Algorithm 1047: FdeSolver, a Julia Package for Solving Fractional Differential Equations

MOEIN KHALIGHI, GIULIO BENEDETTI, and LEO LAHTI, Department of Computing, Faculty of Technology, University of Turku, Turku, Finland

We introduce FdeSolver, an open-source Julia package designed to solve fractional-order differential equations efficiently. The available solutions are based on product-integration rules, predictor–corrector algorithms, and the Newton-Raphson method. The package covers solutions for one-dimensional equations with orders of positive real numbers. For higher-dimensional systems, it supports orders up to one. Incommensurate derivatives are allowed and defined in the Caputo sense. Here, we summarize the implementation for a representative class of problems and compare it with available alternatives in Julia and MATLAB. Moreover, FdeSolver leverages the power and flexibility of the Julia environment to offer enhanced computational performance, and our development emphasizes adherence to the best practices of open research software. To highlight its practical utility, we demonstrate its capability in simulating microbial community dynamics and modeling the spread of COVID-19. This latter application involves fitting the order of derivatives grounded on real-world epidemiological data. Overall, these results highlight the efficiency, reliability, and practicality of the FdeSolver Julia package.

CCS Concepts: • **Mathematics of computing** → **Mathematical software performance**; **Solvers**;

Additional Key Words and Phrases: Julia package, fractional differential equations, memory effects, numerical algorithms, predictor-corrector method, product-integration, Newton-Raphson method

ACM Reference format:
Moein Khalighi, Giulio Benedetti, and Leo Lahti. 2024. Algorithm 1047: FdeSolver, a Julia Package for Solving Fractional Differential Equations. *ACM Trans. Math. Softw.* 50, 3, Article 22 (October 2024), 23 pages.
https://doi.org/10.1145/3680280

1 Introduction

Fractional calculus is the mathematical study of derivatives and integrals of arbitrary orders, encompassing a broad spectrum that includes integer, non-integer, and complex values. Fractional derivatives have emerged from pure mathematics to be applied in various disciplines and real-world scenarios due to advanced computational approaches and new perspectives [43]. Because the

The research was supported by the Academy of Finland (URL: https://www.aka.fi; decision 330887 to LL, MK), the University of Turku Graduate School Doctoral Programme in Technology (UTUGS/DPT) (URL: https://www.utu.fi/en/research/utugs/dpt; to MK), and the Baltic Science Network Mobility Programme for Research Internships (BARI) (URL: https://www.baltic-science.org; to GB). The authors wish to acknowledge the CSC-IT Center for Science, Finland, for computational resources and high-speed networking.

Authors' Contact Information: Moein Khalighi (corresponding author), Department of Computing, Faculty of Technology, University of Turku, Turku, Finland; e-mail: moein.khalighi@utu.fi; Giulio Benedetti, Department of Computing, Faculty of Technology, University of Turku, Turku, Finland; e-mail: giuliobene2000@gmail.com; Leo Lahti (corresponding author), Department of Computing, Faculty of Technology, University of Turku, Turku, Finland; e-mail: leo.lahti@utu.fi.

ACM 1557-7295/2024/10-ART22
https://doi.org/10.1145/3680280

domain of application of fractional calculus has expanded, but analytical solutions are sometimes difficult or even impossible to find, researchers are studying, developing, and using numerical methods to solve **fractional differential equations (FDEs)**.

One significant application of fractional derivatives is incorporating memory effects into the classical system with integer order [1, 13, 24]. Although memory effects in real phenomena are not unique, their common feature is incorporating an average of the previous values into the current value. This inherent characteristic of memory effects has led to multiple interpretations through various definitions of fractional derivatives [6].

Among these, Caputo has defined one of the most popular differential operators with notable advantages [25, 36], which have wide-reaching implications in applied mathematics and practical applications. These advantages include: (1) the derivative of a constant is zero, (2) it includes convolution integrals with a singular power-law kernel, and (3) initial conditions expressed as integer order derivatives; the first one leads to having an operator for modeling systems with higher orders and nonzero initial values, the second one applies to many real-world phenomena with gradually decaying memory effects [10], and the last property is beneficial as it aligns mathematical models more closely with physical reality [7, 36]. Often, initial conditions, obtained from real-world data or measurements, are more naturally and simply represented as integer-order derivatives, such as initial velocity being the first derivative of position over time. Hence, the last attribute not only strengthens the model's mathematical integrity but also enhances its practical significance, making it more straightforward to corroborate with empirical data.

Therefore, we have developed the FdeSolver package based on well-established mathematical algorithms [8, 9, 14] that were formerly designed and implemented as MATLAB routines [16]. These implementations convert the FDE problem to a Volterra integral equation, discretize it via **product-integration (PI)** rules, numerically solve this by using either the **predictor-corrector (PC)** method or the **Newton–Raphson (NR)** method, and finally, reduce the cost of computations by **fast Fourier transform (FFT)**. Furthermore, many other MATLAB toolboxes exist, offering various methods for specific applications, as detailed in literature reviews [26, 43]. However, because MATLAB is a proprietary programming language, its usability is restricted to the holders of a MATLAB license.

FdeSolver is a pioneering open-source Julia package, offering a free and high-performance alternative to MATLAB routines for solving FDEs with Caputo derivatives, overcoming the licensing barriers and dependency on C, C++, or Fortran libraries. While FractionalDiffEq is another key player in the Julia ecosystem, our numerical experiments reveal that FdeSolver not only meets but often exceeds the performance of existing solutions in both MATLAB and Julia. This superior performance is largely attributable to Julia's just-in-time compiled nature, which translates code into machine language for rapid execution, and its unique multiple dispatch feature, which optimizes code dynamically based on the types of parameters passed [4].

The package benefits further from Julia's built-in package manager and robust ecosystem, facilitating interoperability with other tools [4]. Julia's interactive **read-evaluate-print loop (REPL)** enhances user experience by providing immediate code execution and results visualization. FdeSolver's open-source license invites community contributions, encouraging feature enhancements and optimizations. With its blend of computational efficiency, coding flexibility, user-friendliness, and community-driven development, FdeSolver is a premier tool for researchers dealing with FDEs.

This study is based on FdeSolver's Version 1.0.7, archived in the Collected Algorithms repository (http://calgo.acm.org). The package is publicly accessible via its GitHub repository, with a permanent Zenodo DOI (https://doi.org/10.5281/zenodo.5906103). Further insights into its structure, implementation, and usage can be found in the accompanying user manual. Additionally, a dynamically updated user manual is hosted on the package website (https://juliaturkudatascience.github.io/FdeSolver.jl/stable/examples), ensuring it remains current with ongoing updates and new version releases.

We organized the paper as follows: We start with preliminary mathematics and numerical methods in Section 2. The usage of the package for five classes of problems is demonstrated in Section 3, including benchmarks with alternative solvers. We provide two applications of the package in the context of ecological and epidemiological models in Section 4. We comprehensively discuss results in Section 5 and conclude with Section 6.

2 Preliminaries and Numerical Scheme

This section focuses on some primary definitions in fractional calculus, as presented in [25, 36] and the numerical methods used in the package, which is proposed by Diethelm [8, 9] and implemented in MATLAB by Garrappa [16].

Let us suppose $\mathcal{D}_{t_0}^{\beta}$ as a notation of time-fractional Caputo derivative with order of $\beta \in \mathbb{R}^+$ from initial value t_0. Thus, the fractional derivative of a given differentiable function g at time t in the sense of Caputo is defined as:

$$\mathcal{D}_{t_0}^{\beta} g(t) = I_{t_0}^{m-\beta} g^{(m)}(t) = \frac{1}{\Gamma(m-\beta)} \int_{t_0}^{t} \frac{g^{(m)}(\tau) d\tau}{(t-\tau)^{1+\beta-m}}, \tag{1}$$

in which $g^{(m)}(t)$ denotes the mth order derivative of $g(t)$, and m represents the smallest integer that is greater than or equal to the derivative order β. The term $I_{t_0}^{m-\beta}$ is Riemann-Liouville fractional integral of order $m-\beta$ that is defined by

$$I_{t_0}^{\beta} g(t) = \frac{1}{\Gamma(\beta)} \int_{t_0}^{t} \frac{g(\tau) d\tau}{(t-\tau)^{1-\beta}}, \tag{2}$$

where Γ denotes the gamma function.

FdeSolver can solve the following types of equations and systems:

$$\mathcal{D}_{t_0}^{\beta_i} X_i = f_i(t, X), \quad i = 1, \dots, M, \tag{3}$$

with the initial conditions $X_i(t_0) = X_{i,0}$ when $0 < \beta_i \leq 1$ for a system of equations, and $X_1^{(j)}(t_0) = X_{1,0}^{(j)}$ $(j = 0, \dots, m-1)$ for one-dimensional equations when $\beta_1 > 1$. We can rewrite the fractional order system (3) in the following vector form:

$$\mathcal{D}_{t_0}^{\boldsymbol{\beta}} \boldsymbol{X} = \boldsymbol{F}(t, \boldsymbol{X}), \tag{4}$$

where $\boldsymbol{X} = [X_1, \dots, X_M]$, $\boldsymbol{\beta} = [\beta_1, \dots, \beta_M]$, and $\boldsymbol{F} = [f_1, \dots, f_M]$.

The initial value problem (4) is equivalent to the Volterra integral equation [8, 25]

$$\boldsymbol{X}(t) = T_{m-1}[\boldsymbol{X}; t_0] + \frac{1}{\Gamma(\boldsymbol{\beta})} \int_{t_0}^{t} (t-\tau)^{\boldsymbol{\beta}-1} \boldsymbol{F}(\tau, \boldsymbol{X}(\tau)) d\tau, \tag{5}$$

in which $T_{m-1}[X_i; t_0]$ is Taylor polynomial of degree $m-1$ for the function $X_i(t)$ centered at t_0 defined as

$$T_{m-1}[X_i; t_0](t) = \sum_{k=0}^{m-1} \frac{(t-t_0)^k}{k!} X_i^{(k)}(t_0).$$

The integrals in Equation (5) give us history-dependent dynamics, and the presence of power in the kernels provides a nonlocal feature of our fractional order modelings. These, however, cause a non-smooth behavior at the initial time; thus, straightforward numerical methods, such as polynomial approximations, cannot achieve the solution, and only those are acceptable that each step of computation involves the whole history of the solution.

Therefore, based on PI rules [8, 14], we discretise the integral term of Equation (5) in n points and the step size $h > 0$; $t_r = t_0 + rh(r = 0, \dots, n-1)$, so that the Equation (5) turns to

$$\boldsymbol{X}(t_n) = T_{m-1}[\boldsymbol{X}; t_0](t_n) + \frac{1}{\Gamma(\boldsymbol{\beta})} \sum_{r=0}^{n-1} \int_{t_r}^{t_{r+1}} (t_n - \tau)^{\boldsymbol{\beta}-1} \boldsymbol{F}(\tau, \boldsymbol{X}(\tau)) d\tau. \tag{6}$$

Then, we use piece-wise interpolating polynomials for the approximation of integral terms.

