Fractional Order Derivative Computation with a Small Number of Discrete Input Values

Dariusz W. Brzeziński
Institute of Applied Computer Science, Lodz University of Technology, Poland.
Corresponding & presenting author: dbrzezinski@iis.p.lodz.pl

Abstract

High-accurate computer approximation of fractional derivatives and integrals by applying the Grünwald-Letnikov formula requires generally a high number of input values. If required amount can not be supplied, the accuracy of approximation drops drastically. In the paper we solve a difficult and crucial problem in this scope, i.e. when input data consist only of a small number of discrete values. Furthermore, some of the values may be unusable for computational purposes. Our problem solution include appropriate method of input data preprocessing, an interpolation algorithm with extrapolation abilities, central point function discretization schema, recurrent computational method of coefficients and the application of Horner's schema for the core of the Grünwald-Letnikov method: coefficients and function's values multiplication. Numerical method presented in the paper enables computing fractional derivatives and integrals of complicated functions with much higher accuracy than it is possible when the default approach to the Grünwald-Letnikov method computer implementation is applied. This new method usually takes only 10% of function's values required by the default approach for the same computations and it is much less restrictive for their quality. The general novelty of the method is an efficient configuration of existing numerical methods and an enhancement of their abilities by applying modern programming language - Python and arbitrary precision for computations.

Keywords: **Numerical Methods, Finite Differences, Fractional Order Derivatives and Integrals, Accuracy of Numerical Calculations, Arbitrary Precision.**

1 Introduction

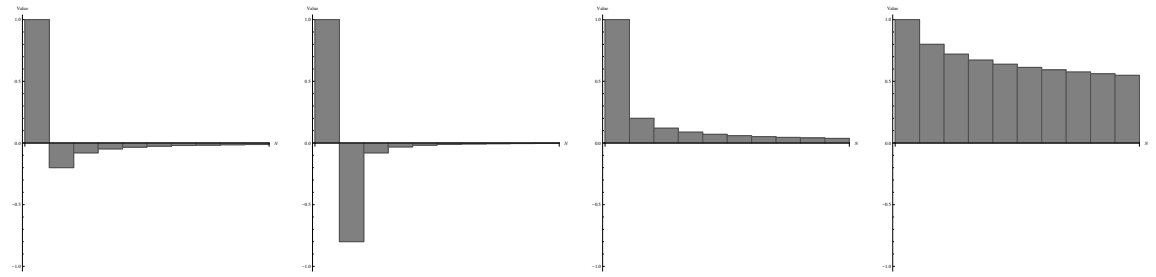
Fractional calculus (FC) or more aptly calculus of any order has been successfully applied for many areas of technical sciences including electrical engineering, electronics and control systems as well as signals analysis and processing. The application for close-loop control systems require computations to be conducted with high accuracy and in precisely provided time. Otherwise, the system control fails.

The time factor of the computations requires the application of mathematical formulas enabling developing fast and compact computer programs. Despite the existence of numerous formulas for numerical approximation of fractional order derivatives and integrals (FOD/I) [1–6], only the popular Grünwald-Letnikov formula (GL) [7–11] fulfills this requirement. Therefore it became the first choice for the purpose of systems control.

The algorithm of this method consists of multiplication of some coefficients (weights) and function's values.

Figure 1 presents first few subsequent coefficients values of the two utmost fractional orders of derivatives/integrals: 0.2 and 0.8.

The remaining orders between 0 and 1 can be deduced from this figure: the first coefficient has always value 1, the second one has a value of an fractional order (negative for derivatives, positive for integrals). The rest of the coefficients decrease their sum to 0.

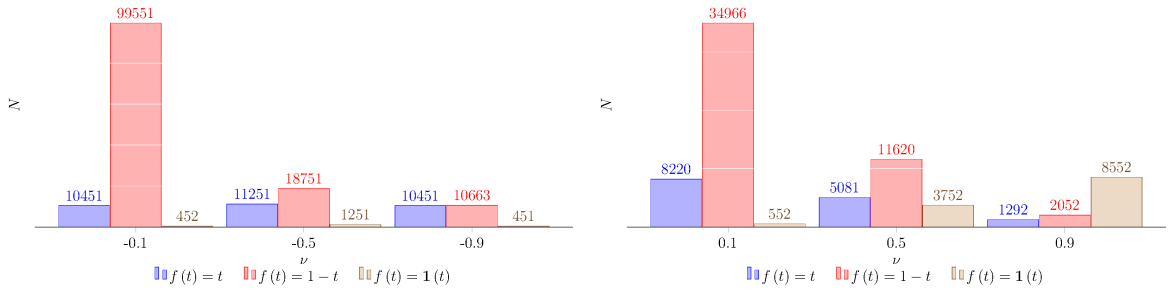


(a) (b) (c) (d)

Figure 1: Ten first coefficients for FOD of order: 0.2 (a), 0.8 (b) and FOI 0.2 (c), 0.8 (d).

The values of subsequent coefficients depend on the fractional order value, i.e. higher orders' coefficients assume higher values than lower orders do. Their amount applied for computation have a direct impact on approximation accuracy.

