0 with oscillation. ∎

Based on the mentioned convergence analysis above, we put forward the following properties for proposed algorithm.

Theorem 3.3 *Under Assumptions* 1 *and* 2, *let* $\boldsymbol{x}_i(k)$ *be generated by Algorithm* 1.

1) *For two iteration orders* α_1 *and* α_2 *with corresponding vector* $\overline{\boldsymbol{x}}_{\alpha_1}(k)$ *and* $\overline{\boldsymbol{x}}_{\alpha_2}(k)$, *respectively. If* $\alpha_1>\alpha_2$, *then* $\overline{\boldsymbol{x}}_{\alpha_1}(k)$ *converges to the optimal value with a faster speed than that of* $\overline{\boldsymbol{x}}_{\alpha_2}(k)$.

2) *For two iteration step-sizes* h_1 *and* h_2 *with corresponding vectors* $\overline{\boldsymbol{x}}_{\alpha_1}(k)$ *and* $\overline{\boldsymbol{x}}_{\alpha_2}(k)$, *respectively. If* $h_1>h_2$, *then* $\overline{\boldsymbol{x}}_{\alpha_1}(k)$ *converges to the optimal value with a faster speed than that of* $\overline{\boldsymbol{x}}_{\alpha_2}(k)$.

Proof 1) Due to $\alpha_1>\alpha_2$, the solutions of (19) are

$$\begin{cases}\overline{x}_{i_1}(k)=\overline{x}_i(0)\mathcal{M}_{\alpha_1,1}(-hp_i,k-1)+\nabla^1\overline{x}_i(0)\mathcal{M}_{\alpha_1,2}(-hp_i,k-1),\\ \overline{x}_{i_2}(k)=\overline{x}_i(0)\mathcal{M}_{\alpha_2,1}(-hp_i,k-1)+\nabla^1\overline{x}_i(0)\mathcal{M}_{\alpha_2,2}(-hp_i,k-1).\end{cases}\tag{30}$$

To analyze the convergent performance, we take $\overline{x}_{i_1}(n_1) = \overline{x}_{i_2}(n_2)$, namely,

$$\sum_{j=0}^{+\infty}\left[\overline{x}_i(0)(-hp_i)^j\frac{\boldsymbol{\Gamma}(j\alpha_1+n_1)}{\boldsymbol{\Gamma}(j\alpha_1+1)\boldsymbol{\Gamma}(n_1)}+L_1(j)\right]=\sum_{j=0}^{+\infty}\left[\overline{x}_i(0)(-hp_i)^j\frac{\boldsymbol{\Gamma}(j\alpha_2+n_2)}{\boldsymbol{\Gamma}(j\alpha_2+1)\boldsymbol{\Gamma}(n_2)}+L_2(j)\right], \tag{31}$$

where $L_1(j) = \nabla^1\overline{x}_i(0)(-hp_i)^j\frac{\boldsymbol{\Gamma}(j\alpha_1+n_1+1)}{\boldsymbol{\Gamma}(j\alpha_1+2)\boldsymbol{\Gamma}(n_1)}$ and $L_2(j) = \nabla^1\overline{x}_i(0)(-hp_i)^j\frac{\boldsymbol{\Gamma}(j\alpha_1+n_2+1)}{\boldsymbol{\Gamma}(j\alpha_2+2)\boldsymbol{\Gamma}(n_2)}$.

For the j-th term of the both sides of (31),

$$\begin{cases}\overline{x}_i(0)(-hp_i)^j\dfrac{\boldsymbol{\Gamma}(j\alpha_1+n_1)}{\boldsymbol{\Gamma}(j\alpha_1+1)\boldsymbol{\Gamma}(n_1)} = \overline{x}_i(0)(-hp_i)^j\dfrac{\boldsymbol{\Gamma}(j\alpha_2+n_2)}{\boldsymbol{\Gamma}(j\alpha_2+1)\boldsymbol{\Gamma}(n_2)},\\ \nabla^1\overline{x}_i(0)(-hp_i)^j\dfrac{\boldsymbol{\Gamma}(j\alpha_1+n_1+1)}{\boldsymbol{\Gamma}(j\alpha_1+2)\boldsymbol{\Gamma}(n_1)} = \nabla^1\overline{x}_i(0)(-hp_i)^j\dfrac{\boldsymbol{\Gamma}(j\alpha_1+n_2+1)}{\boldsymbol{\Gamma}(j\alpha_2+2)\boldsymbol{\Gamma}(n_2)}.\end{cases} \tag{32}$$

For (32),

$$\begin{cases}\dfrac{\boldsymbol{\Gamma}(j\alpha_1+n_1)}{\boldsymbol{\Gamma}(j\alpha_1+1)\boldsymbol{\Gamma}(n_1)} = \dfrac{\boldsymbol{\Gamma}(j\alpha_2+n_2)}{\boldsymbol{\Gamma}(j\alpha_2+1)\boldsymbol{\Gamma}(n_2)},\\ \dfrac{\boldsymbol{\Gamma}(j\alpha_1+n_1+1)}{\boldsymbol{\Gamma}(j\alpha_1+2)\boldsymbol{\Gamma}(n_1)} = \dfrac{\boldsymbol{\Gamma}(j\alpha_1+n_2+1)}{\boldsymbol{\Gamma}(j\alpha_2+2)\boldsymbol{\Gamma}(n_2)}.\end{cases} \tag{33}$$

Defining $f_1(n,\alpha) = \frac{\boldsymbol{\Gamma}(i\alpha+n)}{\boldsymbol{\Gamma}(i\alpha+1)\boldsymbol{\Gamma}(n)}$, we have

$$\begin{aligned} f_1(n,\alpha) &= \frac{\boldsymbol{\Gamma}(i\alpha+n)}{\boldsymbol{\Gamma}(i\alpha+1)\boldsymbol{\Gamma}(n)} = \frac{(i\alpha+n-1)\boldsymbol{\Gamma}(i\alpha+n-1)}{(n-1)\boldsymbol{\Gamma}(i\alpha+1)\boldsymbol{\Gamma}(n-1)}\\ &= \left(\frac{i\alpha}{n-1}+1\right)\frac{\boldsymbol{\Gamma}(i\alpha+n-1)}{\boldsymbol{\Gamma}(i\alpha+1)\boldsymbol{\Gamma}(n-1)}\\ &= \left(\frac{i\alpha}{n-1}+1\right)\left(\frac{i\alpha}{n-2}+1\right)\frac{\boldsymbol{\Gamma}(i\alpha+n-2)}{\boldsymbol{\Gamma}(i\alpha+1)\boldsymbol{\Gamma}(n-2)}\\ &= \prod_{j=1}^{k}\left(\frac{i\alpha}{n-j}+1\right)\frac{\boldsymbol{\Gamma}(i\alpha+n-k)}{\boldsymbol{\Gamma}(i\alpha+1)\boldsymbol{\Gamma}(n-k)}\\ &= \prod_{j=1}^{n-1}\left(\frac{i\alpha}{n-j}+1\right). \end{aligned} \tag{34}$$

