Proceedings of the Federated Conference on Computer Science and Information Systems pp. 297–306
DOI: 10.15439/2018F107
ISSN 2300-5963 ACSIS, Vol. 15

Computation of Gauss-Jacobi Quadrature Nodes and Weights with Arbitrary Precision

Dariusz W. Brzeziński
Institute of Applied Computer Science
Lodz University of Technology
18/22 Stefanowskiego St., 90-924 Łódź, Poland
Email: dbrzezinski@iis.p.lodz.pl

***Abstract*—In the paper there are presented efficient and accurate methods of Gauss-Jacobi nodes and weights computation. They include an enhancement for standard iteration method for Jacobi polynomials zeros finding, weight function formula transformation for increased accuracy of fractional derivatives computation and arbitrary precision application for mitigation of double precision arithmetic flaws. The results of numerical experiments presented in the paper prove high accuracy and efficiency of developed methods for computation of quadratures' nodes and weights, decreased amount of required iterations for polynomials zeros finding and elimination of truncation errors during weights computation. Accuracy of computations depends on height of precision applied for it, which is limited only by accessible hardware.**

I. Introduction

SPECIAL FUNCTIONS are part of mathematics that covers not only well known logarithmic, exponential and trigonometric functions, but also beta, gamma and zeta functions and orthogonal polynomials.

Special functions have numerous applications, not only in mathematics, but also in applied sciences, astronomy, heat conduction, electrical circuits, quantum mechanics and mathematical statistics. More about this subject can be found in [1].

Classical Jacobi orthogonal polynomials are applied in many important scientific areas that include functions' approximation in collocation points method for solutions of ordinary differential equations known as Sturm-Liouville problem and lately - fractional order derivatives and integrals computations by applying Gauss-Jacobi Quadrature [2], [3].

Methods of mathematical formulas implementations in computer programs are crucial part of numerical methods research due to their influence on general accuracy and efficiency of scientific computing. Especially in the case of a basic research as for example computation of polynomials values, their derivatives or their zeros, that can become a part of another computational methods.

Available research on these subjects focus on achieving the highest order of calculated polynomial [4], highest computational speed [5] and lowest computational complexity [6].

Besides of an interesting implementation of algorithms around orthogonal polynomials by applying Julia programming language [4], the majority of results published in scientific papers are obtained by applying computer implementations in Matlab, C++ or Python programming languages and the use of the double precision arithmetic.

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

In the meantime, computational capabilities of computers has been steadily increased and they presently enable using numerous enhancements for the uniform programming languages on the everyday basis. They include *infinite precision computing* for increasing accuracy and correctness of numerical calculations and Nvidia CUDA parallelization technology for their effectiveness, are the best examples in this context.

The term *Infinite Precision Computing* is just a metaphor suggesting, that precision of computation is only limited by an amount of accessible hardware. The more appropriate description of this aspect of computation would be *Arbitrary Precision Computing*.

Arbitrary precision computing has numerous advantages over standard double precision one, e.g. it makes possible for the user to choose a precision for each calculation and for each variable storing a value; it is also not depended on machine or IEEE standard types of data. Therefore, it opens brand new possibilities in terms of accuracy and correctness of scientific computing.

Having that in mind, the primary aim of the following paper is to present high-accuracy computing methods of Jacobi polynomial and its derivative, nodes (zeros of Jacobi polynomials) and weights of Gauss-Jacobi Quadrature.

The secondary aim is to present application usefulness of high-accurate computed Gauss-Jacobi Quadrature for fractional order derivatives and integrals computation.

Presented results of the experiments enable investigating in the future the possibility of their application in spectral methods for solutions of fractional order differential equations and the research of the high-accuracy computation on mitigation of Runge Phenomenon.

The paper is organized as follows: In section II, there are presented mathematical formulas for Jacobi polynomials and their derivatives computation. Section III and IV provides mathematical preliminaries about Gauss-Jacobi Quadrature together with detailed guidance how to adapt it for high-accuracy

fractional derivatives and integrals computation. Section VI describes standard methods for zeros of Jacobi polynomials finding (nodes of quadrature) and their enhancements for increasing computational accuracy and efficiency. Next section VII expands the information on methods of Gauss-Jacobi Quadrature weights computation with details on their enhancements. Both sections include numerous test plots and some preliminary results. Last section VIII presents practical testbed for developed methods in terms of accuracy and efficiency for fractional derivatives of two example functions computation. The paper ends with usual conclusions and future research.

II. Jacobi Polynomial and Its Derivative Computation

Jacobi orthogonal polynomials have two parameters usually denoted as α and β [7] and can be computed by applying Rodrigues' formula [8]

$$P_n^{(\alpha,\beta)}(x) = \frac{(-1)^n}{2n\cdot n!}(1-x)^{-\alpha}(1+x)^{-\beta}\frac{d^n}{dx^n}\left[(1-x)^{n+\alpha}(1+x)^{n+\beta}\right]. \tag{1}$$

Jacobi polynomials are orthogonal with respect to the weight function

$$w(x) = (1-x)^{\alpha}(1+x)^{\beta} \tag{2}$$

only for

$$\alpha,\beta > -1,\ -1 < x < 1$$

and particularly

1) For $\alpha = \beta = 0$ we obtain ultraspherical Jacobi polynomials - Legendre polynomials,
2) For $\alpha = \beta = \frac{1}{2}$ we obtain ultraspherical Jacobi polynomials - Chebyshev polynomials of second kind,
3) For $\alpha = \beta = -\frac{1}{2}$ we obtain ultraspherical Jacobi polynomials - Chebyshev polynomials of first kind,
4) For $\alpha = \beta$ we obtain Gegenbauer polynomial.

Jacobi polynomials $P^{(\alpha,\beta)}(x)$ of order n $P_n^{(\alpha,\beta)}(x)$ can be calculated by applying explicit form of the Rodriguez formula [9]

$$P_n^{(\alpha,\beta)}(x) = 2^{-n}\sum_{k=0}^{n}\binom{n+\alpha}{k}\binom{n+\beta}{n-k}(x-1)^{n-k}(x+1)^k, \tag{3}$$

wherein

$$P_0^{(\alpha,\beta)}(x) = 1,\ P_1^{(\alpha,\beta)}(x) = \frac{1}{2}(\alpha+\beta+2)\,x\frac{1}{2}(\alpha-\beta).$$

The derivative of Jacobi polynomial $P_n'^{(\alpha,\beta)}(x)$ of order n $P_n'^{(\alpha,\beta)}(x)$ can be calculated by applying the following formula

$$\frac{d}{dx}\left[P_n^{(\alpha,\beta)}(x)\right] = \frac{1}{2}(n+\alpha+\beta+1)\,P_{n-1}^{(\alpha+1,\beta+1)}. \tag{4}$$

However, application of formula (3) for computations is impractical. We can replace it by three-term recurrent formula (6) for this purpose resulting from theorem 1.1.1 published in [9].

