J Math Chem (2016) 54:2022–2047
DOI 10.1007/s10910-016-0668-x

ORIGINAL PAPER

Boys function evaluation on graphical processing units

Grzegorz Mazur[1] · **Marcin Makowski**[1] · **Roman Łazarski**[2]

Received: 9 December 2015 / Accepted: 13 July 2016 / Published online: 21 July 2016

Abstract We propose a computational scheme for evaluating the Boys function on General Purpose Graphical Processing Units (GPGPUs). The scheme combines the polynomial and rational approximations, downward and upward recursions, and asymptotic approximations in a manner that facilitates efficient usage of hardware resources. Explicit formulas and implementation details are presented for two standard levels of accuracy.

Keywords Boys function · GPGPU · Rational approximation · Two-electron integrals

1 Introduction

The family of Boys functions [2] plays a role of great importance in calculations of molecular integrals over Gaussian-type basis sets. Namely, they appear in the one-electron nuclear attraction and two-electron Coulomb interaction polycenter integrals [6]. The sheer number of integrals of the latter type required in typical quantum-chemical calculations calls for very efficient techniques of evaluating the underlying Boys functions.

✉ Grzegorz Mazur
mazur@chemia.uj.edu.pl

Marcin Makowski
makowskm@chemia.uj.edu.pl

Roman Łazarski
lazarski.roman@gmail.com

1 Faculty of Chemistry, Jagiellonian University, ul. Ingardena 3, Kraków, Poland

2 Institute of Material Science and Material Technology, Friedrich Schiller University, Löbdergraben 32, 07743 Jena, Germany

Unfortunately, the n-th order Boys function

$$F_n(x) = \int_0^1 t^{2n} e^{-xt^2} dt \tag{1}$$

cannot be expressed in closed analytical form. There exists a solid amount of numerical methods for its evaluation in the literature [4,5,7,9–11]. The most efficient in practical applications are those which rely on pretabulating function values on a dense grid of points and applying relatively short Taylor-type expansions [4,7,12]. It should be noted that such methods require, apart from the necessary arithmetic operations, some rather irregular memory accesses. This, depending on the particular computer architecture, may result in significant efficiency degradation. In the next sections, we analyze several possible ways of approximating the Boys function, focusing particularly on various aspects which may influence their efficient implementation on General Purpose Graphical Processing Units (GPGPUs).

Unfortunately, there is no easy way to estimate how the (numerical) accuracy of the results from the quantum-chemical calculations depend on the accuracy of the underlying Boys function approximation. Nevertheless, the general consensus is that the approximation error of order 10^{-12}–10^{-14} is fully acceptable in standard double-precision calculations. Hence, we aim at the accuracy of the order of 10^{-13} when considering numerical properties of a given approximation.

2 Boys function

2.1 Taylor expansion

Given that

$$\frac{\mathrm{d}F_n(x)}{\mathrm{d}x} = -F_{n+1}(x) \tag{2}$$

and

$$F_n(0) = \frac{1}{2n+1} \tag{3}$$

we easily obtain the explicit form of the Taylor expansion

$$\begin{aligned} F_n(x) &= \sum_{k=0}^{\infty} \frac{x^k}{k!} \frac{\mathrm{d}F_n}{\mathrm{d}x}(0) = \sum_{k=0}^{\infty} (-1)^k \frac{x^k}{k!(2k+2n+1)} \\ &= \sum_{k=0}^{k_{max}} (-1)^k \frac{x^k}{k!(2k+2n+1)} + R_{k_{max}+1} \end{aligned} \tag{4}$$

where

$$R_{k_{max}+1} = \frac{x^{k_{max}+1}}{(k_{max}+1)!(2k_{max}+2n+3)} \underset{k_{max}\to\infty}{\to} 0 \tag{5}$$

To determine the applicability range of the Taylor expansion, we first estimate the dependence of the summand on the expansion order k_{max}. After applying some crude

Table 1 Applicability of the Taylor expansion restricted to the first k_{max} terms

k_{max}	5	15	20	25	30	35	40	50	60	80	110
x_{max}	0.01	1.1	2.2	3.6	5.1	6.7	8.3	10.9	11.0	11.0	11.0
$\tilde{x}_{max}$	0.005	0.75	1.6	2.8	4.0	5.5	7.0	10.1	13.4	20.2	30.8

See text for the details

approximations, this yields

$$\tilde{x}_{max} = \exp\left(\frac{\log(\varepsilon)}{k_{max}} + \log(k_{max}) - 1\right) \tag{6}$$

where $\tilde{x}_{\max}$ is the lower bound of the maximum range in which the expansion holds the requested accuracy ε. While this estimation holds for any degree n of the Boys function, it does not account for numerical inaccuracies. In order to investigate this issue, we have employed standard double-precision arithmetics to check the actual range in which the accuracy estimation holds for several expansion orders and compared them to the values obtained from Eq. (6). Looking at the results collected in Table 1, we see that the estimated $\tilde{x}_{max}$ is smaller than, but fairly close to, the actual x_{max} for an expansion order up to $k_{max} = 50$. After this threshold, the numerical errors build up and degrade accuracy. Hence, the Taylor series cannot be used at values exceeding 11, and it still requires a fairly long expansion at values less than or equal to 11.

As a result, the straightforward Taylor expansion cannot be considered to be an effective method of calculating approximations to the Boys function for any reasonable domain.

2.2 Recurrence relations

After integrating Eq. (1) by parts, we obtain the downward recurrence relation

$$F_n(x) = \frac{2xF_{n+1}(x) + e^{-x}}{2n+1} \tag{7}$$

which can be recast into the upward recurrence relation

$$F_{n+1}(x) = \frac{(2n+1)F_n(x) - e^{-x}}{2x}. \tag{8}$$

Additionally

$$F_0(x) = \int_0^1 e^{-xt^2} dt = \frac{1}{\sqrt{x}} \int_0^{\sqrt{x}} e^{-u^2} du \tag{9}$$

where $u = \sqrt{x}t$. The resulting form can be easily expressed in terms of the error function

$$\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt \tag{10}$$

as

$$F_0(x) = \frac{\sqrt{\pi}}{2\sqrt{x}} \mathrm{erf}(\sqrt{x}). \tag{11}$$

2.2.1 Upward recursion

Together, Eqs. (8) and (11) may seem to form an effective recursive scheme, given that erf(x) can be efficiently calculated. Unfortunately, the formula given by Eq. (8) cannot be universally applied in the case of finite arithmetic, as it is numerically unstable for small values of the argument. This stems from the fact that for small values of the argument

$$F_n(x) = \frac{2xF_{n+1}(x) + e^{-x}}{2n+1} \approx \frac{e^{-x}}{2n+1}. \tag{12}$$

Hence, in the numerator of Eq. (8) we subtract two almost equal numbers, which may lead to numerical instability. For the orders of the Boys function considered, the required accuracy in double precision is obtained only for $x \gtrsim 6$, as verified by numerical experiments.