Similarly, considering $f_2(n,\alpha) = \frac{\boldsymbol{\Gamma}(i\alpha+n+1)}{\boldsymbol{\Gamma}(i\alpha+1)\boldsymbol{\Gamma}(n)}$,

$$f_2(n,\alpha) = n\prod_{j=0}^{n-2}\left(\frac{i\alpha}{n-j}+1\right). \tag{35}$$

Combining (34) with (35), there is a remarkable conclusion that $f_1(n,\alpha)$ and $f_2(n,\alpha)$ are increasing function on n and α. Combining (34) with $\alpha_1 > \alpha_2$, n_1 is smaller than n_2. (31) indicates that a larger α takes a faster convergent rate.

2) The proof of the second condition of Theorem 3.3 is similar to the first condition. Therefore,

$$\begin{cases} (h_1)^i \dfrac{\boldsymbol{\Gamma}(i\alpha+n_1)}{\boldsymbol{\Gamma}(i\alpha+1)\boldsymbol{\Gamma}(n_1)} = (h_2)^i \dfrac{\boldsymbol{\Gamma}(i\alpha+n_2)}{\boldsymbol{\Gamma}(i\alpha+1)\boldsymbol{\Gamma}(n_2)}, \\ (h_1)^i \dfrac{\boldsymbol{\Gamma}(i\alpha+n_1+1)}{\boldsymbol{\Gamma}(i\alpha+2)\boldsymbol{\Gamma}(n_1)} = (h_2)^i \dfrac{\boldsymbol{\Gamma}(i\alpha+n_2+1)}{\boldsymbol{\Gamma}(i\alpha+2)\boldsymbol{\Gamma}(n_2)}, \end{cases} \tag{36}$$

which indicates that a larger h takes a faster convergent rate. ∎

3.3 Distributed Algorithm with Switching Iteration Order

As analyzed in Theorem 3.2, we know that the response speed of Algorithm 1 with $1 < \alpha < 2$ is faster than that of $0 < \alpha \leq 1$ but with oscillation, which may deteriorate the convergent performance.

To improve the convergent performance, we take a center node to access the information $\boldsymbol{e}(k) = c\boldsymbol{e}_1(k) + (1-c)\boldsymbol{e}_2(k)$, where $c \in (0,1)$, $\boldsymbol{e}_1(k) = \sum_{i\in\mathcal{V}} \|\boldsymbol{z}_i - \boldsymbol{H}_i\boldsymbol{x}_i(k)\|$, and $\boldsymbol{e}_2(k) = \sum_{i\in\mathcal{V}}\sum_{j\in\mathcal{N}_i} \|\boldsymbol{x}_i(k) - \boldsymbol{x}_j(k)\|$. Therefore, if $\boldsymbol{e}(k) \geq \delta$, which denotes that $\boldsymbol{x}_i(k)$ is far from the exact solution $\boldsymbol{x}^*$ or its neighbours' solution $\boldsymbol{x}_j(k)$, we take a larger α_2 $(1 < \alpha_2 < 2)$ to achieve a fast response speed and if $\boldsymbol{e}(k) < \delta$, which means that $\boldsymbol{x}_i(k)$ gradually closes to the exact solution $\boldsymbol{x}^*$ and its neighbours' solution $\boldsymbol{x}_j(k)$, $\forall j \in \mathcal{N}_i$, we switch the iteration order to a smaller α_1 $(0 < \alpha_1 \leq 1)$. Then we modify Algorithm 1 with switching iteration order scheme in Algorithm 2.

Algorithm 2 Distributed algorithm with switching iteration order

Initialization: For each $i \in \mathcal{V}$,

h, $\boldsymbol{H}_i \in \mathbb{R}^{r_i\times m}$, $\boldsymbol{z}_i \in \mathbb{R}^{r_i}$, $\boldsymbol{x}_i(0) \in \mathbb{R}^m$, and $\boldsymbol{x}_j(0) \in \mathbb{R}^m, j \in \mathcal{N}_i$, $\alpha = \alpha_2 \in (1,2)$, $\delta > 0$.

Update flows: For each $i \in \mathcal{V}$,

For $k = 1$ to K **do**

$$\nabla^\alpha \boldsymbol{x}_i(k) = h\Big[\sum_{j\in\mathcal{N}_i}(\boldsymbol{x}_j(k-1) - \boldsymbol{x}_i(k-1)) - \boldsymbol{H}_i^{\mathrm{T}}(\boldsymbol{H}_i\boldsymbol{x}_i(k-1) - \boldsymbol{z}_i)\Big],$$

$$\boldsymbol{e}_1(k) = \sum_{i\in\mathcal{V}} \|\boldsymbol{z}_i - \boldsymbol{H}_i\boldsymbol{x}_i(k)\|, \quad \boldsymbol{e}_2(k) = \sum_{i\in\mathcal{V}}\sum_{j\in\mathcal{N}_i} \|\boldsymbol{x}_i(k) - \boldsymbol{x}_j(k)\|,$$

$$\boldsymbol{e}(k) = c\boldsymbol{e}_1(k) + (1-c)\boldsymbol{e}_2(k),$$

If $\boldsymbol{e}(k)/\boldsymbol{e}(0) < \delta$

Switching iteration order α from α_2 to α_1 with $0 < \alpha_1 < 1$.