Our previous research on GL method [12] proved that coefficient amount required for certain level of FOD/I approximation accuracy is determined by "the shape of function". This term describes a behavior of a function in terms of values of its derivatives: (1^{st} and 2^{nd} and higher). If they assume high values, coefficients requirements increase enormously. In this scope: figure 2 presents an indicative number of coefficients (in thousands) required for FOD/I computation of monotonically increasing, monotonically decreasing and constant function in the range $\langle 0, 1\rangle$ with accuracy up to $1.0e^{-04}$ (measured as relative error) by applying a common approach to GL method implementation.



(a) (b)

Figure 2: An indicative number of coefficients N required by respective functions and orders ν for FOI (a) and FOD (b) computation with accuracy up to $1.0e^{-04}$.

The selection of functions in figure 2 is on purpose: it shows that computation of lower order FOD/I of simple monotonically increasing functions with accuracy (expressed as relative error calculated in respect to the exact value) as high as four significant decimal places requires application of 600 of coefficients. For higher accuracy of computation and more complicated functions, there are required many hundreds of thousands of them. In fact, calculation with high accuracy (i.e. with more than four significant decimal places) of a low fractional order of FOD for a high frequency periodic function which bounding box is either constant or decreasing, requires over 2 billion of coefficients. This can become a difficult task even for a state-of-the-art computer. This is pictured in figure 3 with the coefficient requirements for exponential and periodical functions and their combination for an arbitrary selected fractional order 0.45.

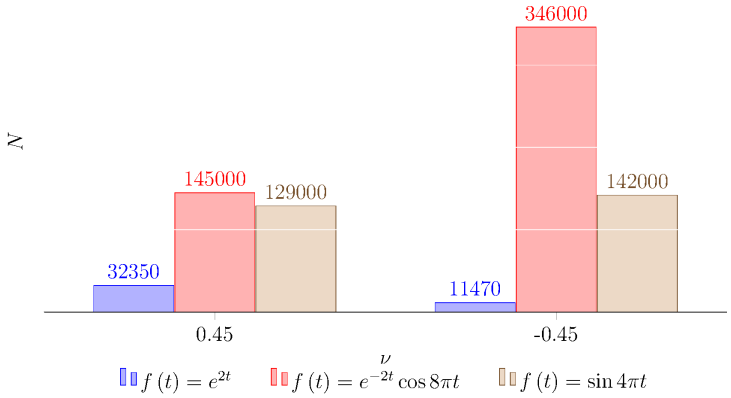


Figure 3: An indicative number of coefficients N required by respective functions and orders ν for FOD and FOI computation with accuracy up to $1.0e^{-04}$.

Long computation time associated with processing of high number of coefficients and multiplying them by function's values makes real-time application difficult. Therefore, our research on GL method was associated from the very beginning with the increase of its efficiency by generally reducing requirements for a number of coefficients for requested accuracy of FOD/I computations. In papers [13, 14] we presented successful attempts achieved by modifying commonly applied form of GL formula with the well known Horner's schema of polynomial's value calculation. Application for real-time computation of an equivalent form of GL formula (Horner's form) enabled us removing from computation up to 20% of coefficients required by application of default approach while still maintaining target accuracy. We refer to it as *calculation tail* of variable length technique.

Furthermore, in paper [15] we presented another successful approach to the research on general accuracy and efficiency increase of GL formula by evaluating some alternative to the commonly applied formulas for coefficients computation and some other forms of GL formula. We also assessed application usefulness for efficiency increase of forward and central point discretization schemas as well as the use of three-point discretization schema.

All the efforts resulted in significant efficiency increase of GL method.

Additional results presented in the same paper showed that the magnitude of errors which influence negatively computational accuracy of scientific calculations can be mitigated or often even eliminated by applying a right form of a mathematical formula and by careful selection of a programing language for its implementation. It usually led to the replacement of the commonly used double precision computer arithmetic with "arbitrary precision" for computation (this term is explained in Section 3).

The following paper presents some interesting results of our latest research on GL formula aimed at solving a practical problem - computation of FOD/I with a small number (often not sufficient for target accuracy) of discrete input data. The number may decrease if some of the data are unusable for calculations with computer. This includes infinite and NaN (not a number) values. In such situation computational accuracy can drop below two digit error expressed in percent. Therefore, for the following research we aimed at increasing computational accuracy to at least two decimal places with the same data amount supplied.

The paper is divided into the following sections: At first there are presented several forms of GL formula which are applied for FOD/I computation. This section also includes a brief description of Horner's form of the GL formula and its application algorithm of "calculation tail" of variable length. Next, there is explained importance of arbitrary precision over standard dou-

ble precision for computations and its positive impact on their accuracy and reliability. Section 4 contains description of computing tools. Section 5 includes details of conducted numerical experiment. Last sections 6 and 7 presents concisely results and conclusions.

