Volume 4, Issue 10, October 2026 ISSN (E): 2938-3803

A SOFTWARE TOOL FOR SOLVING A FRACTIONAL-ORDER DIFFERENTIAL EQUATION: A NUMERICAL EXPERIMENT BASED ON THE MITTAG-LEFFLER FUNCTION AND THE EULER METHOD

Ibragimov Samandar Iskandar o'g'li
Associate Professor of the Department of "Applied Mathematics and Programming Technologies", Bukhara State University

Gaybullayeva Saodat Ulugbekovna
2nd-Year Student of the "Applied Mathematics" Study Program, Bukhara State University

Abstract
A software tool has been developed for the simplest linear equation with a Caputo fractional-order derivative, $CD^{\alpha}y = -\lambda y$. In a window based on Python and Tkinter, the user enters the values of α, λ and T, and the program computes the exact solution via the Mittag-Leffler series and plots its graph. Together with the tool, the error of the fractional-order Euler method was measured as a function of the step h. For α = 0.5, 0.7 and 0.9 the method converges with first order; for α = 0.3 the order of the maximum error remains around 0.55, because the error accumulates near the initial point. If the step is large, the explicit scheme becomes unstable: for α = 0.5, starting from $\lambda h^{\alpha}/\Gamma(\alpha+1) \approx 1.4$ the solution grew without bound. It is shown that summing the series in ordinary floating-point arithmetic is unreliable for large λt^{α} and that the program warns about this. The computations were performed with Python 3.12, NumPy and mpmath; the codes are included in the article.

Keywords: Fractional-order derivative, Caputo derivative, Mittag-Leffler function, Euler method, order of convergence, stability, Python, Tkinter.

Introduction

The term "derivative" (in Uzbek, "hosila") is a literal translation into Russian of the French word "dérivée"; this term was introduced by J. Lagrange in 1797. An integer-order derivative gives only the instantaneous rate of change of a process at the given moment. In many real processes, however, the present state also depends on the past: examples are the seepage of a fluid in a porous medium, the gradual softening of a viscoelastic material, and anomalous diffusion in a complex medium. One way to introduce such memory into an equation is to pass from an integer derivative order to a fractional one. The first attempts to define fractional-order derivatives. Although the subject of fractional-order derivatives and integrals has a long history, it can be regarded as a modern object of research. In fact, Leibniz (1695–1697) and Euler wrote about this in their books. Applied studies, however, have multiplied mainly in recent decades.

What makes fractional-order equations difficult is this very memory, i.e. to find a new value one has to store the entire previous history and reuse it at every step, so the computational cost grows quadratically with the number of steps. An exact solution can be found only in the simplest cases; often there is no choice but to turn to numerical methods. Karimov [1] implemented in Python

Licensed under a Creative Commons Attribution 4.0 International License.

Web of Humanities: Journal of Social Science and Humanitarian Research

Volume 4, Issue 10, October 2026 ISSN (E): 2938-3803

an algorithm based on the Euler method for fractional-order ordinary differential equations and showed that, when the step is reduced, the numerical solution approaches the exact one. Tomášek [2], in his 2023 paper "On Euler Methods for Caputo Fractional Differential Equations", analyzed forward and backward Euler methods for Caputo equations; according to his conclusion, the full history of the solution must be taken into account when computing a new value, and simplified methods that discard part of the history are not always reliable. Khan and co-authors, in "Euler's Numerical Method on Fractional DSEK Model under ABC Derivative" [3], constructed a DSEK model based on the Atangana–Baleanu–Caputo derivative and studied its solutions with the Euler method. In 2026, Daniel Clemente-López and co-authors [4] created the VOFD Python package for variable-order derivatives, and Adams [5], comparing the Riemann–Liouville, Grünwald–Letnikov and GLI algorithms in the differint package, found that the Riemann–Liouville algorithm gave a smaller error in most tests.