End if

End for

We rewrite Algorithm 2 as the following compact form

$$\begin{cases} \boldsymbol{x}(k+1) = (\boldsymbol{I} - h\boldsymbol{F_d})\boldsymbol{x}(k) + h\boldsymbol{z_H} - \boldsymbol{\Phi}_{11}(\alpha_1, k), \quad \boldsymbol{e} \le \delta; \\ \boldsymbol{x}(k) = (2\boldsymbol{I} - h\boldsymbol{F_d})\boldsymbol{x}(k-1) - \boldsymbol{x}(k-2) + h\boldsymbol{z_H} - \boldsymbol{\Phi}_{12}(\alpha_2, k-1), \quad \boldsymbol{e} > \delta, \end{cases} \tag{37}$$

where $\boldsymbol{g}_r(k, \alpha_r) = (-1)^{k+1}\binom{\alpha_r - r}{k+1}$ and $\boldsymbol{\Phi}_{1r}(k) = \boldsymbol{g}_r(k, \alpha_r) * \nabla^r \boldsymbol{x}(k)$. Then we have the following convergence result.

Corollary 3.4 *If Assumptions* 1 *and* 2 *hold, then for all* $k > 0$ *and* $\boldsymbol{x}_i(k)$ *is generated by Algorithm* 2*, then* $\boldsymbol{x}_i(k)$ *converges to the exact solution* $\boldsymbol{x}^*$.

Proof The proof of this corollary can be completed by extending the counterparts of Theorem 3.2. According to the 2-nd part of the proof of Theorem 3.2, if $\boldsymbol{e}(k) \ge \delta$, then $\boldsymbol{e}(k)$ decrease to $\boldsymbol{0}$ with a fast speed. Taking assumption that $\boldsymbol{e}(k) = \delta$ at the k_i-th iteration and referring to the 1-st part of the proof of Theorem 3.2, $\boldsymbol{e}(k)$ monotonously converges to $\boldsymbol{0}$ for $k > k_i$. ∎

Recalling to the proof of Corollary 3.4, Algorithm 2 switches the iteration order α from α_2 to α_1 at the k_i-th iteration, which implies

$$\begin{cases} \boldsymbol{x}(k_i+1) = (\boldsymbol{I} - h\boldsymbol{F_d})\boldsymbol{x}(k_i) + h\boldsymbol{z_H} - \boldsymbol{\Phi}_{11}(\alpha_1, k_i), \\ \boldsymbol{x}(k_i) = (2\boldsymbol{I} - h\boldsymbol{F_d})\boldsymbol{x}(k_i-1) - \boldsymbol{x}(k_i-2) + h\boldsymbol{z_H} - \boldsymbol{\Phi}_{12}(\alpha_2, k_i-1). \end{cases} \tag{38}$$

Note that

$$\begin{aligned} \boldsymbol{x}(k_i+1) - \boldsymbol{x}(k_i) &= (\boldsymbol{I} - h\boldsymbol{F_d})\nabla\boldsymbol{x}(k_i) - \nabla\boldsymbol{x}(k_i-1) + \boldsymbol{\Phi}_{12}(\alpha_2, k_i-1) - \boldsymbol{\Phi}_{11}(\alpha_1, k_i) \\ &\approx \nabla^2\boldsymbol{x}(k_i) - h\boldsymbol{F_d}\nabla\boldsymbol{x}(k_i) + \boldsymbol{\Phi}_{12}(\alpha_2, k_i) - \boldsymbol{\Phi}_{11}(\alpha_1, k_i). \end{aligned} \tag{39}$$

Due to $\boldsymbol{g}_r(k, \alpha_r) = (-1)^{k+1}\binom{\alpha_r - r}{k+1}$, $\boldsymbol{\Phi}_{1r}(k) = \boldsymbol{g}_r(k, \alpha_r) * \nabla^r \boldsymbol{x}(k)$, and $r = 1, 2$,

$$\begin{aligned} &\boldsymbol{x}(k_i+1) - \boldsymbol{x}(k_i) \\ &\approx \sum_{j=0}^{k_i}(-1)^j\binom{\alpha_2 - 2}{j}\nabla^2\boldsymbol{x}(k_i - j) - \sum_{j=0}^{k_i}(-1)^j\binom{\alpha_1 - 1}{j}\nabla\boldsymbol{x}(k_i - j) + (\boldsymbol{I} - h\boldsymbol{F_d})\nabla\boldsymbol{x}(k_i). \end{aligned} \tag{40}$$

Taking $\alpha_2 = \alpha_1 + 1$,

$$\begin{aligned} \boldsymbol{x}(k_i+1) - \boldsymbol{x}(k_i) &\approx \sum_{j=0}^{k_i}(-1)^j\binom{\alpha_1 - 1}{j}[\nabla^2\boldsymbol{x}(k_i - j) - \nabla\boldsymbol{x}(k_i - j)] + (\boldsymbol{I} - h\boldsymbol{F_d})\nabla\boldsymbol{x}(k_i) \\ &= -\nabla^{\alpha_1}\boldsymbol{x}(k_i - 1) + (\boldsymbol{I} - h\boldsymbol{F_d})\nabla\boldsymbol{x}(k_i). \end{aligned} \tag{41}$$

According to (41), there exists a mutation when the iteration order is changed based on (41). Therefore, for the purpose of reducing the mutation, we modify the switching order scheme as follows:

$$\begin{cases} \boldsymbol{x}(k+1) = (\boldsymbol{I} - h\boldsymbol{F_d})\boldsymbol{x}(k) + h\boldsymbol{z_H} - \overline{\boldsymbol{\Phi}}_{11}(\alpha_1, k), \quad \boldsymbol{e} \le \delta; \\ \boldsymbol{x}(k) = (2\boldsymbol{I} - h\boldsymbol{F_d})\boldsymbol{x}(k-1) - \boldsymbol{x}(k-2) + h\boldsymbol{z_H} - \overline{\boldsymbol{\Phi}}_{12}(\alpha_2, k-1), \quad \boldsymbol{e} > \delta, \end{cases} \tag{42}$$

where $\overline{\boldsymbol{\Phi}}_{11}(k) = (1-\gamma)\boldsymbol{\Phi}_{11}(k) + \gamma\boldsymbol{\Phi}_{12}(k)$ and $\gamma \in (0, 1)$.

