Staggered Correction Computations with Enhanced Accuracy and Extremely Wide Exponent Range*

Frithjof Blomquist
Scientific Computing/Software Engineering, University of Wuppertal, Gaußstraße 20, D-42097 Wuppertal, Germany
`blomquist@math.uni-wuppertal.de`

Abstract

In C-XSC a staggered interval arithmetic with precision p is defined as an array of $p+1$ components of type *double*. The disadvantage of this multiple precision arithmetic is the rather small exponent range and particularly the dramatic loss of accuracy, if intermediate results of computations lie near the underflow range. In order to avoid these difficulties a new class `lx_interval` is implemented with objects of the form $2^r \cdot \boldsymbol{x}$, $r \in \mathbb{Z}$ of type *double* and $\boldsymbol{x}$ of type `l_interval`, the original staggered interval data type. Now the intervals $\boldsymbol{x}$ can be scaled in such a way that the basic arithmetic operations can be performed in a remarkable higher accuracy. Numerical examples demonstrate the ability of the new arithmetic.

Keywords: interval computations, high-precision data, staggered interval arithmetic
AMS subject classifications: 65G30, 65G50, 65Y04

1 Introduction

Beside the multiple precision interval packages in [9, 10, 11, 12] based on integer arithmetics, the so called staggered precision arithmetic, implemented in C-XSC, is a special kind of a multiple precision arithmetic based on the underlying floating-point data format (IEEE double format) [1, 2, 3]. In C-XSC a staggered interval $\boldsymbol{x}$ with precision p is defined as an array of $p+1$ components x_i of type double:

$$\boldsymbol{x} := \sum_{i=1}^{p-1} x_i + [x_p, x_{p+1}] = [\mathtt{Inf}(\boldsymbol{x}), \mathtt{Sup}(\boldsymbol{x})] = [\underline{\boldsymbol{x}}, \overline{\boldsymbol{x}}],$$

*Submitted: January 17, 2009; Revised: January 17, 2010; Accepted: February 1, 2010.

$$\texttt{Inf}(\boldsymbol{x}) := \sum_{i=1}^{p-1} x_i + x_p, \quad \texttt{Sup}(\boldsymbol{x}) := \sum_{i=1}^{p-1} x_i + x_{p+1}.$$

The basic arithmetic operations are provided by the data type `dotprecision` (so called long accumulator, C-XSC) and so, e.g. the machine product $\boldsymbol{x} \diamond \boldsymbol{y} \supset \boldsymbol{x} \cdot \boldsymbol{y}$, results in an optimal enclosure of the exact interval $\boldsymbol{x} \cdot \boldsymbol{y}$, because $\boldsymbol{x} \cdot \boldsymbol{y}$ is rounded to the nearest machine numbers $\texttt{Inf}(\boldsymbol{x} \diamond \boldsymbol{y})$ and $\texttt{Sup}(\boldsymbol{x} \diamond \boldsymbol{y})$. The multiplication of two point intervals $\boldsymbol{x}, \boldsymbol{y}$, each of the order 10^{153}, delivers a product of the order $\boldsymbol{x} \diamond \boldsymbol{y} \sim 10^{306}$ with a maximum accuracy of about $2 \cdot 153 + 324 = 630$ decimal digits, where $2^{-1074} \sim 10^{-324}$ is the smallest IEEE number. However, if one or both of the intervals $\boldsymbol{x}, \boldsymbol{y}$ are non-point intervals then the accuracy of the product can't exceed the accuracy of the operand with the greatest relative diameter. Thus, applications of a staggered interval arithmetic are senseful only, if sufficiently small interval operands are used.