Most of the works reviewed provide a ready-made package or a complex model. From the standpoint of a first experiment and educational purposes, however, another question remains open: if we create a simple tool in which parameters are entered in a window and the graph of the solution can be seen immediately, where is its computation reliable and where is it not? In the simplest equation with a known exact solution, how does the Euler method behave depending on α, and when does it become unstable? The aim of the article is to answer these questions with numbers. For this purpose, we create a windowed tool based on Python and Tkinter, reliably compute the Mittag-Leffler reference, measure the error, order of convergence and stability of the fractional-order Euler method, and estimate the computational cost.

Model and exact solution

We consider the following initial value problem:

$$ {}^{C}D^{\alpha}y(t) = -\lambda y(t), \quad y(0) = y_0, \quad 0 < \alpha \le 1, \quad \lambda > 0. \tag{1} $$

Here the Caputo derivative

$$ {}^{C}D^{\alpha}y(t) = (1/\Gamma(1-\alpha)) \int_0^t (t-s)^{-\alpha} y'(s)\, ds \tag{2} $$

is defined by the formula; for $\alpha = 1$ it reduces to the usual $y'(t)$. Using the Laplace transform, the solution of problem (1) is found:

$$ y(t) = y_0 E_{\alpha}(-\lambda t^{\alpha}), \quad E_{\alpha}(z) = \Sigma_{k=0}^{\infty} z^k / \Gamma(\alpha k+1). \tag{3} $$

This is the Mittag-Leffler function. Two cases are particularly convenient for us. For $\alpha = 1$, $E_1(z) = e^z$, i.e. the solution turns out to be the classical $y_0 e^{-\lambda t}$. For $\alpha = 1/2$, there is a closed formula: $E_{1/2}(-z) = e^{z^2}\,\mathrm{erfc}(z)$. Both are suitable for testing the computation code. One more important point: for $0 < \alpha < 1$, at large t the function $E_{\alpha}(-\lambda t^{\alpha})$ decays not exponentially but by a power law, approximately like $t^{-\alpha}/\lambda$ [6]. This is where the "memory" of the fractional-order model shows itself.

Software tool

The conceptual model of the program is shown in Figure 1. The model consists of eight stages, which are connected with the above description as follows. The first three stages (the problem, the model assumptions, the mathematical model) correspond to formulas (1)–(2). In the fourth stage, solution (3) is obtained through the Laplace transform. In the fifth stage, the time grid $t_i = i \cdot h$ ($h = 0.1$) is constructed, the Mittag-Leffler series is summed until its terms become

Licensed under a Creative Commons Attribution 4.0 International License.

Volume 4, Issue 10, October 2026 ISSN (E): 2938-3803

sufficiently small, and the values $y(t_i)$ are stored. The sixth stage is devoted to studying the influence of α, λ, T and h. In the seventh stage, the analytical and numerical solutions are compared, and the absolute and relative errors are estimated. The eighth stage presents the result in the form of graphs and interpretation; the dashed arrows indicate that the model parameters can be re-selected depending on the result. In our work, the role of the numerical solution in the seventh stage is played by the Euler method.