Corollary 3.5 *If Assumptions* 1 *and* 2 *hold and* $\boldsymbol{x}_i(k)$ *is generated by* (42)*, then* $\boldsymbol{x}_i(k)$ *converges to the exact solution* $\boldsymbol{x}^*$*.*

Proof The first formula of (42) is the convex combination of two formulas of Corollary 3.4. So that it is not diffucult to achieve the convergence by referring to the proof of Theorem 3.2 and Corollary 3.4. ∎

According to the analysis of (41), we get that

$$\boldsymbol{x}(k_i+1)-\boldsymbol{x}(k_i)\approx-(1-\gamma)(\nabla^{\alpha_1}\boldsymbol{x}(k_i-1)+(\boldsymbol{I}-h\boldsymbol{F_d})\nabla\boldsymbol{x}(k_i)). \tag{43}$$

Compared with (41), the mutation is weakened by $1-\gamma$ times. The effectiveness of (42) is increasing with the growth of γ. Therefore, the iteration order swithcing acheme in (42) improve the convergence speed of $\boldsymbol{x}(k)$.

3.4 Approximation of Fractional Order Difference Operator

Based on (12), we find that the realization of $\boldsymbol{g}(k)$ is related to the convolution of previous information which causes a plenty of computation at every iteration step. Then, we provide an approximate method for the fractional order difference operator to reduce the computation burden. Firstly, the transfer function of $\boldsymbol{g}(k)$ is

$$G(s)=\sum_{n=0}^{\infty}(-1)^{j+1}\binom{\alpha-1}{j+1}s^{-j}=s\left[(1-s^{-1})^{\alpha-1}-1\right]. \tag{44}$$

However, $G(s)$ has an $(\alpha-1)$-th order fractional integrator. Then, we make a continued fraction expansion for $(1-s^{-1})^{\alpha-1}$ by using Maple toolbox as detailed in [45].

According to the above discussion, we get the following two iteration schemes:

For $0<\alpha<1$, we get $t=1$. Then, we get

$$\boldsymbol{G}_1(s)\approx\frac{a_1+a_2s^{-1}+a_3s^{-2}+a_4s^{-3}}{b_0+b_1s^{-1}+b_2s^{-2}+b_3s^{-3}}=\overline{\boldsymbol{G}}_1(s). \tag{45}$$

Then we take

$$\begin{aligned}\boldsymbol{\Phi}_1(k)&=\overline{\boldsymbol{G}}_1(s)\nabla\boldsymbol{x}(k)\\&=\frac{1}{b_0}[(a_1\nabla\boldsymbol{x}(k)+a_2\nabla\boldsymbol{x}(k-1)+a_3\nabla\boldsymbol{x}(k-2)\\&\quad+a_4\nabla\boldsymbol{x}(k-3))-(b_1\boldsymbol{\Phi}_1(k-1)+b_2\boldsymbol{\Phi}_1(k-2)+b_3\boldsymbol{\Phi}_1(k-3)].\end{aligned} \tag{46}$$

The iteration scheme is

$$\boldsymbol{x}(k)=(\boldsymbol{I}-h\boldsymbol{F_d})\boldsymbol{x}(k-1)+h\boldsymbol{z_H}-\boldsymbol{\Phi}_1(k-1). \tag{47}$$

For $1<\alpha<2$, we get $t=2$. Therefore,

$$\begin{aligned}\boldsymbol{\Phi}_1(k-1)&=\boldsymbol{g}(k-1)*\nabla^2\boldsymbol{x}(k-1)\\&=\sum_{j=0}^{k-1}\binom{\alpha-2}{j+1}\nabla^2\boldsymbol{x}(k-j-1)=\sum_{j=0}^{k-1}\binom{\alpha-2}{j+1}(\nabla\boldsymbol{x}(k-j)-\nabla\boldsymbol{x}(k-j-1))\end{aligned}$$

$$= \sum_{j=0}^{k} \binom{\alpha-2}{j+1} \nabla \boldsymbol{x}(k-j) - \nabla \boldsymbol{x}(0) - \sum_{j=0}^{k-1} \binom{\alpha-2}{j+1} \nabla \boldsymbol{x}(k-j-1)$$
$$= \boldsymbol{\Psi}_1(k) - \boldsymbol{\Psi}_1(k-1), \tag{48}$$

where the last equality follows from $\nabla \boldsymbol{x}(0) = \boldsymbol{0}$ and $\boldsymbol{\Psi}_1(k) = \boldsymbol{g}(k) * \nabla \boldsymbol{x}(k)$. Through Maple toolbox,

$$\boldsymbol{G}_2(s) \approx \frac{c_1 + c_2 s^{-1} + c_3 s^{-2} + c_4 s^{-3}}{d_0 + d_1 s^{-1} + d_2 s^{-2} + d_3 s^{-3}} = \overline{\boldsymbol{G}}_2(s). \tag{49}$$

Then we take

$$\begin{aligned} \boldsymbol{\Psi}_1(k) &= \overline{\boldsymbol{G}}_2(s) \nabla \boldsymbol{x}(k) \\ &= \frac{1}{d_0}[(c_1 \nabla \boldsymbol{x}(k) + c_2 \nabla \boldsymbol{x}(k-1) + c_3 \nabla \boldsymbol{x}(k-2) + c_4 \nabla \boldsymbol{x}(k-3)) \\ &\quad - (d_1 \boldsymbol{\Psi}_1(k-1) + d_2 \boldsymbol{\Psi}_1(k-2) + d_3 \boldsymbol{\Psi}_1(k-3)]. \end{aligned} \tag{50}$$

Because of $\nabla^2 \boldsymbol{x}(k) = \boldsymbol{x}(k) - 2\boldsymbol{x}(k-1) + \boldsymbol{x}(k-2)$, the iteration scheme is

$$\boldsymbol{x}(k) = (2\boldsymbol{I} - h\boldsymbol{F}_{\boldsymbol{d}})\boldsymbol{x}(k-1) - \boldsymbol{x}(k-2) + h\boldsymbol{z}_{\boldsymbol{H}} - \boldsymbol{\Psi}_1(k) + \boldsymbol{\Psi}_1(k-1). \tag{51}$$