According to it, more practical formula for Jacobi polynomial computation $P_n^{(\alpha,\beta)}(x)$, $n = 0,1,2,3,\ldots$ is

$$P_n^{(\alpha,\beta)}(x) = \sum_{i=0}^{n}\frac{(-1)^{n-i}(1+\beta)_i(1+\alpha+\beta)_{n+i}}{m!(n-i)!(1+\beta)_i(1+\beta+\alpha)_n}\left(\frac{x+1}{2}\right)^i, \tag{5}$$

where $(\alpha)_i = \alpha(\alpha+1)\cdots(\alpha+i-1)$, $\alpha_0 = 1$.

From (5) the following computational three-term recurrence formula can be then derived

$$\begin{aligned} P_0^{(\alpha,\beta)}(x) &= 1, \\ P_1^{(\alpha,\beta)}(x) &= \frac{1}{2}\left[(\alpha-\beta)+(\alpha+\beta+2)\,x\right], \\ P_{n+1}^{(\alpha,\beta)}(x) &= (\alpha_n x+\beta_n)P_n^{(\alpha,\beta)}(x) - \gamma_n P_{n-1}^{(\alpha,\beta)}(x), \\ & n = 1,2,\cdots, \end{aligned} \tag{6}$$

where

$$\begin{aligned} \alpha_n &= \frac{(2n+\alpha+\beta+1)(2n+\alpha+\beta+2)}{2(n+1)(n+\alpha+\beta+1)}, \\ \beta_n &= \frac{(2n+\alpha+\beta+1)(\alpha^2-\beta^2)}{2(n+1)(n+\alpha+\beta+1)(2n+\alpha+\beta)}, \\ \gamma_n &= \frac{(n+\alpha)(n+\beta)(2n+\alpha+\beta+2)}{(n+1)(n+\alpha+\beta+1)(2n+\alpha+\beta)}. \end{aligned}$$

Two example plots for Jacobi polynomial $P_n^{(1.5,-0.5)}(x)$ and its derivative $P_n'^{(1.5,-0.5)}(x)$, $x \in \langle -1,1\rangle$ of order $n = 1,2,\ldots 5$ are presented in Figures 1 and 2.

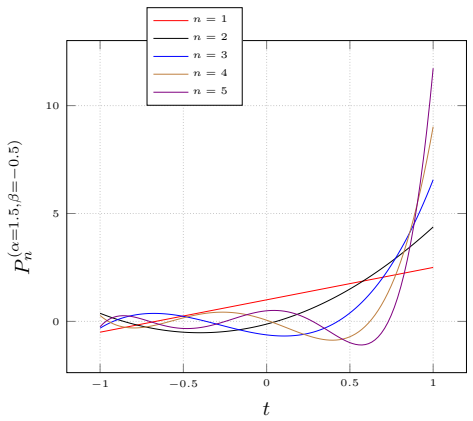

Fig. 1. Graph of Jacobi polynomial $P_n^{(1.5,-0.5)}(x)$, $x \in \langle -1,1\rangle$ of order $n = 1,2,\ldots 5$

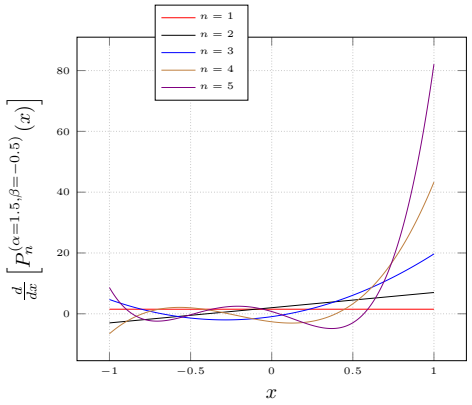

Fig. 2. Graph of Jacobi polynomial 1^{st} derivative $\frac{d}{dx}\left[P_n^{(1.5,-0.5)}(x)\right]$, $x \in \langle -1, 1\rangle$ of order $n = 1, 2, \ldots 5$.

III. Gauss-Jacobi Quadrature

A weight function enabling elimination of problems with integration of functions with singularities at both ends of integration interval is so called *Jacobi weight* (2).

By using some of its properties, there can be increased the accuracy of numerical integration, e.g. for computation of derivatives and integrals of fractional orders. Detailed description will be presented in subsection of section IV.

Jacobi polynomials are orthogonal in respect to weight function (2).

By applying Gauss-Jacobi Quadrature definition, formula for finite integral approximation assumes the following form

$$\int_{-1}^{1} (1-x)^{\alpha} (1+x)^{\beta} \cdot f(x)\, dx \approx \sum_{k=1}^{n} w_k f(x_k), \quad (7)$$

where the nodes of the quadrature x_k are the zeros of Jacobi polynomial $P_n^{(\alpha,\beta)}(x_k)$ of order n.

The weights w_k can be computed by applying the following formula

$$w_k = 2^{\alpha+\beta+1} \frac{\Gamma(\alpha+n+1)\,\Gamma(\beta+n+1)}{n!\,\Gamma(\alpha+\beta+n+1)\,(1-x_k^2)\left[P_n^{(\alpha,\beta)'} x_k\right]^2}, \quad (8)$$

where $P_n'(x_k)$ is 1^{st} derivative $P_n^{(\alpha,\beta)'}(x_k)$ of Jacobi polynomial of order n and $\Gamma(.)$ is Euler Gamma function.

IV. Formulas for Fractional Order Derivatives and Integrals Approximation

Fractional order derivatives and integrals can be approximated by applying numerous formulas representing various approaches to this problem [10]. The most popular are as follows.