2.1 PC Method

The PC approach has been proposed using a generalization of Adams multi-step methods to solve fractional ordinary differential equations [8, 14]. To use this technique for the solution of Equation (6), we start with a preliminary explicit estimation as a *predictor* and then improve it by the iteration of implicit approximations as a *corrector*. Hence, the PI rectangular rule [16] gives the matrix form of the predictor of the solution of Equation (6) for the nth step:

$$\boldsymbol{X}^{[0]}(t_n) = \boldsymbol{T}_0 + \boldsymbol{hbF}_0^{n-1}, \tag{7}$$

and the PI trapezoidal rule [16] provides the framework for the corrector:

$$\boldsymbol{X}^{[\mu]}(t_n) = \boldsymbol{T}_0 + \boldsymbol{hcF}_0 + \boldsymbol{hdF}_1^{n}. \tag{8}$$

To elaborate, Equation (7) is

$$\begin{bmatrix} X_1^{[0]}(t_n) \\ X_2^{[0]}(t_n) \\ \vdots \\ X_M^{[0]}(t_n) \end{bmatrix}^\top = \begin{bmatrix} T(X_1) \\ T(X_2) \\ \vdots \\ T(X_M) \end{bmatrix}^\top$$
$$+ \begin{bmatrix} h^{\beta_1} \\ h^{\beta_2} \\ \vdots \\ h^{\beta_M} \end{bmatrix}^\top \begin{bmatrix} b_{1,n-1} & b_{1,n-2} & \dots & b_{1,0} \\ b_{2,n-1} & b_{2,n-2} & \dots & b_{2,0} \\ \vdots & \vdots & \ddots & \vdots \\ b_{M,n-1} & b_{M,n-2} & \dots & b_{M,0} \end{bmatrix} \begin{bmatrix} f_1(\boldsymbol{X}(t_0)) & \dots & f_M(\boldsymbol{X}(t_0)) \\ f_1(\boldsymbol{X}(t_1)) & \dots & f_M(\boldsymbol{X}(t_1)) \\ \vdots & \ddots & \vdots \\ f_1(\boldsymbol{X}(t_{n-1})) & \dots & f_M(\boldsymbol{X}(t_{n-1})) \end{bmatrix},$$

where $T(X_i) = T_{m-1}[X_i; t_0](t_n)$ and $b_{i,r} = \frac{(r+1)^{\beta_i} - (r)^{\beta_i}}{\Gamma(\beta_i + 1)}$, $i = 1, \dots, M$, $r = 0, \dots, n-1$, and Equation (8) is

$$\begin{bmatrix} X_1^{[\mu]}(t_n) \\ X_2^{[\mu]}(t_n) \\ \vdots \\ X_M^{[\mu]}(t_n) \end{bmatrix}^\top = \begin{bmatrix} T(X_1) \\ T(X_2) \\ \vdots \\ T(X_M) \end{bmatrix}^\top + \begin{bmatrix} h^{\beta_1} \times c_{1,n} \times f_1(\boldsymbol{X}(t_0)) \\ h^{\beta_2} \times c_{2,n} \times f_2(\boldsymbol{X}(t_0)) \\ \vdots \\ h^{\beta_M} \times c_{M,n} \times f_M(\boldsymbol{X}(t_0)) \end{bmatrix}^\top$$
$$+ \begin{bmatrix} h^{\beta_1} \\ h^{\beta_2} \\ \vdots \\ h^{\beta_M} \end{bmatrix}^\top \begin{bmatrix} d_{1,n-1} & d_{1,n-2} & \dots & d_{1,0} \\ d_{2,n-1} & d_{2,n-2} & \dots & d_{2,0} \\ \vdots & \vdots & \ddots & \vdots \\ d_{M,n-1} & d_{M,n-2} & \dots & d_{M,0} \end{bmatrix}$$

$$\times \begin{bmatrix} f_1(\boldsymbol{X}(t_1)) & \dots & f_M(\boldsymbol{X}(t_1)) \\ \vdots & \ddots & \vdots \\ f_1(\boldsymbol{X}(t_{n-1})) & \dots & f_M(\boldsymbol{X}(t_{n-1})) \\ f_1(\boldsymbol{X}^{[\mu-1]}(t_n)) & \dots & f_M(\boldsymbol{X}^{[\mu-1]}(t_n)) \end{bmatrix}, \ \mu = 1, 2, ...,$$

in which

$$c_{i,n} = \frac{(n-1)^{\beta_i+1} - n^{\beta_i}(n - \beta_i - 1)}{\Gamma(\beta_i + 2)},$$

$$d_{i,r} = \begin{cases} \frac{1}{\Gamma(\beta_i+2)}, & \text{if } r = 0, \\ \frac{(r-1)^{\beta_i+1} - 2r^{\beta_i+1} + (r+1)^{\beta_i+1}}{\Gamma(\beta_i+2)}, & \text{if } r = 1, 2, ..., n-1, \end{cases}$$

and, μ is the number of corrections. This is the main algorithm of our package with the convergence rate of $O(h)$ for the predictor and $O(h^{min\{1+\beta,2\}})$ for the corrector [16].

2.2 NR Method

The PC method may not be sufficient for stiff problems, which need too small step size h, leading to high computation costs for accuracy. Hence, we use the more efficient approximation with better stability properties, the implicit method Equation (8). However, this method needs a prior approximation of the current step besides the computed values of the previous steps. So, we rewrite Equation (8) as:

$$\boldsymbol{X}(t_n) = \boldsymbol{\Psi}_{n-1} + \boldsymbol{a}_0 \boldsymbol{F}_n, \tag{9}$$

where $\boldsymbol{\Psi}_{n-1}$ denotes the term including all the explicitly known information of the previous steps

$$\boldsymbol{\Psi}_{n-1} = \begin{bmatrix} T(X_1) \\ T(X_2) \\ \vdots \\ T(X_M) \end{bmatrix}^\top + \begin{bmatrix} h^{\beta_1} \times c_{1,n} \times f_1(\boldsymbol{X}(t_0)) \\ h^{\beta_2} \times c_{2,n} \times f_2(\boldsymbol{X}(t_0)) \\ \vdots \\ h^{\beta_M} \times c_{M,n} \times f_M(\boldsymbol{X}(t_0)) \end{bmatrix}^\top$$
$$+ \begin{bmatrix} h^{\beta_1} \\ h^{\beta_2} \\ \vdots \\ h^{\beta_M} \end{bmatrix}^\top \begin{bmatrix} d_{1,n-1} & d_{1,n-2} & \dots & d_{1,1} \\ d_{2,n-1} & d_{2,n-2} & \dots & d_{2,1} \\ \vdots & \vdots & \ddots & \vdots \\ d_{M,n-1} & d_{M,n-2} & \dots & d_{M,1} \end{bmatrix} \begin{bmatrix} f_1(\boldsymbol{X}(t_1)) & \dots & f_M(\boldsymbol{X}(t_1)) \\ f_1(\boldsymbol{X}(t_2)) & \dots & f_M(\boldsymbol{X}(t_2)) \\ \vdots & \ddots & \vdots \\ f_1(\boldsymbol{X}(t_{n-1})) & \dots & f_M(\boldsymbol{X}(t_{n-1})) \end{bmatrix},$$

and the second term $\boldsymbol{a}_0 \boldsymbol{F}_n$ is related to the initial approximation of the current step

$$\boldsymbol{a}_0 \boldsymbol{F}_n = \begin{bmatrix} d_{1,0} \times f_1(\boldsymbol{X}^{[\mu-1]}(t_n)) \\ d_{2,0} \times f_2(\boldsymbol{X}^{[\mu-1]}(t_n)) \\ \vdots \\ d_{M,0} \times f_M(\boldsymbol{X}^{[\mu-1]}(t_n)) \end{bmatrix}.$$

Garrappa [16] suggested the iterative modified Newton-Raphson method for the solution of Equation (9) with the convergence order $O(h^2)$. With having an initial approximation $\boldsymbol{X}^{[0]}(t_n)$ for $\boldsymbol{X}(t_n)$, we can calculate new improved solutions $\boldsymbol{X}^{[\mu]}(t_n)$ by the following iterative formula

$$\boldsymbol{X}^{[\mu+1]}(t_n) = \boldsymbol{X}^{[\mu]}(t_n) - \left[\boldsymbol{I} - \boldsymbol{a}_0 \boldsymbol{J}_F(\boldsymbol{X}^{[0]}(t_n))\right]^{-1} \left(\boldsymbol{X}^{[\mu]}(t_n) - \boldsymbol{\Psi}_{n-1} - \boldsymbol{a}_0 \boldsymbol{F}_n\right), \tag{10}$$

where I is the identity matrix and $J_F(X(t_n))$ is the Jacobian matrix of F with respect to the variables X_i at t_n (or a derivative for one-dimensional problems) defined as

$$J_F(X(t_n)) = \begin{bmatrix} \frac{\partial f_1}{\partial X_1} & \frac{\partial f_1}{\partial X_2} \cdots & \frac{\partial f_1}{\partial X_M} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_M}{\partial X_1} & \frac{\partial f_M}{\partial X_2} \cdots & \frac{\partial f_M}{\partial X_M} \end{bmatrix} (t_n).$$

Then, we only need an initial approximation for each step that we achieve from the last evaluated approximation $X^{[0]}(t_{r+1}) = \mathbf{X}(t_r)$, $(r = 0, \ldots, n-1)$. It might seem this assumption leads to the method's inefficiency. Still, the high-order convergences of the two blended algorithms and a sufficient number of iterations guarantee efficiency unless the variables X or derivative functions F alter very rapidly (see Example 3.2.2) and the step size is not small enough.

2.3 FFT

In both methods, there are convolution sums in the second term of Equation (7), the third term of Equation (8), and the second term of Equation (9). Thus, the whole evaluation of the solution requires n number of operations for grids $1, \ldots, n$, which proportionally costs $O(n^2)$. Hence, the direct calculation of the matrix is not reasonable when the number of grid points n is sufficiently large. However, using FFT, as elaborately explained in [11], can substantially reduce the computational cost proportional to $O(n(log_2 n)^2)$.

Here, we briefly describe exploiting the FFT algorithm in the PC and NR methods. Suppose $Y_n = \sum_{j=1}^{n} d_{n-j} f_i(X(t_j))$ is one of the last sums of our matrix products. The idea is to split each sum into two halves of the computation interval. The Fourier Transform converts the convectional coefficients of one part into frequencies, multiplies once, and finally, converts back the sum evaluation. This process requires only $O(2nlog_2 2n)$ proportional operations and it can be reduced if we recursively repeat by splitting the interval. However, we need to start the evaluation from the initial time to a smaller length of the interval $r < n$

$$Y_q = \sum_{j=1}^{q} d_{q-j} f_i(X(t_j)), \quad q = 1, 2, \ldots, r.$$

This is the initial vector of values that can be directly evaluated, and the length r is fixed, which could be any small integer number power of two for convenience (where we consider $r = 2^4$). Then we calculate the values of the following r points of the interval:

$$Y_q = Y_1^r + Y_r^q = \sum_{j=1}^{r} d_{q-j} f_i(X(t_j)) + \sum_{j=r}^{q} d_{q-j} f_i(X(t_j)), \quad q = r+1, r+2, \ldots, 2r,$$

where FFT algorithm can evaluate the partial sum Y_1^r, with a computational cost proportional to $O(2r \log_2 2r)$ instead of $O(r^2)$. For the computation of the next $2r$ values, we have two splits from $2r$ to $3r$ and from $3r$ to $4r$:

$$Y_q = Y_1^{2r} + Y_{2r}^q = \sum_{j=1}^{2r} d_{q-j} f_i(X(t_j)) + \sum_{j=2r}^{q} d_{q-j} f_i(X(t_j)), \quad q = 2r+1, 2r+2, \ldots, 3r,$$

$$\begin{aligned} Y_q = Y_1^{2r} + Y_{2r}^{3r} + Y_{3r}^q &= \sum_{j=1}^{2r} d_{q-j} f_i(X(t_j)) \\ &+ \sum_{j=2r}^{3r} d_{q-j} f_i(X(t_j)) + \sum_{j=3r}^{q} d_{q-j} f_i(X(t_j)), \quad q = 3r+1, 3r+2, \ldots, 4r, \end{aligned}$$

where FFT algorithm can evaluate the partial sums Y_1^{2r} and Y_{2r}^{3r}, with computational costs proportional to $O(4r \log_2 4r)$ and $O(2r \log_2 2r)$, respectively. Similarly, this process can be repeated until it reaches the nth point.