2.2.2 Downward recursion

Let us recall Eq. (7), which gives a formula for downward recursion. It has to be noted that the downward recursion, despite being obtained from the same relationship as the upward one, exhibits strikingly different numerical properties. To demonstrate that, let us assume that an approximation $\tilde{F}_{n+1}(x)$ of the Boys function $F_{n+1}(x)$ differs from the exact value by Δ_{n+1}. Then, it follows from Eq. (7) that

$$\begin{aligned}\Delta_n &= \frac{e^{-x} + 2x(F_{n+1}(x) + \Delta_{n+1})}{2n+1} - F_n(x) \\ &= \frac{e^{-x} + 2xF_{n+1}(x)}{2n+1} + \frac{2x\Delta_{n+1}}{2n+1} - F_n(x) \\ &= F_n(x) + \frac{2x\Delta_{n+1}}{2n+1} - F_n(x) \\ &= \frac{2x\Delta_{n+1}}{2n+1}\end{aligned} \tag{13}$$

which, after k-fold repetition, yields

$$\Delta_n = \frac{(2x)^k \Delta_{n+k}}{(2n+1)(2n+3)\dots(2n+2k-1)}. \tag{14}$$

Given that

$$\lim_{k\to\infty} \frac{(2x)^k}{(2n+1)(2n+3)\dots(2n+2k-1)} = 0, \tag{15}$$

we are able to find values of k that are large enough to obtain Δ_n (calculated from Eq. (14)) smaller than the required precision. Obviously, convergence becomes slower as we consider larger values of x. It should be noted that such an approach is well known as the Miller's algorithm [1] and has been already applied to the evaluation of several special functions.

In order to use the method described above, we have to first decide how to approximate F_{n+k}, and then estimate how large k must be for a given n. We selected the following crude approximation

$$F_{n+k}(x) = \sqrt{\frac{\pi}{4x}}, \tag{16}$$

trading (initial) accuracy for the low computational cost.

Rewriting Eq. (14)

$$\Delta_n = \frac{(2x)^k \Delta_{n+k}}{2^k(n+1/2)(n+3/2)...(n+k-1/2)}. \tag{17}$$

and estimating the right-hand side from above yields

$$\Delta_n \lesssim \frac{x^k \Delta_{n+k}(n-1)!}{(n+k-1)!} \tag{18}$$

we obtain

$$\ln(\Delta_n) \lesssim k\ln(x) + \ln(\Delta_{n+k}) + \ln((n-1)!) - \ln((n+k-1)!). \tag{19}$$

Applying crude upper and lower bounds for the factorial

$$1 + n(\ln n - 1) \le \ln(n!) \le 1 + (n+1)(\ln(n+1) - 1), \tag{20}$$

and taking advantage of the fact that $\Delta_{n+k} < 1$ we may simplify the estimate of the right-hand side to

$$\ln(\Delta_n) \lesssim k\ln x + n\ln n - (n+k)\ln(n+k-1) + k. \tag{21}$$

which for values of k large with respect to n (corresponding to large values of x) yields

$$\ln(\Delta_n) - n\ln(n) \lesssim k(\ln x - \ln k + 1). \tag{22}$$

and finally

$$\exp\left(\frac{\ln(\Delta_n) - n\ln(n)}{k}\right) \lesssim e\frac{x}{k} \tag{23}$$

The argument of the exponential function is negative, hence the left-hand side approaches 1 from the above for large values of k. Hence, we may expect that the upper estimate on the right-hand side depends linearly on x. Numerical experiments seem to support this conclusion.

Linear fit

$$n_t = a_n x + b_n \tag{24}$$

where n_t denotes $n + k$ yields

$$a_n = -0.10453 \cdot n + 3.68823 \tag{25}$$
$$b_n = 0.76625 \cdot n + 16.00000 \tag{26}$$

where the last coefficient was artificially increased to ensure that we obtained the upper bound. This way, the explicit form of the estimated starting index for the downward recursion reads

$$n_t = (-0.10453 \cdot n + 3.68823)x + (0.76625 \cdot n + 16.00000). \tag{27}$$

We just note in passing that our results do not agree with those in [5]. This is so despite the fact that detailed numerical tests seem to confirm validity of our approximation.

2.3 Asymptotics

For large values of the argument we can approximate the Boys function with

$$\tilde{F}_n(x) = \int_0^\infty t^{2n} e^{-xt^2}\,\mathrm{d}t = \frac{(2n-1)!!}{2^{n+1}}\sqrt{\frac{\pi}{x^{2n+1}}}. \tag{28}$$

where the approximation error can be estimated from

$$\tilde{F}_n(x) - F_n(x) = \frac{\Gamma(2n+1, x^2)}{x^{4n+2}}. \tag{29}$$

Specifically for $n = 0$ we get

$$\tilde{F}_0(x) - F_0(x) = \frac{\sqrt{\pi}\,\mathrm{erfc}\,(\sqrt{x})}{2\sqrt{x}}. \tag{30}$$

For $x \gg 1$

$$\frac{\sqrt{\pi}\,\mathrm{erfc}\,(\sqrt{x})}{2\sqrt{x}} \leq \frac{e^{-x}}{2x} \tag{31}$$

which shows that the estimated error decreases quickly as x grows. Actual numerical tests show that an accuracy better than 10^{-13} may be expected from an asymptotic formula for $F_0(x)$ when x is larger than 26.

Higher order Boys functions may be obtained from F_0 using upward recursion in its asymptotic form

$$\tilde{F}_{n+1}(x) = \frac{2n+1}{2x}\tilde{F}_n(x). \tag{32}$$

2.4 Minimax approximation

To get a useful approximation for the non-asymptotic region, the Remez algorithm[3] was used. This method aims to find a polynomial or rational function which brings the least error on a given variable range. Balanced distribution of the error is what distinguishes this method from the Taylor series or Padé-type approximation, which are focused on a single expansion point.

The concept of the Remez algorithm can be introduced in a few points. Let us consider a function $F(x)$ that will be approximated by $R(x)$, with nominator/denominator order n/m, in the range $[a, b]$. The following steps are to be done:

1. Choose $n+m+2$ points, where $R(x)$ will pass through $F(x)$
2. Find coefficients of the rational function by solving a system of equations with these points
3. Calculate error extrema. There will be $n+m+3$ of them, $n+m+1$ between the chosen points and additional 2 points placed between the extreme points and endpoints of the range
4. Choose $n+m+2$ of them
5. For each x set the new value to make errors equal (differing only by sign), obtaining new points
6. Construct a new rational function which passes through all of them
7. Repeat procedure from step 3 until satisfactory precision is achieved

For our purposes we used an efficient implementation of the Remez algorithm in the *minimax* program, which is provided by boost C++ libraries.

To improve convergence of the procedure, we fitted Boys functions multiplied by damping factor in the form of e^{ax}. In this respect our approach differs from Ref. [8], where, instead of applying a damping function, specific form of the Remez algorithm, which takes the asymptotic behaviour of the Boys function into account, was applied.

Table 2 Set of approximations used to evaluate the Boys functions using double-precision arithmetics for various highest orders required (N)

N	Range	Approximation type	Damping factor	Recursion type
0	[0, 13)	Rational [8/9] or polynomial [18]	$e^{-x/3}$	Not applicable
	[13, 28)	Rational [8/9] or polynomial [20]	$e^{-x/3}$	Not applicable
	[28, ∞)	Asymptotic		Not applicable
1	[0, 13)	Rational [8/9] or polynomial [18]	$e^{-x/2}$	Downward
	[13, 28)	Rational [8/9] or polynomial [20]	$e^{-x/3}$	Upward
	[28, ∞)	Asymptotic		Upward
2	[0, 13)	Rational [8/9] or polynomial [16]	$e^{-x/2}$	Downward
	[13, 28)	Rational [8/9] or polynomial [20]	$e^{-x/3}$	Upward
	[28, ∞)	Asymptotic		Upward
3	[0, 13)	Rational [8/8] or polynomial [16]	$e^{-x/2}$	Downward
	[13, 28)	Rational [8/9] or polynomial [20]	$e^{-x/3}$	Upward
	[28, ∞)	Asymptotic		Upward
4	[0, 13)	Rational [9/8] or polynomial [22]	e^{-x}	Downward
	[13, 28)	Rational [8/9] or polynomial [20]	$e^{-x/3}$	Upward
	[28, ∞)	Asymptotic		Upward
5	[0, 13)	Rational [9/8] or polynomial [21]	e^{-x}	Downward
	[13, 28)	Rational [8/9] or polynomial [20]	$e^{-x/3}$	Upward
	[28, ∞)	Asymptotic		Upward
6	[0, 13)	Rational [9/8] or polynomial [20]	e^{-x}	Downward
	[13, 28)	Rational [8/9] or polynomial [20]	$e^{-x/3}$	Upward
	[28, ∞)	Asymptotic		Upward
7	[0, 13)	Rational [9/8] or polynomial [20]	e^{-x}	Downward
	[13, 28)	Rational [8/9] or polynomial [20]	$e^{-x/3}$	Upward
	[28, ∞)	Asymptotic		Upward
8	[0, 13)	Rational [9/8] or polynomial [19]	e^{-x}	Downward
	[13, 28)	Rational [8/9] or polynomial [20]	$e^{-x/3}$	Upward
	[28, ∞)	Asymptotic		Upward