However, even with point intervals $\boldsymbol{x}, \boldsymbol{y}$, defined by an arbitrary precision p, the accuracy of $\boldsymbol{x} \diamond \boldsymbol{y}$ will drastically be reduced, if the machine product $\boldsymbol{x} \diamond \boldsymbol{y}$ lies near the underflow range. For example, the product of two point intervals $\boldsymbol{x}, \boldsymbol{y}$, each of the order 10^{-153}, is of the order 10^{-306} and can be calculated with an accuracy of at most $324 - 306 = 18$ correct decimal digits and of course this result is absolutely unacceptable if for example 200 correct decimal digits are required. The described loss of accuracy can be avoided by scaling $\boldsymbol{x}, \boldsymbol{y}$ appropriately, for instance

$$\widetilde{\boldsymbol{x}} = 2^{+1018} \cdot \boldsymbol{x} \sim 10^{153}, \qquad \widetilde{\boldsymbol{y}} = 2^{+1018} \cdot \boldsymbol{y} \sim 10^{153}.$$

Then an enclosure of the exact interval $\widetilde{\boldsymbol{x}} \cdot \widetilde{\boldsymbol{y}}$ can be computed in high accuracy, and without any overflow we get

$$\boldsymbol{x} \cdot \boldsymbol{y} \subset 2^{-2036} \cdot (\widetilde{\boldsymbol{x}} \diamond \widetilde{\boldsymbol{y}}).$$

With this result the desired enclosure of $\boldsymbol{x} \cdot \boldsymbol{y}$ in high accuracy can be described by the additional exponent -2036 and by the interval $(\widetilde{\boldsymbol{x}} \diamond \widetilde{\boldsymbol{y}}) \sim 10^{+306}$ [14]. In order to get enclosures for the elementary functions with the same maximum accuracy the three other elementary operations are analogously to be reworked.

2 Extended Staggered Interval Arithmetic

In order to avoid intermediate products like $\boldsymbol{x} \diamond \boldsymbol{y} \sim 10^{-306}$ near the underflow range the operands $\boldsymbol{x}, \boldsymbol{y}$ are scaled by $2^{r_x}, 2^{r_y}$ respectively in such a way that the scaled values $\widetilde{\boldsymbol{x}} := 2^{r_x} \diamond \boldsymbol{x}$ and $\widetilde{\boldsymbol{y}} := 2^{r_y} \diamond \boldsymbol{y}$ are both of order 10^{+153}. The exponents r_x, r_y are chosen as integer values of type double. So the product $\widetilde{\boldsymbol{x}} \cdot \widetilde{\boldsymbol{y}}$ is of order 10^{+306} and can be included by $\widetilde{\boldsymbol{x}} \diamond \widetilde{\boldsymbol{y}}$ with a maximum accuracy of 630 digits, if both of the operands are point intervals and in addition no overflow is generated. For the desired product $\boldsymbol{x} \cdot \boldsymbol{y}$ it holds

$$\boldsymbol{x} \cdot \boldsymbol{y} = (\widetilde{\boldsymbol{x}} \cdot \widetilde{\boldsymbol{y}}) \cdot 2^{-r_x - r_y} \subseteq (\widetilde{\boldsymbol{x}} \diamond \widetilde{\boldsymbol{y}}) \cdot 2^{-r_x - r_y} = (\widetilde{\boldsymbol{x}} \diamond \widetilde{\boldsymbol{y}}) \cdot 2^{r},$$

and now $\boldsymbol{x} \cdot \boldsymbol{y}$ can be included in maximum accuracy by the machine product $\widetilde{\boldsymbol{x}} \diamond \widetilde{\boldsymbol{y}}$ together with the factor 2^r, where r is an integer value of type double.

With this preliminary considerations the extended staggered interval arithmetic is defined with an additional factor $2^r, r \in \mathbb{Z}$:

$$(r, \boldsymbol{x}) := 2^r \cdot \boldsymbol{x} = 2^r \cdot \left(\sum_{i=1}^{p-1} x_i + [x_p, x_{p+1}] \right).$$

With the exponent r of type `double` an extremely wide range of the exponents r is realized: $\quad -9007199254740991 \le r \le +9007199254740991 = 2^{53} - 1$.
The greatest absolute value of an interval $\boldsymbol{x}$ is about

$$|\boldsymbol{x}| = 2^{+9007199254740991} \cdot 10^{+308} \sim 10^{2711437152599603}.$$

Thus, in practice, almost all overflow or underflow problems are eliminated.