2 Mathematical Background

The Grünwald-Letnikov FOD/I approximation method can be represented using the following formula

$$ {}_{t_0}D_t^{(\nu)} f(t) = \lim_{h\to 0^+} \frac{1}{h^\nu} \sum_{i=0}^{N-1} (-1)^i \binom{\nu}{i} f(t-ih), \tag{1} $$

in which $\nu \in \mathbf{R}$ is the order of derivative; fractional integral is defined as derivative evaluated for negative order $-\nu < 0$, N denotes an amount of steps in summation, t_0,t is the interval and $h = \frac{t-t_0}{N}$ is the subinterval width.

Formula (1) includes ∞ limitation, which thwarts its computer application. The next formula has it removed, which makes it useful for computational purposes. The formula will be referred as the commonly used form of GL formula for FOD/I computations, in which Γ denotes Euler's Gamma function

$$ {}_{t_0}D_t^{(\nu)} f(t) \simeq \frac{h^{-\nu}}{\Gamma(-\nu)} \sum_{k=0}^{N-1} \frac{\Gamma(k-\nu)}{\Gamma(k+1)} f(t-kh). \tag{2} $$

The formulas (1) and (2) represent backward-difference. GL formula can be applied with central point discretization schema [16]

$$ {}_{t_0}D_t^{(\nu)} f(t) \simeq \frac{h^{-\nu}}{\Gamma(-\nu)} \sum_{k=0}^{N-1} \frac{\Gamma(k-\nu)}{\Gamma(k+1)} f\left(t-\left(k-\frac{\nu}{2}\right)h\right) \tag{3} $$

and with the forward point discretization schema

$$ {}_{t_0}D_t^{(\nu)} f(t) \simeq \frac{h^{-\nu}}{\Gamma(-\nu)} \sum_{k=0}^{N-1} \frac{\Gamma(k-\nu)}{\Gamma(k+1)} f(t-(k-\nu)h). \tag{4} $$

The application of formulas (1)-(4) for computation has some serious restriction: it requires that $t_0 = 0$.

In paper [17] we can find a formula, which removes these restrictions and it has higher accuracy order (second instead of first one)

$$ {}_{t_0}D_t^{(\nu)} f(t) \simeq \frac{h^{-\nu}}{\Gamma(-\nu)} \sum_{k=0}^{N-1} \frac{\Gamma(k-\nu)}{\Gamma(k+1)} f\left(t-\left(k-\frac{\nu}{2}\right)h\right) + \frac{h^{-\nu}}{\Gamma(-\nu)} \frac{(1+\nu)}{2} f(t_0) N^{-1-\nu}. \tag{5} $$

Before presenting Horner's form of the the Grünwald-Letnikov formula we introduce a discrete version of the formula (2) for $h = 1$ and $t = ih$.

For a given discrete-time, real bounded function $f(k) = f_0, f_1, \ldots f_{k-1}, f_k$ GL formula of

Fractional Order Backward Difference (FOBD) is defined as

$$ {}_0^{GL}\Delta_k^{(\nu)} f_k = \sum_{i=0}^{k} a_i^{(\nu)} f_{k-i}, \tag{6} $$

where ν is the FOBD order. Fractional Order Backward Sum (FOBS) is defined as the FOBD evaluated for negative order, f_k is a discrete time function and $a_i^{(\nu)}$ are the coefficients for $i = 0, 1, 2, 3 \ldots, k-1, k$.

The coefficients $a_i^{(\nu)}$ can be calculated by applying several formulas, which include the formula involving factorial function calculation

$$ a_i^{(\nu)} = \begin{cases} 0 & \text{for } i < 0 \\ 1 & \text{for } i = 0 \\ (-1)^i \frac{\nu(\nu-1)\cdots(\nu-i+1)}{i!} & \text{for } i > 0. \end{cases} \tag{7} $$

This formula presents a serious limitation for computational accuracy due to the use of factorial function. It causes the overflow which limits a number coefficients that can be computed to 170. Therefore the following recurrent formula should be applied instead. It is derived from the relation between coefficients $a_i^{(\nu)}$

$$ a_i^{(\nu)} = a_{i-1}^{(\nu)} \left(1 - \frac{1+\nu}{i}\right) \text{ for } i > 0. \tag{8} $$

The next formula presents the algorithm (7) expressed in a matrix-vector form

$$ {}_0^{GL}\Delta_k^{(\nu)} f(k) = \left[a_0^{(\nu)}\, a_1^{(\nu)}\, \cdots\, a_k^{(\nu)}\right] \begin{bmatrix} f_k \\ f_{k-1} \\ \vdots \\ f_0 \end{bmatrix}. \tag{9} $$

The Horner form of GL formula is a formula to which the well known Horner's schema of polynomial's value calculation is applied. Horner's schema possesses some significant computational advantages, e.g. lower computational complexity and a natural method of data input for computation.