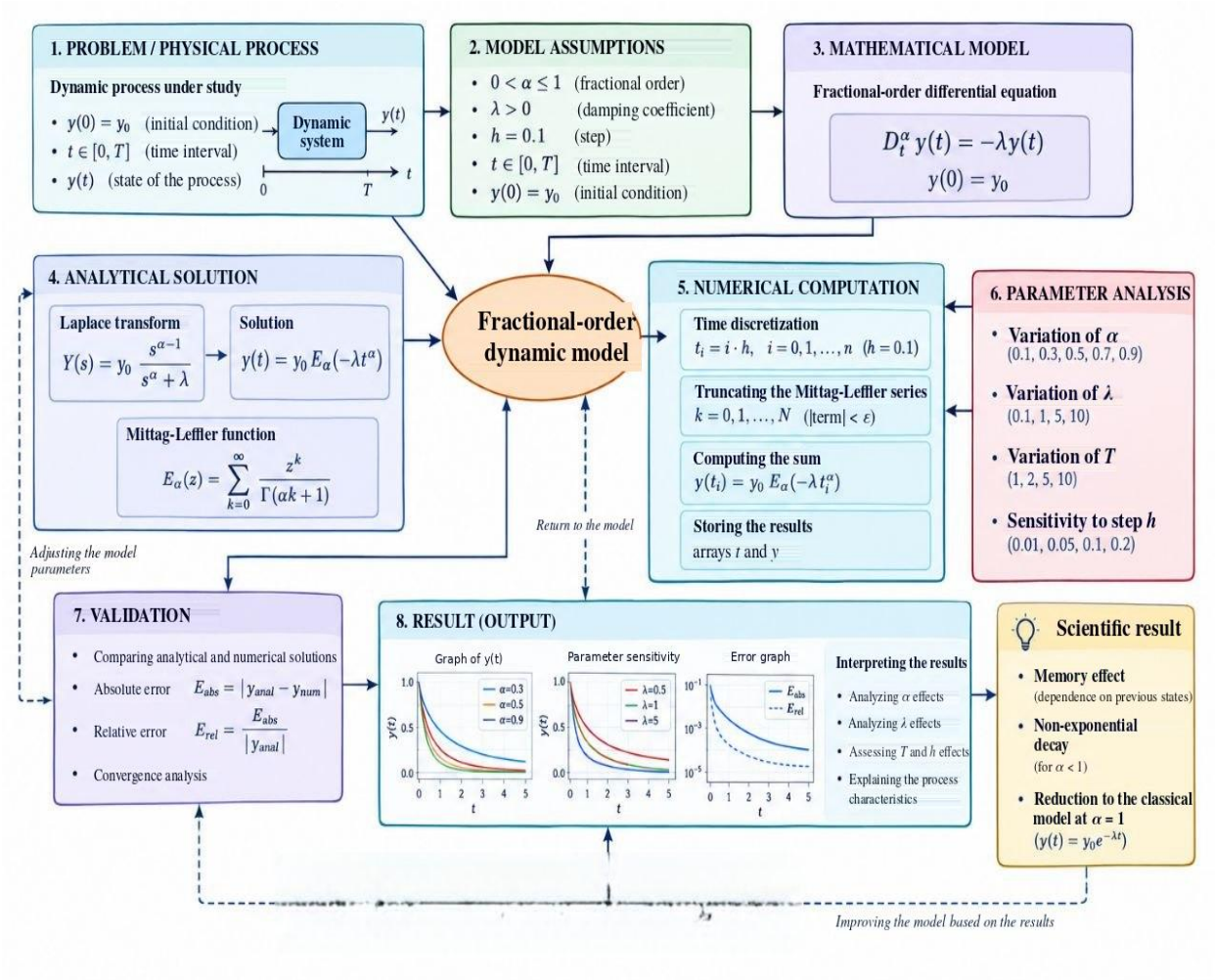


Figure 1. Conceptual model of computing fractional-order derivatives and analyzing the results with a Python program

The tool is created in the form of a Tkinter window (Figure 2). The user enters three values: α ($0 < \alpha \leq 1$), λ and T. When the "Calculate and plot" button is pressed, the program first checks the entered values: if α does not fall within the interval, or λ and T are not positive, it displays a message. Then, on the grid from t = 0 to T with step h = 0.1, it computes y(t) via the Mittag-Leffler series and plots the graph (Figure 3). The initial values are α = 0.5, λ = 1, T = 5. Here the graph gives the values 0.7236, 0.4276 and 0.2323 at t = 0.1, 1 and 5, respectively, which agree to four decimal places with the reference values in Table 2. The program code is given at the end of the article (Listing 1).

Licensed under a Creative Commons Attribution 4.0 International License.

Volume 4, Issue 10, October 2026 ISSN (E): 2938-3803

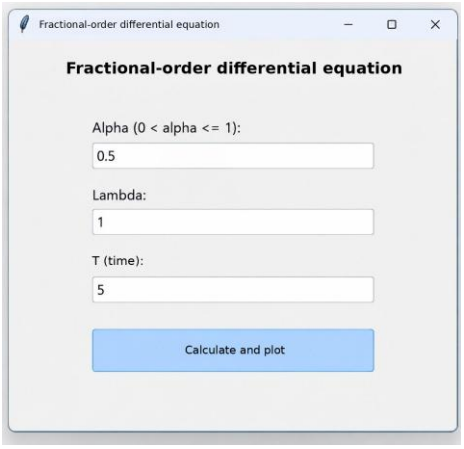


Figure 2. Program window (interface)

