AIMS Mathematics

http://www.aimspress.com/journal/Math

AIMS Mathematics, 11(3): 5798–5823.
DOI: 10.3934/math.2026239
Received: 30 December 2025
Revised: 14 February 2026
Accepted: 25 February 2026
Published: 09 March 2026

Research article

Unified scientific tool to investigate fractional derivatives of arbitrary variable order with time-memory and order-memory: The VOFD Python package

Daniel Clemente-López[1], Jesus M. Munoz-Pacheco[2,*], José de Jesus Rangel-Magdaleno[1] and Lizbeth Vargas-Cabrera[2]

[1] Department of Electronics, Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE), Luis Enrique Erro No. 1, Tonantzintla, Puebla 72840, Mexico

[2] Faculty of Electronics Sciences, Benemérita Universidad Autónoma de Puebla, Av. San Claudio y 18 Sur, Puebla 72570, Mexico

[*] **Correspondence:** Email: jesusm.pacheco@correo.buap.mx.

Abstract: Fractional derivatives of arbitrary order may enhance several practical applications in science and engineering, ranging from physics, chemistry, economics, and botany to robotics, neural networks, data encryption, and internet of things (IoT). In such variable order (VO) fractional derivatives, the memory property changes as a function of time and order. However, despite a strong mathematical background, there are no software tools dedicated to exploiting the distinctive properties of VO derivatives. This is mainly due to the difficulty of capturing the complex properties of VO derivatives by a suitable computational method for numerical simulations. Therefore, this tutorial paper introduces a simple open-source Python library (VOFD Python package which can be downloaded from `http://pypi.org/project/vofd/`) that implements numerical integration schemes based on finite-difference approximations to solve Caputo VO derivatives (V1) and two convolution-based Caputo VO derivatives (V2 and V3). In addition, the proposed package includes a subroutine for generating bifurcation diagrams. Step-by-step examples for the Riccati equation and Chen system, along with error and convergence analyses, are provided to demonstrate the benefits of the proposed tool. The variable-order fractional derivative (VOFD) package provides high-performance numerical routines accelerated with the Numba JIT compiler, significantly reducing computation time for numerical solutions and large-scale bifurcation analysis, enabling efficient exploration of variable-order fractional models by both experts and practitioners.

Keywords: fractional calculus; chaos; non-constant order derivative; open-source software; Numba JIT compiler; finite difference approximations; numerical solver
Mathematics Subject Classification: 34A08, 37D04, 37D45, 37M05, 37M22

1. Introduction

It is well-known that the hallmark of fractional calculus is its memory properties. In a constant fractional-order q, the derivative order has an identical magnitude on the solving interval. Otherwise, the variable-order (VO) derivative permits the derivative order to change with time [1]. As a result of this effect, the fractional order can be a time-dependent function $q(t)$, a time- and order-dependent function $q(t, \tau)$, and an order-dependent function $q(\tau)$. Samko and Ross, essential figures in the development of variable-order fractional derivatives [2, 3], proposed the foundation for understanding variable-order operators and the theoretical basis for further developments and applications [4, 5]. Over the years, several proposals for variable-order operators have appeared in the literature [6–9]. The results obtained with VO derivatives have demonstrated additional advantages over conventional fractional calculus. In works such as [10], it was reported that VO systems have been used to describe hereditary processes more efficiently than constant-order systems. Additional applications of VO systems include control strategies [11, 12] and chaos-based security [13]. In [14], the existence, Hyers-Ulam stability, and trajectory controllability of Hilfer fractional stochastic pantograph equations with random impulses in the pth mean were established. Numerical results confirmed the stability conditions' outcomes. In [15], the existence, uniqueness, and trajectory controllability of mild solutions to the fractional stochastic differential system in the new Banach space setting were analyzed. In addition, controllability in the sense of trajectory for higher-order Riemann-Liouville stochastic differential systems was also given. In [16], the authors presented the existence, uniqueness, and trajectory controllability for higher-order noninstantaneous impulsive fractional neutral stochastic differential equations using Grönwall's inequality and deviating argument, and in [17], the results for the trajectory controllability of higher-order fractional neutral stochastic integrodifferential systems (FNSIDEs) with noninstantaneous impulsive (NI) via state-dependent delay were obtained by the Mönch-type fixed-point theorem and Hausdorff measure. Next, the T-controllability of both coupled nonlinear fractional-order stochastic differential systems with integral boundary conditions via integral contractors and a fractional neutral stochastic integro-delay differential system with a Lipschitz condition was presented in [18] and [19], respectively. In [20], the solutions of the Hilfer fractional stochastic differential pantograph equations were demonstrated. Even in artificial neural networks, fractional calculus has important implications. For instance, in [21], it was introduced using solutions to the fractional Riccati equation. Several numerical experiments validated the proposed approach. A novel, generalized approach was proposed for obtaining exact analytical solutions of nonlinear partial differential equations by applying bilinear neural networks and bilinear residual neural networks in [22] and [23], respectively. In [24] and [25], a symbolic approach combined with a multimodal reasoning algorithm was proposed, which does not require bilinear transformations to find solutions to nonlinear partial differential equations.

In the literature, three main definitions of VO fractional derivatives have been proposed, namely V1, V2, and V3. These definitions differ in their mathematical formulations and underlying principles, yet, they share the common goal of capturing the time-memory and order-memory characteristics of fractional-order in mathematical models. Although, the theoretical framework of VO calculus possesses a solid basis, translating the entire history of time and order memory may be ineffective. For instance, a fractional-order Lorenz chaotic system can be 100 times slower than the classical one. In numerical algorithms, simulation time grows significantly because a fine-grained time step is required for convergence, thereby increasing computational cost; for example, long-time simulations, such as bifurcation diagrams, become almost impractical. In this manner, the development of feasible computational methods and practical software tools remains a challenge.

Several studies have bridged the gap between VO derivatives theory and practice. In [11], the authors proposed a simulation and control implementation for VO chaotic systems using a sliding-mode control strategy on a field-programmable gate array (FPGA). They utilized LabVIEW software to construct these systems and solved the associated differential equations using the VO Adams algorithm. In [13], a novel VO chaotic system was introduced for fast image encryption. This approach used the concept of short memory in the Caputo derivative. Other works, such as [26] and [27], utilized numerical schemes and hardware implementations to address the challenges posed by variable-order derivatives. These efforts demonstrated the feasibility and effectiveness of employing VO calculus in various science and engineering applications, including chaos control and encryption. Further research, exemplified by [28] and [29], has explored numerical analysis, synchronization, and engineering applications of VO systems, demonstrating their potential across diverse domains, from chaotic systems to neural networks.

Therefore, this tutorial paper introduces a simple open-source Python library variable-order fractional derivative (VOFD Python package) that implements numerical integration schemes based on finite-difference approximations to solve Caputo VO derivatives (V1) and two convolution-based Caputo VO derivatives (V2 and V3). By using the Numba JIT compiler and parallelizing tasks, the computational time for long-term numerical solutions is reduced, facilitating efficient exploration of variable-order fractional models by experts and practitioners alike. Therefore, research on software tools for the study of VO fractional derivatives is vital to understanding this important yet underexplored subject. The contributions of this tutorial paper are:

- A comprehensive and open-source Python library (termed as VOFD Python package) and its complete documentation with step-by-step instructions for numerical solving of three types of variable-order fractional derivatives, which can be downloaded from (`http://pypi.org/project/vofd/`).
- Implementation of numerical algorithms for approximating variable-order derivatives with time-memory (V1), weak order-memory (V2), and strong order-memory (V3), using the theoretical formulation in Ref. [4].
- An optimization of such algorithms using the Numba JIT compiler and parallel routines in Python, ensuring high performance and feasibility.
- To the best knowledge of the authors, this is the first time introducing a software package to compute numerical solutions of fractional derivatives of arbitrary variable order. We seek to provide a user-friendly, versatile platform for researchers and practitioners to explore and apply VO calculus to their respective investigations.

2. Mathematical background

2.1. Caputo fractional-order operator

The Caputo definition for a fractional-order derivative is given as the following [30]:

$$
{}^{C}D_t^q y(t) := \begin{cases} \dfrac{1}{\Gamma(m-q)} \displaystyle\int_{t_0}^{t} \dfrac{y^{(m)}(\tau)}{(t-\tau)^{q+1-m}} d\tau, & m-1 < q \le m, \\ \dfrac{d^m}{dt^m} y(t), & q = m, \end{cases} \tag{2.1}
$$

where $t \in [t_0, t_1]$, $m \in N^+$ is the first integer larger than q, with $m = 1$ for $0 < q < 1$, and $\Gamma(\cdot)$ is the Euler gamma function.

2.2. VO derivative type 1 (V1)

The definition of VO derivative type 1 (V1) with time-memory is stated as follows [31]:

$$
{}_{0+}^{V1}D_t^{q(t)} y(t) = \frac{1}{\Gamma(m-q(t))} \int_{0+}^{t} \frac{y^{(m)}(\tau) d\tau}{(t-\tau)^{q(t)+1-m}}, \; m-1 < q(t) \le m. \tag{2.2}
$$

This definition indicates that the system memory varies with time and depends on its current state. Therefore, variable-order fractional derivatives can be used to characterize the variable memory effect of the system. If $q(t)$ is constant, it reduces to the Caputo definition in Eq (2.1).