3 Usage and Benchmarks

In the following, we present examples using FdeSolver to solve some model classes. Certain examples necessitate specific packages and corresponding implementations, as detailed in the accompanying user manual. It is worth noting that we considered the initial conditions of all examples equal to zero, while they could be nonzero as well.

We compared the efficiency, in terms of speed and accuracy, of the PC and NR methods within the FdeSolver. This is contrasted with four solvers from MATLAB counterpart [16] and six methods provided in another Julia package, FractionalDiffEq.jl v0.3.1 [38]. While FractionalDiffEq is comprehensive in offering numerical methods for FDEs, our empirical findings indicated certain limitations in its current version (v0.3.1), particularly for one-dimensional FDEs. These limitations included challenges in parameter flexibility and handling integer order equations, as well as issues with accuracy and convergence in many examples. Consequently, while FractionalDiffEq theoretically supports FDEs, its practical application for one-dimensional equations was found to be constrained, as evidenced by our testing. Moreover, the results suggest that our solver provides a performance that is similar to or better than that of the MATLAB counterparts (Figures 1, 2, and A2), and significantly greater accuracy than any available Julia alternative (Figures 1 and 2(a)).

Let us define some notations for the methods used for solving the examples. As shown in Table 1, Julia solvers employ notations beginning with "J" while MATLAB solvers use the "M" prefix. For instance, J1 indicates the PC method in FdeSlover, corresponding to M1 used in the MATLAB code [16]. Similarly, J2 denotes the NR method in FdeSolver, analogous to M2 in MATLAB.

We benchmarked these methods across five examples. In addition, we considered alternative algorithms: M3, which employs the NR method with PI rectangular rule, and M4, which uses the explicit PI rectangular rule without PC [16].

For one-dimensional examples and Example 3.2.1, we compared these methods against eight additional Julia-developed methods [38]. They include J3, which mirrors the methods of J1 and M1; J4, which aligns with M4; J5, a modified Grünwald–Letnikov approach; J6, which is a one-step Adams-Bashforth method [9]; J7, which is derived from the **fractional-order transfer functions (FOTF)** Toolbox [45]; and J8 to J10, which are based on fractional linear multistep methods [15].

In our study, all execution times were provided in seconds. To accurately assess the performance of various methods, we compared their numerical solutions against known analytical solutions whenever available. In cases where analytical solutions are not accessible, we contrasted the results with those obtained using a more finely resolved grid and optimized configurations. For both MATLAB and Julia implementations, we employed a step size 2^{-10} and specified a relative tolerance 10^{-12} for the solver. This is our benchmark for accuracy, as exact solutions for multi-dimensional models are often difficult or impossible to determine. The deviation between the approximations and exact solutions, derived using solvers with fine step sizes, could either confirm or refute the convergence of these approximations to exact solutions.

All the experiments were carried out in Julia Version 1.7.3 and MATLAB Version 9.12.0.1975300 (R2022a) Update 3 on a computer equipped with a CPU Intel i7-9750H at 2.60 GHz running under the OS Ubuntu 22.04.1 LTS. Results may vary when using different versions than those mentioned above.

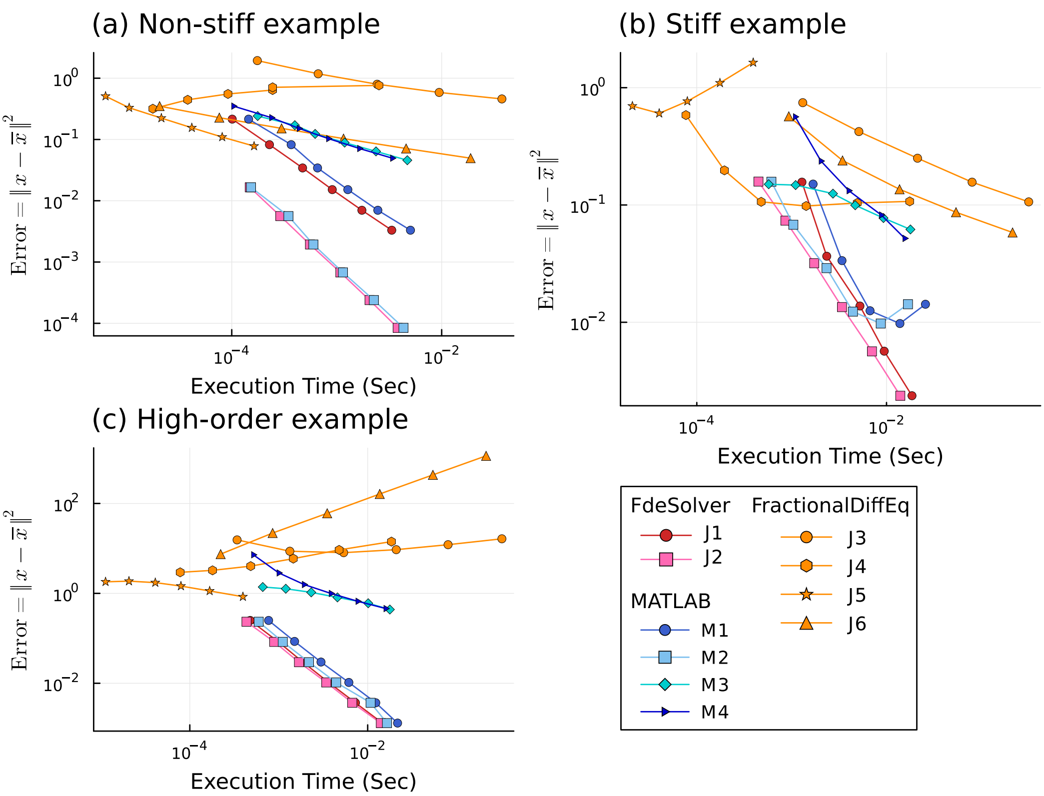

Fig. 1. Error versus execution time of FdeSolver, MATLAB codes, and alternative Julia solvers for solving the one-dimensional differential equation of (a) Example 3.1.1, (b) Example 3.1.2, and (c) Example 3.1.3. The axes are visualized on a logarithmic scale. The unit of the X-axes is second (Sec), and the Y-axes show the Euclidean norm of the difference between the approximations and exact values. For FdeSolver methods, the notations are J1 for the PC method and J2 for the NR method. For MATLAB routines, we use M1 for the PC method, M2 for the NR method, M3 for the NR method with the PI rectangular rule, and M4 for the explicit PI rectangular rule without PC. Within the Julia FractionalDiffEq package, solvers J3 to J6 are used: J3 parallels methods J1 and M1; J4 corresponds to M4; J5 is a modified Grünwald–Letnikov method; and J6 employs a one-step Adams-Bashforth approach.

3.1 One-Dimensional Models

We start solving three model examples described by one differential equation. The first two examples have $\beta \leq 1$ and the third one has $\beta > 1$. For these examples, we examined ten solvers: six developed in Julia, denoted as J1 through J6, and four implemented in MATLAB, labeled M1 through M4. For a detailed list of these methods, please refer to Table 1.

3.1.1 Non-Stiff Example. Consider the following nonlinear FDE [8]:

$$\mathcal{D}_{t_0}^{\beta} X(t) = \frac{40320}{\Gamma(9-\beta)} t^{8-\beta} - 3\frac{\Gamma(5+\beta/2)}{\Gamma(5-\beta/2)} t^{4-\beta/2} + \frac{9}{4}\Gamma(\beta+1) + \left(\frac{3}{2}t^{\beta/2} - t^4\right)^3 - (X(t))^{3/2}, \quad 0 < \beta \leq 1.$$

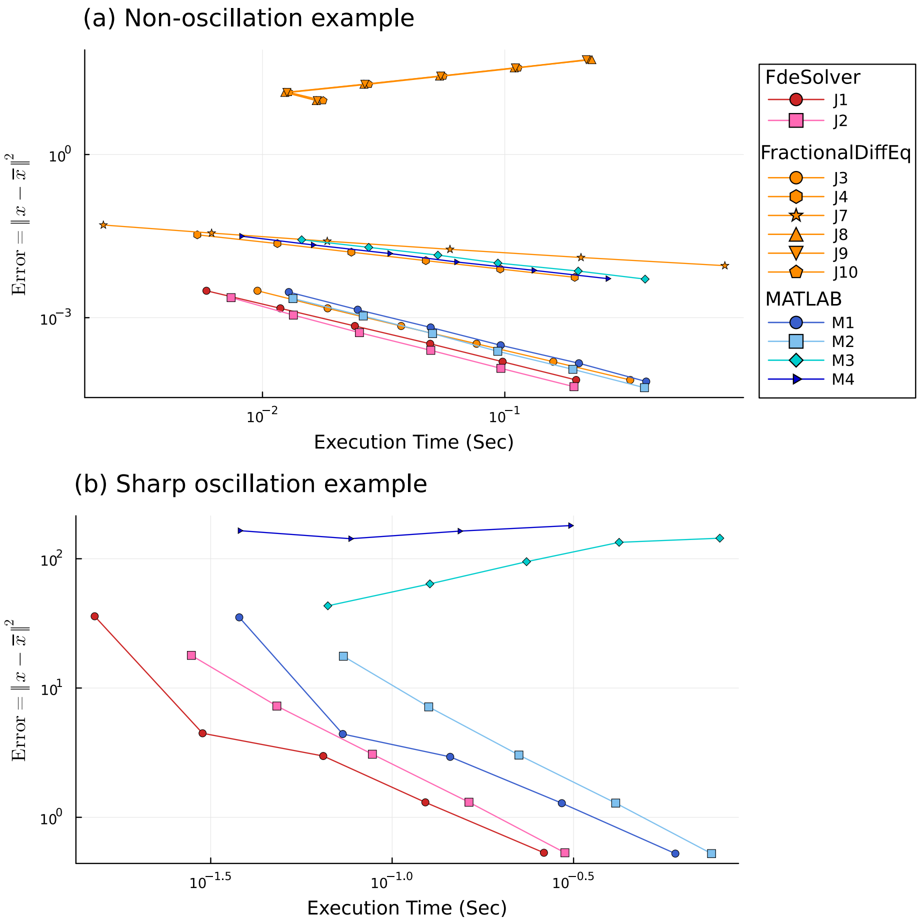

Fig. 2. Error versus execution time of Julia and MATLAB codes for solving for multi-dimensional systems of (a) Example 3.2.1 and (b) Example 3.2.2. The axes are visualized on a logarithmic scale. The unit of the X-axes is second (Sec), and the Y-axes show the accuracy by measuring the Euclidean norm of the difference between the approximations and the results with fine step size 2^{-10}. Methods in FdeSolver are marked as J1 (PC method) and J2 (NR method). For FractionalDiffEq, we denote methods as J3 (PC method), J4 (explicit PI rectangular rule without PC), J7 (based on FOTF Toolbox), and J8-J10 (fractional linear multistep methods). In MATLAB routines, we have M1 (PC method), M2 (NR method), M3 (NR method with PI rectangular rule), and M4 (explicit PI rectangular rule without PC).