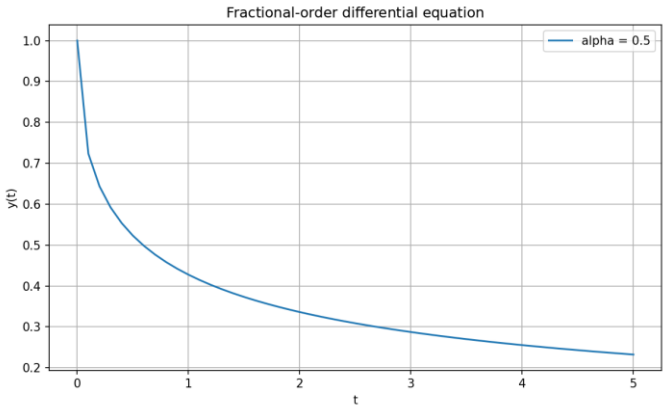


Figure 3. Graph obtained from the window (α = 0.5, λ = 1, T = 5)

The window sums the series in ordinary floating-point numbers. When the parameters are small this is sufficient, but when λt^{α} grows, the alternating-sign terms cancel each other and the result is corrupted. For example, for α = 1, λ = 10, at the point t = 5 the series gives the value $-5.2 \cdot 10^{4}$, whereas the exact value is $1.9 \cdot 10^{-22}$. Therefore the program monitors the size of the largest term: if the estimated loss-of-accuracy error exceeds 10^{-8} or a term exceeds the range of the double type, it gives the user a warning. In the cases α = 1, λ = 10 and α = 0.5, λ = 5 (T = 5) the warning was indeed issued; for α = 0.3 and 0.1 (λ = 1) it was not issued, and the result agreed with the reference to within 10^{-4}.

Computational methods

Reference solution. We return once more to the question of truncating the Mittag-Leffler series, since the reference must be reliable. In our initial approach, the series was truncated at 100 terms and stopped when the last term became smaller than 10^{-10}. We compared this with a high-precision computation (mpmath); the results are in Table 1. In most cases the difference is around 10^{-11}, i.e. as expected. But for α = 0.1 and z = −1, 100 terms were not enough: the series had not yet converged, the error came out as $1.5 \cdot 10^{-7}$, and the program gave no warning at all. The reason is simple: when α is small, Γ(αk+1) grows very slowly, and the terms of the series also decrease slowly.

Table 1. Truncation of the Mittag-Leffler series at 100 terms compared with a high-precision computation (reference)

α	z	Computed terms	100-term code	Reference	Absolute difference
0.5	−2.0	51	0.2553956763	0.2553956763	$2.1 \cdot 10^{-11}$
0.2	−1.0	67	0.4711006890	0.4711006889	$3.5 \cdot 10^{-11}$
0.1	−1.0	100	0.4855643104	0.4855644643	$1.5 \cdot 10^{-7}$
0.1	−0.5	32	0.6543244603	0.6543244603	$2.1 \cdot 10^{-11}$

Licensed under a Creative Commons Attribution 4.0 International License.

Volume 4, Issue 10, October 2026 ISSN (E): 2938-3803

That is why we used the mpmath variant as the reference (Listing 2). In it, the number of digits is increased starting from 50 depending on |z|, the series continues until five consecutive terms are smaller than 10^{-45}, and it stops with an error after 20,000 terms. We tested the reference in two cases. For α = 1, at 501 points the difference from e^{-t} did not exceed $1.1 \cdot 10^{-16}$. For α = 1/2 and z = 1.3, it agreed to 15 digits with $E_{1/2}(-1.3) = e^{1.69}\,\mathrm{erfc}(1.3) = 0.357642669086090$.