Riemann-Liouville integral of fractional order $\nu > 0$

$${}_0^{RL}\mathrm{I}_t^{(\nu)} f(t) = \frac{1}{\Gamma(\nu)} \int_0^t \frac{f(\tau)}{(t-\tau)^{1-\nu}} d\tau, \quad (9)$$

Riemann-Liouville derivative of fractional order $\nu > 0$

$${}_0^{RL}\mathrm{D}_t^{(\nu)} f(t) = \frac{d^n}{dt^n}\left[\frac{1}{\Gamma(n-\nu)} \int_0^t \frac{f(\tau)}{(t-\tau)^{\nu-n+1}} d\tau\right], \quad (10)$$

Caputo derivative of fractional order $\nu > 0$

$${}_0^{C}\mathrm{D}_t^{(\nu)} f(t) = \frac{1}{\Gamma(n-\nu)} \int_0^t \frac{f^{(n)}(\tau)}{(t-\tau)^{\nu-n+1}} d\tau \quad (11)$$

with the following conditions: $f(t) = 0$ for $t \le 0$, $f(0) = 0, f^{(1)} = f^{(2)} \ldots f^{(n)} = 0$.

The following formula presents the relationship between formula (10) and (11)

$${}_0^{RL}\mathrm{D}_t^{(\nu)} f(t) = {}_0^{C} D_t^{(\nu)} f(t) + \sum_{k=0}^{n-1} \frac{t^{k-\nu}}{\Gamma(k-\nu+1)} f^{(k)}(0). \quad (12)$$

Inserting formula (11) to the right side of equation (12) enables derivation of an equivalent to (10) formula for Riemanna-Liouville'a derivative of fractional order

$${}_0^{RL}\mathrm{D}_t^{(\nu)} f(t) = \sum_{k=0}^{n-1} \frac{t^{k-\nu} f^{(k)}(0)}{\Gamma(k-\nu+1)} + \frac{1}{\Gamma(n-\nu)} \int_0^t \frac{f^{(n)}(\tau)}{(t-\tau)^{\nu-n+1}} d\tau. \quad (13)$$

In formulas (9)-(13) ν is a real number such as $n-1 < \nu < n$, n denotes an integer number $n = \lceil \nu \rceil$.

The practical application advantage of Caputo fractional derivative (11) over Riemann-Liouville fractional derivative (10) is, that the first one enable defining initial conditions in terms of classical, integer order derivatives [11]. Therefore, the Riemann-Liouville derivative definition is used more often in theoretical consideration, in which initial conditions must be defined in terms of fractional order integrals [12].

V. Approximation of Fractional Order Derivatives and Integrals by Applying Gauss-Jacobi Quadrature

Using the weight function (2) and integration formula (7), we can "remove" the kernel of the integrand from the formula (9)

$${}_0^{RL}\mathrm{I}_t^{(\nu)} f(t) = \frac{1}{\Gamma(\nu)} \int_0^t \underbrace{(t-\tau)^{\nu-1}}_{\text{kernel}} f(\tau)\, d\tau,$$

substituting $\lambda = \nu - 1, \beta = 0$, we obtain

$$\int_{-1}^{1} (1-t)^{\lambda} f(t)\, dt \approx \sum_{k=1}^{n} w_k f(t_k) = \sum_{k=1}^{n} \frac{2^{\nu}}{(1-t_k^2)\left[P_n^{(\lambda,0)'} t_k\right]^2} f(t_k), \quad (14)$$

where w_k are the weights (8).

Transforming integration interval $[0,t]$ into $\langle -1,1\rangle$

$$\left(\frac{t-t_0}{2}\right)^{\nu}\int_{-1}^{1}\frac{f(u)}{(1-u)^{\lambda}}du$$

where

$$f(u)=f\left(\left(\frac{t-t_0}{2}\right)u+\left(\frac{t+t_0}{2}\right)\right),$$

we obtain formula (15), which can be applied for computing Riemann-Liouville and Caputo fractional derivatives with high accuracy. A formula for fractional integrals computation can be derived in a similar way.

In the formula (15) the difficult part of the integrand - the kernel - equipped with singularity and high increases of function values, is computed using different, much more accurate method - by applying formula for the weights w_k [2].

$$\begin{aligned}{}_{t_0}^{RL}\mathrm{D}_t^{(\nu)}f(t)&=\sum_{k=0}^{n-1}\frac{(t-t_0)^{k-\nu}f^{(k)}(t_0)}{\Gamma(k-\nu+1)}+\frac{1}{\Gamma(n-\nu)}\left(\frac{t-t_0}{2}\right)^{n-\nu}\int_{-1}^{1}\frac{f^{(n)}(u)}{(1-u)^{\nu-n+1}}du\\&=\sum_{k=0}^{n-1}\frac{(t-t_0)^{k-\nu}f^{(k)}(t_0)}{\Gamma(k-\nu+1)}+\frac{1}{\Gamma(n-\nu)}\left(\frac{t-t_0}{2}\right)^{n-\nu}\underbrace{\sum_{k=1}^{n}\frac{2^{\nu}}{(1-u_k^2)\left[P_n^{(\nu-n+1,0)'}u_k\right]^2}}_{w_k}f^{(n)}(u_k),\ n=\lceil\nu\rceil.\end{aligned}\tag{15}$$

VI. Methods of Finding Zeros of Jacobi Polynomials

Formula (7) suggests that the construction of Gauss-Jacobi Quadrature is limited to finding zeros x_k of Jacobi polynomial $P_n^{(\alpha,\beta)}(x_k)$ of order n and determining its derivative $P_n^{(\alpha,\beta)'}(x_k)$.

Polynomial of order n has n distinct zeros [13]. This rule extends for systems of orthogonal polynomials, including Jacobi polynomials. Proof of this theorem can be found in [14].

Chebyshev polynomials of I, II, III and IV kind are special cases of Jacobi polynomial for α and β with values $-0.5,-0.5,\ 0.5,0.5,-0.5,0.5$ and $0.5,-0.5$ respectively.

Zeros of Chebyshev polynomials called Chebyshev points are given by the following formulas [15]:

$$\begin{aligned}x_k&=\cos\frac{(k-0.5)\pi}{n},\\x_k&=\cos\frac{k\pi}{n+1},\\x_k&=\cos\frac{(k-0.5)\pi}{n+0.5},\\x_k&=\cos\frac{k\pi}{n+0.5},\ k=1,2,\dots n.\end{aligned}\tag{16}$$

Finding zeros of Jacobi polynomial with other values α and β is not easy.

However, a standard method for finding zeros of polynomials is an iteration algorithm called Newton-Raphson algorithm [16].

Iteration algorithms usually require first raw approximation for finding a zero. In case of Jacobi polynomial, it can be for example a zero of Chebyshev polynomial of 1^{st} kind (16).