With initial value $X(0) = 0$, the exact solution is $X(t) = t^8 - 3t^{4+\beta/2} + \frac{9}{4}t^{\beta}$. Suppose the final time equals 1 and the order derivative $\beta = 0.5$.

This example has a smooth solution despite its derivative function displaying a nonlinear and nonsmooth equation. We run the solvers for this range of step size of computations: $h = 2^{-n},\ n = 3, \ldots, 8$. Figure 1(a) shows the performance of MATLAB and our Julia equivalence codes, J1 versus M1 and J2 versus M2, along with an evaluation of alternative Julia solvers, J3 to J6. The results of this example indicate that the NR method is more efficient than the PC method. Additionally, our solvers perform slightly faster than MATLAB. Notably, these four solvers—J1, J2, M1, and M2—exhibit greater accuracy than their counterparts, the MATLAB solvers M3 and M4, and the alternative Julia solvers J3 to J6.

Table 1. A List of Solvers Used in This Study for Solving the FDEs

Notation	FdeSolver	Notation	MATLAB [16]	Notation	FractionalDiffEq [38]
J1	PC	M1	FDE_PI12_PC.m	J3	PIPECE
J2	NR	M2	FDE_PI2_Im.m	-	-
-	-	M3	FDE_PI1_Im.m	-	-
-	-	M4	FDE_PI1_Ex.m	J4	PIEX
-	-	-	-	J5	GL
-	-	-	-	J6	Euler
-	-	-	-	J7	NonLinearAlg
-	-	-	-	J8	FLMMBDF
-	-	-	-	J9	FLMMNewtonGregory
-	-	-	-	J10	FLMMTrap

The solvers on the same rows are built on the same algorithms.

A detailed step-by-step Julia code for this example, including the equation, its Jacobian, and how to use the FdeSolver package, can be found in the PDF user manual (Section 3.1.1).

3.1.2 Stiff Example. It is recommended to consider the Jacobian function and use the NR method for stiff problems [16] such as:

$$\mathcal{D}_{t_0}^{\beta} X(t) = \lambda X(t), \quad X(t_0) = X_0,$$

where the exact solution is $X(t) = E_\beta\left((t - t_0)^\beta \lambda\right)$ in which the one parameter Mittag-Leffler function is defined as:

$$E_\beta(t) = \sum_{k=0}^{\infty} \frac{t^k}{\Gamma(\beta k + 1)}.$$

The implementation of this function and other settings are explained in the PDF user manual (Section 3.2.1).

We solved this problem with initial condition $X(0) = 1$, order $\beta = 0.8$, and $\lambda = -10$, to $t = 5$. We also used 4 corrections of the PC method to improve the accuracy of the results.

This example illustrates the superiority of the NR method. We ran the solvers for this range of step size of computations: $h = 2^{-n}$, $n = 3, \ldots, 8$. The PC method diverges for $h > 2^{-4}$; however, the 2-norm error of the NR method for $h = 2^{-3}$ is about 0.16. Our FdeSolver outperforms both MATLAB and Julia alternatives in terms of efficiency and reliability. This is further evidenced in Figure 1(b), which highlights the limited accuracy of Julia solvers J3 to J6. Moreover, it reveals that for smaller step sizes $h < 2^{-7}$, the error margins for MATLAB solvers M1 and M2 notably increase.

3.1.3 High-Order Example. We can solve Example 3.1.1 for an order greater than 1. But, let us consider another popular example, fractional Harmonic motion [47], described as:

$$\mathcal{D}_0^{\beta} X(t) = -\frac{k}{m} X(t), \quad k > 0, \quad m > 0, \quad 1 < \beta \leq 2, \tag{11}$$

where X is the displacement from the equilibrium point, k the spring constant, and m is the inertial mass of the oscillating body. The exact solution for $\beta = 2$ case is given by

$$X(t) = X(0) cos(\omega t) + \frac{X'(0)}{\omega} sin(\omega t), \quad \omega = \sqrt{\frac{k}{m}}. \tag{12}$$

The general analytical solution for other values of β can be derived using the Laplace transform method. This technique involves transforming a function $X(t)$ into a function $Y(s)$ of complex variable s using the equation [27]

$$Y(s) = \mathcal{L}\{X(t)\} = \int_0^\infty e^{-st}X(t)dt.$$

Applying the Laplace transform to both sides of (11) yields [27]

$$\mathcal{L}\{\mathcal{D}_0^\beta X(t)\} = s^\beta Y(s) - s^{\beta-1}X(0) - s^{\beta-2}X'(0),$$

resulting in

$$Y(s) = \frac{X(0)s^{\beta-1}}{\left(s^\beta + \frac{k}{m}\right)} + \frac{X'(0)s^{\beta-2}}{\left(s^\beta + \frac{k}{m}\right)}.$$

The inverse Laplace transform then provides the analytical expression:

$$X(t) = X(0)E_{\beta,1}\left(-\frac{k}{m}t^\beta\right) + X'(0)tE_{\beta,2}\left(-\frac{k}{m}t^\beta\right), \tag{13}$$

where $E_{\beta,1}$ and $E_{\beta,2}$ are two-parameter Mittag-Leffler functions, generally defined as

$$E_{\beta,\alpha}(t) = \sum_{n=0}^{\infty} \frac{t^n}{\Gamma(n\beta+\alpha)}, \quad \beta, \alpha > 0.$$

Specifically, for $\beta = 2$ we have

$$E_{2,1}(-t^2) = \sum_{n=0}^{\infty} \frac{(-t^2)^n}{\Gamma(2n+1)} = \sum_{n=0}^{\infty} \frac{(-1)^n t^{2n}}{(2n)!} = \cos(t),$$

and

$$E_{2,2}(-z^2) = \sum_{n=0}^{\infty} \frac{(-t^2)^n}{\Gamma(2n+2)} = \frac{1}{t}\sum_{n=0}^{\infty} \frac{(-1)^n t^{2n+1}}{(2n+1)!} = \frac{\sin(t)}{t},$$

Thus, it is evident that the solution in Equation (13) aligns with the solution in Equation (12) for the case of $\beta = 2$. Notice that for non-integer orders of β, the Mittag-Leffler functions must be evaluated numerically.

Let us consider $X(0) = 1$ and $X'(0) = 1$ when the parameters are $k = 16$ and $m = 4$, and solve the equation for $\beta = 1.90$ to $t = 10$. The related code for functions and conditions is represented in the user manual (Section 3.1.3).

Figure 1(c) shows the similar performance of FdeSolver and its MATLAB counterparts, while both are better than M4 and M3, and Julia solvers J3 to J6. We run the solvers for this range of step size of computations: $h = 2^{-n},\ n = 2, \ldots, 7$.

3.2 Multi-Dimensional Systems

FdeSolver can solve high dimensional systems of incommensurate FDEs, which means the order derivatives could be unequal. Here, we present two three-dimensional examples of FDE systems without and with oscillation dynamics to evaluate the performance of the PC and NR methods alongside alternative solvers. For the non-oscillation dynamics example, we analyzed with eleven solvers: seven from Julia, identified as J1-4 and J7-10, alongside four MATLAB solvers, M1 to M4. In the sharp oscillation example, we used six solvers: two from our FdeSolver package, J1 and J2, and the same four MATLAB solvers, M1 to M4. Details about these methods can be found in Table 1.

3.2.1 Non-Oscillation Example. The **susceptible-infected-recovered (SIR)** model is the most popular mathematical model for simulating the transmission dynamics of infectious diseases in a population that Kermack and McKendrick have introduced [22]. Recently, its extensions with fractional orders have been commonly used in different contexts. This motivates us to present it here as an example and its generalization as an application in Section 4.2. Suppose the following fractional SIR model [39] for susceptible (S), infected (I), and recovered (R) individuals with the corresponding order derivatives α_S, α_I, and α_R:

$$\begin{aligned}
\mathcal{D}_{t_0}^{\alpha_S} S(t) &= -\beta I(t)S(t),\\
\mathcal{D}_{t_0}^{\alpha_I} I(t) &= \beta I(t)S(t) - \gamma I(t),\\
\mathcal{D}_{t_0}^{\alpha_R} R(t) &= \gamma I(t),
\end{aligned}$$

where β is the infectious rate and γ the recovery rate. We solved this system until $t = 100$ for initial conditions $S(0) = 1 - I(0)$, $I(0) = 0.1$, and $R(0) = 0$, orders $\alpha_S = 0.9, \alpha_I = 0.6$, and $\alpha_R = 0.7$, and parameters $\beta = 0.4$ and $\gamma = 0.04$. Notice that we considered $tol = 10^{-8}$ for the NR method to have a distinguishable result in benchmarking. The user manual (Section 3.2.1) provides the related code.

We added six alternative methods from FractionalDiffEq to benchmark this example. The methods J8, J9, and J10, from the alternative Julia package, fail, the methods J4 and J7 have a similar performance with M3 and M4, and the method J3 is similar to M1, which is a more reliable solver within the FractionalDiffEq (Figure 2(a)).

Approximations taken from M2 and J2 with fine step size $h = 2^{-10}$ and $tol = 10^{-12}$ were the candidates for the exact solutions for measuring the errors. The square norm of the difference of these solutions was about 2.68×10^{-12}. We ran the solvers for $h = 2^{-n},\ n = 2, \ldots, 7$ and Figure 2(a) shows FdeSolver superiority.

3.2.2 Sharp Oscillation Example. The dynamics of the previous example are smooth, and here we challenge the solver by considering a three-species Lotka-Volterra model [23] with a sharp oscillation behavior (Figure A1(b)). This can be defined as:

$$\begin{aligned}
\mathcal{D}_{t_0}^{\beta_1} X_1(t) &= X_1(t)\left(a_1 - a_2 X_1(t) - X_2(t) - X_3(t)\right),\\
\mathcal{D}_{t_0}^{\beta_2} X_2(t) &= X_2(t)\left(1 - a_3 + a_4 X_1(t)\right),\\
\mathcal{D}_{t_0}^{\beta_3} X_3(t) &= X_3(t)\left(1 - a_5 + a_6 X_1(t) + a_7 X_2(t)\right),
\end{aligned}$$

where $0 < \beta_i \leq 1\ (i = 1, 2, 3)$, and all coefficients $a_i > 0\ (i = 1, \ldots, 7)$, and initial values are positive. We solved the system until $t = 60$ for initial conditions $X_1(0) = X_2(0) = X_3(0) = 1$, with parameters $a_1 = a_2 = a_3 = a_5 = a_6 = a_7 = 3$ and $a_4 = 5$, and orders $\beta_1 = 1,\ \beta_2 = 0.9$ and $\beta_3 = 0.7$.

We conducted a comparative analysis between our Julia solvers (J1 and J2) and the MATLAB codes (M1-4). This is a challenging example for solvers such that all methods of FractionalDiffEq fail to solve it even for the fine step sizes. Moreover, the PC method is more efficient than the NR method for the solution of this example due to fast oscillation dynamics at the initial times, as mentioned in Section 2.2. Hence, we set 4 corrections for the PC method and $tol = 10^{-8}$ for both methods for better accuracy and comparison.