Euler scheme

We reduce problem (1) to a Volterra equation: $y(t) = y_0 + (1/\Gamma(\alpha)) \int_0^t (t-s)^{\alpha-1} f(y(s))\,ds$, where $f(y) = -\lambda y$. Dividing the interval [0, T] with step h = T/N, on each interval $[t_j, t_{j+1}]$ we replace f by its value at the left end, and integrate the kernel exactly. As a result

$$y_n = y_0 + (h^\alpha/\Gamma(\alpha+1)) \sum_{j=0}^{n-1} b_{n-1-j} f(y_j), \quad b_k = (k+1)^\alpha - k^\alpha. \tag{4}$$

For α = 1, $b_k = 1$ and (4) turns into the usual formula $y_n = y_{n-1} + h\, f(y_{n-1})$. The sum in (4) lengthens with n: to find y_n, all previous $f(y_j)$ are needed. Therefore the total number of arithmetic operations is proportional to N^2.

Experimental procedure

In the first experiment, $y_0 = 1$, λ = 1, T = 5 were taken. For the order of convergence, α = 0.3, 0.5, 0.7, 0.9 were considered, and α = 1 as well for verification. The number of steps was increased from N = 25 to 1600 by doubling (from h = 0.2 to 0.003125). The error was measured in two ways: the maximum absolute error over the whole grid, $\max|y_n - y(t_n)|$, and the error at the last point t = T. The order of convergence was estimated from the ratio of errors at two consecutive steps, $p = \log_2(e_h/e_{h/2})$. In the second experiment, for α = 0.5, λ = 0.1, 1, 5, 10 and N = 50, 200, 800 were taken. The computations were performed with Python 3.12.3, NumPy 2.4.4 and mpmath 1.3.0, in a Linux environment with a single processor core.

Results

How the parameters change the solution

As can be seen from Figure 4 (a) and Table 2, as α decreases the solution first drops faster and then slows down. At t = 0.1, y = 0.6321 for α = 0.3, and 0.9048 for α = 1. The curves intersect at approximately t ≈ 0.7–0.8, after which the order is reversed: at t = 5, the solution for α = 0.3 is 0.3372, while the classical solution is 0.0067, i.e. about 50 times smaller. The process does not decay as quickly as in the classical case and leaves a long "tail". As α approaches one, the solution passes continuously into the classical e^{-t}: the difference between α = 0.9 and α = 1 at t = 1 is only 0.008. λ (Figure 4, b) controls the decay rate: for α = 0.5, the value at t = 5 is 0.0503 for λ = 5 and 0.0252 for λ = 10, i.e. when λ doubles, the value decreases approximately twofold. This corresponds to the power-law decay of the form $t^{-\alpha}/\lambda$ mentioned above.

Licensed under a Creative Commons Attribution 4.0 International License.

Volume 4, Issue 10, October 2026 ISSN (E): 2938-3803

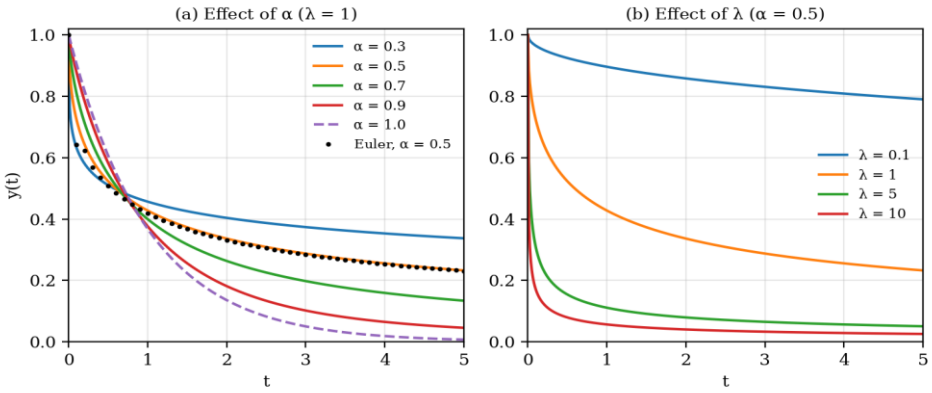


Figure 4. $y(t) = E_\alpha(-\lambda t^\alpha)$ ($y_0 = 1$): (a) different α, black dots — Euler method for α = 0.5 (h = 0.1); (b) different λ

Table 2. Solution values $y(t) = E_\alpha(-t^\alpha)$ (λ = 1)