See text for details

3 Implementation

3.1 GPGPU-specific aspects

General Purpose Graphical Processing Units are becoming ubiquitous in computational centres on the promise of significant calculation speedup at relatively low cost. Unfortunately, the promise is not an easy one to fulfill. In practice, often substantial rework is required due to significant hardware dissimilarity. This is also the case for an efficient GPGPU implementation of any special function.

First, it must be remembered that although the throughput of GPU memory is much higher than the host RAM memory, the access latencies are also high. Such latencies

Table 3 Set of approximations used to evaluate Boys functions using single-precision arithmetics for various highest orders required (N)

N	Range	Approximation type	Damping factor	Recursion type
0	[0, 13)	Rational [6/7] or polynomial [14]	$e^{-x/3}$	Not applicable
	[13, ∞)	Asymptotic		Not applicable
1	[0, 13)	Rational [6/7] or polynomial [14]	$e^{-x/2}$	Downward
	[13, ∞)	Asymptotic		Upward
2	[0, 13)	Rational [6/7] or polynomial [14]	$e^{-x/2}$	Downward
	[13, ∞)	Asymptotic		Upward
3	[0, 13)	Rational [6/7]	$e^{-x/2}$	Downward
	[0, 8)	Polynomial [10]	$e^{-2x/3}$	Downward
	[8, 13)	Polynomial [8]	$e^{-x/3}$	Upward
	[13, ∞)	Asymptotic		Upward
4	[0, 13)	Rational [6/7]	$e^{-x/2}$	Downward
	[0, 8)	Polynomial [12]	$e^{-2x/3}$	Downward
	[8, 13)	Polynomial [8]	$e^{-x/3}$	Upward
	[13, ∞)	Asymptotic		Upward
5	[0, 8)	Rational [4/5] or polynomial [10]	$e^{-2x/3}$	Downward
	[8, 13)	Rational [2/3] or polynomial [8]	$e^{-x/3}$	Upward
	[13, ∞)	Asymptotic		Upward
6	[0, 8)	Rational [4/5] or polynomial [10]	$e^{-2x/3}$	Downward
	[8, 13)	Rational [2/3] or polynomial [8]	$e^{-x/3}$	Upward
	[13, ∞)	Asymptotic		Upward
7	[0, 8)	Rational [4/5] or polynomial [10]	$e^{-2x/3}$	Downward
	[8, 13)	Rational [2/3] or polynomial [8]	$e^{-x/3}$	Upward
	[13, ∞)	Asymptotic		Upward
8	[0, 8)	Rational [4/5] or polynomial [10]	$e^{-2x/3}$	Downward
	[8, 13)	Rational [2/3] or polynomial [8]	$e^{-x/3}$	Upward
	[13, ∞)	Asymptotic		Upward

For more details see the text

may be well hidden in arithmetic burden, provided that the memory access pattern is very regular, optimally resulting in consecutive threads accessing consecutive memory addresses. This is not the case for Boys function evaluation algorithms that rely on pretabulation and short Taylor expansions, where the access, due to distribution of function arguments, is rather random. As a result, arithmetic-heavy and memory-light implementation may be preferred.

Second, the performance of GPU units degrades a lot if so called warp divergence occurs. In modern GPU architectures the threads are grouped into warps of 16 or 32, which are executed in parallel on the same multi-processor. If all threads in a warp enter the same branch of the code, then full speed can be expected. On the contrary, if every branch is visited by some thread, the efficiency is such as if every thread

Table 4 Polynomial coefficients ($a_\mathrm{m}x^\mathrm{m}$) for rational approximations used to evaluate F_0 to F_2 using double-precision arithmetics

m	$F_0(x), x \in [0, 13)$		$F_0(x), x \in [13, 28)$	
	Nominator	Denominator	Nominator	Denominator
0	0.99999999999999995700241	1.0	1.02241837924423924866	1.0
1	0.110797197786631479568	0.110797197786377266728	0.114650580864037928698	0.136951673337325655053
2	0.0447039731090986429495	0.000259528667137113921197	0.0370559849570588636147	−0.0180199379732503914666
3	0.00173561872693777888307	−0.000366831695730316839546	0.00070264488753193594737	0.000681696308108648850972
4	0.000399414007760856993659	$-0.493272041137073499076 \times 10^{-5}$	0.000226242532551670921154	$-0.629509987120141096118 \times 10^{-5}$
5	$0.488275205694491957804 \times 10^{-5}$	$0.76237111832045313941 \times 10^{-6}$	$-0.106401343255264539364 \times 10^{-5}$	$-0.303867297213349637263 \times 10^{-6}$
6	$0.114315819494419468355 \times 10^{-5}$	$0.446404578868329808188 \times 10^{-8}$	$0.435705981848460995997 \times 10^{-6}$	$0.117435181167263649392 \times 10^{-7}$
7	$0.100290453231804032913 \times 10^{-8}$	$-0.128927743593561789607 \times 10^{-8}$	$-0.328585595083913821085 \times 10^{-8}$	$-0.187119412616382912596 \times 10^{-9}$
8	$0.101496827289892561636 \times 10^{-8}$	$0.333637026721052224101 \times 10^{-10}$	$0.230224133910752021883 \times 10^{-9}$	$0.15072842445210102341 \times 10^{-11}$
‘9		$-0.282456869187885785253 \times 10^{-12}$		$-0.506080089114735146947 \times 10^{-14}$

m	$F_1(x), x \in [0, 13)$		$F_2(x), x \in [0, 13)$	
	Nominator	Denominator	Nominator	Denominator
0	0.333333333333333343903809	1.0	0.199999999999999895504	1.0
1	−0.0307524706862147488255	0.00774258794288160515724	−0.0330288190512119548088	0.0491416190293807021025
2	0.0112709195324634180829	−0.00469869690623180240936	0.00634115076552028862305	−0.0033988198700741889733
3	−0.000605839430179249726908	$-0.121959609027647601215 \times 10^{-4}$	−0.000509053250273039707197	−0.000195090866338372474454
4	$0.945034785957639723001 \times 10^{-4}$	$0.111138619313123589917 \times 10^{-4}$	$0.447569124464951732549 \times 10^{-4}$	$0.628049935932727280036 \times 10^{-5}$
5	$-0.312045278230439621214 \times 10^{-5}$	$-0.73084355391951055271 \times 10^{-7}$	$-0.199968924134809166274 \times 10^{-5}$	$0.378469794927881112829 \times 10^{-6}$