2.3. VO derivative type 2 (V2) and type 3 (V3)

The definitions of VO derivative type 2 (V2) and type 3 (V3) with weak and strong order-memory are given in (2.3) and (2.4), respectively. These definitions possess the derivative order convolved, so there is embedded memory in the derivative order itself [31]. Between the two, this effect is more substantial in the V3 definition [4].

$$
{}_{0+}^{V2}D_t^{q(t)} y(t) = \int_{0+}^{t} \frac{1}{\Gamma(m-q(\tau))} \frac{y^{(m)}(\tau) d\tau}{(t-\tau)^{q(\tau)+1-m}}, \; m-1 < q(t) \le m, \tag{2.3}
$$

$$
{}_{0+}^{V3}D_t^{q(t)} y(t) = \int_{0+}^{t} \frac{1}{\Gamma(m-q(t-\tau))} \frac{y^{(m)}(\tau) d\tau}{(t-\tau)^{q(t-\tau)+1-m}}, \; m-1 < q(t) \le m. \tag{2.4}
$$

2.4. Discretization of the VO derivatives

The approximations of the VO derivatives V1, V2, and V3 are realized by applying finite difference approximations as follows [4]:

$$
{}_{0+}^{V1}D_t^{q(t_{k+1})} y(t_{k+1}) \approx \frac{h^{-q(t_{k+1})}}{\Gamma(m+1-q(t_{k+1}))} \sum_{j=0}^{k} \psi_{m,k,j} \Delta_h^m y(t_{k-j}), \tag{2.5}
$$

$$
{}_{0+}^{V2}D_t^{q(t_{k+1})} y(t_{k+1}) \approx \sum_{j=0}^{k} \frac{h^{-q(t_{j+1})}}{\Gamma(m+1-q(t_{j+1}))} \varphi_{m,j} \Delta_h^m y(t_{k-j}), \tag{2.6}
$$

$$ {}^{V3}_{0+}D_t^{q(t_{k+1})}y(t_{k+1}) \approx \sum_{j=0}^{k} \frac{h^{-q(t_{k-j})}}{\Gamma(m+1-q(t_{k-j}))}\lambda_{m,k,j}\Delta_h^m y(t_{k-j}), \tag{2.7} $$

where ${}^{V1}D_{t_k}^{q(t_k)}y(t_k)$ and ${}^{V2}D_{t_k}^{q(t_k)}y(t_k)$ are the approximations of the VO derivatives (2.2) and (2.3), respectively, and $t_{k+1} = (n+1)h$, where $h \approx 0$ is the uniform step size. The coefficients are $\psi_{m,k,j} = [(j+1)^{m-q(t_{k+1})} - j^{m-q(t_{k+1})}]$, $\varphi_{m,j} = [(j+1)^{m-q(t_{j+1})} - j^{m-q(t_{j+1})}]$, and $\lambda_{m,k,j} = [(j+1)^{m-q(t_{n-j})} - j^{m-q(t_{n-j})}]$. The operator $\Delta_h^m y(t_k)$ denotes the forward difference term,

$$ \Delta_h^m y(t_k) = \sum_{i=0}^{m}(-1)^i \binom{m}{i} y(t_{k+(m-i)h}). \tag{2.8} $$

Stability and convergence remarks: The approximations in Eqs (2.5)–(2.7) are history-dependent finite-difference (convolution) formulas. Under standard smoothness assumptions on $y(t)$ and boundedness of the order function $q(t)$, these discretizations are consistent, that is, the local truncation error vanishes as h approaches to 0. When they are embedded into a numerical scheme for initial value problems $D_t^{q(t)}y(t) = f(t, y(t))$, stability and convergence follow under the usual assumptions for fractional finite-difference forward methods. In particular, if f is Lipschitz in y, and the time step size h is sufficiently small, the resulting discrete solution remains bounded and converges to an approximated continuous-time solution over finite time intervals [4, 32].

3. Solution of variable-order (VO) derivatives

This section outlines the key steps involved to obtain an approximated solution of variable order derivatives V1, V2, and V3 in Eqs (2.5)–(2.7), respectively. Also, it offers insights into their practical implementation.

3.1. Solution of the V1 (time-memory) VO derivative

In Eq (2.5), the summation involves terms of the form $\Delta_h^m y(t_{k-j})$ multiplied by coefficients $\psi_{m,k,j}$. Expanding this expression leads to the determination of $y(t_{n+1})$ as follows:

$$ \begin{aligned} {}^{V1}_{0+}D_t^{q(t_{k+1})}y(t_{k+1}) \approx & \frac{h^{-q(t_{k+1})}}{\Gamma(m+1-q(t_{k+1}))} \\ & \times \left(\psi_{m,k,0}\Delta_h^m y(t_k) + \sum_{j=1}^{k} \psi_{m,k,j}\Delta_h^m y(t_{k-j})\right). \end{aligned} \tag{3.1} $$

Considering $m = 1$, then $\psi_{m,k,0} = 1$, and $\Delta_h^m y(t_k) = y(t_{k+1}) - y(t_k)$. Hence, the expression (3.1) becomes:

$$ \begin{aligned} {}^{V1}_{0+}D_t^{q(t_{k+1})}y(t_{k+1}) \approx & \frac{h^{-q(t_{k+1})}}{\Gamma(m+1-q(t_{k+1}))} \\ & \times \left(y(t_{k+1}) - y(t_k) + \sum_{j=1}^{k} \psi_{m,k,j}\Delta_h^m y(t_{k-j})\right). \end{aligned} \tag{3.2} $$

Finally, by solving $y(t_{k+1})$ in (3.2), the solution for the approximation of the V1 VO derivative is given as:

$$\begin{aligned} y(t_{k+1}) \approx & y(t_k) + h^{q(t_{k+1})}\Gamma(m+1-q(t_{k+1})) \times {}_{0+}^{V1}D_t^{q(t_{k+1})} y(t_{k+1}) \\ & - \sum_{j=1}^{k} \psi_{m,k,j} \Delta_h^m y(t_{k-j}). \end{aligned} \tag{3.3}$$

3.2. Solution of the V2 (weak order-memory) VO derivative

In Expression (2.6), the summation involves terms of the form $\Delta_h^m y(t_{k-j})$ multiplied by coefficients $\varphi_{m,j}$. Expanding this expression leads to the determination of $y(t_{k+1})$ as follows:

$$\begin{aligned} {}_{0+}^{V2}D_t^{q(t_{k+1})} y(t_{k+1}) \approx & \frac{h^{-q(t_1)}}{\Gamma(m+1-q(t_1))} \varphi_{m,0} \Delta_h^m y(t_k) \\ & + \sum_{j=1}^{k} \frac{h^{-q(t_{j+1})}}{\Gamma(m+1-q(t_{j+1}))} \varphi_{m,j} \Delta_h^m y(t_{k-j}). \end{aligned} \tag{3.4}$$

Considering $m = 1$, then $\varphi_{m,0} = 1$, and $\Delta_h^m y(t_j) = y(t_{j+1}) - y(t_j)$. Hence, the expression (3.4) becomes

$$\begin{aligned} {}_{0+}^{V2}D_t^{q(t_{k+1})} y(t_{k+1}) \approx & \frac{h^{-q(t_1)}}{\Gamma(m+1-q(t_1))} \Big(y(t_{k+1}) - y(t_k) \Big) \\ & + \sum_{j=1}^{k} \frac{h^{-q(t_{j+1})}}{\Gamma(m+1-q(t_{j+1}))} \varphi_{m,j} \Delta_h^m y(t_{k-j}). \end{aligned} \tag{3.5}$$

Finally, by solving $y(t_{k+1})$ in (3.5), the solution for the approximation of the V2 VO derivative is given as

$$\begin{aligned} y(t_{k+1}) \approx & y(t_k) + h^{q(t_1)}\Gamma(m+1-q(t_1)) \\ & \times \left({}_{0+}^{V2}D_t^{q(t_{k+1})} y(t_{k+1}) - \sum_{j=1}^{k} \frac{h^{-q(t_{j+1})}}{\Gamma(m+1-q(t_{j+1}))} \varphi_{m,j} \Delta_h^m y(t_{k-j}) \right). \end{aligned} \tag{3.6}$$

3.3. Solution of the V3 (strong order-memory) VO derivative

In expression (2.7), the summation involves terms of the form $\Delta_h^m y(t_{k-j})$ multiplied by coefficients $\lambda_{m,k,j}$. Expanding this expression leads to the determination of $y(t_{k+1})$ as follows:

$$\begin{aligned} {}_{0+}^{V3}D_t^{q(t_{k+1})} y(t_{k+1}) \approx & \frac{h^{-q(t_0)}}{\Gamma(m+1-q(t_0))} \lambda_{m,k,0} \Delta_h^m y(t_k) \\ & + \sum_{j=1}^{k} \frac{h^{-q(t_{k-j})}}{\Gamma(m+1-q(t_{k-j}))} \lambda_{m,k,j} \Delta_h^m y(t_{k-j}). \end{aligned} \tag{3.7}$$

Considering $m = 1$, then $\lambda_{m,k,0} = 1$ and $\Delta_h^m y(t_j) = y(t_{j+1}) - y(t_j)$. Hence, the expression (3.7) becomes:

$$\begin{aligned} {}_{0+}^{V3}D_t^{q(t_{k+1})} y(t_{k+1}) = & \frac{h^{-q(t_0)}}{\Gamma(m+1-q(t_0))} \Big(y(t_{k+1}) - y(t_k) \Big) \\ & + \sum_{j=1}^{k} \frac{h^{-q(t_{k-j})}}{\Gamma(m+1-q(t_{k-j}))} \lambda_{m,k,j} \Delta_h^m y(t_{k-j}). \end{aligned} \tag{3.8}$$

Finally, by solving $y(t_{k+1})$ in (3.8), the solution for the approximation of the V2 VO derivative is given as:

$$
\begin{aligned}
y(t_{k+1}) \approx & y(t_k) + h^{q(t_0)}\Gamma(m+1-q(t_0)) \\
& \times \left({}^{V3}_{0+}D_t^{q(t_{k+1})} y(t_{k+1}) - \sum_{j=1}^{k} \frac{h^{-q(t_{k-j})}}{\Gamma(m+1-q(t_{k-j}))} \lambda_{m,k,j} \Delta_h^m y(t_{k-j}) \right).
\end{aligned}
\tag{3.9}
$$

4. VOFD Python package

The VOFD Python package is a comprehensive toolkit for solving variable-order differential equations efficiently. The source code is available in the Git repository (`https://github.com/DClementeL/VOFD/tree/main/VOFD`), and the package can be installed directly from PyPI (`https://pypi.org/project/vofd`). The VOFD package provides several subroutines for approximating the V1-, V2-, and V3-type variable-order derivatives. The proposed package permits the study, analysis, modeling, and simulation of complex physical systems and engineering problems. The VOFD Python package also facilitates scientific exploration by streamlining a user-friendly workflow for numerical solutions. It therefore serves as a valuable tool for numerical analysis and scientific research. The main features of the package are given as follows:

1) **Modular structure for clarity:** The package adopts a modular design, allowing users to easily navigate and utilize specific functionalities for different types of variable order derivatives, V1, V2, and V3.
2) **Solid mathematical foundation:** The VOFD package is built on a solid mathematical foundation. We have conducted extensive numerical analyses of variable-order systems to demonstrate their usefulness and accuracy.
3) **Enhanced performance with Numba JIT and multiprocessing:** The package integrates Numba-based just-in-time compilation to accelerate numerical kernels at runtime. It also supports multiprocessing for parameter sweeps and data-intensive workflows, thereby reducing overall computation time [33].