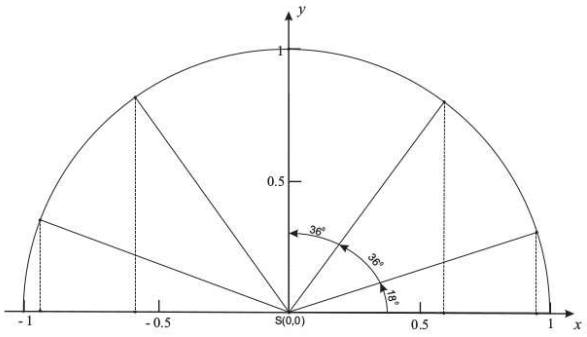

Fig. 3. Chebyshev points,$n=5$ of Chebyshev polynomial of the I kind.

Then, the Newton-Raphson method is used for finding highly accurate location of that zero.

This method is usually fast-convergent, especially for orthogonal polynomials.

Let s denote consecutive iteration. Each iteration of standard Newton-Raphson method requires computation of polynomial value and its derivative. Both values can be used for approximating k^{th} zero in the following way

$$x_k^{s+1}=x_k^s-P_n(s_k^s)/P_n'(x_k^s),\tag{17}$$

where $P_n^{(\alpha,\beta)}(x)$ is Jacobi polynomial computed using recurrent relationship (6).

Its derivative $P_n^{\prime(\alpha,\beta)}(x)$ can be computed by another recurrence relation

$$P_{n+1}'=P_n+(x-\alpha)P_n'-\beta_nP_{n-1}'.\tag{18}$$

A. Accuracy of the Standard Iteration Method

For the purpose of assessing accuracy of finding zeros of Jacobi polynomial P_n for arbitrary selected values of n by

applying standard iteration method, relative error measure e_r and norm $\|e_r\|_{L_\infty}$

$$\|e_r\|_{L_\infty} = max_i \frac{\|x_i - \hat{x}_i\|}{\|x_i\|}. \tag{19}$$

are used.

The norm (19) assess similarity of two vectors, e.g. with zeros values by applying standard iteration method $\hat{x}_i$ and their exact values. In both cases, there are applied standard double precision for computations and first raw approximations of zeros by applying formulas (16) proposed in [17].

TABLE I
Relative error $\|e_r\|_{L_\infty}$ for selected order n of Jacobi polynomial.

n	Chebyshev I	Chebyshev IV
50	2.89e-15	5.66e-15
100	9.55e-15	1.04e-14
200	1.37e-14	1.10e-14
300	3.18e-14	2.02e-14
400	2.19e-14	4.11e-14
500	2.00e-14	1.19e-14
1000	5.36e-14	7.39e-14

Application of first raw approximation of zeros by applying formulas (16) in Newton-Raphson method enabled finding n distinct zeros with double precision exactness of each zero after 8-10 iterations.

B. Enhancements of the Standard Iteration Method

1) More accurate first raw approximations of zeros: Application of more accurate first raw approximations of zeros can reduce an amount of required iterations for finding an exact location of a zero in Newton-Raphson method.

As it is to see in figure 4 with plots of Jacobi polynomials of order 5 and 6, in the middle part, the polynomials are similar to sine and cosine functions. Near boundary, at $x = 1$ they become compressed. Therefore, we can draw a conclusion that an amount of zeros of Jacobi polynomial increases towards end of the interval $[-1, 1]$. This conclusion suggests that we should use different formulas for first raw approximations for zeros in Newton-Raphson method for middle and boundary parts of Jacobi polynomial.

According to [18] an universal formula for the middle part of the Jacobi polynomials, zeros $4, n-2$ for $k = 1, 2, 3, \ldots, n$ is (20).

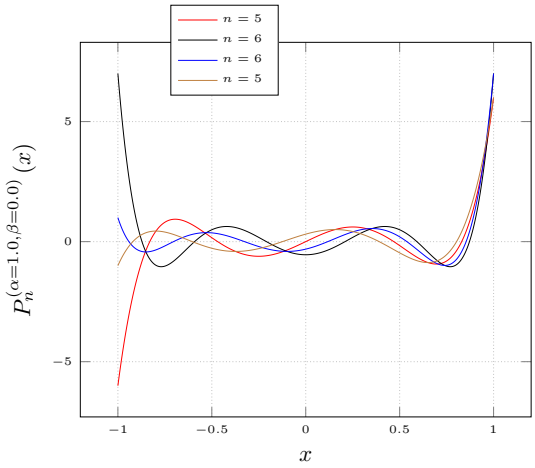

Fig. 4. Jacobi polynomials $n = 5$ and $n = 6$.

$$x_k = -\cos(\theta_k + \delta\theta_k) + O\left(n^{-4}\right) \tag{20}$$

where

$$\theta_k = \pi(2k + \beta - 0.5)/\sigma,$$
$$\delta\theta_k = \left[\left(0.25 - \beta^2\right)\cot\left(\frac{\theta_k}{2}\right) - \left(0.4 - \alpha^2\right)\tan\left(\frac{\theta_k}{2}\right)\right]/\sigma^2,$$
$$\sigma = 2n + \alpha + \beta + 1.$$

For approximating zeros for nodes $[1, 2, n-1, n]$ i.e. for both ends of the interval we can use formula by [18] that uses zeros of Bessel functions of order 5 $J^5_{\alpha,k}$, e.g. [19]

$$x_k = \cos(\theta_k + \delta\theta_k) + O\left(J^5_{\alpha,k} n^{-7}\right) \tag{21}$$

where

$$\theta_k = \frac{J^5_{\alpha,k}}{\nu},$$
$$\delta\theta_k = -\theta_k\left[\frac{4 - \alpha^2 - 15\beta^2}{720\nu^4}\left(\frac{J^2_{\alpha,k}}{2}\right) + \alpha^2 - 1\right],$$
$$\nu = 0.5\sqrt{\sigma^2 + (1 - \alpha^2 - 3\beta^2)/3}.$$

2) Application of reflection formula: For finding zeros of Jacobi polynomial for $\alpha \neq \beta$, there can be applied reflection formula

$$P_n^{(\alpha,\beta)}(-x) = (-1)^n P_n^{(\beta,\alpha)}(x),$$
$$\frac{d}{dx}\left[P_n^{(\alpha,\beta)}(-x)\right] = (-1)^{n-1}\frac{d}{dx}\left[P_n^{(\beta,\alpha)}(x)\right], \tag{22}$$

which results from formula for Jacobi polynomial of order n (1).