Next, we introduce a new method to solve the problem of a mutation when switching iteration order α. The main reason of mutation is that $\boldsymbol{x}(k)$ is updating continuously before switching but the $\boldsymbol{\Phi}_1(k)$ of smaller α does not update after switching. Hence, we need to utilize the known information to deploy the initial condition of $\boldsymbol{\Phi}_1(k)$ of smaller α so that the process of converting iteration order is smooth without a mutation. For example, we switch $\alpha_1 = 1.5$ to $\alpha_2 = 1.3$ when $k = k_s$. At this moment, we have some known informations that are the value of $\boldsymbol{x}(k_s + 1)$ and the $\boldsymbol{\Phi}_1(k_s - 1)$ of $\alpha_1 = 1.5$, however, the value of $\boldsymbol{\Phi}_1(k_s - 1)$ of $\alpha_2 = 1.3$ is in initial condition and unchanged. Therefore, there exists a mutation in this process. Our goal is that $\boldsymbol{x}(k_s) = \boldsymbol{x}(k_s + 1)$ through deploying the initial condition of $\boldsymbol{\Phi}_1(k_s - 1)$ of $\alpha_2 = 1.3$. Via (51), before switching, we get

$$\boldsymbol{x}(k_s) = (2\boldsymbol{I} - h\boldsymbol{F}_{\boldsymbol{d}})\boldsymbol{x}(k_s - 1) - \boldsymbol{x}(k_s - 2) + h\boldsymbol{z}_{\boldsymbol{H}} - \boldsymbol{\Psi}_1(k_s) + \boldsymbol{\Psi}_1(k_s - 1). \tag{52}$$

After switching, we obtain

$$\boldsymbol{x}(k_s + 1) = (2\boldsymbol{I} - h\boldsymbol{F}_{\boldsymbol{d}})\boldsymbol{x}(k_s) - \boldsymbol{x}(k_s - 1) + h\boldsymbol{z}_{\boldsymbol{H}} - \boldsymbol{\Psi}_1(k_s + 1) + \boldsymbol{\Psi}_1(k_s), \tag{53}$$

where $\boldsymbol{\Psi}_2(k) = \boldsymbol{g}(k) * \nabla \boldsymbol{x}(k)$. Then we need to deploy the initial condition of $\boldsymbol{\Psi}_2(k_s)$ of $\alpha = 1.3$. Based on (49), we assume that

$$\begin{aligned} \boldsymbol{\Psi}_2(k) &= \frac{1}{e_0}[(f_1 \nabla \boldsymbol{x}(k) + f_2 \nabla \boldsymbol{x}(k-1) + f_3 \nabla \boldsymbol{x}(k-2) + f_4 \nabla \boldsymbol{x}(k-3)) \\ &\quad - (e_1 \boldsymbol{\Psi}_2(k-1) + e_2 \boldsymbol{\Psi}_2(k-2) + e_3 \boldsymbol{\Psi}_2(k-3)]. \end{aligned} \tag{54}$$

Therefore,

$$\begin{aligned}
\boldsymbol{\Psi}_2(k-1) &= \frac{1}{e_0}[(f_1\nabla\boldsymbol{x}(k-1)+f_2\nabla\boldsymbol{x}(k-2)+f_3\nabla\boldsymbol{x}(k-3)+f_4\nabla\boldsymbol{x}(k-4))\\
&\quad-(e_1\boldsymbol{\Psi}_2(k-2)+e_2\boldsymbol{\Psi}_2(k-3)+e_3\boldsymbol{\Psi}_2(k-4)],\\
\boldsymbol{\Psi}_2(k-2) &= \frac{1}{e_0}[(f_1\nabla\boldsymbol{x}(k-2)+f_2\nabla\boldsymbol{x}(k-3)+f_3\nabla\boldsymbol{x}(k-4)+f_4\nabla\boldsymbol{x}(k-5))\\
&\quad-(e_1\boldsymbol{\Psi}_2(k-3)+e_2\boldsymbol{\Psi}_2(k-4)+e_3\boldsymbol{\Psi}_2(k-5)],\\
\boldsymbol{\Psi}_2(k-3) &= \frac{1}{e_0}[(f_1\nabla\boldsymbol{x}(k-3)+f_2\nabla\boldsymbol{x}(k-4)+f_3\nabla\boldsymbol{x}(k-5)+f_4\nabla\boldsymbol{x}(k-6))\\
&\quad-(e_1\boldsymbol{\Psi}_2(k-4)+e_2\boldsymbol{\Psi}_2(k-5)+e_3\boldsymbol{\Psi}_2(k-6)],\\
\boldsymbol{\Psi}_2(k-4) &= \frac{1}{e_0}[(f_1\nabla\boldsymbol{x}(k-4)+f_2\nabla\boldsymbol{x}(k-5)+f_3\nabla\boldsymbol{x}(k-6)+f_4\nabla\boldsymbol{x}(k-6))\\
&\quad-(e_1\boldsymbol{\Psi}_2(k-5)+e_2\boldsymbol{\Psi}_2(k-6)+e_3\boldsymbol{\Psi}_2(k-7)]. \qquad (55)
\end{aligned}$$

For convenience of calculation, we assume that the values of $\boldsymbol{\Psi}_2(k-6)$ and $\boldsymbol{\Psi}_2(k-7)$ are $\mathbf{0}$. Therefore, we compute the values of $\boldsymbol{\Psi}_2(k_s)$ and $\boldsymbol{\Psi}_2(k-1)$ via (52)–(55). By doing this, we restart initial condition of $\boldsymbol{\Psi}_2(k)$ so that there exist no mutation at the switching iteration. The response speed and convergence speed of fractional order algorithm are better than the original algorithm which is $\alpha=1$. Because the amplitude of α $(1<\alpha<2)$ is larger when α is larger but the response speed of α is faster, we make a multiple switching. For example, we switch $\alpha_1=1.5$ to $\alpha_2=1.3$ firstly, then switch $\alpha_2=1.3$ to $\alpha_3=1.1$, finally switch $\alpha_3=1.1$ to $\alpha_4=1$. In the above process of switching, the δ becomes small gradually, besides, we take a restarting initial condition of $\boldsymbol{\Phi}_1(k)$ when making a switching. By doing this, we take the advantages of each α to further improve the response speed and convergence speed of fractional order algorithm.