Approximations taken from M2 and J2 with fine step size $h = 2^{-10}$ and $tol = 10^{-12}$ were the candidates for the exact solutions for measuring the errors. The square norm of the difference of these solutions was about 1.62×10^{-10}. We considered $h = 2^{-n},\ n = 4, \ldots, 8$, and Figure 2(b) illustrates that M4 and M3 are not reliable since they tend to diverge as the step size decreases. Furthermore, the PC method has a better performance than the NR method in both Julia and MATLAB due to the sharp oscillations of the dynamics (see 2.2 and Figure A1).

3.2.3 Random Values. To assess the generality of the results, we considered random values for the parameters and conditions of the studied examples. Thus, we ran the solvers for the methods J1, J2, M1, M2, M3, and M4 (see Table 1), to solve Example 3.1.1 40 times with random values including $0 < \beta \leq 2$ and 30 times each Example of 3.1.2–3.2.1. We excluded Example 3.2.2 due to the difficulty of controlling conditions for making a solvable FDE.

Figure A2 shows the distribution of errors and execution times of the solutions of the problems with random conditions, illustrating the similar or improved performance of the FdeSolver Julia package compared to the MATLAB alternatives. Codes related to the randomization step are available in the user manual (Section 3.2.3).

4 Applications

Let us next present two examples of applications of fractional calculus in community dynamics and epidemiology, where fractional derivatives have been used to describe memory effects or the influence of past events on population dynamics [24, 39].

4.1 Simulation of Microbial Community Dynamics

In microbial ecology, long-term memory has been observed in the context of antibiotic-tolerant persister cells [41], which results from phenotypic heterogeneity [19, 37] that has a power-law scale feature. In our recent study, we numerically examined the effect of memory in ecosystem dynamics of interacting communities [24], using some of the methods converging in the FdeSolver package. We investigated extensions to interaction models, including the Generalized Lotka-Volterra model with fractional orders described as:

$$\mathcal{D}_{t_0}^{\beta_i} = X_i \left(b_i f_i(\{X_k\}) - k_i X_i \right), \quad f_i(\{X_k\}) = \prod_{\substack{k=1 \\ k \neq i}}^{N} \frac{K_{ik}^n}{K_{ik}^n + X_k^n}, \quad i = 1, \ldots, N, \tag{14}$$

where N is the number of species, X_i species abundances, b_i growth rates, k_i death rates, and f_i inhibition functions such that K_{ij} and n denote interaction constants and Hill coefficients [20]. If we determine $1 - \beta_i$ as the memory index of the dynamics of species abundance i, then a decrease in the order of the corresponding derivative leads to an increase in its memory.

The FdeSolver package thus allows us to replicate the simulation results that we recently reported [24] to demonstrate and assess system dynamics under pulse and periodic perturbations in the presence of ecological memory. We have included the codes in Section 4.1 of the user manual.

Resistance and resilience are two key factors in determining the stability of an ecological system. Resistance describes a system's ability to maintain its existing state when subjected to external disturbances, while resilience refers to its capability to return to its original state after being disturbed [40, 42]. To investigate how memory influences resistance and resilience within a community facing disruptions, we altered the growth rates of species over time (Figure 3(a) and (c)) and observed the subsequent effects.

Our findings indicate that memory strengthens a community's resistance to disturbances. Specifically, memory enables competing species to coexist for extended periods [24]. Moreover, when memory is a factor, transitions between alternative states of the community are more gradual following a short-term disturbance (Figure 3(b)) or are completely averted in some cases (Figure 3(d)).

Regarding resilience, our study reveals that the role of memory is complex. In the immediate aftermath of a disturbance, memory speeds up the community's recovery. However, as the community returns to its original state, memory slows down this recovery (Figure 3(b)). Thus, depending on the time frame, memory can either accelerate or decelerate recovery, enhancing or diminishing resilience accordingly. In terms of long-lasting effects, memory can exert its influence

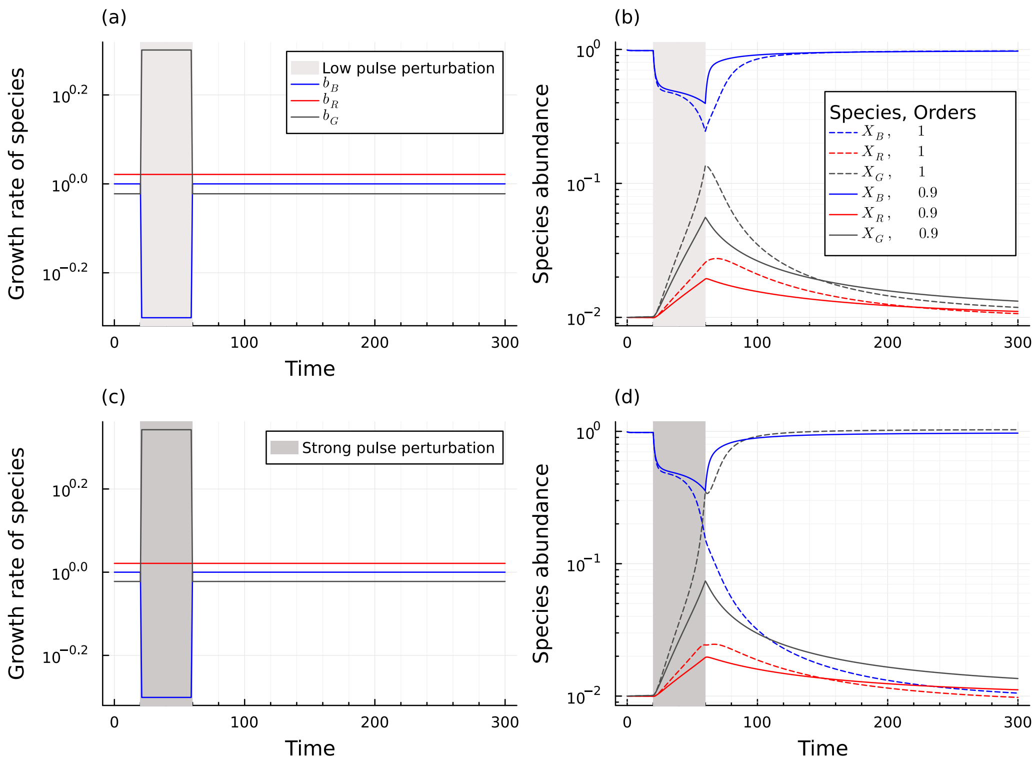

Fig. 3. This is the replication of Figure 2 from [24] using the mature FdeSolver Julia package to illustrate the influence of commensurate fractional derivatives on resistance and resilience of the 3-species community model (14). Let us suppose X_B, X_R, and X_G as the abundance of blue, red, and gray species, where indexes B, R, and G indicate the colors and present through all parameters. We solve the model with initial conditions $X_B(0) = 0.99$, $X_R(0) = 0.01$, and $X_B(0) = 0.01$, and parameters $K_{i,j} = 0.1, \forall i \neq j$, $n = 2$, $k_i = 1$. The order of the derivatives is set to integer one and fractional values $\beta_B = \beta_R = \beta_G = 0.9$. (a) The growth rates of the system during relaxation are $b_B = 1$, $b_R = 0.95$, and $b_G = 1.05$. A pulse perturbation is applied to the system during a time window indicated via the gray background by lowering the growth rate of the blue species ($b_B = 0.5$) and raising that of the gray species ($b_G = 2$). (b) The perturbation temporarily moves the system away from the original stable state. However, fractional orders increase resistance to perturbation since the community (solid lines) is not displaced as far from its initial state as the system's dynamics with integer order (dashed lines). (c) A slightly stronger pulse perturbation is applied to the growth rate of the gray species ($b_G = 2.2$). (d) It triggers a shift toward an alternative stable state dominated by the gray species in the system with integer order (dashed lines). However, fractional orders can also entirely prevent a state shift (solid lines) by increasing resistance and resilience to perturbation.

over multiple time scales due to the slow dissipation of the impact of prior states. Additionally, memory fortifies resilience in systems with multiple stable states by favoring the original stable state, as demonstrated in Figure 3(d).

4.2 Epidemiological Analysis of COVID-19 Transmission Dynamics

Recent studies highlight the growing importance of mathematical tools, especially fractional calculus modeling, in analyzing infectious diseases, addressing the intricate dynamics and complexities of studying epidemic spread [5, 34]. Following the global spread of the Coronavirus pandemic in

2019, infectious disease transmission analysis has become very popular, and a large amount of time series data on COVID-19 occurrences has been published [12]. Although this stimulated progress in epidemiological models, predicting the pandemic's spread proved challenging. We demonstrate how the FdeSolver package can speed up simulations and improve model fits in real data.

Methods for modeling general transmission dynamics are available; for instance, the Julia package Pathogen.jl [2] provides many tools to simulate and infer transmission networks and to model parameters of infectious disease spread. Our work can complement and extend such models by providing new tools for incorporating memory effects. These effects have been linked to the spread and control of epidemics, for instance, by changes in precautionary measures, such as vaccinations and lockdowns [39]. The COVID-19 evolution was suggested to exhibit power-law scaling features [21], which motivates using Caputo derivatives.

A recent compartmental model with super-spreader class [32] was extended with fractional order derivatives [31, 33]. In this model, the population is constant. It is divided into eight epidemiological compartments: susceptible individuals (S), exposed individuals (E), symptomatic and infectious individuals (I), super-spreaders individuals (P), infectious but asymptomatic individuals (A), hospitalized individuals (H), recovery individuals (R), and dead individuals (F). Here, we study an incommensurate fractional orders form of the model defined as:

$$
\begin{aligned}
\mathcal{D}_{t_0}^{\alpha_S} S(t) &= -\beta \frac{I}{N} S - l\beta \frac{H}{N} S - \beta' \frac{P}{N} S, \\
\mathcal{D}_{t_0}^{\alpha_E} E(t) &= \beta \frac{I}{N} S + l\beta \frac{H}{N} S + \beta' \frac{P}{N} S - \kappa E, \\
\mathcal{D}_{t_0}^{\alpha_I} I(t) &= \kappa \rho_1 E - (\gamma_a + \gamma_i) I - \delta_i I, \\
\mathcal{D}_{t_0}^{\alpha_P} P(t) &= \kappa \rho_2 E - (\gamma_a + \gamma_i) P - \delta_p P, \\
\mathcal{D}_{t_0}^{\alpha_A} A(t) &= \kappa (1 - \rho_1 - \rho_2) E, \\
\mathcal{D}_{t_0}^{\alpha_H} H(t) &= \gamma_a (I + P) - \gamma_r H - \delta_h H, \\
\mathcal{D}_{t_0}^{\alpha_R} R(t) &= \gamma_i (I + P) + \gamma_r H, \\
\mathcal{D}_{t_0}^{\alpha_F} F(t) &= \delta_i I + \delta_p P + \delta_h H,
\end{aligned}
\tag{15}
$$

where $0 < \alpha_i \leq 1$ ($i = S, E, I, P, A, H, R, F$) are the order derivatives corresponding to the compartments and not necessarily equal, and N is total population, where $N = S + E + I + P + A + H + R + F$. Table 2 describes the parameters with their values.

Let us consider the time series of Spain and Portugal's daily new confirmed cases from cumulative confirmed cases reported by the center for systems science and engineering at Johns Hopkins University [12]. We took into account three types of the system equation (15) in terms of the derivatives: with integer orders (M1), commensurate fractional orders (Mf1), and incommensurate fractional orders (Mf8). The approximated values for the parameters, the initial conditions, and the population size were derived from [31, 33].