It enables reducing computational effort of polynomial values to the right part of the interval, i.e. $x \in [0, 1]$. It means that we only require to compute zeros from the right part of the interval and copy them to the second one $x \in [-1, 0]$.

3) Application of arbitrary precision: enables increasing overall accuracy of computations.

To be able to solve a difficult numerical problem according to a set goal, we have to make some crucial decisions regarding applied hardware, programming tools and techniques for that purpose. This includes a selection of an appropriate computer programming language, mathematical libraries and hardware.

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. And it is the main task for a computer scientist, because the newest hardware gives the opportunity to solve many problems, which appeared "unsolvable" not long time ago. The application of *infinite 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.

In this paper there is presented application of arbitrary precision for increasing accuracy of computations.

The standard double precision computer arithmetic was replaced by arbitrary precision for most parts. This move made possible unlocking full potential of developed algorithms by using available hardware.

Double precision arithmetic commonly applied in scientific numerical calculations is optimized for speed and has many flaws which influence negatively 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 only defines that the results must be rounded correctly to the destination's precision and not defines 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 returns significantly different results depending on the implementation of the IEEE standard.

Arbitrary precision makes possible for the user to choose a precision for calculation and for each variable storing a value and it is nor machine or IEEE standard types depended. It is only limited by accessible hardware.

Arbitrary precision can be applied for calculating important constants like π or increase general accuracy of the mathematical computations. 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 derivatives and integrals calculating.

Still, application of arbitrary precision has drawbacks:

Arbitrary precision is simulated and therefore, depending on chosen precision, calculations with the help of it require more time to complete than by applying standard data types optimized to run on standard processors - even with the use of FPGAs (field programmable gate arrays), which can be fully programmed by the user.

Another challenge is a requirement of special computational algorithms which can handle different data structures.

Nevertheless, arbitrary precision application already became a part of standard computations without the consent of the user. The process named *constant folding with arbitrary precision* is used in preprocessing phase to increase the accuracy of constants before they can be handled with standard precision data types. This procedure [20] involves replacing constant expressions with their final value in order to reduce the need of recomputing the same result every time the program executes the code line containing the constant. When the compiler flag -01 is inserted GCC complier uses the GNU MPFR library with version 4.3 to handle constant folding and evaluate mathematical applied to constants at compile time at arbitrary precision.

The GNU MPFR library is an arbitrary precision package for C/C++ [21] and is based on the GNU Multiple-Precision Library (GMP) [22]. MPFR supports arbitrary precision floating-point variables and provides exact rounding of all implemented mathematical functions [23]. The code is portable, i.e. it will produce the same result independently from the hardware.

The GNU MPFR library is written in C and thus it can not use operator overloading. Even the most basic arithmetic operations have to be conducted using function calls. Therefore MPFR includes multiple functions for each operation and for each supported data type.

C. Results

Table II presents accuracy of computed zeros of Jacobi polynomial in form of relative error (19) calculated in respect to the exact values obtained by applying Czebyshev points (16) for 50, 100, 500 and 1000 digits precision.

It is worth noting, that the proposed enhancements of the standard iteration method enables finding zeros with arbitrary precision. The level of exactness depends on how high precision is selected applied for computations.

Additionally, application of more accurate first raw approximations of zeros (20) and (21) before Newton-Rapshon method starts, decreases an amount of required iterations until exact zero position is found.

Computation time complexity of running program is presented in figure 5. The time depends directly on the hight of precision selected for computations: for polynomial order $n < 500$ and up to 100 digits precision, the time is similar to double precision computations, for $n > 500$ and more than 100 digits precision, the complexity is 2^n, and $n!$ is for 1000 and more digits precision.

VII. Methods of Jacobi Weights Computation

A. Standard Approach

A standard approach to the problem of Jacobi weights computation is the direct use of formula (8).

In this formula proposed in [24], weight is computed by using value of derivative of Jacobi polynomial of order n and a value of Jacobi polynomial of order $n-1$.

TABLE II
RELATIVE ERROR $\|e_r\|_{L_\infty}$ OF FOUND ZEROS OF $P_n(x)$ FOR SELECTED n IN RESPECT TO (16).

	Czebyszew I			Czebyszew II			Czebyszew III			Czebyszew IV		
n	50	time(s)	iter	100	time(s)	iter	500	time(s)	iter	1000	time(s)	iter
50	1.50e-49	0.047	6	3.77e-99	0.053	7	6.24e-499	0.102	7	8.31e-647	0.150	6
100	6.84e-49	0.0168	6	6.74e-99	0.195	7	1.40e-498	0.392	7	5.24e-641	0.562	7
200	9.52e-49	0.588	7	7.43e-99	0.765	6	1.35e-498	1.323	7	1.89e-641	2.073	7
300	1.66e-48	1.236	6	2.22e-99	1.673	6	1.31e-498	3.012	7	3.64e-640	4.514	7
500	1.49e-48	3.089	6	1.59e-99	4.349	6	9.05e-498	8.034	7	1.58e-648	12.598	7
1000	5.61e-48	11.867	6	2.60e-98	15.913	6	7.96e-498	34.37	7	8.99e-709	49.015	7
2000	7.00e-48	47.353	6	5.45e-98	63.445	6	3.78e-497	126.664	7	8.99e-709	49.173	7

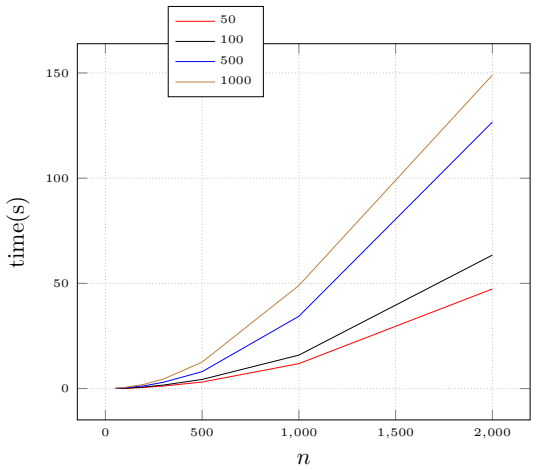

Fig. 5. Computation time complexity of finding zeros of Jacobi polynomials of selected order n with 50, 100, 500 and 100 digits precision.

However, due to $1-x_k^2$ expression occurrence in the denominator of (8), high truncation error is expected if standard double precision is applied.