α	t = 0.1	t = 0.5	t = 1	t = 2	t = 5
0.3	0.6321	0.5104	0.4566	0.4037	0.3372
0.5	0.7236	0.5232	0.4276	0.3362	0.2323
0.7	0.8092	0.5458	0.3996	0.2632	0.1337
0.9	0.8781	0.5826	0.3761	0.1811	0.0452
1.0	0.9048	0.6065	0.3679	0.1353	0.0067

Accuracy of the Euler method. For α = 1 the method turns into the usual Euler method and gives the expected result. When the step was halved, the error also decreased approximately twofold ($1.9 \cdot 10^{-2}$ at h = 0.1, $1.2 \cdot 10^{-3}$ at h = 0.00625). The fractional-order cases are in Table 3 and Figure 5. For α = 0.5, 0.7 and 0.9 the order at the last step equals 0.98, 1.02 and 1.00, respectively, i.e. the method is first-order. For α = 0.7 the order was around 1.3 in the initial steps and gradually fell to 1. For α = 0.3 the picture is different: the order grew very slowly from 0.49 to 0.55. Even when we reduced the step 64-fold, the maximum error decreased only 9-fold (from 0.268 to 0.030).

Table 3. Maximum absolute error of the Euler method $\max|y_n - y(t_n)|$ (T = 5, λ = 1)

N	h	α = 0.3	α = 0.5	α = 0.7	α = 0.9
25	0.2	$2.7 \cdot 10^{-1}$	$1.5 \cdot 10^{-1}$	$7.1 \cdot 10^{-2}$	$3.9 \cdot 10^{-2}$
50	0.1	$1.9 \cdot 10^{-1}$	$8.0 \cdot 10^{-2}$	$2.9 \cdot 10^{-2}$	$1.8 \cdot 10^{-2}$
100	0.05	$1.3 \cdot 10^{-1}$	$4.3 \cdot 10^{-2}$	$1.1 \cdot 10^{-2}$	$8.8 \cdot 10^{-3}$
200	0.025	$9.3 \cdot 10^{-2}$	$2.2 \cdot 10^{-2}$	$4.8 \cdot 10^{-3}$	$4.3 \cdot 10^{-3}$
400	0.0125	$6.4 \cdot 10^{-2}$	$1.2 \cdot 10^{-2}$	$2.3 \cdot 10^{-3}$	$2.2 \cdot 10^{-3}$
800	0.00625	$4.4 \cdot 10^{-2}$	$5.9 \cdot 10^{-3}$	$1.1 \cdot 10^{-3}$	$1.1 \cdot 10^{-3}$
1600	0.003125	$3.0 \cdot 10^{-2}$	$3.0 \cdot 10^{-3}$	$5.5 \cdot 10^{-4}$	$5.3 \cdot 10^{-4}$
Order p		0.55	0.98	1.02	1.00
Order p, at t = T		1.02	1.00	1.00	1.00

Web of Humanities: Journal of Social Science and Humanitarian Research

Licensed under a Creative Commons Attribution 4.0 International License.

Volume 4, Issue 10, October 2026 ISSN (E): 2938-3803

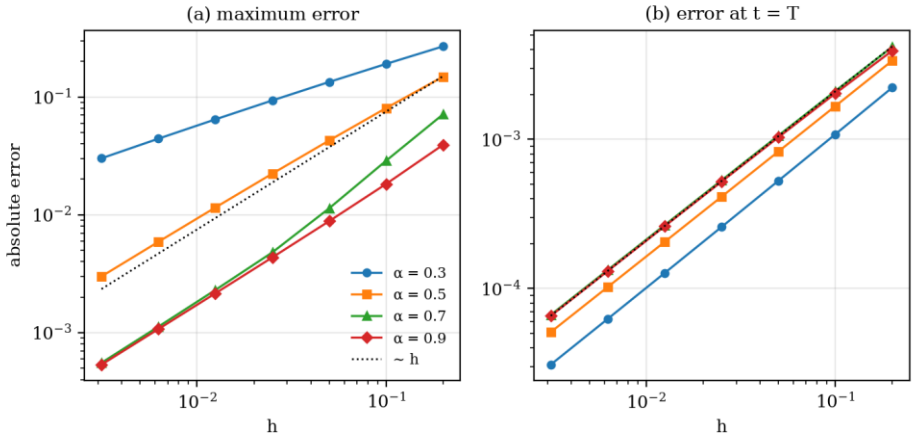


Figure 5. Dependence of the error on the step (log–log scale): (a) maximum error, (b) error at t = T. The dotted line is first-order growth ~ h