Remark 3.6 As shown in (12), the computation of fractional order dynamics at every step is a convolution, it needs all the history information and large computation burdens. For dealing with this issue, we construct $G(s)$ as the transfer function of $g(t)$ in (12) and provide its approximation to reduce the computation burdens but still have a similar performance of fractional order dynamics. Besides, the work is helpful for us to reduce further the mutation by taking a restartion initial condition.

4 Numerical Simulations

In this section, we take $\boldsymbol{H}$ and $\boldsymbol{z}$ as follows to verify the effectiveness of the proposed method.

$$\boldsymbol{H}=\begin{bmatrix} 1.20 & -0.60\\ -0.40 & -1.00\\ -0.60 & 2.50\\ -1.25 & -1.00 \end{bmatrix},\quad \boldsymbol{z}=\begin{bmatrix} 0.36\\ -1.32\\ 2.02\\ -2.00 \end{bmatrix},$$

where the exact solution is $\boldsymbol{x}^* = [0.80\ 1.00]^{\mathrm{T}}$. Consider a multi-agent system with 4 agents to solve this equation and the communication topology is shown in Figure 1.

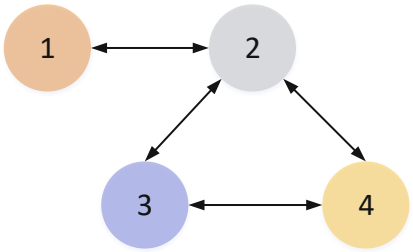
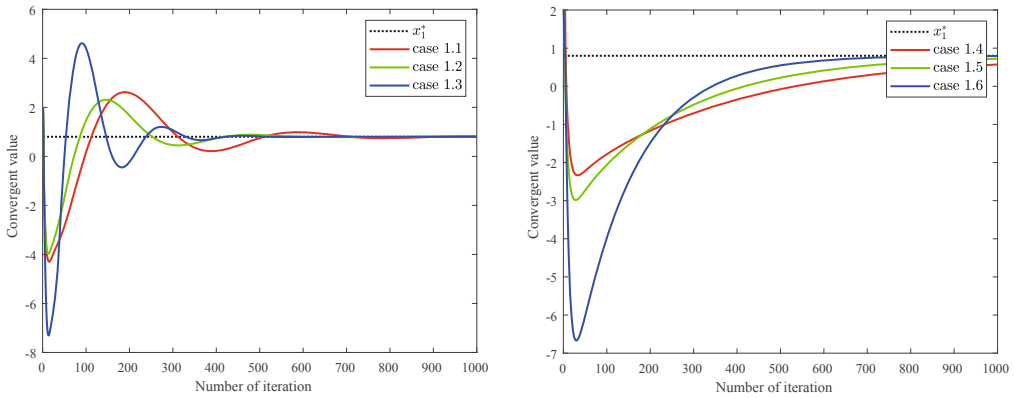


Figure 1 The communication graph

To verify the effectiveness of the results of this paper, we consider the following three examples.

Example 4.1 We take different α and step-size h as following cases to verify Theorems 3.2 and 3.3 and present the curve of the first element of $\boldsymbol{x}_1(k)$ in Figure 2.

$$\begin{cases} \text{Case 1.1}: \alpha = 1.30, \quad h = 0.02; \\ \text{Case 1.2}: \alpha = 1.30, \quad h = 0.03; \\ \text{Case 1.3}: \alpha = 1.50, \quad h = 0.03; \\ \text{Case 1.4}: \alpha = 0.80, \quad h = 0.02; \\ \text{Case 1.5}: \alpha = 0.80, \quad h = 0.03; \\ \text{Case 1.6}: \alpha = 0.95, \quad h = 0.03. \end{cases} \tag{56}$$



(a) (b)

Figure 2 (a) The numerical simulations of $\boldsymbol{x}(k)$ with $0 < \alpha \leq 1$ and different step-sizes; (b) The numerical simulations of $\boldsymbol{x}(k)$ with $1 < \alpha < 2$ and different step sizes

From Figure 2, we have

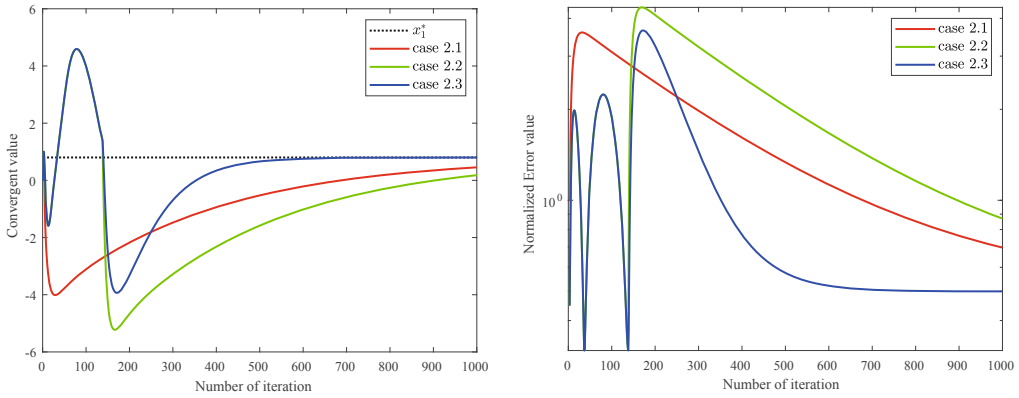
1) For $0 < \alpha \leq 1$, $\boldsymbol{x}_1(k)$ asymptotically converges to $\boldsymbol{x}^*$. Moreover, there exists a k_1 such that $\boldsymbol{x}_1(k)$ asymptotically converges to $\boldsymbol{x}^*$ for $k \geq k_1$;

2) For $1 < \alpha < 2$, $\boldsymbol{x}_1(k)$ asymptotically converges to $\boldsymbol{x}^*$ with oscillation;

3) A larger iteration order α or step-size h can take a faster convergence and response speed.