Additionally, deducing from 4, an amount of zeros in Jacobi polynomials increases quadratically with increasing n (hence the distance between then decreases) towards bounds of integration interval $[-1,1]$. It causes the following problem: if standard double precision is applied, for enough large n, zeros become indistinguishable.

B. Enhancement of Standard Approach

Instead of formula (8), an equivalent formula is suggested to apply

$$w_k = 2^{\alpha+\beta+1}\frac{\Gamma(\alpha+n+1)\,\Gamma(\beta+n+1)}{n!\Gamma(\alpha+\beta+n+1)}\frac{1}{\left[\frac{d}{d\theta}P_n(\cos\theta_k)\right]^2}, \tag{23}$$

in which $\frac{d}{d\theta}P^{(\alpha,\beta)}$ is derivative of Jacobi polynomial of order n and $\theta_k = \cos^{-1}x_k$, x_k are zeros of Jacobi polynomial of order n.

The conversion into trigonometric functions space has been proposed by [25]. It enables omitting the expression $1-x_k^2$ and hence reduce truncation error at the same time.

C. Results

Table III presents accuracy of Jacobi weights computation in form of relative error (19) calculated in respect to the exact values obtained by applying Czebyshev weights [15] for 50, 100, 500 and 1000 digits precision. Application arbitrary precision enables computing Jacobi weights with high-accuracy. However, computational accuracy is not so straightforward depended on an amount of digits of precision applied for computations. It is caused by the fact that computational complexity is increased by zeros of finding of the polynomial of a given order.

General time complexity of Jacobi weights w_k computation presented in figure 6 depends directly on precision applied for computations: for $n<500$ and up to 100 digits precision it is similar to double precision, $n>500$ and more than 100 digits precision it is 2^n, and $n!$ for more than 1000 digits precision.

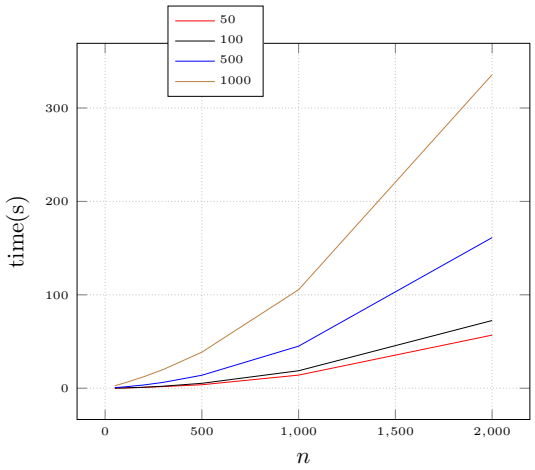

Fig. 6. Computation time complexity of Jacobi weights for selected order n with 50, 100, 500 and 100 digits precision.

VIII. FRACTIONAL ORDER DERIVATIVES AND INTEGRALS COMPUTATION

The most useful method of presenting practical capabilities of algorithms proposed in the following paper is computation of values of integrals and derivatives of fractional order of two exponential functions by applying formulas (9) and (13) with the help of Gauss-Jacobi Quadrature (15).

TABLE III
RELATIVE ERROR $\|e_r\|_{L_\infty}$ OF COMPUTED WEIGHTS w_k FOR SELECTED n IN RESPECT TO (16).

	Czebyszew I			Czebyszew II			Czebyszew III			Czebyszew IV		
n	50	czas(s)	iter	100	czas(s)	iter	500	czas(s)	iter	1000	czas(s)	iter
50	1.46e-46	0.060	8	1.63e-97	0.109	9	9.33e-326	0.640	10	1.14e-323	2.651	10
100	3.73e-46	0.256	8	2.49e-96	0.339	9	9.64e-317	1.440	10	4.51e-319	5.632	11
200	8.39e-45	0.910	8	1.31e-95	1.051	9	1.86e-318	3.469	10	1.97e-318	12.322	11
300	6.53e-45	1.781	8	1.23e-95	2.170	9	2.57e-317	6.252	10	3.40e-317	20.000	11
500	1.01e-45	3.728	8	3.91e-94	5.265	9	1.56e-317	13.895	11	2.06e-320	38.619	11
1000	8.47e-43	14.03	8	8.53e-94	18.710	9	2.72e-326	45.04	11	1.71e-349	105.491	11
2000	7.26e-42	56.852	8	1.14e-93	72.479	9	3.08e-318	161.266	11	4.74e-318	335.694	11

To assess accuracy of computations of fractional derivatives and integrals, it is required to computed exact values with high-accuracy for relative error computation. In case of fractional order derivative and integral computations, the effective accuracy assessment is difficult, sometimes not possible due to general lack of formulas for exact values.

Despite the availability of a handful of analytical formulas for fractional order $\nu = \frac{1}{2}$ and some computational formulas, they are accessible for selected types of functions only. Some other formulas are in form of series expansion only. As it is in case of exponential functions.

For the error (19) computation Mittag-Leffler function is used.

A. Mittag-Leffler Function Computation

The Mittag-Leffler function [26] is a direct generalization of the exponential function e^{at} and it plays a major role in fractional calculus. The one, two and three-parameter representations of the Mittag-Leffler function can be defined in terms of a power series as

$$E_\alpha(at) := \sum_{k=0}^{\infty} \frac{at^k}{\Gamma(\alpha k + 1)},\ \alpha > 0, \tag{24}$$

$$E_{\alpha,\beta}(at) := \sum_{k=0}^{\infty} \frac{at^k}{\Gamma(\alpha k + \beta)},\ \alpha, \beta > 0. \tag{25}$$

When $\beta = 1$, $E_{\alpha,1}(at) = E_\alpha(at)$.

$$E^\gamma_{\alpha,\beta}(at) := \sum_{k=0}^{\infty} \frac{(\gamma)_k}{\Gamma(\alpha k + \beta)} \frac{at^k}{k!},\ \alpha, \beta > 0, \tag{26}$$

in which $(\gamma)_k$ is Pochhammer symbol [27]

$$(\gamma)_k := \frac{\Gamma(\gamma + k)}{\Gamma(k)}.$$

When $\gamma = 1$, $E^1_{\alpha,\beta}(at) = E^\gamma_{\alpha,\beta}(at)$, and when $\gamma = \beta = 1$, $E^1_{\alpha,1}(at) = E_\alpha(at)$. Some particular cases of the Mittag-Leffler function are: $E_0(at) = \frac{1}{1-at}$, $E_1(at) = e^{at}$ $E_2(at) = cosh\sqrt{at}$, $E_{1,2}(at) = \frac{e^{at}-1}{t}$, $E_{2,2}(at) = \frac{sinh(at^{1/2})}{at^{\frac{1}{2}}}$, $E_{\frac{1}{2},2}(at) = e^{at^2}\, erfc(-at)$. Papers [28] and [29] present comprehensive knowledge of computing the Mittag-Leffler function and its first derivative.