3 Maximum Accuracy of the Basic Operations

Due to the ambiguity of the notation $(r, \boldsymbol{x}) \subseteq (r - s, 2^s \mathbin{\Diamond\!\!\!\!\cdot}\, \boldsymbol{x})$ the exponent $s \in \mathbb{Z}$ of the scaling factor 2^s can be chosen in such a way that, e.g. for a multiplication $(r_x, \boldsymbol{x}) \cdot (r_y, \boldsymbol{y}) \subseteq (r_x - s_x, 2^{s_x} \mathbin{\Diamond\!\!\!\!\cdot}\, \boldsymbol{x}) \cdot (r_y - s_y, 2^{s_y} \mathbin{\Diamond\!\!\!\!\cdot}\, \boldsymbol{y})$, for both of the operands the relations $2^{s_x} \cdot |\boldsymbol{x}| \sim 2^{s_y} \cdot |\boldsymbol{y}| \sim 10^{+153}$ are valid. In this case we have

$$(r_x, \boldsymbol{x}) \cdot (r_y, \boldsymbol{y}) \subseteq (r_x - s_x + r_y - s_y, (2^{s_x} \mathbin{\Diamond\!\!\!\!\cdot}\, \boldsymbol{x}) \mathbin{\Diamond\!\!\!\!\cdot}\, (2^{s_y} \mathbin{\Diamond\!\!\!\!\cdot}\, \boldsymbol{y}))$$

and because of $|(2^{s_x} \mathbin{\Diamond\!\!\!\!\cdot}\, \boldsymbol{x}) \mathbin{\Diamond\!\!\!\!\cdot}\, (2^{s_y} \mathbin{\Diamond\!\!\!\!\cdot}\, \boldsymbol{y})| \sim 10^{+306}$ an overflow is avoided, and if the accuracy of the operands is given by about $153 + 324 = 477$ decimal digits then the product will have nearly the same accuracy.

Concerning the division operation

$$(r_x, \boldsymbol{x})/(r_y, \boldsymbol{y}) \subseteq (r_x - s_x - r_y + s_y, (2^{s_x} \mathbin{\Diamond\!\!\!\!\cdot}\, \boldsymbol{x}) \mathbin{\Diamond\!\!\!\!/}\, (2^{s_y} \mathbin{\Diamond\!\!\!\!\cdot}\, \boldsymbol{y}))$$

the exponents s_x, s_y are chosen in such a way that for the numerator and denominator the following relations $2^{s_x} \cdot |\boldsymbol{x}| \sim 10^{306}$, $2^{s_y} \cdot |\boldsymbol{y}| \sim 10^{153}$ are valid. Then the interval quotient $(2^{s_x} \mathbin{\Diamond\!\!\!\!\cdot}\, \boldsymbol{x}) \mathbin{\Diamond\!\!\!\!/}\, (2^{s_y} \mathbin{\Diamond\!\!\!\!\cdot}\, \boldsymbol{y})$ is of the order 10^{153} and again a maximum accuracy of about $153 + 324 = 477$ can be achieved, if both of the operands have an accuracy of the same order.

Concerning the addition operation

$$(r_x, \boldsymbol{x}) + (r_y, \boldsymbol{y}) \subseteq (r_x - s_x, (2^{s_x} \mathbin{\Diamond\!\!\!\!\cdot}\, \boldsymbol{x})) \mathbin{\Diamond\!\!\!\!+}\, (r_y - s_y, (2^{s_y} \mathbin{\Diamond\!\!\!\!\cdot}\, \boldsymbol{y}))$$