4.1. Functional components of the package

Figure 1 shows the directory organization of the VOFD Python package. It is structured in two main files as follows: (i) the `vofd/` source directory, containing the VO numerical routines and utilities; and (ii) the `examples/` directory, which provides ready-to-run scripts covering typical simulation and bifurcation workflows. The package modules are described as follows:

- **vofd/__init__.py:** Initializes the package and discloses the main public functions (VO solvers, bifurcation routines, and I/O utilities).
- **vofd/maxima.py:** Provides routines for extracting local maxima from simulated trajectories, supporting signal processing and bifurcation diagram computation.
- **vofd/plot_data.py:** Delivers plotting utilities for time-evolution graphs, phase portraits, and bifurcation diagrams.
- **vofd/vo_core.py:** Implements the core routines of the numerical schemes for VO derivatives (V1, V2, V3).

- **vofd/sim_manager.py:** Provides high-level wrappers that interface the lower-level numerical routines (`vo_core.py`), enabling a simple user workflow without exposing internal implementation details. This module contains `v1_alg`, `v2_alg`, `v3_alg`, and their bifurcation variants `v1_bifurcation`, `v2_bifurcation`, and `v3_bifurcation`, respectively.
- **examples:** Collection of complete workflows. The example names follow a descriptive convention:
  - **Simulations:**
    `sim_vo_relaxation.py`,
    `sim_vo_riccati.py`,
    `sim_vo_chen.py`,
    `sim_5Dvo_Lorenz.py`.
  - **Bifurcation workflows:**
    `bif_vo_chen.py`,
    `bif_5Dvo_Lorenz.py`.
- **License:** Specifies the distribution of this package under the MIT License.
- **README.md:** Contains the package description, installation instructions, and usage guidance.
- **pyproject.toml:** Defines the package metadata, dependencies, and build configuration (compliant with PEP 621).

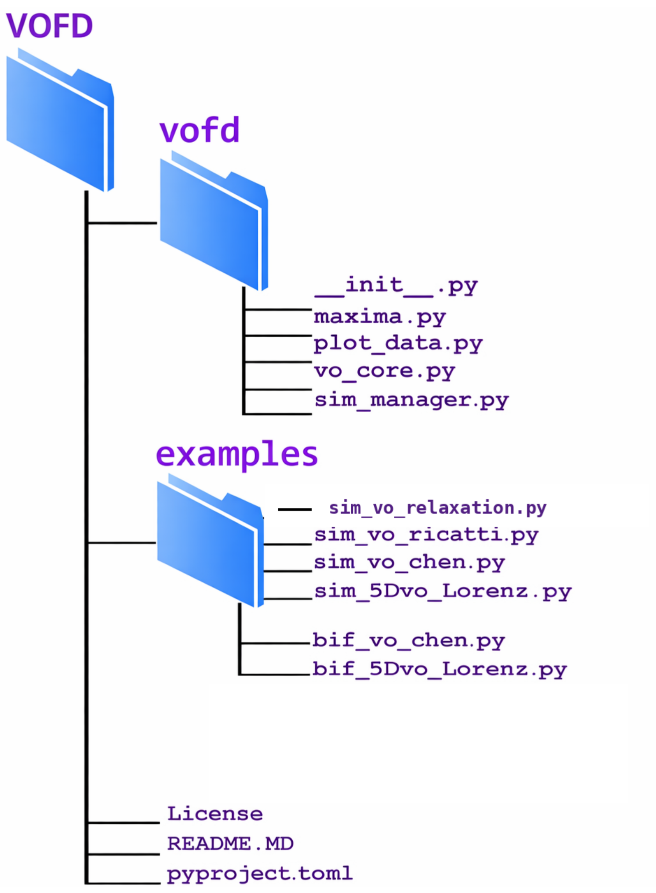


Figure 1. Directory organization of the VOFD Python package.

The modules described above operate together as a layered architecture that separates numerical computation, simulation control, data processing, and visualization. At the lowest level, `vo_core.py` implements the numerical schemes for variable-order derivatives, and `sim_manager.py` applies these schemes to user-defined dynamical systems. The plotting utilities in `plot_data.py` and `maxima.py` process the resulting time series to gather phase portraits and bifurcation diagrams, respectively. Finally, the scripts in the `examples/` directory include several examples to facilitate the understanding of the proposed package.

Indeed, the package offers a user-friendly, compact, and stable interface in which the user performs only the next steps: defines the dynamical system, selects a VO formulation (V1, V2, or V3), and calls helper routines to export and plot results. The functions below summarize this public workflow.

- **vo_system**: User-defined function specifying the right-hand side of the VO dynamical system to be evaluated.
- **vo_algorithm**: Numerical routine that contains the VO solver used for time-domain simulations. In `simulation_setup.py`, the options are `v1_alg`, `v2_alg`, and `v3_alg`; `bifurcation_setup.py` gives the options for bifurcation diagrams named `v1_bifurcation`, `v2_bifurcation`, and `v3_bifurcation`.
- **save_time_series**: Exports time-series results to text files (commonly used in time-domain simulation scripts).
- **save_pairwise_plots**: Generates and saves pairwise phase-plane figures such as y_1–y_2, y_1–y_3, and y_2–y_3 projections (used in time-domain simulation scripts).
- **save_bifurcation_data**: Stores bifurcation sweep results (parameter values and extracted maxima) into text files (used in bifurcation scripts).
- **create_xy_figure**: Unified plotting function used to deliver a single graph for time-evolution plots, phase portraits, and bifurcation diagrams.

In the following section, we demonstrate the use of the VOFD Python package to analyze variable-order dynamical systems.

4.2. Using the VOFD Python package to solve the variable-order relaxation system

The relaxation Eq (4.1) is one of the most elementary test problems in fractional calculus because it isolates the effect of the derivative operator itself without additional nonlinearities [34]. Hence, it is well-suited to examining how variable-order definitions (V1–V3) influence the decay behavior. Moreover, this linear model enables new users to apply the proposed package before moving on to more complex nonlinear problems. Let us express the VO relaxation equation as follows:

$$ {}_{0+}D_t^{q(t)} y(t) = -y(t), \tag{4.1}$$

with initial conditions $y(0) = 1$, $t \in [0, 5]$ and $q(t) = q_0 + (q_1 - q_0)\frac{t}{5}$ with $q_0 = 0,55$, $q_1 = 0.95$. The analytical solution of relaxation system (4.1) under constant order can be derived using the Mittag-Leffler function [31],

$$ y(t) = \sum_{k=0}^{\infty} \frac{-t^{\alpha_0 k}}{\Gamma(\alpha_0 k + 1)}. \tag{4.2}$$

We now solve the variable-order relaxation system using the VOFD Python package.

1) **Creating the main .py file:** A single Python file (e.g., `sim_vo_relaxation.py`) contains the full workflow: system definition, VO solver selection, time discretization, variable-order definition, solver invocation, and post-processing.
2) **Importing libraries:** The script imports NumPy for array handling and time discretization, and Numba-JIT to compile the system. From VOFD, the script imports the VO solvers (`v1_alg`, `v2_alg`, `v3_alg`) and the utilities `create_xy_figure` and `save_time_series`.

```
import numpy as np
from numba import jit
from vofd import v1_alg, v2_alg, v3_alg, create_xy_figure, save_time_series
```

3) **Defining the variable-order system:** The relaxation model is defined as a Numba-compatible function. The function takes the state variable `y` and returns the right-hand side of the relaxation equation.

```
@jit(nopython=True)
def Relaxation(y):
    Dq_y = -y
    return Dq_y
```

4) **Solver selection:** The user specifies the system function and selects the VO algorithm (V1, V2, or V3) by setting `vo_algorithm` accordingly.

```
vo_system    = Relaxation
vo_algorithm = v3_alg
```

5) **Time discretization and variable-order definition:** The simulation is performed on a uniform grid with step size h over total duration t_{sim}. The VO profile is evaluated over the same grid, and the initial condition is set to $y(0) = 1$.

```
h     = 0.01
t_sim = 5
Stime = np.arange(0, t_sim + h, h)
q0, q1 = 0.55, 0.95
q = q0 + (q1 - q0)*(Stime / t_sim)
y0 = np.array([[1.0]])
```

6) **Executing the VO algorithm and assembling `(t,y)`:** The solver is invoked using the user-defined system, the time-varying order `q`, the initial condition `y0`, and step size h. The output is stacked with the time vector to form the two-row array expected by the plotting/export utilities.

```
y = vo_algorithm(vo_system, q, y0, h)
y = np.vstack((Stime, y))
```