Table 4 continued

m	$F_1(x), x \in [0, 13)$		$F_2(x), x \in [0, 13)$	
	Nominator	Denominator	Nominator	Denominator
6	$0.271978126968244926363 \times 10^{-6}$	$-0.157370468737939165095 \times 10^{-7}$	$0.988949573195282803189 \times 10^{-7}$	$-0.992449720139712244 85 \times 10^{-8}$
7	$-0.462539908969141806287 \times 10^{-8}$	$0.410078145345644005506 \times 10^{-9}$	$-0.215528408817525733515 \times 10^{-8}$	$-0.429332907627276088202 \times 10^{-9}$
8	$0.236715461483360410 64 \times 10^{-9}$	$-0.194442657318426506569 \times 10^{-12}$	$0.570546583786746380737 \times 10^{-10}$	$0.180990972805032721716 \times 10^{-10}$
9		$-0.639489344621524669492 \times 10^{-13}$		$-0.182183510048296788429 \times 10^{-12}$

See text for details

Table 5 Polynomial coefficients ($a_{\mathrm{m}}x^{\mathrm{m}}$) for rational approximations used to evaluate F_3 to F_6 using double-precision arithmetics

m	$F_3(x), x \in [0, 13)$		$F_4(x), x \in [0, 13)$	
	Nominator	Denominator	Nominator	Denominator
0	0.142857142857140123028	1.0	0.111111111111103293052	1.0
1	−0.028445070492668875682	0.0786622843281632097491	−0.0139148832167939799845	−0.307052130771314686981
2	0.00448013774134009182352	−0.00108133056044008381254	0.00162696748977928592813	0.0424983395858807281545
3	−0.000386775982963851036211	−0.000237037804224883588737	$-0.662119157028470038327 \times 10^{-4}$	−0.00346361103289143243358
4	$0.282373440418903158296 \times 10^{-4}$	$-0.140764551127625276 79 \times 10^{-5}$	$0.382534107683271793372 \times 10^{-5}$	0.000181816669094072273715
5	$-0.131362432259290846434 \times 10^{-5}$	$0.356787924535046051312 \times 10^{-6}$	$-0.177532033480554620673 \times 10^{-7}$	$-0.62947667863295986372 \times 10^{-5}$
6	$0.523742625966482847805 \times 10^{-7}$	$0.275903056721026043953 \times 10^{-8}$	$0.310603292916945772772 \times 10^{-8}$	$0.140352027202573610484 \times 10^{-6}$
7	$-0.118309230356978606483 \times 10^{-8}$	$-0.382351840804085482137 \times 10^{-9}$	$0.750555827378940471864 \times 10^{-10}$	$-0.184220719784567695199 \times 10^{-8}$
8	$0.221839001441544997885 \times 10^{-10}$	$0.473716067612956790153 \times 10^{-11}$	$0.223659213588257882707 \times 10^{-11}$	$0.108946158463872109164 \times 10^{-10}$
9			$0.468739800011107957362 \times 10^{-13}$	

m	$F_5(x), x \in [0, 13)$		$F_6(x), x \in [0, 13)$	
	Nominator	Denominator	Nominator	Denominator
0	0.0909090909090900209071	1.0	0.0769230769230768028335	1.0
1	−0.0130301767374693016374	−0.297178097958598184711	−0.0118856947156836392233	−0.287847364637266699036
2	0.0013260888771875645004	0.0397938645158868996942	0.00112502456750246073178	0.0373186934862629924654
3	$-0.633878911074329746771 \times 10^{-4}$	−0.00313671182516856689049	$-0.565558758140881035171 1 \times 10^{-4}$	−0.00284702366635621453112
4	$0.270743454605471121973 \times 10^{-5}$	0.000159211576126655384656	$0.212098960130182099066 \times 10^{-5}$	0.000139818631832717886793
5	$-0.402585007659069045478 \times 10^{-7}$	$-0.532902241007815526731 \times 10^{-5}$	$-0.396038580921588426564 \times 10^{-7}$	$-0.452700644288278691678 \times 10^{-5}$

Table 5 continued

m	$F_5(x), x \in [0, 13)$		$F_6(x), x \in [0, 13)$	
	Nominator	Denominator	Nominator	Denominator
6	$0.1272805752760812009 \times 10^{-8}$	$0.1148649794485457607 76 \times 10^{-6}$	$0.725479358762515386567 \times 10^{-9}$	$0.943756500625232443777 \times 10^{-7}$
7	$0.143926976271825541774 \times 10^{-10}$	$-0.145757165393316312658 \times 10^{-8}$	$0.172003075424807920485 \times 10^{-11}$	$-0.115821140053297774 78 \times 10^{-8}$
8	$0.468185137961393994136 \times 10^{-12}$	$0.833486895658022964945 \times 10^{-11}$	$0.113330602043689503305 \times 10^{-12}$	$0.640567581849310007961 \times 10^{-11}$
9	$0.854314270571347003835 \times 10^{-14}$		$0.182699988705610865882 \times 10^{-14}$	

See text for details

Table 6 Polynomial coefficients ($a_\mathrm{m}x^\mathrm{m}$) for rational approximations used to evaluate F_7 to F_8 using double-precision arithmetics

m	$F_7(x), x \in [0, 13)$		$F_8(x), x \in [0, 13)$	
	Nominator	Denominator	Nominator	Denominator
0	0.06666666666666666478716	1.0	0.05882352941176470256\u200b92	1.0
1	−0.01075832801463152565\u200b67	−0.27902197904301076332\u200b3	−0.00972971755668365536806	−0.27066835635836070542\u200b6
2	0.00097384656191378881374\u200b4	0.03504991268148847877\u200b31	0.00085300753624023778654\u200b5	0.03296747139191046290\u200b36
3	$-0.494809728602069413199 \times 10^{-4}$	−0.00258977232264322921418	$-0.430790531180315330578 \times 10^{-4}$	−0.00236088139671802840686
4	$0.173253766996539336755 \times 10^{-5}$	0.00012313819144327816\u200b66	$0.144284871685167219013 \times 10^{-5}$	0.00010875539911763751\u200b2542
5	$-0.344256195863412298356 \times 10^{-7}$	$-0.385895118054266647982 \times 10^{-5}$	$-0.289460003401400700751 \times 10^{-7}$	$-0.330084932130781955135 \times 10^{-5}$
6	$0.499158519374328551681 \times 10^{-9}$	$0.778489043599582352\u200b47 \times 10^{-7}$	$0.374834280391756985694 \times 10^{-9}$	$0.644741698828656415338 \times 10^{-7}$
7	$-0.101387461661133031138 \times 10^{-11}$	$-0.924387662800835670148 \times 10^{-9}$	$-0.144526191277955085148 \times 10^{-11}$	$-0.741093143710942224069 \times 10^{-9}$
8	$0.308182280903291080284 \times 10^{-13}$	$0.494634610453459953655 \times 10^{-11}$	$0.922536904337508397092 \times 10^{-14}$	$0.383823674794415057013 \times 10^{-11}$
9	$0.444163621006646951654 \times 10^{-15}$		$0.120038117604205730523 \times 10^{-15}$	