it is first assumed $(r_x, \boldsymbol{x})$ to be the operand with the greatest absolute value and s_x is chosen in such a way that $|2^{s_x} \mathbin{\Diamond\!\!\!\!\cdot}\, \boldsymbol{x}| \sim 10^{+306}$ is valid. In order to enable the addition the exponent s_y must be chosen to realize $r_x - s_x = r_y - s_y$ and then it holds

$$(r_x, \boldsymbol{x}) + (r_y, \boldsymbol{y}) \subseteq (r_x - s_x, (2^{s_x} \mathbin{\Diamond\!\!\!\!\cdot}\, \boldsymbol{x}) \mathbin{\Diamond\!\!\!\!+}\, (2^{s_y} \mathbin{\Diamond\!\!\!\!\cdot}\, \boldsymbol{y})),$$

and now the staggered machine addition $(2^{s_x} \mathbin{\Diamond\!\!\!\!\cdot}\, \boldsymbol{x}) \mathbin{\Diamond\!\!\!\!+}\, (2^{s_y} \mathbin{\Diamond\!\!\!\!\cdot}\, \boldsymbol{y})$ can be performed with maximum accuracy.

With the above new staggered interval operators $\{ \mathbin{\Diamond\!\!\!\!+}\,, \mathbin{\Diamond\!\!\!\!\cdot}\,, \mathbin{\Diamond\!\!\!\!/}\, \}$ the basic arithmetic operation results can be computed with a maximum accuracy of about 470 decimal digits and the user must not reflect about any appropriate scaling operations. Furthermore, due to the new wide exponent range, in practice all overflow and underflow problems are vanished.

Compared to the classical staggered interval arithmetic the enlargement of the run time by the new staggered interval arithmetic of about 25 percent seems to be absolutely acceptable.

3.1 Complex Division

Now the complex division is considered, which is a suitable example to test the ability of the new staggered interval arithmetic with the four basic arithmetic operators. With the complex numbers $z = a + i \cdot b,\ w = c + i \cdot d,\ i = \sqrt{-1}$ the imaginary part of the complex-valued quotient z/w is given by

$$\Im(z/w) = \frac{b \cdot c - a \cdot d}{c^2 + d^2} \in \mathbb{R}.$$

With the real values $a = b = 10^{300}, c = 10^{155}, d = 10^{155} - 1$, which by a given precision of 480 decimal digits can all be enclosed by point intervals, a simple C-XSC program delivers an inclusion of $\Im(z/w)$ with 476 correct decimal digits:

$$\Im(z/w) \in [\underbrace{5.00\ldots002499\ldots99}_{476 \text{ decimal digits}}86\ldots, 5.00\ldots002400\ldots0095\ldots] \cdot 10^{-11}.$$

It should be noticed that by use of the classical staggered interval arithmetic, i.e. without the factor 2^r, the products $b \cdot c$ and $a \cdot d$ will generate an overflow. Of course this overflow can be avoided dividing a, b, c, d by 10^{+155}. However, the new difference $d/10^{155} = 1 - 10^{-155}$ can then be enclosed with only 155 decimal digits, because 10^{-155} cannot be included by a point interval, and finally $\Im(z/w)$ can only be enclosed with 168 correct decimal digits. Thus, the new staggered interval arithmetic delivers automatically a nearly triple fold accuracy without an intermediate overflow and without any additional scaling problems.

4 Elementary Functions

Beside the discussed basic arithmetic operations $\{\Diamond\kern-0.62em{+}\,, \Diamond\kern-0.62em{-}\,, \Diamond\kern-0.55em{\cdot}\,, \Diamond\kern-0.6em{/}\}$, all necessary programming tools together with a set of 42 real standard functions $f(\boldsymbol{x})$ are implemented by a new class `lx_interval` in C-XSC, a C^{++} class library for extended scientific computing. For sufficiently tight intervals $\boldsymbol{x}$, with $|\boldsymbol{x}| \gg 1$ or $|\boldsymbol{x}| \ll 1$, optimal inclusions of rather sophisticated expressions are computable with high accuracy, even in cases where Computer Algebra Systems like Mathematica or Maple generate premature overflows or underflows. Furthermore, also wide intervals are permitted and deliver guaranteed inclusions with at least 16 correct decimal digits. The new class `lx_interval` can additionally be included by a C-XSC program already using ordinary staggered intervals (C-XSC data type `l_interval`). Thus, critical code segments can use the new modified staggered intervals (new data type `lx_interval`). In table **??** the 44 real standard functions $f(\boldsymbol{x})$ are listed. Additionally in the new class `lx_real` the same tools and elementary functions are implemented for point arguments $(r, x) := 2^r \cdot x$, with x of the C-XSC data type `l_real`.