7) **Post-processing and outputs:** A descriptive name is constructed from the system and algorithm identifiers. The function `create_xy_figure` generates the time-evolution plot directly from the stacked `(t,y)` array. Optionally, the user may provide a plot specification dictionary (`spec`) to customize the figure style (labels, plot type, color, line thickness, grid, and output name). If `spec` is omitted, `create_xy_figure(xy)` uses default settings and still produces a valid plot. Finally, `save_time_series` exports the computed results using the same `(t,y)` array and the constructed system name.

```
system_name = f"{vo_system.name}_{vo_algorithm.name}"
```

```
plot_param = {
    "x_label": "Time(s)",
    "y_label": "y",
    "figure_name": system_name,
    "plot_type": "line",
    "color": "blue",
    "thickness": 1.4,
    "grid": False}
create_xy_figure(y, plot_param)
save_time_series(y, system_name)
```

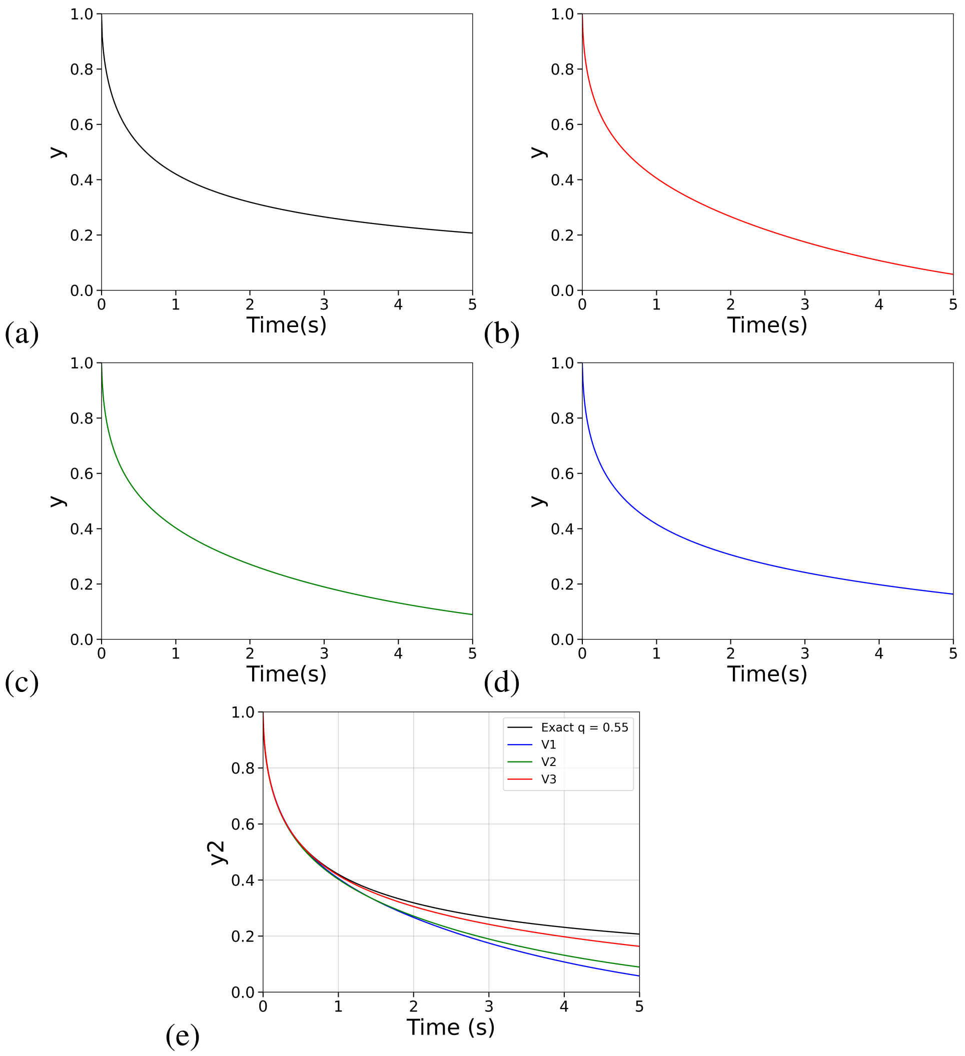


Figure 2. Simulation results of the variable-order relaxation system. (a) Analytical constant-order solution with $q(t) = 0.55$, (b), (c), and (d) Numerical variable-order solutions for $v1$, $v2$, and $v3$, respectively, with the variable order ramp $q(t) = q_0 + (q_1 - q_0)t/5$. (e) Comparison of the four previous results.

8) **Results:** By repeating the solver-selection step with `v1_alg`, `v2_alg`, and `v3_alg`, the solution of the relaxation system can be directly compared under the three VO formulations. The resulting

trajectories are shown in Figure 2. For this variable-order linear system, all solutions remain monotone-decreasing, which makes the relative effect of the VO definition easy to interpret: the decay rate depends on how each VO formulation weights time- and order-memory when the order is time varying. For an increasing ramp (from 0.55 to 0.95), the effective memory weakens as $q(t)$ approaches 1, so the solution tends to decay more rapidly at later times. To provide a constant-order reference, the relaxation system is first solved with $q(t) \equiv q_0 = 0.55$ using the closed-form Mittag-Leffler solution for the constant-order fractional relaxation equation [31]. The comparison shows that V1 exhibits the fastest decay (smallest $y(t)$), V3 the slowest (largest $y(t)$), and V2 an intermediate response. This behavior is consistent with the main distinction between V1 and the kernel-based formulations. V1 approximation weights the full history using the instantaneous value of $q(t)$ at the current time, which makes the response more sensitive to temporal variations of the order, whereas V2 and V3 embed the fractional order inside convolution kernels, leading to a smoother modulation of the decay as $q(t)$ evolves.

4.3. Using the VOFD Python package to solve the variable-order Riccati system

The Riccati equation appears in a wide range of applications in physics, engineering, and applied mathematics, including quantum mechanics, control theory, and finance [35–37]. For this reason, it is frequently adopted as a benchmark problem for assessing numerical methods in fractional calculus. In the present work, the particular motivation for selecting this system is to demonstrate that the package is capable of reproducing the well-established results with V1, V2, and V3 definitions of the variable-order fractional derivative, as originally analyzed in [4].

The variable order Riccati system is given by

$$ {}_{0^+}D_t^{q(t)} y(t) = -y(t)^2 + 2y(t) + 1, \tag{4.3}$$

subject to the initial condition $y(0) = 0$, with $t \in [0, 5]$ and $q(t) = 0.6 - 0.1 sin(\pi t)$.

We aim to solve the variable-order Riccati system using the VOFD Python package as follows:

1) **Creating the main `.py` file:** We start by creating a Python file (e.g., `sim_vo_ricatti.py`) that contains the complete setup: system definition, solver selection, time discretization, and variable-order.
2) **Importing libraries:** The script imports NumPy for numerical arrays and time discretization and Numba to JIT-compile the system function. From the VOFD package, the script imports a variable-order solver (e.g., `v1_alg`), a plotting utility (`create_xy_figure`), and a function to export the simulated trajectory (`save_time_series`).

```
import numpy as np
from numba import jit
from vofd import v1_alg, create_xy_figure, save_time_series
```

3) **Defining the variable-order system:** The Riccati model is defined as a Numba-compatible function. In the current version of the script, the function takes only the state variable `y` and returns the right-hand side of the Riccati dynamics.

```
@jit(nopython=True)
def Ricatti(y):
    Dq_y = -y*y + 2*y + 1
    return Dq_y
```

4) **Solver selection:** The user selects both the variable-order system function and the VO algorithm to be used. The current script is configured to solve Riccati using the V1 operator by setting `vo_algorithm = v1_alg`.

```
vo_system    = Ricatti
vo_algorithm = v1_alg
```

5) **Time discretization and variable-order definition:** The simulation is defined on a uniform time grid with step size h over a total duration t_{sim}. The variable order is then evaluated over this grid as $q(t) = 0.6 - 0.1\sin(\pi t)$, where t corresponds to the entries of `Stime`. The initial condition is set as $y(0) = 0$.

```
h     = 0.01
t_sim = 5
Stime = np.arange(0, t_sim + h, h)

q  = 0.6 - 0.1*np.sin(np.pi*Stime)
y0 = np.array([[0.0]])
```

6) **Executing the VO Algorithm and assembling `(t,y)`:** The selected VO solver is invoked using the user-defined system, the time-varying order `q`, the initial condition `y0`, and the step size h. The returned trajectory is then stacked together with the time vector to form a two-row array, where the first row is `Stime`, and the second row is $y(t)$. This `(t,y)` array is the input expected by the plotting and exporting utilities used in the current script.

```
y = vo_algorithm(vo_system, q, y0, h)
y = np.vstack((Stime, y))
```

7) **Post-processing and outputs:** Similar to the previous example, a descriptive name is constructed from the system and algorithm identifiers. The function `create_xy_figure` generates the time-evolution plot directly from the stacked `(t,y)` array, and `save_time_series` exports the computed results using the same `(t,y)` array and the constructed system name.

```
system_name = f"{vo_system.__name__}_{vo_algorithm.__name__}"

# Default plot (no styling dictionary required)
#create_xy_figure(y)
# Optional styling via 'spec'
plot_param = {
    "x_label": "Time(s)",
    "y_label": "y",
    "figure_name": system_name,
    "plot_type": "line",      # "line" or "scatter"
    "color": "blue",
    "thickness": 1.4,
    "grid": False
}
create_xy_figure(y, plot_param)
save_time_series(y, system_name)
```

8) **Results:** By repeating the solver-selection step with `v1_alg`, `v2_alg`, and `v3_alg`, the Riccati response can be compared under the three VO formulations. The resulting trajectories can

be observed in Figure 3, and they are consistent with the reference behaviors reported in [4]. From Figure 3, we observe that V3 produces the smoothest evolution, with essentially no visible perturbations. V2 yields a low-ripple response, and V1 exhibits the most pronounced oscillations, particularly for $t \in [2, 5]$. So, the instantaneous order $q(t)$ in the V1 formulation shapes the weighting of past states, which produces more sensitivity to temporal variations of the fractional order.