The reason lies in where the error accumulates. For α = 0.3 and 0.5, the maximum error always occurs at the first node, at t = h. For example, for N = 800 and α = 0.3 it equals 0.044, which is about 5.5 percent of the solution value (0.801). Near zero the solution changes close to $1 - \lambda t^{\alpha}/\Gamma(\alpha+1)$, and its derivative is infinite at t = 0; setting f equal to $f(y_0)$ in the first step cannot capture this sharp drop. This explanation is our conjecture: we did not test it in a separate experiment (for example, by refining the grid near zero). For α = 0.7 the maximum is at t ≈ 0.22, and for α = 0.9 at t ≈ 0.74 (N = 1600).

At t = T the picture is completely different. As the bottom row of Table 3 and Figure 5 (b) show, the order is close to 1 for all α (1.02, 1.00, 1.00, 1.00). Thus, far from the initial layer, the method works with first order, almost indifferent to α.

Stability

The second experiment (Table 4) showed something unexpected: if the step is not reduced as λ increases, the explicit scheme completely destroys the solution. It is convenient to explain this through the combination $q = \lambda h^{\alpha}/\Gamma(\alpha+1)$ (in parentheses in the table). Up to q ≈ 0.9 the solution is positive and bounded, but the error is large, 0.38 (the same for λ = 5 and λ = 10, because q is the same). At q = 1.78, however, the solution changed sign and grew from 10^7 to 10^{31}. In a separate scan (200 steps, λ = 1, choosing h from the given q), for α = 0.5 the solution began to oscillate from q ≥ 1, and from q ≈ 1.4 it grew without bound. For α = 0.9 the growth began at q = 2, and for α = 1 (the usual Euler method) at q > 2. These are our observations, not a theoretical proof; nevertheless, the practical meaning is clear: if λ is large, h must be chosen so that $q \ll 1$.

Table 4. Maximum error for α = 0.5 and (in parentheses) $q = \lambda h^{\alpha}/\Gamma(\alpha+1)$, T = 5

λ	N = 50 (h = 0.1)	N = 200 (h = 0.025)	N = 800 (h = 0.00625)
0.1	$9.8\cdot10^{-4}$ (0.04)	$2.5\cdot10^{-4}$ (0.02)	$6.2\cdot10^{-5}$ (0.01)
1.0	$8.0\cdot10^{-2}$ (0.36)	$2.2\cdot10^{-2}$ (0.18)	$5.9\cdot10^{-3}$ (0.09)
5.0	$7.0\cdot10^{7}$ (1.78)	$3.8\cdot10^{-1}$ (0.89)	$1.2\cdot10^{-1}$ (0.45)
10.0	$1.4\cdot10^{25}$ (3.57)	$7.1\cdot10^{31}$ (1.78)	$3.8\cdot10^{-1}$ (0.89)

Licensed under a Creative Commons Attribution 4.0 International License.

Volume 4, Issue 10, October 2026 ISSN (E): 2938-3803

Computation time. For N = 2000 the computation took 0.007 s, for N = 4000 0.019 s, and for N = 8000 0.051 s. When the number of steps was doubled, the time increased 2.5 and 2.7 times, i.e. the quadratic (4-fold) law had not yet fully manifested itself. Probably, at these sizes the np.dot call is executed very quickly and the share of the O(N) cost of loop control is large. The number of operations nevertheless grows like N^2, so over a long time interval this computation becomes expensive. This is the situation noted by Tomášek [2]: saving by discarding the history is not always possible.

Discussion

The main convenience of the tool is that one can instantly see the graph of the solution by changing a parameter: the influence of α, the influence of λ and the transition to the classical case ($\alpha \to 1$) are observed in the window. Its weak point is also clear: summing the series in ordinary floating-point arithmetic is unreliable for large λt^{α}, which is why a warning was introduced. For reliable results at such parameters, one should use a high-precision computation (Listing 2) or other representations of the Mittag-Leffler function; we did not try them.