By applying the same assumptions as in case of the Grünwald-Letnikov definition, introducing new coefficients $c_i^{(\nu)}$

$$ c_i^{(\nu)} = \begin{cases} 0 & \text{for } i < 0 \\ 1 & \text{for } i = 0 \\ 1 - \frac{1+\nu}{i} & \text{for } i > 0 \end{cases} \tag{10} $$

we apply Horner's schema to (6). We obtain [18]

$$ {}_0^{H}\Delta_k^{(\nu)} f(k) = c_0^{(\nu)} \left[f_k + c_1^{(\nu)} \left[f_{k-1} + c_2^{(\nu)} \left[f_{k-2} + \cdots + c_{k-1}^{(\nu)} \left[f_1 + c_k^{(\nu)} \left[f_0\right]\right]\right]\right]\right]. \tag{11} $$

The formulas (6) and (11) are equivalent, i.e.

$$ {}_0^{GL}\Delta_k^{(\nu)} f_k = {}_0^{H}\Delta_k^{(\nu)} f_k. $$

However,

$$\lim_{i\to\infty} a_i^{(\nu)} = 0,$$

$$\lim_{i\to\infty} c_i^{(\nu)} = 1, \tag{12}$$

and for some $i > k - L$

$$a_i^{(\nu)} \approx 0,$$

$$c_i^{(\nu)} \approx 1$$

We can use the property of $c_i^{(\nu)}$ coefficients to modify the formula (11)

$$\begin{aligned} &{}_0^{Hs}\Delta_{k,L}^{(\nu)} f(k) \\ &= \begin{cases} c_0^{(\nu)}\left[f_k + c_1^{(\nu)}\left[f_{k-1} + c_2^{(\nu)}\left[f_{k-2} + \cdots + c_{k-1}^{(\nu)}\left[f_1 + c_k^{(\nu)}\left[f_0\right]\right]\right]\right]\right] & \text{for } k \le L \\ c_0^{(\nu)}\left[f_k + c_1^{(\nu)}\left[f_{k-1} + \cdots + c_{k-L}^{(\nu)}\left[\sum_{i=0}^{L} f_i\right]\right]\right] & \text{for } k > L. \end{cases} \end{aligned} \tag{13}$$

Application of formula (13) enables reducing up to 20% requirements for an amount of coefficients during FOD/I computation, i.e. for samples $k > L$ we assume $c_i^{(\nu)} = 1$ and replace multiplication with summation of the remaining function values.

The L value is an arbitrary number and is to be set empirically. However, its value must be selected carefully, i.e. the removal of L coefficients is not to decrease accuracy of calculations over a permissible error.

Based on our past experience, we apply $L = 20\%$ less coefficients required for FOD/I computation with target accuracy by the default approach (see as examples numbers of coefficients presented in Figures 2 and 3), e.g. if a function requires 600 coefficients for the accuracy $1.0e^{-04}$, then we calculate 20% of this number (which in the case is 120). From now on, $L = 480$ for the use with formula (13). After this operation, there are applied $600 - 120 = 480$ coefficients.

For full reasoning and procedure description how to set L, please refer to the papers mentioned in introduction.

The next table presents computational accuracy decrease over $1.0e^{-04}$ after removal of $L = 20\%$ of former number of coefficients required for this accuracy (see again Figures 2 and 3) by applying formula (13).

Table 1: Relative error increase over $1.0e^{-04}$ after removal of $\mathbf{L = 20\%}$ of coefficients number required for this accuracy for selected fractional orders ν (positive - FOD, negative - FOI).

	Function $f(t)$				
	t	$1-t$	$\mathbf{1}(t)$	$e^{-2t}\sin(8\pi t)$	$\cos(4\pi t)$
0.1	0.28	0.02	2.49	4.39	0.1
0.5	2.15	0.08	2	1.05	0.25
0.9	5.25	0.04	0.53	0.09	0.17
−0.1	0.22	0	2.44	8.84	0.17
−0.5	0.6	0.04	1.74	7.32	1.79
−0.9	0.19	0.01	0.42	3.16	7.96

3 The Importance of Arbitrary Precision

Limiting factor to the accuracy of computation by applying the formulas (2)-(5) is a precision which uses the computer to store data that are being supplied to it.

Consider calculating the value of $\Gamma(z)$ for large values of z. At $z = 171$ the approximate value provided by the computer programmed to use standard double precision arithmetic is $7.26e306$. At $z = 171$ the computer begins to refer to the value of $\Gamma(z)$ as 'Inf'. Hence, for the formulas (1)-(2) there is no practical use of calculating coefficients beyond the 171^{st}.

The above example of overflow occurrence shows that the selection of uniform C++ equipped with the standard mathematical library as a main programming tool is not enough nowadays to take full advantage of available hardware. The application of arbitrary precision computing for increasing the accuracy and the correctness of numerical calculations and Nvidia CUDA parallelization technology for their effectiveness, are the best examples in this context [19].

Application of arbitrary precision makes it possible for the user to choose precision for calculation and for each variable storing a value. It is not machine-depended or IEEE standard types. With its help we can - among the others - increase general accuracy of mathematical computations. However, its application purpose is above all to increase accuracy of numerical calculations, e.g. by eliminating under- and overflows, increasing accuracy of a polynomial zeros finding and derivative and integral calculating.

The importance of elimination of limited precision in computer calculations was aptly presented by Toshio Fukushima in *The Astronomical Journal* in 2001 by giving the following example: "In the days of powerful computers, the errors of numerical integration are the main limitation in the research of complex dynamical systems, such as the long-term stability of our solar system and of some exoplanets [...]" and gives an example where using double precision leads to an accumulated round-off error of more than 1 radian for angular position of planets [20].