Along the same line of reasoning, extended complex staggered intervals $\boldsymbol{z}$ are defined by

$$\boldsymbol{z} = (r_r, \boldsymbol{x}) + i \cdot (r_i, \boldsymbol{y}), \quad r_r, r_i \in \mathbb{Z}, \quad i := \sqrt{-1}.$$

The basic arithmetic operations together with a set of 41 elementary complex transcendental functions are implemented in the new class `lx_cinterval`.

The same tools and elementary functions are also available for complex point arguments $z = (r_r, x) + i \cdot (r_i, y) \in \mathbb{C}, \quad r_r, r_i \in \mathbb{Z}, \quad i := \sqrt{-1}$ implemented in the new class `lx_complex`.

Elementary Functions of Type `lx_interval` or `lx_real`

Function Term	**C-XSC Name**	**Informations**
$\|x\|$	abs(x)	interval of the absolute values
x^2	sqr(x)	interval of the squares
$\sqrt{x}$	sqrt(x)	Inf(x) ≥ 0
$\sqrt[n]{x}$	sqrt(x, n)	$2 \leq n \leq +2147483647$
$\sqrt{1+x}-1$	sqrtp1m1(x)	Inf(x) ≥ -1
$\sqrt{1-x^2}$	sqrt1mx2(x)	Sup($\|x\|$) ≤ 1
$\sqrt{1+x^2}$	sqrt1px2(x)	any intervall x
$\sqrt{x^2-1}$	sqrtx2m1(x)	Inf($\|x\|$) ≥ 1
$\sqrt{x^2+y^2}$	sqrtx2y2(x, y)	x, y almost arbitrary
x^n	power(x, n)	x almost arbitrary, $n \in \mathbb{Z}$
x^y	pow(x, y)	x, y almost arbitrary
$(1+x)^y$	xp1_pow_y(x, y)	x, y almost arbitrary
e^x	exp(x)	Sup($\|x\|$) $\leq 6.24331476 \cdot 10^{15}$
2^x	exp2(x)	Sup($\|x\|$) $\leq 6.24331476 \cdot 10^{15}$
10^x	exp10(x)	Sup($\|x\|$) $\leq 6.24331476 \cdot 10^{15}$
$e^x - 1$	expm1(x)	Sup($\|x\|$) $\leq 6.24331476 \cdot 10^{15}$
$\ln(x)$	ln(x)	Inf(x) > 0
$\log_2(x)$	log2(x)	Inf(x) > 0
$\log_{10}(x)$	log10(x)	Inf(x) > 0
$\ln(1+x)$	lnp1(x)	Inf(x) > -1
$\ln(\sqrt{x^2+y^2})$	ln_sqrtx2y2(x, y)	Inf($\|x\|$) + Inf($\|y\|$) > 0