See text for details

Table 7 Polynomial coefficients ($a_m x^m$) for polynomial approximations used to evaluate F_0 to F_2 using double-precision arithmetics

m	$F_0(x), x \in [0, 13)$	$F_0(x), x \in [13, 28)$	$F_1(x), x \in [0, 13)$	$F_2(x), x \in [0, 13)$
0	0.99999999999986319384	1.23367583242159180431	0.33333333333333998565	0.20000000000019000163
1	$0.77780499532525206025 3 \times 10^{-12}$	−0.25420503970371396167 3	−0.03333333333369707753 9	−0.04285714285793787965 73
2	0.04444444443709288000 53	0.17549689617245724726 1	0.01309523809854340483 8	0.00912698413250344125 126
3	−0.00282186946095460121 89	−0.04539934464926982933 14	−0.00085978837169799155 2157	−0.00106421357933126289 364
4	0.00070546731740844305 5588	0.01048354410615183635 03	0.00015857986873280178 0485	0.00011845795262488164 55
5	$-0.57007398076884891471 9 \times 10^{-4}$	−0.00174453211392308405 505	$-0.78336278367017401594 9 \times 10^{-5}$	$-0.94639343805450150090 6 \times 10^{-5}$
6	$0.64559200031403667176 \times 10^{-5}$	0.00023358127559697577 764	$0.92453933201111767351 6 \times 10^{-6}$	$0.71986181879790801343 \times 10^{-6}$
7	$-0.48721287944166568931 4 \times 10^{-6}$	$-0.24902996565626661442 2 \times 10^{-4}$	$-0.36809688795585536986 4 \times 10^{-7}$	$-0.43701238044817310814 2 \times 10^{-7}$
8	$0.37804454685479862156 \times 10^{-7}$	$0.21672680885927253476 7 \times 10^{-5}$	$0.31664776809392039897 2 \times 10^{-8}$	$0.25336227254291179248 5 \times 10^{-8}$
9	$-0.24419284891467819355 2 \times 10^{-8}$	$-0.15463378181124990348 \times 10^{-6}$	$-0.10695357248077100689 6 \times 10^{-9}$	$-0.12409354945040752326 5 \times 10^{-9}$
10	$0.14897679292733336512 8 \times 10^{-9}$	$0.91118708670889443947 8 \times 10^{-8}$	$0.73764617120146878229 7 \times 10^{-11}$	$0.58315790891759885437 2 \times 10^{-11}$
11	$-0.80726779489364577929 3 \times 10^{-11}$	$-0.44379161594830459714 3 \times 10^{-9}$	$-0.25084526414334648957 1 \times 10^{-12}$	$-0.23931101930758129341 3 \times 10^{-12}$
12	$0.39842321741942614825 2 \times 10^{-12}$	$0.17850403257208068793 1 \times 10^{-10}$	$0.17146290310169392265 8 \times 10^{-13}$	$0.93101174584333876950 4 \times 10^{-14}$
13	$-0.17218968190944520244 8 \times 10^{-13}$	$-0.58983429155994275743 4 \times 10^{-12}$	$-0.85916290081280982928 5 \times 10^{-15}$	$-0.30682452533625256275 2 \times 10^{-15}$
14	$0.63381635389274019293 1 \times 10^{-15}$	$0.15870436806658026548 6 \times 10^{-13}$	$0.55124132558912559115 9 \times 10^{-16}$	$0.85691773876122031712 1 \times 10^{-17}$
15	$-0.18693827721726560517 \times 10^{-16}$	$-0.34265101948535840142 2 \times 10^{-15}$	$-0.25269704287092477553 3 \times 10^{-17}$	$-0.16397390779920190367 3 \times 10^{-18}$
16	$0.40937960115924995065 \times 10^{-18}$	$0.58062271382952877719 5 \times 10^{-17}$	$0.93832072516684846861 5 \times 10^{-19}$	$0.18412239322942078777 1 \times 10^{-20}$
17	$-0.58222353068690065718 4 \times 10^{-20}$	$-0.74551044394173630242 \times 10^{-19}$	$-0.20971010073699311492 3 \times 10^{-20}$	
18	$0.40141030578768086596 8 \times 10^{-22}$	$0.68453306929194962405 3 \times 10^{-21}$	$-0.20971010073699311492 3 \times 10^{-20}$	
19		$-0.40250612302206646466 \times 10^{-23}$		
20		$0.11535994642419470202 6 \times 10^{-25}$		

See text for details

Table 8 Polynomial coefficients ($a_m x^m$) for polynomial approximations used to evaluate F_3 to F_6 using double-precision arithmetics

m	$F_3(x), x \in [0, 13)$	$F_4(x), x \in [0, 13)$	$F_5(x), x \in [0, 13)$	$F_6(x), x \in [0, 13)$
0	0.1428571428571297l5732	0.1111111111111155267118	0.0909090909090418894218	0.0769230769231433304501
1	−0.039682539681940222284	0.0202020201984917831837	0.0139860139895943769452	0.0102564102519954893211
2	0.0077561327515897280585	0.00310800315490172272381	0.00186480182131100939763	0.00120663654955329500618
3	−0.00100593849229835733416	0.000414400166537300036615	0.000219388664619585428667	0.000127014154283341090658
4	0.000106548522461521082494	$0.487536847112118858613 \times 10^{-4}$	$0.230929845836704818761 \times 10^{-4}$	$0.120971042304921501513 \times 10^{-4}$
5	$-0.905104526359044126978 \times 10^{-5}$	$0.513069895607636063669 \times 10^{-5}$	$0.220022434901947812293 \times 10^{-5}$	$0.105117158529560522249 \times 10^{-5}$
6	$0.665793244280159113221 \times 10^{-6}$	$0.490126749263780962495 \times 10^{-6}$	$0.190370038783173650663 \times 10^{-6}$	$0.848202367967884206378 \times 10^{-7}$
7	$-0.421033032496292895812 \times 10^{-7}$	$0.413787595121766077484 \times 10^{-7}$	$0.159517538791307783747 \times 10^{-7}$	$0.578595792393278564403 \times 10^{-8}$
8	$0.237950308105312128429 \times 10^{-8}$	$0.407434118761132044935 \times 10^{-8}$	$0.778814490754225626272 \times 10^{-9}$	$0.648765149611935152251 \times 10^{-9}$
9	$-0.119585419261532337924 \times 10^{-9}$	$-0.557063911311579936462 \times 10^{-10}$	$0.223906416425992668526 \times 10^{-9}$	$-0.528368725925082466781 \times 10^{-10}$
10	$0.544052665333592656135 \times 10^{-11}$	$0.125963215708502986724 \times 10^{-9}$	$-0.411326622269351458591 \times 10^{-10}$	$0.244156884389639498406 \times 10^{-10}$
11	$-0.220367182975008587518 \times 10^{-12}$	$-0.289977041663717227377 \times 10^{-10}$	$0.117336468783720353353 \times 10^{-10}$	$-0.488184480722150335347 \times 10^{-11}$
12	$0.782440549317872283015 \times 10^{-14}$	$0.669449581858860305726 \times 10^{-11}$	$-0.221098993037740027072 \times 10^{-11}$	$0.856794083815237453902 \times 10^{-12}$
13	$-0.230246069026969031004 \times 10^{-15}$	$-0.115947780116820567014 \times 10^{-11}$	$0.344743879871198540581 \times 10^{-12}$	$-0.113780539824004076151 \times 10^{-12}$
14	$0.522272868973222138142 \times 10^{-17}$	$0.163508260357701435893 \times 10^{-12}$	$-0.419060036720456287824 \times 10^{-13}$	$0.119287675670696340377 \times 10^{-13}$
15	$-0.792007050838327411146 \times 10^{-19}$	$-0.182816325963274730756 \times 10^{-13}$	$0.403167314131026508448 \times 10^{-14}$	$-0.961413167058437051429 \times 10^{-15}$
16	$0.610271838657186329254 \times 10^{-21}$	$0.162534908889415658976 \times 10^{-14}$	$-0.301124512725180177882 \times 10^{-15}$	$0.589705623061355288522 \times 10^{-16}$