To calculate the Mittag-Leffler fractional order derivative/integral we combine the Riemann-Liouville fractional derivative of the power function $(t - t_0)^p$, $p \in \mathbf{R}$ and the Mittag-Leffler function (24) or (25)

$${}_{t_0}D_t^{(\nu)} E_{\alpha,\beta}(at) = t^{-\nu} \sum_{k=0}^{\infty} \frac{\Gamma(k+1)\, at^k}{\Gamma(k+1-\nu)\,\Gamma(\alpha k + \beta)} \tag{27}$$

and for calculations of fractional order integral of the Mittag-Leffler function, we apply the following formula

$${}_{t_0}D_t^{(-\nu)} E_{\alpha,\beta}(at) = at^\nu \sum_{k=0}^{\infty} \frac{(at^\nu)^k}{\Gamma(\alpha k + \beta + \nu)}. \tag{28}$$

B. Computing Environment Configuration

All computations described in the paper were conducted using PC computer with Intel i7 2600K Processor, 8 GB of RAM armed with full open-source operating system and compiler: Ubuntu 16.04 LTS 64-bit Linux OS and gcc 5.4.0 compiler.

The computer system can be described as high-performance, because its computational power is enormous. However, it dates from 2011. Therefore it also can be described as commonly used.

To complete the picture that is important for time complexity assessment, calculations were also conducted on an older notebook with Intel Core 2 Duo Processor 2.4 GHz with 4 GB of memory from 2008 with exactly the same software configuration.

C. Results

Figures 7 and 9 present plots of fractional integral and derivative of two exponential functions. Figures 8 and 10 present plots of relative error (19) for $n = 8, 16, 32$ computed with 100 digits precision.

As it is to see, steady 100 digits accuracy can be obtained by applying Jacobi polynomial of order $n = 32$ for computations.

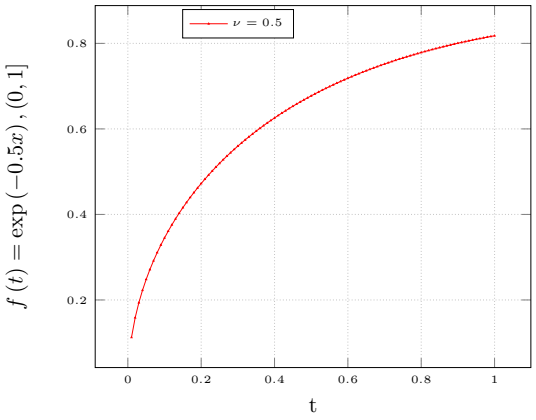

Fig. 7. Plot of fractional integral of order $\nu = 0.5$ of function $f(t) = \exp^{-0.5x}$ in interval $(0, 1]$.

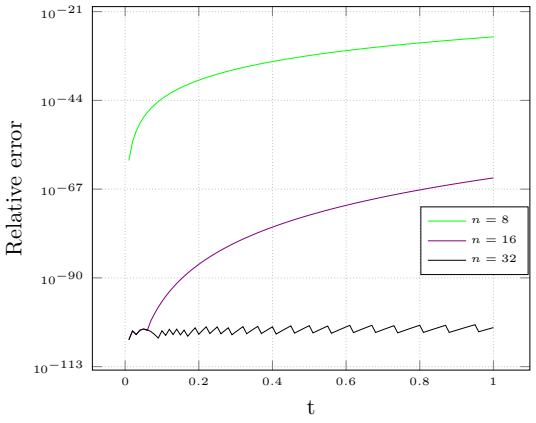

Fig. 8. Plot of relative error of fractional integral of order $\nu = 0.5$ of function $f(t) = \exp^{-0.5x}$ in interval $(0, 1]$.

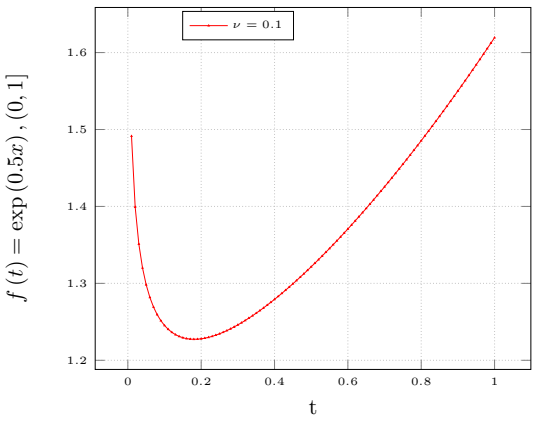

Fig. 9. Plot of fractional integral of order $\nu = 0.5$ of function $f(t) = \exp^{0.5x}$ in interval $(0, 1]$.

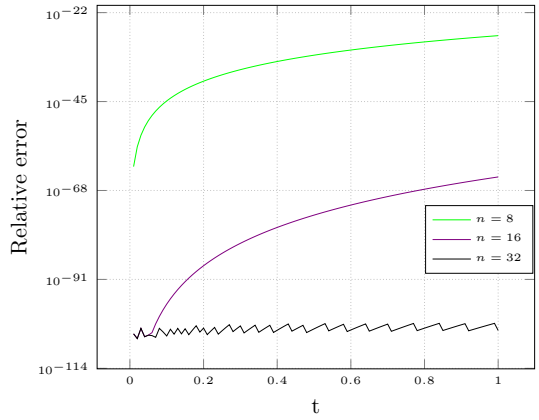

Fig. 10. Plot of relative error of fractional integral of order $\nu = 0.5$ of function $f(t) = \exp^{0.5x}$ in interval $(0, 1]$.

IX. Conclusions

The aim of the following research was to develop the most efficient and accurate numerical algorithms for Gauss-Jacobi Quadrature nodes and weights computation. In the paper, we discuss efficient mathematical formulas for nodes and wights computations and accurate methods of their computer implementations.