Elementary Functions of Type `lx_interval` or `lx_real`		
Function Term	**C-XSC Name**	**Informations**
$\sin(x)$	$\sin(x)$	$\mathrm{Sup}(\lvert x\rvert) < 10^{308}$
$\sin(n\pi + x)$	$\mathrm{sin\_n}(x, n)$	$\mathrm{Sup}(\lvert x\rvert) < 10^{308},\ n \in \mathbb{Z}$
$\cos(x)$	$\cos(x)$	$\mathrm{Sup}(\lvert x\rvert) < 10^{308}$
$\cos((n + 1/2)\pi + x)$	$\mathrm{cos\_n}(x, n)$	$\mathrm{Sup}(\lvert x\rvert) < 10^{308},\ n \in \mathbb{Z}$
$\tan(x)$	$\tan(x)$	$\mathrm{Sup}(\lvert x\rvert) < 10^{308}$
$\cot(x)$	$\cot(x)$	$\mathrm{Sup}(\lvert x\rvert) < 10^{308}$
$\arcsin(x)$	$\mathrm{asin}(x)$	$\mathrm{Sup}(\lvert x\rvert) \leq 1$
$\arccos(x)$	$\mathrm{acos}(x)$	$\mathrm{Sup}(\lvert x\rvert) \leq 1$
$\arctan(x)$	$\mathrm{atan}(x)$	any interval x
$\mathrm{arccot}(x)$	$\mathrm{acot}(x)$	any interval x
$\sinh(x)$	$\sinh(x)$	$\mathrm{Sup}(\lvert x\rvert) \leq 6.24331476 \cdot 10^{15}$
$\cosh(x)$	$\cosh(x)$	$\mathrm{Sup}(\lvert x\rvert) \leq 6.24331476 \cdot 10^{15}$
$\tanh(x)$	$\tanh(x)$	any interval x
$\coth(x)$	$\coth(x)$	any interval $x \neq 0$
$\mathrm{arsinh}(x)$	$\mathrm{asinh}(x)$	any interval x
$\mathrm{arcosh}(x)$	$\mathrm{acosh}(x)$	$\mathrm{Inf}(x) \geq 1$
$\mathrm{arcosh}(1 + x)$	$\mathrm{acoshp1}(x)$	$\mathrm{Inf}(x) \geq 0$
$\mathrm{artanh}(x)$	$\mathrm{atanh}(x)$	$-1 < \mathrm{Inf}(x) \leq \mathrm{Sup}(x) < +1$
$\mathrm{artanh}(1 - x)$	$\mathrm{atanh1m}(x)$	$0 < \mathrm{Inf}(x) \leq \mathrm{Sup}(x) < 2$
$\mathrm{artanh}(-1 + x)$	$\mathrm{atanhm1p}(x)$	$0 < \mathrm{Inf}(x) \leq \mathrm{Sup}(x) < 2$
$\mathrm{arcoth}(x)$	$\mathrm{acoth}(x)$	any interval $x \neq 0$
$\mathrm{arcoth}(+1 + x)$	$\mathrm{acothp1}(x)$	$\mathrm{Inf}(x) > 0$
$\mathrm{arcoth}(-1 - x)$	$\mathrm{acothm1m}(x)$	$\mathrm{Inf}(x) > 0$

Table 1: Elementary Functions of Type `lx_interval` or `lx_real`

In Table 1, the formulation 'any interval x' has more or less the meaning

$$-2^{+9007199254740991} < \mathrm{Inf}(x) \leq \mathrm{Sup}(x) < 2^{+9007199254740991} \sim 10^{2711437152599295}.$$

As a first application the values $e = 2.718\ldots$ and $\pi = 3.141\ldots$ can be included with the C-XSC function calls $\mathtt{exp}(\boldsymbol{x})$ and $4 * \mathtt{atan}(\boldsymbol{x})$ respectively, where $\boldsymbol{x}$ is an extended staggered point interval including the value 1. The accuracy of the two enclosures is realized with 467 and 475 correct decimal digits respectively. The same function calls with $\boldsymbol{x}$ of type `l_interval` deliver an accuracy of the enclosures of only 307 and 323 correct decimal digits respectively. Thus, with the new type `lx_interval` a considerable enlargement of the accuracy is given more or less by factor 1.5.

Please notice that often used values like $e, \pi, \ln(\pi)\sqrt{\pi}, \sqrt{2}, \ldots$ are enclosed in C-XSC as constants of type `lx_interval` with a maximum accuracy of about 624 correct decimal digits. Thus, in practice the above function calls are fully unnecessary.