Example 4.2 Based on Theorem 3.2 and Theorem 3.3, we find that $1 < \alpha < 2$ takes a faster response speed but with oscillation. We propose a switching iteration order scheme to improve the convergence speed. However, there exists a mutation at the switching step. For reducing mutation, we design a switching scheme (42) by introducing a convex combination parameter γ. We take this case to verify the effectiveness of Algorithm 2 and the iteration order swithcing scheme in (42) and provide the simulation results in Figure 3.

$$\begin{cases} \text{Case 2.1} : \alpha = 0.80, \quad h = 0.02; \\ \text{Case 2.2} : \alpha = 1.30 \text{ to } 0.80, \quad h = 0.02, \delta = 0.39, c = 0.6; \\ \text{Case 2.3} : \alpha = 1.30 \text{ to } 0.80, \quad h = 0.02, \delta = 0.39, c = 0.6, \gamma = 0.75. \end{cases} \tag{57}$$

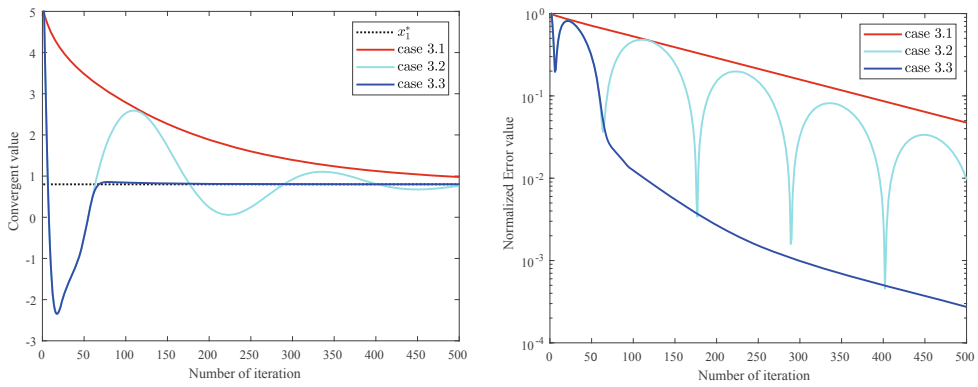


(a) (b)

Figure 3 (a) The curves of the first element of $\boldsymbol{x}_1(k)$; (b) The curves of the normalized errors $\|\boldsymbol{x}(k) - \boldsymbol{x}^*\|/\|\boldsymbol{x}(1) - \boldsymbol{x}^*\|$

Example 4.3 In Corollary 3.5, we put forward a switching iteration order scheme with a convex combination parameter γ to reduce the mutation. However, the mutation is not removed. Finally, we design a multi-switching iteration order scheme with restarting initial condition to remove the mutation. The new scheme not only removes this mutation,but also increases the convergence and response speed. The effectiveness of this method is demonstrated by Figure 4.

$$\begin{cases} \text{Case 3.1} : \alpha = 1.00, \quad h = 0.02; \\ \text{Case 3.2} : \alpha = 1.50, \quad h = 0.02; \\ \text{Case 3.3} : \text{Multi} - \text{switching } \alpha, \quad h = 0.02, \delta = 0.07. \end{cases} \tag{58}$$



(a) (b)

Figure 4 (a) The curves of the first element of $\boldsymbol{x}_1(k)$ with restarting initial condition; (b) The curves of the normalized errors $\|\boldsymbol{x}(k) - \boldsymbol{x}^*\|/\|\boldsymbol{x}(1) - \boldsymbol{x}^*\|$

5 Conclusion

The paper proposed a novel distributed algorithm with fractional order dynamics to solve linear algebraic equations. By introducing fractional order dynamics with more design freedom, it yielded a better convergent performance than that of the conventional first order algorithm. For $0 < \alpha < 2$, we proved that the proposed algorithm asymptotically converges to the optimal solution. Moreover, we designed a iteration order switching schemes to further accelerate the convergence rate and provided numerical examples to illustrate the effectiveness of the proposed algorithms.

References

[1] Hui Q, Haddad W M, and Bhat S P, Semistability, finite-time stability, differential inclusions, and discontinuous dynamical systems having a continuum of equilibria, *IEEE Transactions on Automatic Control*, 2009, **54**(10): 2465–2470.

[2] Chen C T, Introduction to the linear algebraic method for control system design, *IEEE Control Systems Magazine*, 1987, **7**(5): 36–42.

[3] Zhu S Y, Chen C L, Xu J M, et al., Mitigating quantization effects on distributed sensor fusion: A least squares approach, *IEEE Transactions on Signal Processing*, 2018, **66**(13): 3459–3474.

[4] Braunstein A, Muntoni A P, Pagnani A, et al., Compressed sensing reconstruction using expectation propagation, *Journal of Physics A: Mathematical and Theoretical*, 2020, **53**(18): 184001.

[5] Dafchahi F N, A new refinement of Jacobi method for solution of linear system equations $Ax = b$, *International Journal of Contemporary Mathematical Sciences*, 2008, **3**(17): 819–827.

[6] Vatti V and Gonfa G G, Refinement of generalized Jacobi (RGJ) method for solving system of linear equations, *International Journal of Contemporary Mathematical Sciences*, 2011, **6**(3): 109–116.

[7] Hurt J, Some stability theorems for ordinary difference equations, *SIAM Journal on Numerical Analysis*, 1967, **4**(4): 582–596.

[8] Ortega J M and Rockoff M L, Nonlinear difference equations and Gauss-Seidel type iterative methods, *SIAM Journal on Numerical Analysis*, 1966, **3**(3): 497–513.

[9] Wen Z W, Yin W T, and Zhang Y, Solving a low-rank factorization model for matrix completion by a nonlinear successive over-relaxation algorithm, *Mathematical Programming Computation*, 2012, **4**(4): 333–361.

[10] Allahviranloo T, Successive over relaxation iterative method for fuzzy system of linear equations, *Applied Mathematics and Computation*, 2005, **162**(1): 189–196.

[11] Yi P and Hong Y G, Distributed cooperative optimization and its applications, *Scientia Sinica Mathematica*, 2016, **46**(10): 1547–1564.

[12] Yang T, Yi X L, Wu J F, et al., A survey of distributed optimization, *Annual Reviews in Control*, 2019, **47**: 278–305.

[13] Shi W, Ling Q, Wu G, et al., Extra: An exact first-order algorithm for decentralized consensus optimization, *SIAM Journal on Optimization*, 2015, **20**(2): 944–966.

[14] Cheng S S, Liang S, and Fan Y, Distributed solving Sylvester equations with fractional order dynamics, *Control Theory and Technology*, 2021, **19**(2): 249–259.