Table 8 continued

m	$F_3(x), x \in [0, 13)$	$F_4(x), x \in [0, 13)$	$F_5(x), x \in [0, 13)$	$F_6(x), x \in [0, 13)$
17		$-0.11304561111364938509 1 \times 10^{-15}$	$0.17202915963897866281 8 \times 10^{-16}$	$-0.26561180218588406576 4 \times 10^{-17}$
18		$0.60428485268044252188 3 \times 10^{-17}$	$-0.72601265005950205336 6 \times 10^{-18}$	$0.83540210221081597290 5 \times 10^{-19}$
19		$-0.23983880429464745892 5 \times 10^{-18}$	$0.21473790023242299129 1 \times 10^{-19}$	$-0.16442986599378801223 6 \times 10^{-20}$
20		$0.66941012227589073782 8 \times 10^{-20}$	$-0.39903165538825352707 \times 10^{-21}$	$0.15675683376425307591 9 \times 10^{-22}$
21		$-0.11777431106851480223 5 \times 10^{-21}$	$0.35894634805731470494 7 \times 10^{-23}$	
22		$0.10028167659867893295 5 \times 10^{-23}$		

See text for details

Table 9 Polynomial coefficients ($a_{\mathrm{m}}x^{\mathrm{m}}$) for polynomial approximations used to evaluate F_7 and F_8 using double-precision arithmetics

m	$F_7(x), x \in [0, 13)$	$F_8(x), x \in [0, 13)$
0	0.0666666666666792786671	0.0588235294117413001689
1	0.00784313725406306245489	0.00619195046580679172754
2	0.000825593404530716987562	0.000589709553908030086275
3	$0.786279016549200026579 \times 10^{-4}$	$0.512791490423893838672 \times 10^{-4}$
4	$0.68373071139961302575 \times 10^{-5}$	$0.410220940108251083417 \times 10^{-5}$
5	$0.546842299560632490664 \times 10^{-6}$	$0.304027512169341505033 \times 10^{-6}$
6	$0.406444764184000626435 \times 10^{-7}$	$0.208278263397454997549 \times 10^{-7}$
7	$0.270895233610719952673 \times 10^{-8}$	$0.142947761244936735112 \times 10^{-8}$
8	$0.222039480215198281916 \times 10^{-9}$	$0.480870922358984868218 \times 10^{-10}$
9	$-0.445936980318832982556 \times 10^{-11}$	$0.157663667276142991773 \times 10^{-10}$
10	$0.496897710542915876124 \times 10^{-11}$	$-0.251057490699620640587 \times 10^{-11}$
11	$-0.918362655379709565603 \times 10^{-12}$	$0.543386306099847053127 \times 10^{-12}$
12	$0.164772634229469199918 \times 10^{-12}$	$-0.780853640736970586768 \times 10^{-13}$
13	$-0.21774171352195316967 \times 10^{-13}$	$0.905278144826231820914 \times 10^{-14}$
14	$0.229062800486870433577 \times 10^{-14}$	$-0.791305688413093701383 \times 10^{-15}$
15	$-0.18471514394046369986 \times 10^{-15}$	$0.52533115807928187281 \times 10^{-16}$
16	$0.113559718171813908632 \times 10^{-16}$	$-0.254000319857859504737 \times 10^{-17}$
17	$-0.51243661072967266004 \times 10^{-18}$	$0.855735765242934406053 \times 10^{-19}$
18	$0.161690346832485458817 \times 10^{-19}$	$-0.179599072412377437745 \times 10^{-20}$
19	$-0.319386590104278379248 \times 10^{-21}$	$0.183644573632662066635 \times 10^{-22}$
20	$0.306891710154503330058 \times 10^{-23}$	

See text for details

would execute all code branches. If this kind of divergence is common, the expected cost of the algorithm execution should be evaluated as the number of operations in all branches summed up rather than an average of the single branches as it would be in CPU-based implementation.

Third, contrary to modern CPUs case, for GPUs there is a significant difference between the efficiencies of arithmetic operations executed on single and double precision floating numbers. It suggests that single-precision implementation should be provided together with standard double-precision one if it is expected that many function evaluations can be performed with lower accuracy.

3.2 Implementation details

For double- and single-precision implementations, we aimed at the maximum absolute errors of 10^{-13} and 3×10^{-7}, respectively. A few different approximations were combined to achieve the assumed accuracy with minimal computational effort. The

Table 10 Polynomial coefficients ($a_{\mathrm{m}} x^{\mathrm{m}}$) for rational approximations used to evaluate F_0 to F_3 using single-precision arithmetics

m	$F_0(x), x \in [0, 13)$		$F_1(x), x \in [0, 13)$	
	Nominator	Denominator	Nominator	Denominator
0	0.99999999997	1.0	0.33333333327	1.0
1	0.10182466997	0.10182466908	−0.030168012966	0.0094959550182
2	0.042019978617	−0.0024244613291	0.010792367586	−0.0059589848643
3	0.0012191985475	−0.00048448137333	−0.00053781624709	$-0.31006295741 \times 10^{-5}$
4	0.00031828251549	$0.79127138367 \times 10^{-5}$	$0.78359910422 \times 10^{-4}$	$0.17693394915 \times 10^{-4}$
5	$0.13860913077 \times 10^{-5}$	$0.12470554653 \times 10^{-5}$	$-0.1948342434 \times 10^{-5}$	$-0.3833822025 \times 10^{-6}$
6	$0.6036650408 \times 10^{-6}$	$-0.55553265779 \times 10^{-7}$	$0.14315491546 \times 10^{-6}$	$-0.10042169732 \times 10^{-7}$
7		$0.70468423997 \times 10^{-9}$		$0.31030312301 \times 10^{-9}$
m	$F_2(x), x \in [0, 13)$		$F_3(x), x \in [0, 13)$	
	Nominator	Denominator	Nominator	Denominator
0	0.20000000002	1.0	0.14285714287	1.0
1	−0.031353599493	0.057517719504	−0.025659473576	0.098161464615
2	0.0058076178004	−0.0042716201489	0.0038066113064	−0.00037958573997
3	−0.00041280093614	−0.00028307084872	−0.00027882655047	−0.0003451125009
4	$0.32640532199 \times 10^{-4}$	$0.11212521115 \times 10^{-4}$	$0.17703110615 \times 10^{-4}$	$-0.59813693555 \times 10^{-5}$
5	$-0.10365544258 \times 10^{-5}$	$0.68320444781 \times 10^{-6}$	$-0.56543841868 \times 10^{-6}$	$0.60099930813 \times 10^{-6}$
6	$0.38220461118 \times 10^{-7}$	$-0.38041283796 \times 10^{-7}$	$0.15670519224 \times 10^{-7}$	$0.81994373903 \times 10^{-8}$
7		$0.49753170236 \times 10^{-9}$		$-0.48168251558 \times 10^{-9}$

See text for details

Table 11 Polynomial coefficients ($a_m x^m$) for rational approximations used to evaluate F_4 to F_8 using single-precision arithmetics

m	$F_4(x), x \in [0, 13)$		$F_5(x), x \in [0, 8)$	
	Nominator	Denominator	Nominator	Denominator
0	0.11111111111	1.0	0.090909090817	1.0
1	−0.021240751205	0.12701505644	−0.012882875391	0.037775527998
2	0.0027490742272	0.0030926172754	0.0012910778188	−0.0038041638458
3	−0.00019542515743	−0.00029527837661	$-0.57275005071 \times 10^{-4}$	−0.00019909424941
4	$0.10477917547 \times 10^{-4}$	$-0.16183306152 \times 10^{-4}$	$0.20477047928 \times 10^{-5}$	$0.46739788572 \times 10^{-5}$
5	$-0.31269230045 \times 10^{-6}$	$0.26711491256 \times 10^{-6}$		$0.3111544053 \times 10^{-6}$
6	$0.65609019819 \times 10^{-8}$	$0.31321729923 \times 10^{-7}$		
7		$-0.70871419526 \times 10^{-9}$		
m	$F_5(x), x \in [8, 13)$		$F_6(x), x \in [0, 8)$	
	Nominator	Denominator	Nominator	Denominator
0	0.84162618815	1.0	0.076923076935	1.0
1	−0.0034169244126	−0.11935391469	−0.010963827464	0.057470247038
2	0.010593928314	0.0050797037293	0.00097991779983	−0.0025644227799
3		$-0.76791269967 \times 10^{-4}$	$-0.41235535291 \times 10^{-4}$	−0.00024596402831
4			$0.11978837582 \times 10^{-5}$	$0.45952872392 \times 10^{-7}$
5				$0.40279289638 \times 10^{-6}$