Integer derivatives imply a memory-less system, while fractional values show a system influenced by its past and present. Such memory effects indicate that past disease exposures can shape current susceptibilities; prior exposure might offer some immunity, or past outbreaks might prompt increased caution. The consistent α in Mf1 suggests a simplified, uniform memory across disease stages, whereas Mf8's varied α values show distinct memory effects for different disease progression stages.

We estimated the parameter infection rate β by fitting the models to the data. For the fractional models (Mf1 and Mf8), we optimized the values of order derivatives and the parameter β to achieve the best fit. All the fitted values are listed in Table 3. The residual of the fitting is measured by

Table 2. Description of the Parameters Used in Model (15) and Their Values Taken from [31, 33]

Notation	Description	Value	Units
β	Transmission coefficient from infected Individuals	Fitted	day^{-1}
κ	rate at which exposed become infectious	0.25	day^{-1}
l	Relative transmissibility of hospitalized patients	1.56	Dimensionless
β'	High transmission coefficient due to superspreaders	7.65	day^{-1}
κ	Rate at which exposed become infectious	2.55	day^{-1}
ρ_1	Rate at which exposed people become infected I	0.58	Dimensionless
ρ_2	Rate at which exposed people become super-spreaders	0.001	Dimensionless
γ_a	Rate of being hospitalized	0.94	day^{-1}
γ_i	Recovery rate without being hospitalized	0.27	day^{-1}
γ_r	Recovery rate of hospitalized patients	0.50	day^{-1}
δ_i	Disease induced death rate due to infected class	1/23	day^{-1}
δ_p	Disease induced death rate due to super-spreaders	1/23	day^{-1}
δ_h	Disease induced death rate due to hospitalized class	1/23	day^{-1}

Table 3. Optimized Values of the Order Derivatives $\alpha_{S,E,I,P,A,H,R,F}$ and Parameter β of System Equation (15) with Integer Orders (Model M1) Commensurate Orders (Model Mf1) Incommensurate Orders (Model Mf8) for Fitting COVID-19 Data from Spain and Portugal with Their Errors Measured by RMSD

	Fitted values for Spain's data			Fitted values for Portugal's data		
Models	M1	Mf1	Mf8	M1	Mf1	Mf8
β	1.852210	2.503197	2.520522	1.638658	2.548180	2.841678
$\alpha_{S,E,I,P,A,H,R,F}$	-	0.829366	-	-	0.775394	-
α_S	-	-	0.829128	-	-	0.764514
α_E	-	-	0.865748	-	-	0.724861
α_I	-	-	0.685968	-	-	0.875856
α_P	-	-	0.500000	-	-	1.000000
α_A	-	-	0.749881	-	-	0.749835
α_H	-	-	0.809659	-	-	0.611715
α_R	-	-	0.749881	-	-	0.749835
α_F	-	-	0.749881	-	-	0.749835
RMSD	1,746.57	1,016.33	1,012.72	264.696	137.448	134.067

root mean square deviation (RMSD) defined as: $\mathrm{RMSD}(\mathrm{Y}, \widehat{\mathrm{Y}}) = \sqrt{\frac{1}{\mathrm{n}} \sum_{\mathrm{t}=1}^{\mathrm{n}} (\widehat{\mathrm{Y}}_{\mathrm{t}} - \mathrm{Y}_{\mathrm{t}})^2}$, where n is the number of data points, Y the vector of approximations, and $\widehat{Y}$ real data. In this context, Y is a scalar, $Y = \beta$, for the integer order model M1. For the fractional order models, Mf1 and Mf8, Y becomes a vector: for Mf1, it includes two variables, $Y = (\beta, \alpha_{S,E,I,P,A,H,R,F})$, and for Mf8, it expands to a vector with nine variables, $Y = (\beta, \alpha_S, ..., \alpha_F)$. Thus, in these models, the order derivatives function as additional parameters for the fitting process. Utilizing advanced optimization methods, we effectively determine the optimal values for these parameters.

We used the StatsBase.jl package to apply the RMSD for the real values and the sum of all infectious compartments $I+P+H$ approximated from the models. To minimize the residual, we used two functions from the Optim.jl package [28]: the Limited-memory Broyden–Fletcher–Goldfarb–Shanno algorithm, and Simulated Annealing for MINimization, for problems with bounds constraints. The

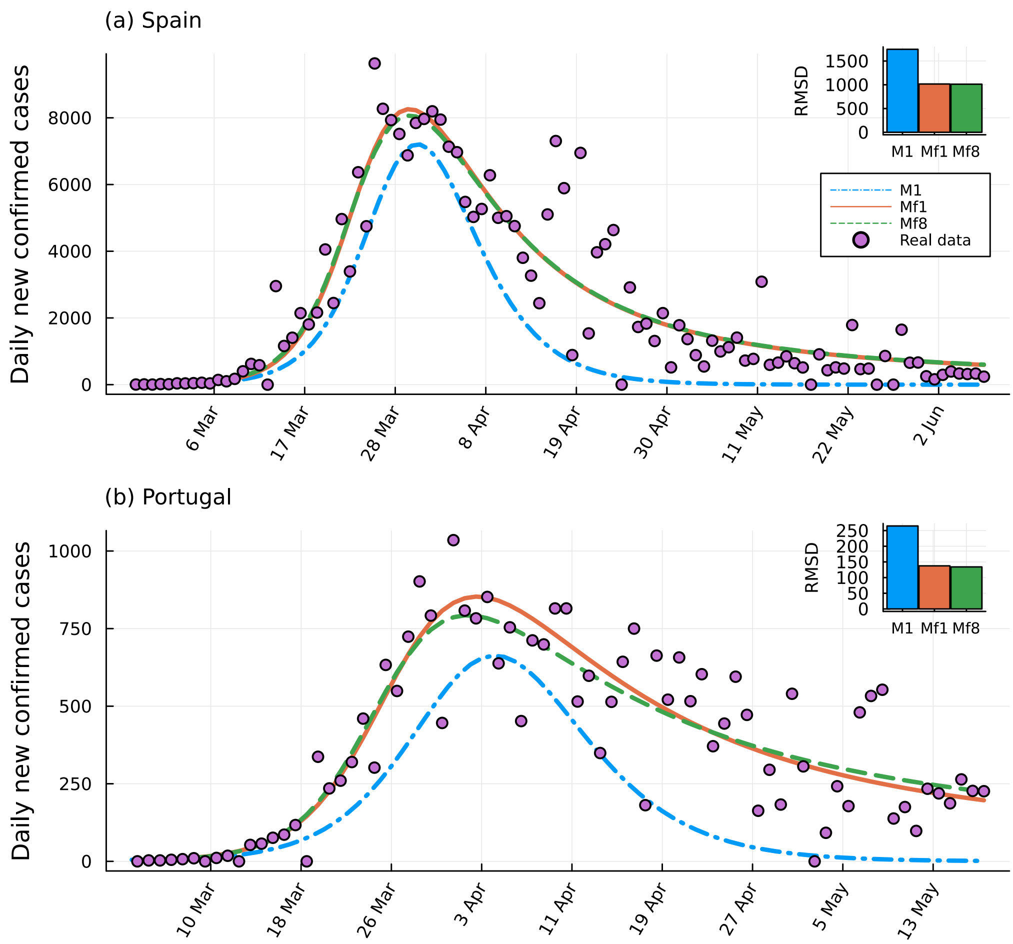

Fig. 4. The comparison between real data on the daily new confirmed cases as retrieved from center for systems science and engineering [12] and the estimation of $I + P + H$ from the system equation (15) with integer order derivatives (model M1 shown by the dash-dotted blue line), commensurate fractional derivatives (model Mf1 shown by the solid orange line), and incommensurate fractional derivatives (model Mf8 shown by the green dashed line). The circles indicate the real data for (a) Spain from 25 February to 7 June and (b) Portugal from 3 March to 17 May. The infection rate β is fitted for the M1 model, the commensurate order for the Mf1 model together with β are optimized, and finally, the eight incommensurate orders and β for model Mf8 are optimized to fit real data. The specified error bars are based on **root mean square deviation (RMSD)** such that the fewer values indicate the less residual variance. The fitted values are listed in Table 3.

optimization process was configured to terminate upon meeting predefined criteria, such as a specific number of iterations or achieving an acceptable error tolerance level in each iteration. The user manual presents the related codes (Section 4.2).

Figure 4 compares the simulations from our models to real data. The discrepancies in residual errors highlight that solely fitting one parameter in model M1 does not provide an optimal fit. Enhancing the model's flexibility by optimizing the order of derivatives noticeably increases its accuracy. However, while adjusting the order of derivatives can elevate the precision of our model, the computational expense rises significantly. For instance, optimizing for eight orders in model Mf8 is more resource-intensive than for a single order in model Mf1. This greater computational demand does not always translate to a proportionate increase in performance. Figure 4 and Table 3 illustrate this, showing the minimal performance difference between Mf1 and Mf8. In a biological context, these mathematical adjustments allow us to better represent the intricate facets of disease spread, including diverse transmission routes (e.g., symptomatic, asymptomatic, super-spreaders) and variations in disease severity.

When comparing the models, the fractional order models suggest a faster and more intense disease spread, peaking earlier and higher. The higher estimated transmission coefficient β, as shown in Table 3, indicates a more aggressive disease transmission. Furthermore, the order values in the fractional models suggest a significant influence of past dynamics on the current state of the disease. These memory effects and the higher transmission coefficient could explain the earlier and more pronounced peak. Conversely, with its lower transmission coefficient, the integer order model estimates a slower and less aggressive disease progression. The results underline the critical role of memory effects, such as lingering immunity or behavioral changes due to past exposures, in shaping disease dynamics, emphasizing the need for swift interventions and the delay risk.

5 Discussion

In this work, we evaluated the efficiency and reliability of the FdeSolver in various scenarios, as illustrated in our figures. This discussion aims to understand better these results, their implications, and the package's applications.

Figure 1(a) highlights the efficiency of the NR method over the PC method and shows Julia's slightly superior performance over MATLAB. In stiff problems, as shown in Figure 1(b), our FdeSolver outperforms its MATLAB counterparts, particularly for step sizes $h < 2^{-7}$. For high-order problems, FdeSolver matches MATLAB's performance and excels over other methods, as depicted in Figure 1(c). Across all one-dimensional cases, alternative solvers provided by FractionalDiffEq demonstrate inadequate or insufficient accuracy and require adjustments to avoid errors. In the context of the non-oscillatory SIR model, FdeSolver exhibits superiority over FractionalDiffEq methods, as shown in Figure 2(a). In the sharp-oscillatory example of the Lotka-Volterra model (Figure A1), FdeSolver stands out as other methods falter, as seen in Figure 2(b). Conclusively, a meticulous assessment employing randomized parameters and conditions, illustrated in Figure A2, reaffirms FdeSolver's Julia package as exemplary, surpassing MATLAB versions and clearly showing its flexibility and performance in solving various problems.

The applicability of FdeSolver in ecology is exemplified by its implementation in Microbiome-Analysis.jl — a leading package for cutting-edge microbiome data analysis. Here, the functionality of FdeSolver is applied to simulate and model complex microbial communities. In practice, this allows us to predict the behavior of natural microbial systems and rigorously evaluate artificial bioprocess outcomes. Leveraging these design insights will significantly reduce implementation costs and enhance production yields.