The next example demonstrates that by use of the extended staggered interval arithmetic extremely great function values can be included. With a precision of 480 decimal digits and with the argument $x = 6.24331476 \cdot 10^{15}$ the function value $y = e^x$ is included by

$$y = e^{6243314760000000} \in \underbrace{1.065543708933\ldots\mathbf{8812868979}}_{451 \text{ decimal digits}} {}^{91\ldots}_{48\ldots} \cdot 10^{2711437149053125}$$

with an accuracy of 451 decimal digits. With the Algebra Systems *Mathematica* or *Maple* the calculation of an approximation of $y = e^x$ fails due to an internal overflow. It should be noticed that already with $x = 709,78$ the inclusion of $y = e^x$ generates an overflow error message using the simple IEEE data type `interval`. However, the above inclusion can be tested with Mathematica by calculating the following approximations step-by-step

$$\alpha := 6243314760000000/\texttt{Log}[10] - 2711437149053125 = 0.0275712\ldots$$

$$\text{and} \quad 10^\alpha = 1.0655437089338451549 20\ldots 190017030217 60\mathbf{8812868979}70592\ldots$$

5 Taylor Arithmetic

We now consider functions f which are the composition of arithmetic operations and elementary functions in one variable. For functions of such a structure the Taylor coefficients or the derivatives of a given order p can recursively be calculated by applying appropriate derivative rules [4, 5, 6].

As an example we consider

$$f(x) = \begin{cases} e^{-1/x^2}, & x \neq 0, \\ 0, & x = 0. \end{cases}$$

For all $x \in \mathbb{R}$ the function f is infinitely differentiable and it holds [13] that

$$\lim_{x \to 0} f^{(k)}(x) = 0 = f^{(k)}(0), \quad k = 0, 1, 2, \ldots \ . \tag{1}$$

Due to $f^{(k)}(0) = 0$ the Taylor series of $f(x)$ with the point of expansion $x_0 = 0$ is the zero function and so the Taylor series does not equal $f(x)$ for all $x \neq 0$. Consequently, f is not analytic at the origin.

In C-XSC the class `lx_itaylor` is implemented for calculating enclosures of derivatives $f^{(k)}(x)$ up to order p, i.e. $k = 0, 1, 2, \ldots, p$. With this tool $f^{(k)}(x_1)$, $x_1 = 1.266 \cdot 10^{-8}$, is enclosed by the following four intervals, $k = 0, 1, 2, 3$:

$$
\begin{aligned}
f^{(0)}(x_1) &\in \underbrace{4.713035205041\ldots1394415677241}_{451\text{ decimal digits}}{}^{62\ldots}_{54\ldots}\cdot 10^{-2709673099980608},\\
f^{(1)}(x_1) &\in \underbrace{4.645468958788\ldots1086791545442}_{451\text{ decimal digits}}{}^{99\ldots}_{79\ldots}\cdot 10^{-2709673099980584},\\
f^{(2)}(x_1) &\in \underbrace{4.578871344729\ldots6036340742486}_{450\text{ decimal digits}}{}^{90\ldots}_{89\ldots}\cdot 10^{-2709673099980560},\\
f^{(3)}(x_1) &\in \underbrace{4.513228476517\ldots6189655217124}_{451\text{ decimal digits}}{}^{86\ldots}_{78\ldots}\cdot 10^{-2709673099980536}.
\end{aligned}
$$

The rather small values of $f^{(k)}(x_1)$ attest but not prove the above limit relation.