Double precision computer arithmetic is optimized for speed and has many flaws which negatively influence the accuracy of computations, e.g. limitations of number values which double precision variables can hold or no programmer influence on mathematical operations rounding.

However, it is the lack of clarity in handling of intermediate results which troubles the most, i.e. the floating-point standard [21] only defines that the results must be rounded correctly to the destination's precision and fails to define the precision of destination variable. This choice is commonly made by a system or a programming language. The user can not influence it in any way. Therefore, the same program can return significantly different results depending on the implementation of the IEEE standard.

Arbitrary precision application is applied in conjunction with special libraries which include their own data structures and mathematical functions.

There are many programming languages, which can be used with arbitrary precision. They include Python. Python is an object oriented script language, which achieves a higher abstract level than for example C++, i.e. an individual programmer can achieve the same results in a much shorter time and with far fewer lines of code. It also has especially clean and straightforward syntax. It can lead to programs' shorter executing time.

An important advantage of programming using Python is availability of ready to use libraries. They enable solving a scientific problem by focusing rather on selecting the right tools and by adopting them if necessary instead of designing a new algorithm from ground up. Therefore, we selected it as the main programming language for our research.

4 Tools for the Problem Solving

The requirement of an efficient solution to the problem presented in the introduction includes constructing an interpolation polynomial using supplied values. It is crucial for arbitrary selection of a discretization point schema, i.e. central (10) or forward (11) one or the one with three points.

We applied SciPy - an open source Python library for scientific and technical computing. SciPy provides a module for interpolation based on the FITPACK library of FORTRAN functions, which is assumed as reliable.

Central point discretization formula (3) requires at least one point beyond t to be accessible for the interpolation. Unfortunately standard routines in SciPy do not allow to include any points from outside the interpolation range. At first our solution to make scipy.interpolate give an extrapolated result beyond the input range included modifying an interpolating algorithm based on spline interpolation by adding:

- Constant extrapolation: extrapolating left and right values as constant beyond the range
- Linear extrapolation: writing a wrapper around an interpolation function, which does linear extrapolation
- Manually inserted points and values to the initial array
- scipy.interpolate.splrep (with degree 1 and no smoothing)

Due to unsatisfactory accuracy and speed concerns, we have decided to apply InterpolatedUnivariateSpline from the same library scipy.interpolate instead. It does interpolation and extrapolation and can be applied in conjunction with mpmath. mpmath is a free (BSD licensed) Python library for real and complex floating-point arithmetic with arbitrary precision. It is based on GNU GMP and GNU MPFR libraries. It enables switching from double to arbitrary precision computation by applying Python programming language.

The mathematical library mapmath is required not only to increase overall exactness of computations, but also due to accessibility of excellent implementations of gamma and reciprocal gamma functions required for FOD/I calculations by applying formulas (2)-(5).

5 Details to the Numerical Experiment

In our previous numerical experiments with GL method the number of coefficients (and function's values at the same time) was commonly limited to 600, because we applied such an amount for real-time calculations (in our case the maximum number of coefficients was determined by amount of available memory in a test DSP-system).

However, to present high efficiency of FOD/I computing method presented in the paper, our experiment was conducted with only 60 values as an input. Additionally, up to 20% of them were randomly assigned ∞ and NaN values to mimic unsuitability for computation, which often occur if experiment data were collected without the knowledge of computing input data requirements.

Unlike simple, monotonically increasing, decreasing or constant functions, which require relatively moderate number of coefficients (up to 600) for accuracy up to four significant decimal places, high-frequency, exponential and periodical functions, which have "dramatical shape changes" require millions or more of them. For this reason, general efficiency and accuracy of the method was assessed against two sets of functions.

Set one included constant, monotonically increasing and decreasing functions:

$\mathbf{f_1}$ $f(t) = \mathbf{1}(t) \in (0,1)$

$\mathbf{f_2}$ $f(t) = tt - t, t \in (0,1)$

$\mathbf{f_3}$ $f(t) = t^{0.1}, t \in (0,1)$.

Set two included more complicated functions:

$\mathbf{f_4}$ $f(t) = te^{-t}, t \in (0,10)$

$\mathbf{f_5}$ $f(t) = e^{-2t}\cos 8\pi, t \in (0,1)$

$\mathbf{f_6}$ $f(t) = e^{-2t}, t \in (0,1)$

$\mathbf{f_7}$ $f(t) = e^{t}\sin t, t \in (0,\pi)$

$\mathbf{f_8}$ $f(t) = 1.5\cos 2t + 2.2\cos 4t, t \in (0,2\pi)$

$\mathbf{f_9}$ $f(t) = \sin 2\pi \cos t, t \in (0,5)$.

The functions were tabulated before supplying them to the program, i.e. the actual input to a program was in form of a vector with discrete values.

At first we applied interpolation method described in section 4 to construct an interpolation polynomial using the vector with the supplied values. Next, we interpolated 600 values for FOD/I calculations using central (3) and forward (4) discretization schemas. Finally FOD/I were computed.