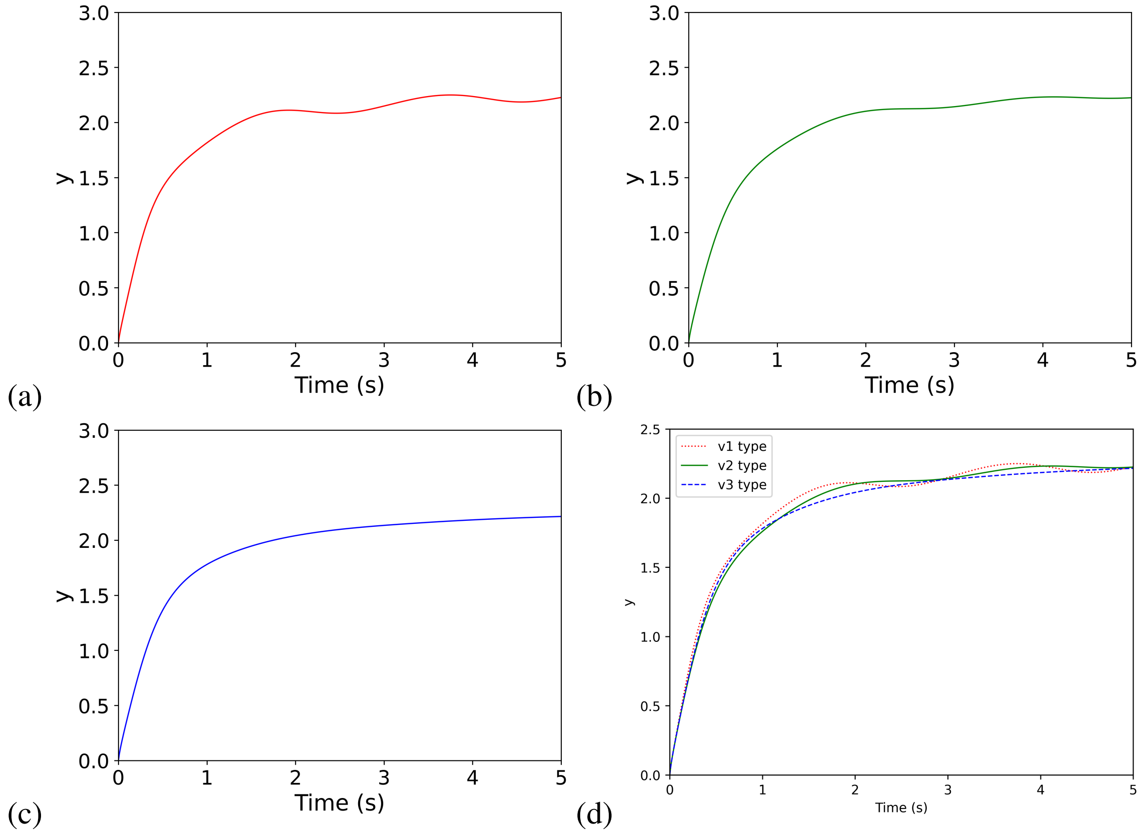


Figure 3. Simulation results of the fractional Riccati system for the three variable-order types: (a) V1, (b) V2, (c) V3. (d) Comparison of all three variable order derivatives.

4.4. Using the VOFD Python package to solve the variable-order chaotic Chen system

In this section, we introduce the variable order model of the chaotic Chen system given by

$$\begin{aligned}
{}_{0+}D_t^{q(t)} y_1(t) &= a(y_2(t) - y_1(t)),\\
{}_{0+}D_t^{q(t)} y_2(t) &= (c-a)y_1(t) - y_1(t)y_3(t) + cy_2(t),\\
{}_{0+}D_t^{q(t)} y_3(t) &= y_1(t)y_2(t) - by_3(t),
\end{aligned} \tag{4.4}$$

considering the parameters $a = 40$, $b = 3$, $c = 28$, and variable-order derivative $q(t) = 0.9 + 0.1 sin(\pi t)$. The procedure for solving the variable-order chaotic Chen system using the VOFD Python package is similar to that for the Riccati example. Herein, we define the simulation setup as follows:

```
import numpy as np
```

```
from numba import jit
from vofd import v1_alg, save_pairwise_plots, save_time_series
#---------------------------------------------------------
@jit(nopython=True)
def Chen_System(y):
        y1,y2,y3 = y
        a = 40
        b = 3
        c = 28
        Dqy1 = a*(y2-y1)
        Dqy2 = (c-a)*y1 - y1*y3 + c*y2
        Dqy3 = y1*y2 - b*y3
        return np.array([Dqy1,Dqy2,Dqy3])
#---------------------------------------------------------

# ========================================================
# 2. Solver selection
# ========================================================
vo_system        = Chen_System
vo_algorithm = v1_alg
# ========================================================
# 3. Time discretization
# ========================================================
h      = 0.005 # step size
t_sim = 30.0  # simulation time
Stime = np.arange(0, t_sim + h, h)        #Time span array
# ========================================================
# 4. Variable order definition & Initial conditions
# ========================================================

q      = 0.9 + 0.1*np.sin(np.pi*Stime)     #Variable order
y0     = np.array([ [1.0], [0.0], [1.0] ]) #Initial Conditions
# ========================================================
# 6. Solver invocation
# ========================================================

y   = vo_algorithm(vo_system,q,y0,h)
# ====================================================
# 7. Post-processing
# ====================================================
system_name = f"{vo_system.__name__}_{vo_algorithm.__name__}"
save_pairwise_plots(y,system_name)
save_time_series(y,system_name)
```

The simulation results for each variable-order derivative are depicted in Figure 4. The chaotic attractors of the Chen system are obtained using the proposed variable-order derivatives. Moreover, the dynamical variations observed across the three cases emphasize the influence of memory dependency on the chaotic dynamics of variable-order systems. To illustrate the different dynamics, additional bifurcation analysis is crucial.

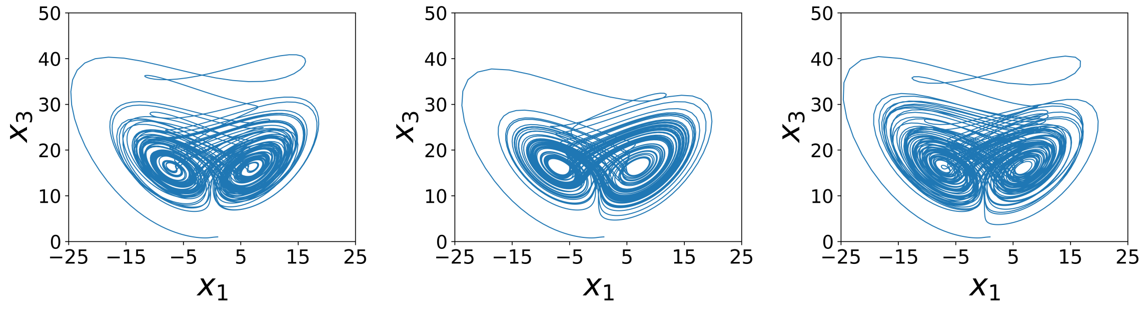


Figure 4. Simulation results of the variable order chaotic Chen system with the VOFD Python package.

4.5. Bifurcation diagrams of the variable-order chaotic Chen system

Bifurcation diagrams are essential tools for comprehending the behavior of dynamical systems. Their graphical representation enables researchers to identify critical points in the system behavior, including routes to chaos. Therefore, the VOFD Python package provides tools for generating bifurcation diagrams for the variable order types V1, V2, and V3.

1) **Creating the main py file:** We start by creating a new Python file, typically named `bif_vo_chen.py` (or `bifurcation_setup.py`) according to its functionality. This file serves as the main script for importing the required modules and generating bifurcation diagrams for the variable-order Chen system. This file should be saved in the VOFD working directory.
2) **Importing libraries:** The script imports `NumPy` for numerical operations and the `Numba` `@jit` decorator to accelerate the system equations. The VOFD utilities are then imported to perform the bifurcation sweep and export results. In particular, `v1_bifurcation` provides the numerical scheme for the VO derivative, `bifurcation_process` coordinates the parameter sweep, `create_xy_figure` generates the bifurcation plot, and `save_bifurcation_data` exports the computed data.

```
import numpy as np
from numba import jit
from vofd import (v1_bifurcation, bifurcation_process, create_xy_figure,
save_bifurcation_data)
```

3) **Defining the variable-order system:** We define the Chen system as a Python function that receives the state vector `y` and the bifurcation parameter (here named `bif_param`). Applying the Numba `@jit` decorator ensures compatibility with the package routines and reduces computation time by compiling the function to machine code. The argument `bif_param` corresponds to the active control parameter being swept during the bifurcation computation.

```
@jit(nopython=True)
def Chen_System(y,bif_param):
        y1,y2,y3 = y
        a = bif_param
        b = 3
        c = 28
        Dqy1 = a*(y2-y1)
        Dqy2 = (c-a)*y1 - y1*y3 + c*y2
        Dqy3 = y1*y2 - b*y3
        return np.array([Dqy1,Dqy2,Dqy3])
```

4) **Bifurcation setup:** We select the system and VO algorithm, define the numerical parameters, and construct the time array used throughout the simulation. The variable-order function `q(t)` is evaluated over the entire time domain, and the interval for the bifurcation parameter is established using a lower limit `L_inf`, an upper limit `L_sup`, and an increment `delta`. Each value in this interval is processed independently by the bifurcation routine.

```
vo_system    = Chen_System
vo_algorithm = v1_bifurcation
h     = 0.005
t_sim = 30.0
Stime = np.arange(0, t_sim + h, h)
q  = 0.9 + 0.1*np.sin(np.pi*Stime)
y0 = np.array([[1.0],[0.0],[1.0]])
L_inf = 40.0
L_sup = 60.0
delta = 0.1
```

5) **Generating the bifurcation diagram:** The bifurcation diagram is generated by calling the `bifurcation_process`, which applies the selected VO algorithm to the system for each value of the bifurcation parameter in the specified interval. This function is executed inside an `if __name__ == '__main__'` block to ensure correct behavior when using Python's multiprocessing backend, preventing the script from being relaunched as a subprocess on some operating systems.