[15] Mo L P, Liu X D, Cao X B, et al., Distributed second-order continuous-time optimization via adaptive algorithm with nonuniform gradient gains, *Journal of Systems Science and Complexity*, 2020, **33**(6): 1914–1932.

[16] Wang J X, Fu K L, Gu Y, et al., Convergence of distributed gradient-tracking-based optimization algorithms with random graphs, *Journal of Systems Science and Complexity*, 2021, **34**(4): 1438–1453.

[17] Yi P and Li L, Distributed nonsmooth convex optimization over Markovian switching random networks with two step-sizes, *Journal of Systems Science and Complexity*, 2021, **34**(1): 1–21.

[18] Liang S, Zeng X L, and Hong Y G, Distributed nonsmooth optimization with coupled inequality constraints via modified Lagrangian function, *IEEE Transactions on Automatic Control*, 2017, **63**(6): 1753–1759.

[19] Zeng X L, Yi P, and Hong Y G, Distributed continuous-time algorithm for constrained convex optimizations via nonsmooth analysis approach, *IEEE Transactions on Automatic Control*, 2016, **62**(10): 5227–5233.

[20] Yuan D M, Hong Y G, Daniel W C H, et al., Optimal distributed stochastic mirror descent for strongly convex optimization, *Automatica*, 2018, **90**: 196–203.

[21] Wang Y H, Zhao W X, Hong Y G, et al., Distributed subgradient-free stochastic optimization algorithm for nonsmooth convex functions over time-varying networks, *SIAM Journal on Control and Optimization*, **57**(4): 2821–2842.

[22] Benner P and Breiten T, On optimality of approximate low rank solutions of large-scale matrix equations, *Systems & Control Letters*, 2014, **67**: 55–64.

[23] Mou S S, Liu J, and Morse A S, A distributed algorithm for solving a linear algebraic equation, *IEEE Transactions on Automatic Control*, 2015, **63**(11): 2863–2878.

[24] Liu J, Mou S S, and Morse A S, Asynchronous distributed algorithms for solving linear algebraic equations, *IEEE Transactions on Automatic Control*, 2017, **63**(2): 372–385.

[25] Lei J L, Yi P, Shi G D, et al., Distributed algorithms with finite data rates that solve linear equations, *SIAM Journal on Optimization*, 2020, **30**(2): 1191–1222.

[26] Shi G D, Anderson B D O, and Helmke U, Network flows that solve linear equations, *IEEE Transactions on Automatic Control*, 2017, **62**(6): 2659–2674.

[27] Deng W, Zeng X L, and Hong Y G, Distributed computation for solving the Sylvester equation based on optimization, *IEEE Control Systems Letters*, 2019, **4**(2): 414–419.

[28] Zeng X L, Liang S, Hong Y G, et al., Distributed computation of linear matrix equations: An optimization perspective, *IEEE Transactions on Automatic Control*, 2018, **64**(5): 1858–1873.

[29] Bhaya A and Kaszkurewicz E, Steepest descent with momentum for quadratic functions is a version of the conjugate gradient method, *Neural Networks*, 2004, **17**(1): 65–71.

[30] Muehlebach M and Jordan M, A dynamical systems perspective on Nesterov acceleration, *International Conference on Machine Learning*, PMLR, 2019.

[31] Su W J, Boyd S, and Candès E J, A differential equation for modeling Nesterov's accelerated gradient method: Theory and insights, *Journal of Machine Learning Research*, 2016, **17**: 1–43.

[32] Wei Y H, Gao Q, Cheng S S, et al., Description and analysis of the time domain response of nabla discrete fractional order systems, *Asian Journal of Control*, 2020, **23**(4): 1911–1922.

[33] Wang Y and Liang S, Two-DOF lifted LMI conditions for robust D-stability of polynomial matrix polytopes, *International Journal of Control, Automation and Systems*, **11**(3): 636–642.

[34] Li Y L, Meng X, Zheng B C, et al., Parameter identification of fractional order linear system based on Haar wavelet operational matrix, *ISA Transactions*, 2015, **59**: 79–84.

[35] Wei Y H and Chen Y Q, Converse Lyapunov theorem for nabla asymptotic stability without conservativeness, *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, DOI: 10.1109/TSMC.2021.3051639, 2021.

[36] Wei Y H, Lyapunov stability theory for nonlinear nabla fractional order systems, *IEEE Transactions on Circuits and Systems II: Express Briefs*, DOI: 10.1109/TCSII.2021.3063914, 2021.

[37] Lu J G and Chen G R, Robust stability and stabilization of fractional order interval systems: An LMI approach, *IEEE Transactions on Automatic Control*, 2009, **54**(6): 1294–1299.

[38] Chen L P, He Y G, Chai Y, et al., New results on stability and stabilization of a class of nonlinear fractional order systems, *Nonlinear Dynamics*, 2014, **75**(4): 633–641.

[39] Wei Y Q, Liu D Y, Boutat D, et al., Modulating functions based model-free fractional order differentiators using a sliding integration window, *Automatica*, 2021, **130**: 109679.

[40] Sheng H, Chen Y Q, and Qiu T S, *Fractional Processes and Fractional-order Signal Processing: Techniques and Applications*, Springer Science & Business Media, Berlin, 2011.

[41] Alieva T and Bastiaans M J, On fractional Fourier transform moments, *IEEE Signal Processing Letters*, 2000, **7**(11): 320–323.

[42] Li H S, Luo Y, and Chen Y Q, A fractional order proportional and derivative (FOPD) motion controller: Tuning rule and experiments, *IEEE Transactions on Control Systems Technology*, 2009, **18**(2): 516–520.

[43] Cheng S S, Wei Y H, Chen Y Q, et al., A universal modified LMS algorithm with iteration order hybrid switching, *ISA Transactions*, 2017, **67**: 67–75.

[44] Podlubny I, *Fractional Differential Equations: An Introduction to Fractional Derivatives, Fractional Differential Equations, to Methods of Their Solution and Some of Their Applications*, Elsevier, Academic Press, San Diego, 1998.

[45] Chen Y Q, Vinagre B M, and Podlubny I, Continued fraction expansion approaches to discretizing fractional order derivatives-an expository review, *Nonlinear Dynamics*, 2004, **38**(1): 155–170.