For comparison purposes, FOD/I were computed by applying the commonly used reference formula (6) with 60 values as well to present the real-life accuracy.

The FOD/I exact values required for accuracy assessment were computed by applying analytical formulas (if available) or by using high-accuracy integration method [22, 23]. This method involves Gauss-Jacobi Quadrature application for integration and is reliable for computation of FOD/I using Riemann-Liouville and Caputo formulas [24] with accuracy up to 120 significant decimal places.

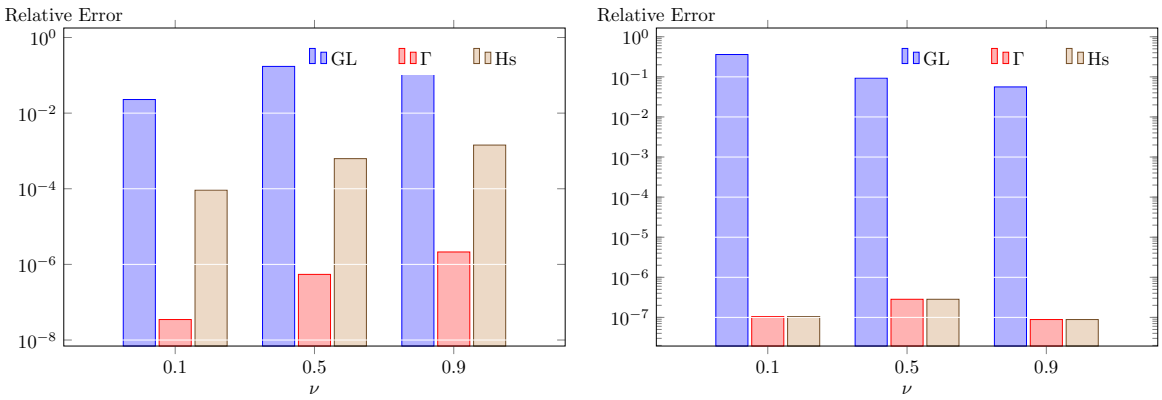
6 Results

FOD/I computational accuracy is assessed as the relative error

$$e_r(m) = \left| 1 - \frac{v_c}{v_e} \right|, \tag{14}$$

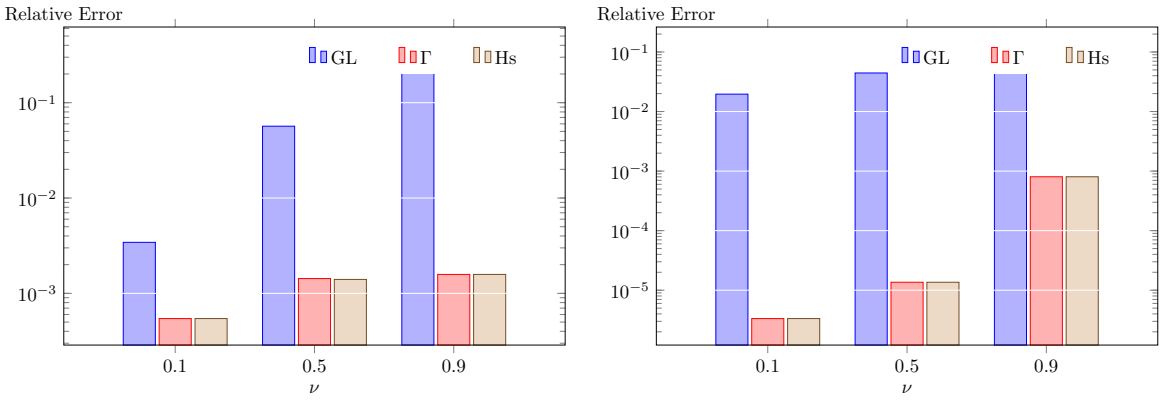
in which: v_c is a calculated value, v_e is a value assumed as exact and m denotes a number of source input values for FOD/I computation.

Figures 4-8 present accuracy of FOD/I computation denoted as relative error (14) for: GL - a classical Grünwald-Letnikov formula (2) applied with $m = 60$ function's values (to present the accuracy of computations, which can be expected by applying only real supplied functions' values); Γ denotes computations by applying the formula (5) and Hs - the formula (13) - both with 600 interpolated values, which are computed by applying the combination of techniques described in the paper (to present the accuracy which can be obtained despite the only 60 supplied real function's values).