Our package has been fundamental in elevating recent epidemiological research and driving innovation in creating and refining mathematical models for infectious diseases. Its application was central in the intricate modeling of the Ebola epidemic, supporting the formulation of a robust 8-dimensional compartmental model [29]. Notably, it also bolstered a COVID-19 study, aiding in developing a fractional model that embraced environmental pathogens, ensuring meticulous numerical simulations and precise data fitting from South African cases [30]. The utilization of our package, particularly on the high-performance computing system PUHTI at the Finnish IT Center for Science, demonstrates its adaptability and instrumental role in enhancing epidemiological models' accuracy and comprehensiveness.

One of the key strengths of the FdeSolver package is its exceptional speed and adaptability, primarily due to its foundation in the Julia programming language. Julia is a compiled language, enhancing execution speed and resource efficiency. Its interactive REPL environment adds user-friendliness, benefiting those requiring a hands-on approach. The open-source package encourages community contributions for feature enhancements, optimizations, and bug fixes. This ensures that FdeSolver remains versatile and current in fast-evolving research and development sectors.

Additionally, FdeSolver leverages Julia's unique multiple dispatch feature. Multiple dispatch in Julia is a paradigm where the method called during a function call is determined by the types of all arguments, not just the first one. Consequently, it dynamically selects the appropriate function version based on input types, thus increasing performance and coding flexibility. Unlike other programming languages, where functions are object-specific, Julia's independent functions make multiple dispatch more potent. This mechanism is particularly beneficial for our package, which deals with various types of FDEs. Depending on the type of equation and associated parameters passed to a function in FdeSolver, Julia's multiple dispatch system ensures that the most optimized and suitable algorithm is selected for computation. This not only streamlines the coding process but also results in more efficient and faster computations, a significant advantage in computational research and application development.

Lastly, Julia's support for static and dynamic types, as utilized in FdeSolver, significantly enhances code efficiency and flexibility. Static typing allows for early type verification and optimized performance, crucial for computationally intensive tasks in FdeSolver. Meanwhile, dynamic typing offers the adaptability to handle various data types and structures, essential in diverse research scenarios. This balanced approach in type specification ensures that FdeSolver can efficiently tackle a wide range of computational challenges, illustrating its significance in computational research and application development.

6 Conclusion

Efficient methods for optimizing FDE models can provide valuable tools for dynamical systems analysis in many application fields. The scarcity of user-friendly utilities for fractional calculus in modern open-source computational languages, such as Julia, has hampered the adoption and development of these methods as part of the broader data science ecosystem. We have introduced a new Julia package that helps to bridge this gap, offering a collection of efficient, customizable, and seamlessly integrated utilities designed for numerical analysis and simulation of FDE models.

Our FdeSolver package provides numerical solutions for FDEs with Caputo derivatives in univariate and multivariate systems. The algorithm is based on PI rules, and we have optimized the computational efficiency by incorporating the FFT technique. The benchmarking experiments showed better speed and accuracy performance than the currently available alternatives in MATLAB and Julia. Furthermore, we have demonstrated two applications of complex systems for simulating and fitting the models to real data. The package adheres to good practices in open research software development, including open licensing, unit testing, and comprehensive documentation with reproducible examples in the accompanying user manual.

While maintaining adherence to fundamental numerical schemes, FdeSolver exhibits enhanced adaptability, facilitating its application across diverse research scenarios. Its intrinsic flexibility and ease of integration make it a practical and efficient choice for computational tasks, extending the applicability of the numerical scheme it employs. This adaptability is a significant advantage of FdeSolver over existing schemes, promoting broader and more efficient utilization in computational research.

Future extensions of FdeSolver could include fractional linear multistep methods, improving efficiency beyond PI rules [15]. Time delays in species interactions, influencing ecological stability [46], could be addressed by extending the package based on delay fractional equations [18]. Support for models with time-varying derivatives [35] is also planned, thus enhancing applicability in problems like infectious disease transmission with multiple memory effects [21]. Due to the flexibility of fractional models, automated cross-validation will be developed to manage overfitting risks. Integration of Bayesian approximation from Turing.jl package [17] and GPU support [3] is the future priority, aiming for enhanced performance and broader integration within the Julia ecosystem.

Thus, FdeSolver exemplifies the implementation of robust computational methodologies in scientific computing, aligning with the guidelines presented in [44]. Its development demonstrates the importance of such practices in ensuring reproducibility, reliability, and efficiency in scientific research. The adherence of FdeSolver to these standards makes it not just a technical achievement but also a model for future scientific software development, bridging gaps in the data science ecosystem and facilitating broader adoption of fractional calculus methods in diverse scientific fields.

Appendix

There are two additional figures in this Appendix, one showing the dynamics of the multi-dimensional example systems (Figure A1) and the other showing the speed and accuracy of solving the problems with random conditions (Figure A2).

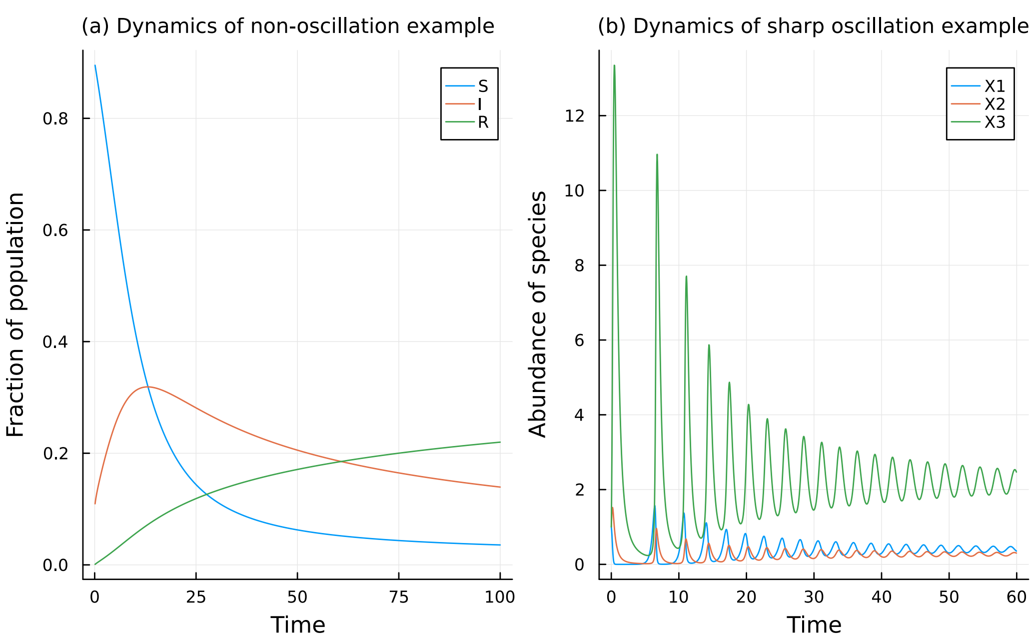

Fig. A1. Illustration of the dynamics of multi-dimensional examples. (a) Non-oscillation dynamics of the susceptible (S), infected (I), and recovered (R) population fractions for Example 3.2.1. (b) Sharp oscillation dynamics of the abundance of species X_1, X_2, and X_3 for Example 3.2.2.

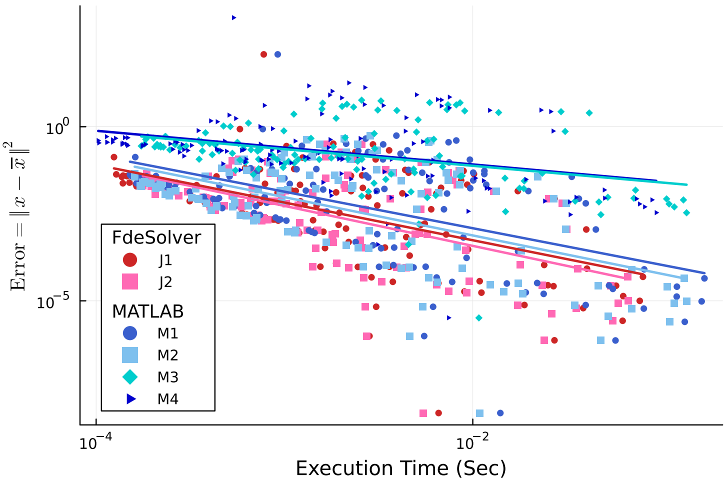

Fig. A2. Scatter plot of error versus execution time for several solutions of Examples 3.1.1–3.2.1 with random parameters and conditions. The axes are visualized on a logarithmic scale. The unit of the X-axes is second (Sec), and the Y-axes show the Euclidean norm of the difference between the approximations and exact values. The lines on the background are the linear fit of the logarithm of the scatter points, and the color of the lines corresponds to the methods with the same color. For FdeSolver methods, we use J1 to represent the PC method and J2 for the NR method. In MATLAB routines, M1 stands for the PC method, M2 for the NR method, M3 signifies the NR method with the PI rectangular rule, and M4 denotes the explicit PI rectangular rule without PC.

References

[1] Mohammad M. Amirian, Isaac N. Towers, Zlatko Jovanoski, and Andrew J. Irwin. 2020. Memory and mutualism in species sustainability: A time-fractional Lotka-Volterra model with harvesting. *Heliyon* 6, 9 (2020), e04816. DOI: https://doi.org/10.1016/j.heliyon.2020.e04816

[2] Justin Angevaare, Zeny Feng, and Rob Deardon. 2022. Pathogen.jl: Infectious disease transmission network modeling with Julia. *Journal of Statistical Software* 104, 4 (2022), 1–30. DOI: https://doi.org/10.18637/jss.v104.i04

[3] Tim Besard, Christophe Foket, and Bjorn De Sutter. 2019. Effective extensible programming: Unleashing Julia on GPUs. *IEEE Transactions on Parallel and Distributed Systems* 30, 4 (2019), 827–841. DOI: https://doi.org/10.1109/TPDS.2018.2872064

[4] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B. Shah. 2017. Julia: A fresh approach to numerical computing. *SIAM Review* 59, 1 (2017), 65–98. DOI: https://doi.org/10.1137/141000671

[5] Yuli Chen, Fawang Liu, Qiang Yu, and Tianzeng Li. 2021. Review of fractional epidemic models. *Applied Mathematical Modelling* 97 (2021), 281–307. DOI: https://doi.org/10.1016/j.apm.2021.03.044

[6] Edmundo Capelas de Oliveira and José António Tenreiro Machado. 2014. A review of definitions for fractional derivatives and integral. *Mathematical Problems in Engineering* 2014 (Jun. 2014), Article 238459. DOI: https://doi.org/10.1155/2014/238459

[7] Kai Diethelm and N. J. Ford. 2010. The analysis of fractional differential equations. In *Jean-Michel Morel, Bernard Teissier*. Lecture Notes in Mathematics, Springer, Berlin, Heidelberg, Vol. 2004, 3–12.