Results of numerical experiments presented in the paper prove that application of more accurate raw first approximations of zeros in the standard iteration method of polynomial zeros finding leads to significant decrease of an mount of iterations required for finding high-accurate zero location.

The proposed enhancements for the standard iteration method of determining zeros of Jacobi polynomial enables decreasing an amount of iterations required for finding each zero and increasing their accuracy many hundred times; changes to the weight function formula and its computation by applying cosine function enables massive reduction of truncation errors and increasing overall accuracy of computations.

The enhanced methods programmed by applying excellent arbitrary precision libraries GNU GMP and GNU MPFR together with C++ programming language enable computation of Gauss-Jacobi Quadratures and nodes and wights with arbitrary precision, i.e. with precision limited only by accessible hardware (computer memory).

Results of computations of fractional order derivatives and integrals of example exponential functions prove that modified Gauss-Jacobi Quadrature that uses high-accurately computed nodes and weight enables their computation with steady 100-digits precision with only 32 sampling points at most. It is worth nothing that standard numerical integration methods', e.g. Newton-Cotes quadratures' accuracy is limited to a few digits at best for the same computations [30], [31].

High accurately computed nodes of Jacobi polynomial are an excellent starting point for research on mitigation of Runge phenomenon. High-accurate methods of fractional order derivatives and integrals computation can be useful for constructing more efficient and accurate spectral methods

for solutions of fractional differential equations. This in turn enables more accurate simulations of physical processes and systems.

ACKNOWLEDGEMENT

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

REFERENCES

[1] S. Wolfram. (2005) The history and future of special functions. http://www.stephenwolfram.com/publications/history-future-special-functions/.

[2] 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*, vol. 62, no. 4, pp. 723–733, 2014.

[3] D. W. Brzeziński, "Comparison of fractional order derivatives computational accuracy - right hand vs left hand definition," *Applied Mathematics and Nonlinear Sciences*, vol. 2, no. 1, pp. 237–248, 2017.

[4] A. Townsend, S. Olver *et al.* (2018) Fastgaussquadrature.jl. https://github.com/ajt60gaibb/FastGaussQuadrature.jl#fastgaussquadraturejl/.

[5] A. Glaser, X. Liu, and V. Rokhlin, "A fast algorithm for the calculation of the roots of special functions," *J. Sci. Comput.*, vol. 29, pp. 1420–1438, 2007.

[6] N. Hale and A. Townsend, "Fast and accurate computation of gauss-legendre and gauss-jacobi quadrature nodes and weights," Oxford Centre for Collaborative Applied Mathematics, 2012, oCCAM Preprint Number 12/79.

[7] M. Abramowitz and I. A. Stegun, *Handbook of Mathematical Functions. Applied Mathematics Series*. Cambridge University Press, 1968.

[8] G. Szegö, *Ortogonal Polynomials*. American Mathematical Society, Colloquiam Publications, Volume 23, 1939.

[9] D. Funaro, *Polynomial Approximation of Differential Equations*. Springer-Verlag., 1992.

[10] M. D. Ortigueira, J. A. T. Machado, and J. S. da Costa, "Which differintegration?" *IEE Proceedings - Vision, Image and Signal Processing*, vol. 152, no. 6, 2005.

[11] Y. Povstenko, *Linear Fractional Diffusion-Wave Equation for Scientists and Engineers*. Cham, Heidelberg, New York, Dodrecht, London: Birkhauser, Springer, 2015.

[12] J. Jiang, D. Cao, and H. Chen, "Boundary value problems for fractional differential equation with causal operators," *Applied Mathematics and Nonlinear Sciences*, vol. 1, no. 1, pp. 11–22, 2016.

[13] D. Xin, *Numerical Methods for Stochastic Computations: A Spectral Method Approach*. Princeton University Press Press, 2000.

[14] E. D. Rainville, *Special Functions*. Chelsea Publications Company, 1960.

[15] J. C. Mason and D. C. Handcomb, *Chebyshev Polynomials*. Champan & Hall/CRC New York, 2003.

[16] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes: The Art of Scientific Computing, Third Edition*. Cambridge University Press, 2008.

[17] K. Petras, "On the computation of the gauss-legendre quadrature form with a given precision," *Jour. Comp. Appl. Math.*, vol. 112, pp. 253–267, 1999.

[18] W. Gautschi and C. Giordano, "Luigi gatteschi's work on asymptotics of special functions and their zeros," *Numerical Algorithms*, vol. 49, pp. 11–31, 2008.

[19] H. Gerber, "First hundert zeros of $j_0(x)$ accurate to 19 significant figures." *Math. Comp.*, vol. 23, pp. 319–322, 1969.

[20] N. Brisebarre and J. M. Müller, "Correctly rounded multiplication by arbitrary precision constants," *IEEE Transactions on Computers*, vol. 57, no. 2, pp. 165–174, 2008.

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

[22] T. Granlund *et al.*, *gmp: GMP is a free library for arbitrary precision arithmetic (version 6.0.0a)*, 2015, `https://gmplib.org/`.

[23] N. Brisebarre and J. M. Müller, "Correct rounding of algebraic functions," *Theoretical Informatics and Applications*, vol. 47, pp. 71–83, 2007.

[24] V. I. Krylov, *Priblizhennoe wychislenie integralov, 2e izd.* Mockba: Nauka, 1967.

[25] P. N. Schwarztrauber, "On computing the points and weights for gauss-legendre quadrature," *SIAM Jour. Sci.Comput*, vol. 24, pp. 945–954, 2002.

[26] P. Humbert and R. P. Agarwal, "Sur la fonction de mittag-leffler et quelques-unes de ses géneéralisations," *Bull. Sci. Math. Ser. II*, vol. 77, pp. 180–185, 1953.

[27] R. K. Saxenna, A. M. Mathai, and H. J. Haubold, "On generalized fractional kinetic equations," *Physica A: Statistical Mechanics and its Applications*, vol. 344, pp. 657–664, 2004.

[28] R. Gorenflo, J. Loutchko, and Y. Luchko, "Computation of the mittag-leffler function and its derivative," *Fractional Calculus & Applied Analysis*, vol. 4, pp. 491–518, 2002.

[29] R. Garrappa, "Numerical evaluation of two and three parameter mittag-leffler functions," *SIAM J. Numer. Anal.*, vol. 53, no. 3, pp. 1350–1369, 2015.

[30] 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*, vol. 1, no. 1, pp. 23–43, 2016.

[31] 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*, vol. 40, pp. 151–162, 2016.