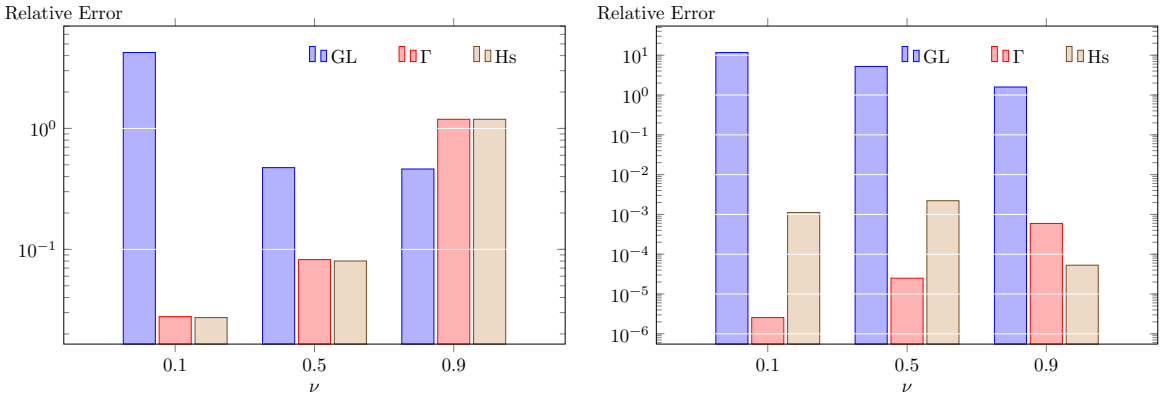
(a) (b)

Figure 4: Computational accuracy of FD, order ν for: (a) f_1 and (b) f_2.



(a) (b)

Figure 5: Computational accuracy of FD, order ν for: (a) f_3 and (b) f_4.



(a) (b)

Figure 6: Computational accuracy of FD, order ν for: (a) f_5 and (b) f_6.

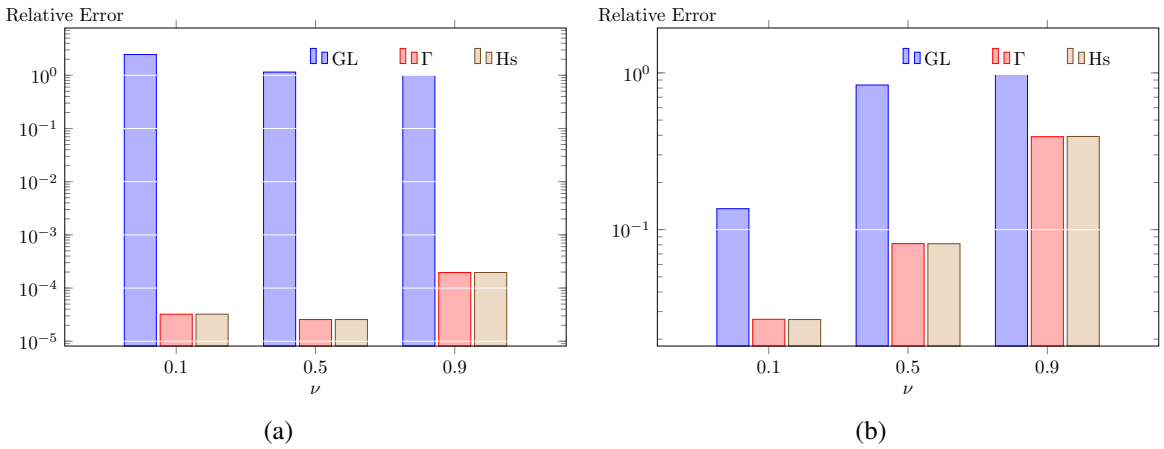


Figure 7: Computational accuracy of FD, order ν for: (a) f_7 and (b) f_8.

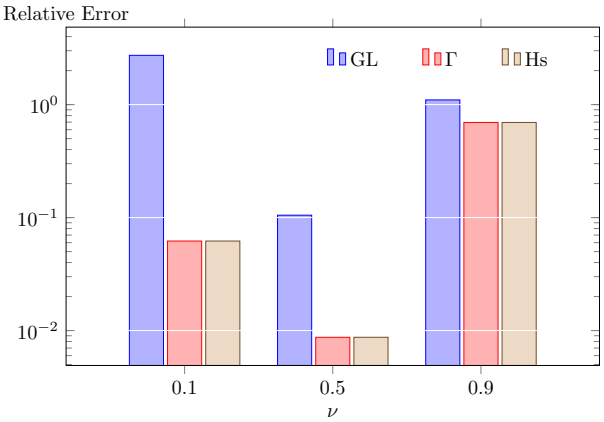


Figure 8: Computational accuracy of FD, order ν for f_9.

7 Conclusions

The Grünwald-Letnikov method requires a high number input values for high accuracy FOD/I computation. If required number of values can not be supplied, the accuracy of computation drops drastically, particularly for complicated functions.

To solve this deficiency problem we proposed a new numerical method combining existing numerical techniques, arbitrary precision computation and modern programming language.

The method presented in the paper enables computation of FOD/I with similar or higher accuracy with only 10% of input values required by the traditional approach to FOD/I computation by applying the Grünwald-Letnikov method.

Application of the simplified Horner's form of the Grünwald-Letnikov formula for the core of the method (multiplication of the coefficients and function's values) decreases again by up to 20% of those 10% the requirements for input values during computation without noticeable accuracy drop over the assumed level.

Additionally, the developed method enables "repairing" input values unusable for computational purposes.