[8] Kai Diethelm, Neville J. Ford, and Alan D. Freed. 2002. A predictor-corrector approach for the numerical solution of fractional differential equations. *Nonlinear Dynamics* 29, 1–4 (2002), 3–22. DOI: https://doi.org/10.1023/A:1016592219341

[9] Kai Diethelm, Neville J. Ford, and Alan D. Freed. 2004. Detailed error analysis for a fractional Adams method. *Numer. Algorithms* 36, 1 (2004), 31–52. DOI: https://doi.org/10.1023/B:NUMA.0000027736.85078.be

[10] Kai Diethelm, Roberto Garrappa, Andrea Giusti, and Martin Stynes. 2020b. Why fractional derivatives with nonsingular kernels should not be used. *Fractional Calculus and Applied Analysis* 23, 3 (Jun. 2020), 610–634. DOI: https://doi.org/10.1515/fca-2020-0032

[11] Kai Diethelm, Roberto Garrappa, and Martin Stynes. 2020a. Good (and not so good) practices in computational methods for fractional calculus. *Mathematics* 8, 3 (2020), 324. DOI: https://doi.org/10.3390/math8030324

[12] Ensheng Dong, Hongru Du, and Lauren Gardner. 2020. An interactive web-based dashboard to track COVID-19 in real time. *The Lancet Infectious Diseases* 20, 5 (2020), 533–534. DOI: https://doi.org/10.1016/S1473-3099(20)30120-1

[13] Leila Eftekhari and Mohammad M. Amirian. 2022. Stability analysis of fractional order memristor synapse-coupled hopfield neural network with ring structure. *Cognitive Neurodynamics* 17, 4 (Aug. 2022), 1045–1059. DOI: https://doi.org/10.1007/s11571-022-09844-9

[14] Roberto Garrappa. 2010. On linear stability of predictor–corrector algorithms for fractional differential equations. *International Journal of Computer Mathematics* 87, 10 (2010), 2281–2290. DOI: https://doi.org/10.1080/00207160802624331

[15] Roberto Garrappa. 2015. Trapezoidal methods for fractional differential equations: Theoretical and computational aspects. *Mathematics and Computers in Simulation* 110 (2015), 96–112. DOI: https://doi.org/10.1016/j.matcom.2013.09.012

[16] Roberto Garrappa. 2018. Numerical solution of fractional differential equations: A survey and a software tutorial. *Mathematics* 6, 2 (2018), 16. DOI: https://doi.org/10.3390/math6020016

[17] Hong Ge, Kai Xu, and Zoubin Ghahramani. 2018. Turing: A language for flexible probabilistic inference. In *Proceedings of the 21st International Conference on Artificial Intelligence and Statistics* (Proceedings of Machine Learning Research, Vol. 84)i, Amos Storkey and Fernando Perez-Cruz (Eds.). PMLR, 1682–1690. Retrieved from https://proceedings.mlr.press/v84/ge18b.html

[18] M. Ghasemi, M. Fardi, and R. Khoshsiar Ghaziani. 2015. Numerical solution of nonlinear delay differential equations of fractional order in reproducing kernel Hilbert space. *Applied Mathematics and Computation* 268 (2015), 815–831. DOI: https://doi.org/10.1016/j.amc.2015.06.012

[19] Chaitanya S. Gokhale, Stefano Giaimo, and Philippe Remigi. 2021. Memory shapes microbial populations. *PLOS Computational Biology* 17, 10 (2021), 1–16. DOI: https://doi.org/10.1371/journal.pcbi.1009431

[20] Didier Gonze, Leo Lahti, Jeroen Raes, and Karoline Faust. 2017. Multi-stability and the origin of microbial community types. *The ISME Journal* 11, 10 (2017), 2159. DOI: https://doi.org/10.1038/ismej.2017.60

[21] Hadi Jahanshahi, Jesus M. Munoz-Pacheco, Stelios Bekiros, and Naif D. Alotaibi. 2021. A fractional-order SIRD model with time-dependent memory indexes for encompassing the multi-fractional characteristics of the COVID-19. *Chaos, Solitons & Fractals* 143 (2021), Article 110632. DOI: https://doi.org/10.1016/j.chaos.2020.110632

[22] William Ogilvy Kermack and Anderson G. McKendrick. 1927. A contribution to the mathematical theory of epidemics. *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character* 115, 772 (1927), 700–721.

[23] Moein Khalighi, Leila Eftekhari, Soleiman Hosseinpour, and Leo Lahti. 2021. Three-species Lotka-Volterra model with respect to Caputo and Caputo-Fabrizio fractional operators. *Symmetry* 13, 3 (2021), 368. DOI: https://doi.org/10.3390/sym13030368

[24] Moein Khalighi, Guilhem Sommeria-Klein, Didier Gonze, Karoline Faust, and Leo Lahti. 2022. Quantifying the impact of ecological memory on the dynamics of interacting communities. *PLOS Computational Biology* 18, 6 (2022), 1–21. DOI: https://doi.org/10.1371/journal.pcbi.1009396

[25] Anatoly Aleksandrovich Kilbas, Hari M. Srivastava, and Juan J. Trujillo. 2006. *Theory and Applications of Fractional Differential Equations*, Vol. 204. Elsevier.

[26] Zhuo Li, Lu Liu, Sina Dehghan, YangQuan Chen, and Dingyü Xue. 2017. A review and evaluation of numerical tools for fractional calculus and fractional order controls. *International Journal of Control* 90, 6 (2017), 1165–1181. DOI: https://doi.org/10.1080/00207179.2015.1124290

[27] Francesco Mainardi. 2020. Why the Mittag-Leffler function can be considered the queen function of the fractional calculus? *Entropy* 22, 12 (2020), Article 1359.

[28] Patrick Kofod Mogensen, John Myles White, Asbjørn Nilsen Riseth, Tim Holy, Miles Lubin, Christof Stocker, Andreas Noack, Antoine Levitt, Christoph Ortner, Blake Johnson, Dahua Lin, Kristoffer Carlsson, Yichao Yu, Christopher Rackauckas, Josua Grawitter, Alex Williams, Alexey Stukalov, Ben Kuhn, Benoît Legat, Jeffrey Regier, cossio, Michael Creel, Ron Rock, Thomas R. Covert, Benoit Pasquier, Takafumi Arakaki, Andrew Clausen, and Arno Strouwen. 2022. JuliaNLSolvers/Optim.jl: Release 1.7.3. DOI: https://doi.org/10.5281/zenodo.7068265

[29] Faïçal Ndaïrou, Moein Khalighi, and Leo Lahti. 2023. Ebola epidemic model with dynamic population and memory. *Chaos, Solitons & Fractals* 170 (2023), Article 113361. DOI: https://doi.org/10.1016/j.chaos.2023.113361

[30] Moein Khalighi, Faïçal Ndaïrou, and Leo Lahti. 2024. Epidemic transmission modeling with fractional derivatives and environmental pathogens. *International Journal of Biomathematics* (2024). DOI: https://doi.org/10.1142/S1793524524500852

[31] Faïçal Ndaïrou, Iván Area, Juan J. Nieto, Cristiana J. Silva, and Delfim F. M. Torres. 2021. Fractional model of COVID-19 applied to Galicia, Spain and Portugal. *Chaos, Solitons & Fractals* 144 (2021), Article 110652. DOI: https://doi.org/10.1016/j.chaos.2021.110652

[32] Faïçal Ndaïrou, Iván Area, Juan J. Nieto, and Delfim F. M. Torres. 2020. Mathematical modeling of COVID-19 transmission dynamics with a case study of Wuhan. *Chaos, Solitons & Fractals* 135 (2020), Article 109846. DOI: https://doi.org/10.1016/j.chaos.2020.109846

[33] Faïçal Ndaïrou and Delfim F. M. Torres. 2021. Mathematical analysis of a fractional COVID-19 model applied to Wuhan, Spain and Portugal. *Axioms* 10, 3 (2021), 135. DOI: https://doi.org/10.3390/axioms10030135

[34] Kottakkaran Sooppy Nisar, Muhammad Farman, Mahmoud Abdel-Aty, and Jinde Cao. 2023. A review on epidemic models in sight of fractional calculus. *Alexandria Engineering Journal* 75 (2023), 81–113. DOI: https://doi.org/10.1016/j.aej.2023.05.071

[35] Sansit Patnaik, John P. Hollkamp, and Fabio Semperlotti. 2020. Applications of variable-order fractional operators: A review. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 476, 2234 (2020), Article 20190498. DOI: https://doi.org/10.1098/rspa.2019.0498

[36] Igor Podlubny. 1998. *Fractional Differential Equations: An Introduction to Fractional Derivatives, Fractional Differential Equations, to Methods of Their Solution and Some of Their Applications*. Elsevier.

[37] Daniel A. Power, Richard A. Watson, Eörs Szathmáry, Rob Mills, Simon T. Powers, C. Patrick Doncaster, and B Czapp. 2015. What can ecosystems learn? Expanding evolutionary ecology with learning theory. *Biology Direct* 10, 1 (2015), 69. DOI: https://doi.org/10.1186/s13062-015-0094-1

[38] Qingyu Qu and Jadon Clugston. 2023. SciFracX/FractionalDiffEq.jl: v0.3.1. DOI: https://doi.org/10.5281/zenodo.7789037

[39] Meghdad Saeedian, Moein Khalighi, Nahid Azimi-Tafreshi, Gholamreza Jafari, and Marcel Ausloos. 2017. Memory effects on epidemic evolution: The susceptible-infected-recovered epidemic model. *Physical Review E* 95 (2017), Article 022409. DOI: https://doi.org/10.1103/PhysRevE.95.022409

[40] Ashley Shade, Hannes Peter, Steven Allison, Didier Baho, Mercé Berga, Helmut Buergmann, David Huber, Silke Langenheder, Jay Lennon, Jennifer Martiny, Kristin Matulich, Thomas Schmidt, and Jo Handelsman. 2012. Fundamentals of microbial community resistance and resilience. *Frontiers in Microbiology* 3 (2012), 417. DOI: https://doi.org/10.3389/fmicb.2012.00417

[41] Emrah Şimşek and Minsu Kim. 2019. Power-law tail in lag time distribution underlies bacterial persistence. *PNAS* 116, 36 (2019), 17635–17640. DOI: https://doi.org/10.1073/pnas.1903836116

[42] Felix Sommer, Jacqueline Moltzau Anderson, Richa Bharti, Jeroen Raes, and Philip Rosenstiel. 2017. The resilience of the intestinal microbiota influences health and disease. *Nature Reviews Microbiology* 15, 10 (2017), 630–638. DOI: https://doi.org/10.1038/nrmicro.2017.58

[43] HongGuang Sun, Yong Zhang, Dumitru Baleanu, Wen Chen, and YangQuan Chen. 2018. A new collection of real world applications of fractional calculus in science and engineering. *Communications in Nonlinear Science and Numerical Simulation* 64 (2018), 213–231. DOI: https://doi.org/10.1016/j.cnsns.2018.04.019

[44] Greg Wilson, Jennifer Bryan, Karen Cranston, Justin Kitzes, Lex Nederbragt, and Tracy K. Teal. 2017. Good enough practices in scientific computing. *PLOS Computational Biology* 13, 6 (2017), 1–20. DOI: https://doi.org/10.1371/journal.pcbi.1005510

[45] Dingyü Xue. 2019. *FOTF Toolbox for Fractional-Order Control Systems*. De Gruyter, Berlin, 237–266. DOI: https://doi.org/10.1515/9783110571745-011

[46] Yuguang Yang, Kevin R. Foster, Katharine Z. Coyte, and Aming Li. 2023. Time delays modulate the stability of complex ecosystems. *Nature Ecology & Evolution* 7, 10 (Oct. 2023), 1610–1619. DOI: https://doi.org/10.1038/s41559-023-02158-x

[47] Azhar Ali Zafar, Grzegorz Kudra, Jan Awrejcewicz, Thabet Abdeljawad, and Muhammad Bilal Riaz. 2020. A comparative study of the fractional oscillators. *Alexandria Engineering Journal* 59, 4 (2020), 2649–2676. DOI: https://doi.org/10.1016/j.aej.2020.04.029

Received 23 December 2022; revised 8 February 2024; accepted 25 June 2024