Table 11 continued

m	$F_7(x), x \in [0, 8)$		$F_8(x), x \in [0, 8)$	
	Nominator	Denominator	Nominator	Denominator
0	0.066666666695	1.0	0.058823529432	1.0
1	−0.0095362385039	0.072642707148	−0.0081414871477	0.089664902286
2	0.00079626501998	−0.0011118013754	0.00063958176829	0.00082984754463
3	$-0.32648649855 \times 10^{-4}$	−0.00023059643023	$-0.25190736987 \times 10^{-4}$	−0.00017823912387
4	$0.83466714489 \times 10^{-6}$	$-0.31233168879 \times 10^{-5}$	$0.59927822663 \times 10^{-6}$	$-0.5950523028 \times 10^{-5}$
5		$0.29783342062 \times 10^{-6}$		$0.99015612579 \times 10^{-7}$

See text for details

Table 12 Polynomial coefficients ($a_m x^m$) for polynomial approximations used to evaluate F_0 to F_6 using single-precision arithmetics

m	$F_0(x), x \in [0, 13)$	$F_1(x), x \in [0, 13)$	$F_2(x), x \in [0, 13)$	$F_3(x), x \in [0, 8)$
0	0.99999999997	0.33333333337	0.2	
1	$0.10517138239 \times 10^{-8}$	−0.033333334614	−0.042857142916	0.14285714333
2	0.044444437391	0.013095245899	−0.042857142916	−0.015873029547
3	−0.0028218508744	−0.00085980717059	−0.0010642142047	0.0031265690149
4	0.0007054417187	0.0001586035548	0.00011845862539	−0.00015429127982
5	$-0.56986210777 \times 10^{-4}$	$-0.78515236529 \times 10^{-5}$	$-0.94643636719 \times 10^{-5}$	$0.2071047221 \times 10^{-4}$
6	$0.64444776797 \times 10^{-5}$	$0.93330691782 \times 10^{-6}$	$0.7200350148 \times 10^{-6}$	$-0.47034988332 \times 10^{-6}$
7	$-0.4829651868 \times 10^{-6}$	$-0.39729078626 \times 10^{-7}$	$-0.43746279275 \times 10^{-7}$	$0.95712296255 \times 10^{-7}$
8	$0.36683054262 \times 10^{-7}$	$0.38452571889 \times 10^{-8}$	$0.25408654374 \times 10^{-8}$	$-0.49611781677 \times 10^{-8}$
9	$-0.22267122414 \times 10^{-8}$	$-0.21842640175 \times 10^{-9}$	$-0.12464644119 \times 10^{-9}$	$0.89610272135 \times 10^{-9}$
10	$0.11855126518 \times 10^{-9}$	$0.20252504781 \times 10^{-10}$	$0.57972960549 \times 10^{-11}$	$-0.53774960016 \times 10^{-10}$
11	$-0.48836232337 \times 10^{-11}$	$-0.12697607584 \times 10^{-11}$	$-0.22465110382 \times 10^{-12}$	
12	$0.15031034021 \times 10^{-12}$	$0.68439748155 \times 10^{-13}$	$0.74095504278 \times 10^{-14}$	
13	$-0.2931260792 \times 10^{-14}$	$-0.21123266249 \times 10^{-14}$	$-0.16548571769 \times 10^{-15}$	
14	$0.28336866216 \times 10^{-16}$	$0.37016384408 \times 10^{-16}$	$0.21800774606 \times 10^{-17}$	
m	$F_3(x), x \in [8, 13)$	$F_4(x), x \in [0, 8)$	$F_5(x), x \in [0, 8)$	$F_6(x), x \in [0, 8)$
0	1.5919889454	0.11111111111	0.090909090923	0.076923076923
1	−0.51008317416	−0.016835016857	−0.016317016685	−0.015384615372
2	0.2373739543	0.0025468360226	0.0022533039007	0.002061337284
3	−0.04467894453	−0.00018513759332	−0.00018638152519	−0.00018023087994
4	0.0063977380109	$0.15738955748 \times 10^{-4}$	0.13997300219	$0.13051974522 \times 10^{-4}$

Table 12 continued

m	$F_3(x), x \in [8, 13)$	$F_4(x), x \in [0, 8)$	$F_5(x), x \in [0, 8)$	$F_6(x), x \in [0, 8)$
5	-0.00055071554382	$-0.70088082967 \times 10^{-6}$	$-0.74587608096 \times 10^{-6}$	$-0.73425458613 \times 10^{-6}$
6	$0.32266506392 \times 10^{-4}$	$0.4646245116 \times 10^{-7}$	$0.40133123958 \times 10^{-7}$	$0.36987584385 \times 10^{-7}$
7	$-0.10699998611 \times 10^{-5}$	$-0.13373754984 \times 10^{-8}$	$-0.16060406391 \times 10^{-8}$	$-0.15155033238 \times 10^{-8}$
8	$0.18338481239 \times 10^{-7}$	$0.10125834709 \times 10^{-9}$	$0.76821973182 \times 10^{-10}$	$0.56970480153 \times 10^{-10}$
9		$-0.38228871873 \times 10^{-11}$	$-0.2582582586 \times 10^{-11}$	$-0.15512086776 \times 10^{-11}$
10		$0.40310834767 \times 10^{-12}$	$0.83713632646 \times 10^{-13}$	$0.30941139171 \times 10^{-13}$
11		$-0.18148826214 \times 10^{-13}$		
12		$0.70935229499 \times 10^{-15}$		

See text for details

Table 13 Polynomial coefficients ($a_{\mathrm{m}}x^{\mathrm{m}}$) for polynomial approximations used to evaluate F_7 and F_8 using single-precision arithmetics

m	$F_7(x), x \in [0, 8)$	$F_8(x), x \in [0, 8)$
0	0.066666666664	0.058823529408
1	−0.014379084884	−0.013415892568
2	0.001914917627	0.0017936994585
3	−0.00017236192564	−0.00016440085309
4	$0.12372629538 \times 10^{-4}$	$0.11807215021 \times 10^{-4}$
5	$-0.71143283929 \times 10^{-6}$	$-0.68655339553 \times 10^{-6}$
6	$0.3526511292 \times 10^{-7}$	$0.33942919132 \times 10^{-7}$
7	$-0.1465365545 \times 10^{-8}$	$-0.14203746108 \times 10^{-8}$
8	$0.51826324626 \times 10^{-10}$	$0.49482626741 \times 10^{-10}$
9	$-0.13349095264 \times 10^{-11}$	$-0.12589659976 \times 10^{-11}$
10	$0.20369202875 \times 10^{-13}$	$0.17598734267 \times 10^{-13}$

See text for details

choices that were made for each considered case are summarized in Tables 2 and 3. The orders of the employed polynomial or rational approximations are presented together with the form of exponential damping factors and the type of recursion used to generate all but highest order (downward recursion) or all but 0-th order (upward recursion) Boys functions. The damping factors $d_k(x)$, actual functions $f_k(x)$ fitted by minimax procedure and the required Boys function $F_k(x)$ are connected according to the following formula:

$$F_k(x) = d_k(x) f(x), \tag{33}$$

where k is 0 or N depending on the type of recursion that is used in a given range of x.