Several practical conclusions follow from the results on the Euler method. First, regardless of α, the method gives the long-time value of the solution with first order. If the problem is concerned with the value at t = T, reducing the step works as expected. The initial layer is another matter. When α is small, the solution changes sharply at the beginning, and capturing this part accurately on a uniform grid is costly: for α = 0.3 the maximum error remains $3 \cdot 10^{-2}$ even after 1600 steps. In such cases, refining the grid near zero or using a higher-accuracy scheme makes sense, but we did not try this. Second, the step must be chosen to match λ; otherwise the solution loses all meaning (Table 4).

The limits of the work are clear. Only a single linear equation was considered, which is why an exact solution existed; for nonlinear f there is no reference, and the error has to be estimated differently. Only the explicit scheme was tested, and stability was assessed only through observation. The time measurements were taken in a single environment. At the next stage, it is planned to compare with implicit and predictor–corrector schemes [7], to use a nonuniform grid and nonlinear cases, and also to add the Euler solution to the window.

Conclusion

For the equation ${}^{C}D^{\alpha}y = -\lambda y$ in the Caputo sense, a windowed software tool was created based on Python and Tkinter: the user enters α, λ and T, and the program computes the exact solution via the Mittag-Leffler series and plots its graph. Summing the series in ordinary floating-point arithmetic corrupts the result for large λt^{α} (for α = 1, λ = 10, t = 5, $-5.2 \cdot 10^{4}$ is obtained instead of $1.9 \cdot 10^{-22}$), so the program warns in such cases. The exact solution was compared with the fractional-order Euler method. As α decreases, the solution falls faster at first and then slows down: at t = 5 the value for α = 0.3 is about 50 times larger than in the classical case. The Euler method is first-order for all tested α in terms of the value at t = T; in terms of the maximum error, however, first order was observed only for $\alpha \geq 0.5$, and for α = 0.3 the order remained around 0.55. The explicit scheme oscillates when $q = \lambda h^{\alpha}/\Gamma(\alpha+1)$ exceeds approximately 1, and for α = 0.5 it grows without bound when q exceeds ≈ 1.4. For N = 8000 the computation took 0.05 s, and the cost grows quadratically with N.

Licensed under a Creative Commons Attribution 4.0 International License.

Volume 4, Issue 10, October 2026 ISSN (E): 2938-3803

References

1. Karimov M. M. Numerical solution of fractional-order ordinary differential equations by the Euler method // Scientific Practical Conference. — 2026. — Vol. 1, No. 1. — P. 159–163. — URL: https://d-pressa.com/index.php/spc/article/view/1257
2. Tomášek P. On Euler methods for Caputo fractional differential equations // Archivum Mathematicum. — 2023. — Vol. 59, No. 3. — P. 287–294. — DOI: 10.5817/AM2023-3-287
3. Khan F. S., Khalid M., Bazighifan O., El-Mesady A. Euler's numerical method on fractional DSEK model under ABC derivative // Complexity. — 2022. — Vol. 2022. — Article ID 4475491. — DOI: 10.1155/2022/4475491
4. Clemente-López D., Muñoz-Pacheco J. M., Rangel-Magdaleno J. de J., Vargas-Cabrera L. Unified scientific tool to investigate fractional derivatives of arbitrary variable order with time-memory and order-memory: The VOFD Python package // AIMS Mathematics. — 2026. — Vol. 11, No. 3. — P. 5798–5823. — DOI: 10.3934/math.2026239
5. Adams M. differint: A Python package for numerical fractional calculus. — 2019. — arXiv:1912.05303. — URL: https://arxiv.org/abs/1912.05303
6. Podlubny I. Fractional Differential Equations. — San Diego: Academic Press, 1999. — 340 p.
7. Diethelm K. The Analysis of Fractional Differential Equations. — Berlin: Springer, 2010. — (Lecture Notes in Mathematics, Vol. 2004).

Web of Humanities: Journal of Social Science and Humanitarian Research

Licensed under a Creative Commons Attribution 4.0 International License.