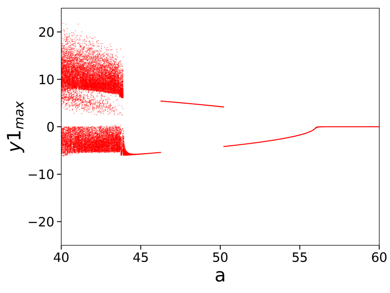
```
if __name__ == "__main__":
    X = bifurcation_process( vo_algorithm, vo_system, q, y0, h,
    delta, L_inf,L_sup)
```

6) **Post-processing and outputs:** After completing the bifurcation sweep, the results are plotted using `create_xy_figure`. In this workflow, providing a plot specification dictionary is essential: bifurcation diagrams require a `scatter` representation, whereas default plotting options are intended for time-series or phase-portrait style curves. Therefore, the `plot_param` (plot spec) must explicitly set `plot_type="scatter"` and provide axis labels (e.g., a vs. $y1_{\max}$) to ensure the bifurcation diagram is displayed as expected.

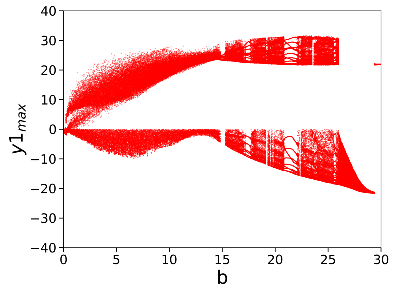
```
    system_name = f"{vo_system.__name__}_{vo_algorithm.__name__}"
    plot_param = {
        "x_label": "a",
        "y_label": "$y1_{max}$",
        "figure_name": system_name,
        "plot_type": "scatter",
        "color": "blue",
        "grid": False
        }
    create_xy_figure(X, plot_param)
    save_bifurcation_data(X, system_name)
```

7) **Results**: Figure 5 presents the obtained results for the variable-order Chen system. All graph information is detailed in accordance with the approach in [38]. In each subgraph, the vertical axis shows the local maxima (peak values) of $y_1(t)$ extracted from the post-transient portion of

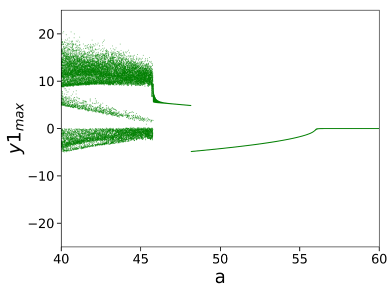
the numerical trajectory plotted against the swept parameter. The bifurcation diagrams reveal transitions from stationary or weakly oscillatory regimes toward bounded oscillatory dynamics, followed by chaotic behavior. In several parameter intervals, the system undergoes crisis-like global bifurcations, where chaotic attractors are destroyed or created abruptly as control parameters cross critical values. These transitions exhibit qualitative features reminiscent of Hopf-bifurcation-type behavior in the sense that oscillatory dynamics emerge from previously nonoscillatory regimes.



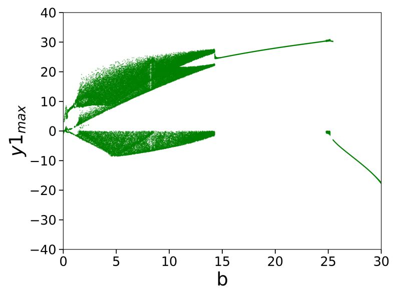
(a) Type V1: parameter a.



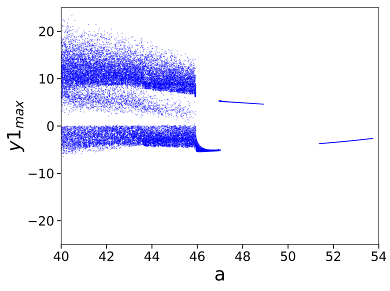
(b) Type V1: parameter b.



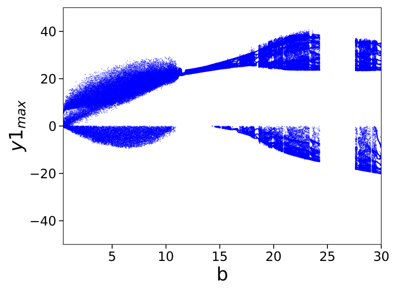
(c) Type V2: parameter a.



(d) Type V2: parameter b.



(e) Type V3: parameter a.



(f) Type V3: parameter b.

Figure 5. Bifurcation diagrams of the variable-order Chen system under the three VO definitions (V1, V2, and V3). For all cases, the variable order is $q(t) = 0.9 + 0.1\sin(\pi t)$, the integration step is $h = 0.005$, the simulation time is $t \in [0, 30]$, and the initial condition is $y(0) = [1,\ 0,\ 1]^{\mathsf{T}}$. (a), (c), (e) Chen's parameter a is swept over $a \in [40, 60]$ with increment $\Delta a = 0.01$ while keeping $b = 3$ and $c = 28$ fixed. (b), (d), (f) Chen's parameter b is swept over $b \in [0, 30]$ with increment $\Delta b = 0.01$ while keeping $a = 40$ and $c = 28$ fixed.

5. Accuracy and efficiency of numerical solutions

The accuracy of numerical solutions quantifies the deviation of a numerical solution from the exact solution, whereas the efficiency is evaluated in terms of the convergence order of the algorithm. The convergence order provides an understanding of how the errors in numerical solutions decrease as the computational resolution increases. When the exact solutions are not accessible, the accuracy and efficiency of numerical solutions can be realized with the mean absolute error (MAE*) and the experimental convergence order rate ($ECOR*$) [4].

Mean absolute error (MAE)*

The MAE* quantifies the average magnitude of errors in numerical solutions as follows:

- **Absolute error (AE^*)**: The absolute difference between the numerical solutions at two levels of discretization, usually involving halving the step size:

$$AE^* = |y_i^N - y_i^{2N}|, \tag{5.1}$$

  where y_i^N and y_i^{2N} represent the solutions at the ith point using step sizes h and $h/2$ corresponding to N and $2N$ points, respectively.
- **Mean absolute error (MAE^*)**:

$$MAE^* = \frac{1}{N}\sum_{i=1}^{N} AE^*. \tag{5.2}$$

  This represents the mean of all absolute errors AE^* across the N evaluated points.

Experimental convergence order rate ($ECOR*$)

The $ECOR*$ is the rate at which the error decreases as the step size is reduced. It is calculated as follows.

- **Convergence behavior:** Assuming that $h1$ and $h2$ are the step sizes used for MAE_1^* and MAE_2^*, respectively, and $h1 = kh2$, the ratio of the MAEs provides insight into the convergence behavior:

$$\frac{MAE_1^*}{MAE_2^*} = k^p, \tag{5.3}$$

  where $k \in \mathbb{R}^+$ and p is the order of convergence, indicating the rate at which errors decrease.
- **Experimental convergence order (ECO^*):** The ECOR is obtained by solving for p in expression 5.3 as follows:

$$ECOR^* = \log_k\left(\frac{MAE_1^*}{MAE_2^*}\right). \tag{5.4}$$

  This formula computes the convergence order p, providing a precise measure of how quickly errors are reduced as the step size is halved (if $k = 2$, then $ECOR^*$ evaluates the base-2 logarithm of the MAE ratio).

In Table 1 are shown the obtained results of MAE^* and $ECOR^*$ for the variable order Riccati system. We observe that MAE^* decreases as h decreases, indicating improved accuracy with finer step sizes. The $ECOR^*$ results for the V1 and V2 variable order types are 0.959 and 0.953, respectively, which are close to 1, indicating that the methods exhibit high convergence rates. On the other hand, the V3 variable-order type exhibits a lower convergence rate than the V1 and V2 types. Although it is still converging, the V3 method does not benefit as much from smaller step sizes.

Table 1. MAE*, ECO*, ECOR*.

Variable order type	Step size	MAE*	ECO*	ECOR*
V1	$h = \frac{5}{1000}$	1.429×10^{-4}	1.670	
	$h = \frac{5}{2000}$	7.348×10^{-5}	1.588	$\log_k \left(\frac{\mathrm{MAE}(h=\frac{5}{1000})}{\mathrm{MAE}(h=\frac{5}{2000})} \right) = 0.959$
V2	$h = \frac{5}{1000}$	1.009×10^{-4}	1.736	
	$h = \frac{5}{2000}$	5.210×10^{-5}	1.261	$\log_k \left(\frac{\mathrm{MAE}(h=\frac{5}{1000})}{\mathrm{MAE}(h=\frac{5}{2000})} \right) = 0.953$
V3	$h = \frac{5}{1000}$	1.290×10^{-4}	1.690	
	$h = \frac{5}{2000}$	8.269×10^{-5}	1.568	$\log_k \left(\frac{\mathrm{MAE}(h=\frac{5}{1000})}{\mathrm{MAE}(h=\frac{5}{2000})} \right) = 0.641$

6. Overview of the VOFD package performance

This section evaluates the computational efficiency and scalability of the VOFD Python package. Performance is analyzed from two complementary perspectives. First, we conduct a theoretical assessment of algorithmic complexity with respect to problem size. Later, an empirical throughput evaluation was conducted on representative PC and ARM-based platforms.