The polynomial coefficients obtained by minimax procedure are collected in Tables 4, 5, 6, 7, 8, 9, 10, 11, 12 and 13. Both rational and polynomial approximations are provided. The approximations which are suitable for both double- and single-precision computations are presented.

3.3 Microoptimizations

Several code microoptimizations are applied in the actual implementation. The Horner scheme is used to evaluate polynomial values. Fused multiply-add (fma) operations are employed to make better use of arithmetic units. Divisions, which are very costly on GPUs, are replaced by the sequence of inverse square root (rsqrt) and multiplication operations. The exponential function arguments in damping factors were chosen so that the number of required function evaluations is minimized. As in two-electron integral calculations usually not only $F_n(x)$ value is needed, but also all lower order Boys function values ($F_o(x) \ldots F_{n-1}(x)$), all the required values are computed in a single call. For illustrative purposes we present in Fig. 1 an examplary code performing F_0 and F_1 computation using double precision arithmetics.

```
void boys1(double x, double F[2])
{
    double b0, b1;

    if (x < 28) {
        const double e_16 = exp(-(1.0 / 6.0) * x);
        const double e_13 = e_16 * e_16;
        const double e_12 = e_13 * e_16;
        const double e = e_12 * e_12;
        const double two_x = 2 * x;

        double n;
        double d;

        if (x < 13) {
            // fma() performs fused multiply-add calculation
            const double n = fma(x, fma(x, fma(x, fma(x, fma(x, fma(x,
            fma(x, fma(x, 0.2367154614833604 1064e-9,
            -0.4625399089691418062 87e-8), 0.2719781269682449263 63e-6),
            -0.3120452782304396212 14e-5), 0.9450347859576397230 01e-4),
            -0.0006058394301792497 26908), 0.0112709195324634180829),
            -0.0307524706862147488255), 0.333333333333343903809);
            const double d = fma(x, fma(x, fma(x, fma(x, fma(x, fma(x,
            fma(x, fma(x, fma(x, -0.6394893446215246694 92e-13,
            -0.1944426573184265065 69e-12), 0.4100781453456440055 06e-9),
            -0.1573704687379391650 95e-7), -0.7308435539195105527 1e-7),
            0.111138619313123589917e-4), -0.121959609027647601215e-4),
            -0.00469869690623180240936), 0.00774258794288160515724), 1.0);

            // rsqrt() performs inverse square root calculation
            const double rsd = rsqrt(d);
            b1 = n * rsd * rsd * e_12;
            b0 = fma(two_x, b1, e);
        } else {
            // fma() performs fused multiply-add calculation
            n = fma(x, fma(x, fma(x, fma(x, fma(x, fma(x, fma(x, fma(x,
            0.230224133910752021883e-9, -0.328585595083913821085e-8),
            0.435705981848460995997e-6), -0.106401343255264539364e-5),
            0.000226242532551670921154), 0.000702644887531935947 37),
            0.0370559849570588636147), 0.114650580864037928698),
            1.02241837924423924866);
            d = fma(x, fma(x, fma(x, fma(x, fma(x, fma(x, fma(x, fma(x,
            fma(x, -0.506080089114735146947e-14,
            0.150728424452101023 41e-11), -0.187119412616382912596e-9),
            0.117435181167263649392e-7), -0.303867297213349637263e-6),
            -0.629509987120141096118e-5), 0.000681696308108648850972),
            -0.0180199379732503914666), 0.136951673337325655053), 1.0);

            // rsqrt() performs inverse square root calculation
            const double rsd = rsqrt(d);
            const double rs2x = rsqrt(two_x);
            b0 = n * rsd * rsd * e_13;
            b1 = (1 * b0 - e) * rs2x * rs2x;
        }
    } else {
        const double rsx = rsqrt(x);
        // M_PI is a macro representing an approximation of the Pi constant
        b0 = sqrt(M_PI) / 2.0 * rsx;
        b1 = 0.5 * b0 * rsx * rsx;
    }

    F[0] = b0;
    F[1] = b1;
}
```

Fig. 1 Implementation of $F_0(x)$ and $F_1(x)$ computation using double precision arithmetics. See text for details

4 Summary

We have developed a computational scheme for the evaluation of the Boys function which is suitable for execution on General Purpose Graphical Processing Units (GPGPUs). The scheme combines the polynomial and rational approximations, downward and upward recursions, and asymptotic approximations. This formulation differs greatly from the CPU-specific one, which stems from fundamental architectural differences between typical GPGPUs and standard processing units.

The proposed formulation allows the exploitation of computational power offered by modern graphical processors, which is an important part of efficient implementation of two-electron integral calculation on GPGPUs.

Acknowledgments This scholarly work was made thanks to POWIEW project. The project is co-funded by the European Regional Development Fund (ERDF) as a part of the Innovative Economy program. The research was carried out with the equipment purchased thanks to the financial support of the European Regional Development Fund in the framework of the Polish Innovation Economy Operational Program (Contract No. POIG.02.01.00-12-023/08).
This work was partially supported by the PLGrid NG project (POIG.02.03.00-12-138/13), co-funded by the European Regional Development Fund as part of the Innovative Economy program.
We would like to express our gratitude to Dr. James Hooper for the help with improving the manuscript.

References

1. W.G. Bickley, L.J. Comrie, J.C.P. Miller, D.H. Sadler, A.J. Thompson, *Bessel Functions. Part II: Functions of Positive Integer Order, British Association for the Advancement of Science, Mathematical Tables* (Cambridge University Press, Cambridge, 1952)
2. S.F. Boys, Electronic wave functions. I. A general method of calculation for the stationary states of any molecular system. Proc. R. Soc. A **200**, 542 (1950)
3. W. Fraser, A survey of methods of computing minimax and near-minimax polynomial approximations for functions of a single independent variable. J. ACM **12**(3), 295–314 (1965). doi:10.1145/321281.321282
4. P.M.W. Gill, M. Head-Gordon, J.A. Pople, Efficient computation of two-electron repulsion integrals and their nth-order derivatives using contracted gaussian basis sets. J. Phys. Chem. **94**, 5564–5572 (1990)
5. I.I. Guseinov, B.A. Mamedov, Evaluation of the boys function using analytical relations. J. Math. Chem. **40**, 179–183 (2006)
6. T. Helgaker, P. Jorgensen, J. Olsen, *Molecular Electronic Structure Theory* (Wiley, New York, 2000)
7. K. Ishida, Ace algorithm for the rapid evaluation of the electron-repulsion integral over gaussian-type orbitals. Int. J. Quantum Chem. **59**, 209–218 (1996)
8. G.O. Morrell, L.J. Schaad, Approximations for the functions $F_m(z)$ occuring in molecular calculations with a gaussian basis. J. Chem. Phys. **54**, 1965–1967 (1971)
9. M. Primorac, New expansion of the boys function. Int. J. Quantum Chem. **68**(5), 305–315 (1998)
10. V. Saunders, An introduction to molecular integral evaluation, in *Computational Techniques in Quantum Chemistry and Molecular Physics*, ed. by G. Diercksen, B.T. Sutcliffe, A. Veillard (D. Reidel Publishing Company, Dordrecht, 1975)
11. I. Shavit, The gaussian function in calculations of statistical mechanics and quantum mechanics, in *Methods in Computational Physics*, vol. 2, ed. by B. Alder, S. Fernbach, M. Rotenberg (Academic Press, Cambridge, 1963)
12. A.K.H. Weiss, C. Ochsenfeld, A rigorous and optimized strategy for the evaluation of the Boys function kernel in molecular electronic structure theory. J. Comput. Chem. **36**(18), 1390–1398 (2015). doi:10.1002/jcc.23935