The computational method described in the paper combines several programming techniques that include application of Python programming language and accompanying mathematical library (mpmath) for arbitrary precision of computations. This combination enables

elimination of common errors associated with double precision computer mathematics and increases significantly accuracy and reliability of scientific computation.

Acknowledgement

The work was created as a result of the research project no. 2016/23/D/ST6/01709 financed from the funds of the National Science Center, Poland.

References

[1] R. Herrmann. *Fractional Calculus. An Introduction for Physicists, 2nd. ed.* World Scientific, Singapore, 2014.

[2] D. Baleanu, K. Diethlem, E. Scalas, and J.J. Trujillo. *Fractional Calculus. Models and Numerical Methods*. World Scientific, Singapore, 2012.

[3] J. T. Machado. Numerical calculation of the left and right fractional derivatives. *Journal of Computational Physics*, 293:96–103, 2015.

[4] L. Debnath and D. Bhatta. *Integral Transforms and Their Applications, Third Edition*. CRC Press, Taylor & Francis Group, Boca Raton London New York, 2015.

[5] P. J. Nahin. *Inside Interesting Integrals*. Springer-Verlag, NY, 2015.

[6] C. Li and F. Zeng. *Numerical Methods for Fractional Calculus*. Chapman & Hall, 2015.

[7] I. Podlubny. *Fractional Differential Equations*. Academic Press, INC, San Diego Ca, 1999.

[8] M. D. Ortigueira, J. A. T. Machado, and J. Sa da Costa. Which differintegration? *Proceedings - Vision, Image and Signal Processing*, 152(6), 2005.

[9] M. D. Ortigueira. Fractional central differences and derivatives. *Journal of Vibration and Control*, 14(9-10):1255–1266, 2008.

[10] M.D. Ortigueira. *Fractional Calculus for Scientists and Engineers*. Springer-Verlag, NY, 2011.

[11] M. D. Ortigueira and J. A. T. Machado. What is a fractional derivative. *J.Comput. Phys*, 2014.

[12] D. W. Brzeziński and P. Ostalczyk. About utility of the simplified grünwald-letnikov formula equivalent horner form. *Discontinuity, Nonlinearity, and Complexity. Special Issue: "Fractional Dynamics and Systems with Power-Law Memory" (Eds. M.Edelman and J. Tenreiro Machado)*, 4(4), 2015.

[13] D. W. Brzeziński and P. Ostalczyk. The grünwald-letnikov formula and its horner's equivalent form accuracy comparison and evaluation for application to fractional order pid controler. In *IEEE Explore Digital Library: IEEE Conference Publications-17th International Conference On Methods and Models In Automation and Robotics (MMAR)*, pages 579–584, 2012.

[14] P. Ostalczyk, D. W. Brzeziński, P. Duch, M. Łaski, and D. Sankowski. The variable, fractional-order discrete-time pd controller in the iisv1.3 robot arm control. *Central European Journal of Physics*, 11(6):750–759, 2013.

[15] D. W. Brzeziński and P. Ostalczyk. About accuracy increase of fractional order derivative and integral computations by applying the grünwald-letnikov formula. *Communications in Nonlinear Science and Numerical Simulation*, 40:151–162, 2016.

[16] K. Oldham and J. Spanier. *The Fractional Calculus. Theory and Applications of Differentiation and Integration to Arbitrary Order*. Academic Press, INC, San Diego Ca, 1974.

[17] Y. Takeuchi and R. Suda. New numerical computation formula and error analysis of some existing formulae in fractional derivatives and integrals. In *Proceedings to the Fifth Symposium on Fractional Differentiation and its Applications*, Hohai University, Nanjing, China, 2012.

[18] P. Ostalczyk. Fractional-order backward difference equivalent forms. In *Fractional Differentiation and Its Applications. Systems Analysis, Implementation and Simulation, System Identification and Control*. 1995.

[19] J. M. Müller, N. Brisebarre, F. De Dinechin, C. P. Jeannerod, V. Lefevre, G. Melquiond, N. Revol, D. Stehle, and S. Torres. *Handbook of Floating-Point Arithmetic*. Birkhauser, New York, NY, 2010.

[20] K. R. Ghazi, V. Lefevre, P. Theveny, and P. Zimmermann. Why and how to use arbitrary precision. *IEEE Computer Society*, 12(3):1–5, 2001.

[21] Microprocessor Standards Committee. *IEEE Standard for Floating-Point Arithmetic*, 2008. `http//dox.doi.org/10.1109/IEEESTD.2008.4610935`.

[22] D. W. Brzeziński and P. Ostalczyk. High-accuracy numerical integration methods for fractional order derivatives and integrals computations. *Bulletin of the Polish Academy of Sciences Technical Sciences*, 62(4):723–733, 2014.

[23] D. W. Brzeziński. Accuracy problems of numerical calculation of fractional order derivatives and integrals applying the riemann-liouville/caputo formulas. *Applied Mathematics and Nonlinear Sciences*, 1(1):23–43, 2016.

[24] D. W. Brzeziński. Comparison of fractional order derivatives computational accuracy - right hand vs left hand definition. *Applied Mathematics and Nonlinear Sciences*, 2(1):237–248, 2017.