6.1. Computational complexity of the VOFD package

The computational complexity of the VOFD package is bounded by its most expensive numerical algorithm. In this manner, the V3 derivative (2.4) represents the highest computational workload due to its dependence on the time $(t - \tau)$. Consequently, all history-dependent contributions must be recomputed at each time step.

Let N denote the total number of discretization steps over the simulation interval. At time step k, the numerical evaluation of the V3 derivative involves the following:

1) Summation over k historical terms.
2) Evaluation of time-varying weights for each term.
3) A constant number of arithmetic operations per term.

Thus, the computational cost at a single time step scales linearly with k, that is:

$$C_k = O(k). \tag{6.1}$$

To compute the solution over the entire simulation window, these operations must be repeated for all time steps $k = 1, 2, \ldots, N$. Hence, the total computational cost is given by

$$C_{\text{total}} = \sum_{k=1}^{N} O(k), \tag{6.2}$$

yielding

$$C_{\text{total}} = O\left(\frac{N(N+1)}{2}\right) = O(N^2). \tag{6.3}$$

6.2. Performance metrics

Following the computational complexity analysis presented above, this subsection evaluates the practical execution performance of the VOFD Python package. The objective is to quantify the effective data-processing throughput, measured in megabits per second (Mbps), at a fixed problem scale. In this manner, the performance test is conducted by computing the solution of the variable-order Chen system 400 times, each with a distinct initial condition and for each variable-order type. For each run, the total amount of numerical data generated and the corresponding execution time are recorded. The throughput is then calculated as the ratio between the total data processed and the total execution time.

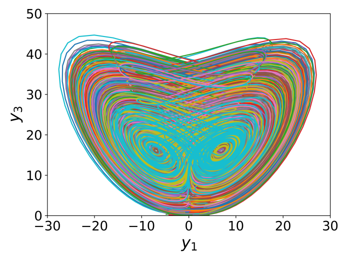
The evaluation setup includes the following computing platforms:

- **PC setup.** Equipped with a 64-bit 2.7 GHz processor with two cores, four logical processors, and 8 GB of RAM.
- **ARM SoC setup.** Employed a Raspberry Pi 4 equipped with a 64-bit 1.5 GHz quad-core processor and 8 GB of RAM.

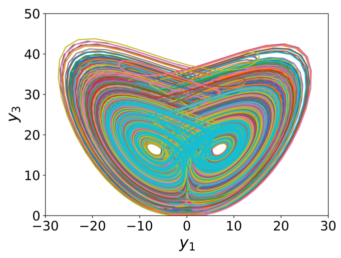
Figure 6 illustrates the phase portraits obtained during the benchmarking process, and the corresponding throughput values are reported in Table 2. To facilitate comparison across architectures operating at different clock frequencies, the throughput results are normalized to a reference clock rate of 1 GHz. Under the fixed problem configuration considered, similar throughput metrics are observed on both the PC and ARM-based platforms at their nominal clock rates. After normalization, the ARM-based SoC exhibits higher throughput per unit clock frequency for the same numerical workload, which is relevant for low-power computing scenarios.

Table 2. Throughput results of the VOFD Python package.

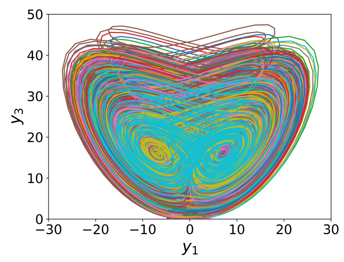
Variable order type	Platform	Clock (GHz)	Throughput (Mbps) at default clock rate	Throughput (Mbps) at 1 GHz clock rate
V1	PC	2.7	1.674	0.620
	Arm-based SoC	1.5	1.674	1.115
V2	PC	2.7	8.344	3.090
	Arm-based SoC	1.5	7.310	4.873
V3	PC	2.7	0.880	0.325
	Arm-based SoC	1.5	0.719	0.479



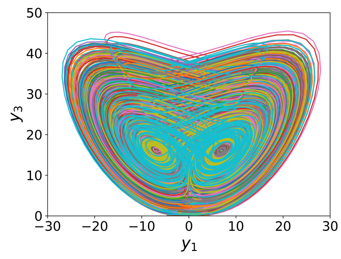
(a) Type V1, PC implementation.



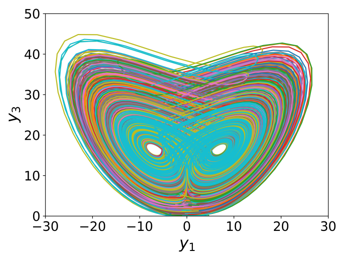
(b) Type V2, PC implementation.



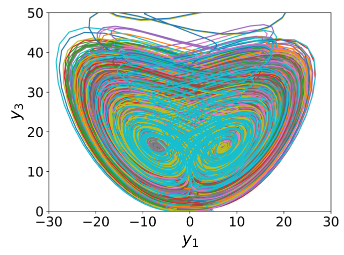
(c) Type V3, PC implementation.



(d) Type V1, ARM-based implementation.



(e) Type V2, ARM-based implementation.



(f) Type V3, ARM-based implementation.

Figure 6. Phase portraits depicting 400 solutions of the variable-order Chen system under different variable-order definition types using the VOFD Python package. Each colored trajectory corresponds to one solution obtained from a distinct initial condition.

7. Conclusions

The VOFD Python package provides a practical computational framework for modeling and simulating variable-order (VO) systems based on the three main derivative definitions V1, V2, and V3. By supporting these forms and offering optimized numerical routines through Numba JIT and multiprocessing, the package enables researchers to explore a wide range of dynamic behaviors with improved computational efficiency.

The numerical evaluation based on mean absolute error (MAE*) and experimental convergence order rate (ECOR*) demonstrated that the V1 and V2 types achieve high accuracy and convergence close to first order. In contrast, the V3 type exhibited a slightly lower convergence rate, which may be attributed to the increased complexity in its memory structure. These results provide guidance for users when selecting an appropriate VO formulation, depending on the desired trade-off between memory representation and numerical performance.

The computational performance analysis showed consistent behavior across both PC and ARM-based SoC platforms. With throughput measured in Mbps, the tests highlight the efficiency of the package's core algorithms, particularly when normalized to a 1 GHz clock rate. The ARM-based platform achieved slightly higher normalized throughput, suggesting that the implementation is suitable not only for traditional computing environments but also for embedded systems where energy efficiency and scalability are important.

Additionally, VOFD is implemented for vector-valued systems through the user-defined function interface, and it has been tested on higher-dimensional VO models. An illustrative script is provided in the examples directory of the Python package as bif_5Dvo_Lorenz.py. Thus, dimensionality is not a strict functional limitation. In this regard, the primary practical constraint is the computational burden imposed by history-dependent evaluations, which becomes more pronounced as simulation time increases. Due to VO finite-difference formulations being inherently history-dependent, the computational cost increases with the number of time steps and becomes more pronounced in large parameter sweeps for bifurcation analyses or long-time simulations. In chaotic regimes, small numerical differences may lead to different time evolutions; therefore, chaotic characterization should be supported by dynamical indicators.

Future work will extend the package by incorporating additional analysis tools such as Lyapunov exponent estimation and Poincaré map generation. These additions will enable a more comprehensive characterization of variable-order dynamics, including stability assessment and transitions among stationary, oscillatory, and chaotic behaviors while preserving compatibility with the existing VOFD simulation and bifurcation workflows.

Author contributions

Daniel Clemente-López: Methodology, software, validation, writing–original draft, writing–review & editing; Jesus M. Munoz-Pacheco: Conceptualization, investigation, methodology, software, supervision, validation, writing–original draft, writing–review & editing; José De Jesus Rangel-Magdaleno: Supervision, validation, writing–original draft, writing–review & editing; Lizbeth Vargas-Cabrera: Investigation, validation, writing–original draft, writing–review & editing. All authors have read and agreed to the published version of the manuscript.

Use of Generative-AI tools declaration

The authors declare they have not used Artificial Intelligence (AI) tools in the creation of this article.

Data availability

Data supporting the findings of this study are available in the following repositories. The source code is available on the git repository (`https://github.com/DClementeL/VOFD/tree/main/VOFD`), and the package can be installed directly from PyPI (`https://pypi.org/project/vofd`).

Acknowledgments

The last author thanks SECIHTI/Mexico for their support through the scholarship CVU 105600. The second author acknowledges BUAP for the support via the project VIEP 2025–2026.

Conflict of interest

Prof. Jesus M. Munoz-Pacheco is the Guest Editor of special issue “Recent Advances in Chaotic System and Applications” for AIMS Mathematics. Prof. Jesus M. Munoz-Pacheco was not involved in the editorial review and the decision to publish this article. The authors declare no conflicts of interest.

References

1. S. C. Ma, Y. F. Xu, W. Yue, Numerical solutions of a variable-order fractional financial system, *J. Appl. Math.*, **2012** (2012), 417942. https://doi.org/10.1155/2012/417942
2. S. G. Samko, B. Ross, Integration and differentiation to a variable fractional order, *Integr. Transf. Spec. F.*, **1** (1993), 277–300. https://doi.org/10.1080/10652469308819027
3. S. G. Samko, Fractional integration and differentiation of variable order, *Anal. Math.*, **21** (1995), 213–236. https://doi.org/10.1007/BF01911126
4. B. P. Moghaddam, J. T. Machado, Extended algorithms for approximating variable order fractional derivatives with applications, *J. Sci. Comput.*, **71** (2017), 1351–1374. https://doi.org/10.1007/s10915-016-0343-1
5. R. Garrappa, A. Giusti, F. Mainardi, Variable-order fractional calculus: A change of perspective, *Commun. Nonlinear Sci.*, **102** (2021), 105904. https://doi.org/10.1016/j.cnsns.2021.105904
6. L. E. S. Ramirez, C. F. M. Coimbra, On the selection and meaning of variable order operators for dynamic modeling, *Int. J. Differ. Equat.*, **2010** (2010), 846107. https://doi.org/10.1155/2010/846107
7. D. Sierociuk, W. Malesza, M. Macias, On a new definition of fractional variable-order derivative, *Proceedings of the 14th International Carpathian Control Conference (ICCC)*, Rytro, Poland, 2013, 340–345. 10.1109/CarpathianCC.2013.6560566
8. J. V. da C. Sousa, J. T. Machado, E. C. de Oliveira, The ψ-hilfer fractional calculus of variable order and its applications, *Comp. Appl. Math.*, **39** (2020), 296. https://doi.org/10.1007/s40314-020-01347-9
9. Z.-Y. Zhang, Z.-X. Lin, L.-L. Guo, Variable-order fractional derivative under hadamard's finite-part integral: Leibniz-type rule and its applications, *Nonlinear Dyn.*, **108** (2022), 1641–1653. https://doi.org/10.1007/s11071-022-07281-1
10. H. G. Sun, A. L. Chang, Y. Zhang, W. Chen, A review on variable-order fractional differential equations: mathematical foundations, physical models, numerical methods and applications, *Fract. Calc. Appl. Anal.*, **22** (2019), 27–59. https://doi.org/10.1515/fca-2019-0003
11. L. F. Ávalos-Ruiz, C. J. Zúñiga-Aguilar, J. F. Gómez-Aguilar, R. F. Escobar-Jiménez, H. M. Romero-Ugalde, FPGA implementation and control of chaotic systems involving the variable-order fractional operator with mittag–leffler law, *Chaos Soliton. Fract.*, **115** (2018), 177–189. https://doi.org/10.1016/j.chaos.2018.08.021
12. Z. C. Wei, A. Akgul, U. E. Kocamaz, I. Moroz, W. Zhang, Control, electronic circuit application and fractional-order analysis of hidden chaotic attractors in the self-exciting homopolar disc dynamo, *Chaos Soliton. Fract.*, **111** (2018), 157–168. https://doi.org/10.1016/j.chaos.2018.04.020
13. G.-C. Wu, Z.-G. Deng, D. Baleanu, D.-Q. Zeng, New variable-order fractional chaotic systems for fast image encryption, *Chaos*, **29** (2019), 083103. https://doi.org/10.1063/1.5096645
14. R. Kasinathan, R. Kasinathan, D. Chalishajar, D. Kasinathan, M. Rautaray, Trajectory controllability of hilfer fractional stochastic pantograph equations with random impulses, *Complex Anal. Oper. Theory*, **19** (2025), 176. https://doi.org/10.1007/s11785-025-01801-8

15. D. Chalishajar, D. Kasinathan, R. Kasinathan, R. Kasinathan, Trajectory controllability of higher-order riemann-liouville fractional stochastic systems via integral contractors in a new banach space, *B. Sci. Math.*, **204** (2025), 103659. https://doi.org/10.1016/j.bulsci.2025.103659

16. D. Kasinathan, R. Kasinathan, R. Kasinathan, K. Shanmugasundaram, D. Chalishajar, Trajectory controllability of fractional neutral stochastic dynamical systems of order $\alpha \epsilon (1, 2]$ with deviating argument, *J. Appl. Math.*, **2025** (2025), 2539363. https://doi.org/10.1155/jama/2539363

17. D. Kasinathan, D. Chalishajar, R. Kasinathan, R. Kasinathan, S. Karthikeyan, Trajectory controllability of higher-order fractional neutral stochastic system with non-instantaneous impulses via state-dependent delay with numerical simulation followed by hearth wall degradation process, *Int. J. Dynam. Control*, **13** (2025), 104. https://doi.org/10.1007/s40435-025-01605-w

18. D. Chalishajar, D. Kasinathan, R. Kasinathan, R. Kasinathan, Viscoelastic kelvin–voigt model on ulam-hyer's stability and t-controllability for a coupled integro fractional stochastic systems with integral boundary conditions via integral contractors, *Chao Soliton. Fract.*, **191** (2025), 115785. https://doi.org/10.1016/j.chaos.2024.115785

19. D. Chalishajar, D. Kasinathan, R. Kasinathan, R. Kasinathan, T-Controllability of higher-order fractional stochastic delay system via integral contractor, *Journal of Control and Decision*, **2024** (2024), 2379993. https://doi.org/10.1080/23307706.2024.2379993

20. R. Kasinathan, R. Kasinathan, D. Chalishajar, D. Baleanu, V. Sandrasekaran, The averaging principle of hilfer fractional stochastic pantograph equations with non-lipschitz conditions, *Stat. Probabil. Lett.*, **215** (2024), 110221. https://doi.org/10.1016/j.spl.2024.110221

21. J. W. Wang, Y. Q. Liu, L. M. Yan, K. L. Han, L. B. Feng, R. F. Zhang, Fractional sub-equation neural networks (fSENNs) method for exact solutions of space-time fractional partial differential equations, *Chaos*, **35** (2025), 043110. https://doi.org/10.1063/5.0259937

22. R. F. Zhang, S. Bilige, Bilinear neural network method to obtain the exact analytical solutions of nonlinear partial differential equations and its application to p-gBKP equation, *Nonlinear Dyn.*, **95** (2019), 3041–3048. https://doi.org/10.1007/s11071-018-04739-z

23. H. W. Zhang, R. F. Zhang, Q. R. Liu, A novel multi-modal neurosymbolic reasoning intelligent algorithm for BLMP equation, *Chinese Phys. Lett.*, **42** (2025), 100002. https://doi.org/10.1088/0256-307X/42/10/100002

24. R. F. Zhang, M. C. Li, Bilinear residual network method for solving the exactly explicit solutions of nonlinear evolution equations, *Nonlinear Dyn.*, **108** (2022), 521–531. https://doi.org/10.1007/s11071-022-07207-x

25. X. R. Xie, R. F. Zhang, Neural network-based symbolic calculation approach for solving the Korteweg–de vries equation, *Chaos Soliton. Fract.*, **194** (2025), 116232. https://doi.org/10.1016/j.chaos.2025.116232

26. M. S. Hashemi, M. Inc, A. Yusuf, On three-dimensional variable order time fractional chaotic system with nonsingular kernel, *Chaos Soliton. Fract.*, **133** (2020), 109628. https://doi.org/10.1016/j.chaos.2020.109628

27. M. F. Tolba, H. Saleh, B. Mohammad, M. Al-Qutayri, A. S. Elwakil, A. G. Radwan, Enhanced fpga realization of the fractional-order derivative and application to a variable-order chaotic system, *Nonlinear Dyn.*, **99** (2020), 3143–3154. https://doi.org/10.1007/s11071-019-05449-w

28. N. Medellín-Neri, J. M. Munoz-Pacheco, O. Félix-Beltrán, E. Zambrano-Serrano, Analysis of a variable-order multi-scroll chaotic system with different memory lengths, In: *Nonlinear dynamics and applications: Proceedings of the ICNDA 2022*, Cham: Springer, 2022, 1181–1191. https://doi.org/10.1007/978-3-030-99792-2_100

29. B. Wang, H. Jahanshahi, B. Arıcıoğlu, B. Boru, S. Kacar, N. D. Alotaibi, Variable-order fractional neural network: Dynamical properties, data security application, and synchronization using a novel control algorithm with a finite-time estimator, *J. Franklin I.*, **360** (2023), 13648–13670. https://doi.org/10.1016/j.jfranklin.2022.04.036

30. H. Ur Rehman, M. Darus, J. Salah, A note on caputo's derivative operator interpretation in economy, *J. Appl. Math.*, **2018** (2018), 1260240. https://doi.org/10.1155/2018/1260240

31. H. G. Sun, W. Chen, H. Wei, Y. Q. Chen, A comparative study of constant-order and variable-order fractional models in characterizing memory property of systems, *Eur. Phys. J. Spec. Top.*, **193** (2011), 185–192. https://doi.org/10.1140/epjst/e2011-01390-6

32. R. Lin, F. W. Liu, V. Anh, I. Turner, Stability and convergence of a new explicit finite-difference approximation for the variable-order nonlinear fractional diffusion equation, *Appl. Math. Comput.*, **212** (2009), 435–445. https://doi.org/10.1016/j.amc.2009.02.047

33. D. Clemente-López, J. M. Munoz-Pacheco, J. de J. Rangel-Magdaleno, Experimental validation of iot image encryption scheme based on a 5-d fractional hyperchaotic system and numba jit compiler, *Internet of Things*, **25** (2024), 101116. https://doi.org/10.1016/j.iot.2024.101116

34. A. Giusti, I. Colombaro, R. Garra, R. Garrappa, A. Mentrelli, On variable-order fractional linear viscoelasticity. *Fract. Calc. Appl. Anal.*, **27** (2024), 1564–1578. https://doi.org/10.1007/s13540-024-00288-y

35. L. Bougoffa, New conditions for obtaining the exact solutions of the general riccati equation, *Sci. World J.*, **2014** (2014), 401741. https://doi.org/10.1155/2014/401741

36. G. J. Taneja, V. K. Tayal, K. Pandey, Optimal control design for process liquid level system, *2021 9th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*, Noida, India, 2021, 1–4. https://doi.org/10.1109/ICRITO51393.2021.9596374

37. P. P. Boyle, W. Tian, F. Guan, The riccati equation in mathematical finance, *J. Symb. Comput.*, **33** (2002), 343–355. https://doi.org/10.1006/jsco.2001.0508

38. J. Wang, Z. Li, The impact of standard wiener process on the qualitative analysis and traveling wave solutions of stochastic nonlinear kodama equation in the Stratonovich sense, *AIMS Mathematics*, **10** (2025), 24997–25010. https://doi.org/10.3934/math.20251107