Now if f is evaluated with a complex variable $z = u + i \cdot v \in \mathbb{C}$ then, in case of $v \neq 0$, the two equations in (1) are no longer valid. In order to confirm this fact the derivatives $f^k(z)$ are included for $z_1 = 10^{-10} + 1.266 \cdot 10^{-8} \cdot i$ and $k = 0, 1, 2$. Again, with a precision of 480 decimal digits, the enclosures can be calculated in high accuracy by use of the class `lx_citaylor` implemented in C-XSC. The following C-XSC program

```
#include <iostream>
#include "lx_cinterval.hpp"
#include "lx_citaylor.hpp"

using namespace cxsc;
using namespace std;
using namespace taylor;

int main()
{
   stagprec = 30;    // Provides a precision of about
                     // 30*16=480 decimal digits.
   int p = 2;        // Taylor expansion of order p
   lx_cinterval z1;  // Inclusion for point of expansion
   lx_interval a,b;

   a = lx_interval(-318,"[1.0e308,1.0e308]");
   // a: Inclusion of the real part 10^(-10);

   b = lx_interval(-316,"[1.266e308,1.266e308]");
   // b: Inclusion of the imaginary part 1.266*10^(-8);

   z1 = lx_cinterval(a,b);
   // z1 includes 10^(-10) + 1.266*10^(-8) * i;

   lx_citaylor z(p,z1); // Constructor call for argument z

   lx_citaylor f;
   f = exp(-1/sqr(z));  // function call
```

```
        cout << SetDotPrecision(16*stagprec,16*stagprec)
             << Scientific;

        // Output of derivatives:
        for (int k=0; k<=p; k++)
        cout << "derivative " << get_j_derivative(f,k) << endl;
    }
```

delivers the inclusions:

$$\begin{aligned}
f^{(0)}(z_1) \quad &\in \quad \underbrace{+3.60063256687\ldots109774110254}_{451\text{ decimal digits}}{}^{74\ldots}_{55\ldots}\cdot 10^{+2709165962646103} + i\,\cdot \\
&\qquad \underbrace{-1.68511626706\ldots002820954547}_{451\text{ decimal digits}}{}^{45\ldots}_{53\ldots}\cdot 10^{+2709165962646104}, \\
f^{(1)}(z_1) \quad &\in \quad \underbrace{+1.65192845042\ldots017601046042}_{451\text{ decimal digits}}{}^{60\ldots}_{52\ldots}\cdot 10^{+2709165962646128} + i\,\cdot \\
&\qquad \underbrace{+3.94119556748\ldots009130997319}_{452\text{ decimal digits}}{}^{74\ldots}_{57\ldots}\cdot 10^{+2709165962646127}, \\
f^{(2)}(z_1) \quad &\in \quad \underbrace{-4.26900054509\ldots350293370921}_{450\text{ decimal digits}}{}^{29\ldots}_{31\ldots}\cdot 10^{+2709165962646151} + i\,\cdot \\
&\qquad \underbrace{+1.61843337313\ldots899158851447}_{452\text{ decimal digits}}{}^{89\ldots}_{05\ldots}\cdot 10^{+2709165962646152}.
\end{aligned}$$

As was expected, near the origin, where $f(z)$ is not analytic, the modulus of a derivative $f^k(z),\ z \neq 0$, can dramatically increase.

6 Concluding remarks

Another useful application of the new extended interval staggered arithmetic is the inclusion of all zeros of a given differentiable real function f using the interval Newton method [7, 8]. In C-XSC this task can be performed with the program `lx_nlfz_ex.cpp` available under
`/www.math.uni-wuppertal.de/org/WRST/index_de.html`

References

[1] F. Blomquist, W. Hofschuster, and W. Krämer, *Real and Complex Staggered (Interval) Arithmetics with Wide Exponent Range,* Preprint BUW-WRSWT 2008/1, Scientific Computing/Software Engineering, University of Wuppertal, 2008.

[2] W. Krämer, *Mehrfachgenaue reelle und intervallmäßige Staggered-Correction Arithmetik mit zugehörigen Standardfunktionen*, Technical Report of the Institute of Applied Mathematics, University of Karlsruhe, 1988.

[3] R. Lohner, “Interval arithmetic in staggered correction format”, In: E. Adams and U. Kulisch (Eds.), *Scientific Computing with Automatic Result Verification*, Academic Press, San Diego, California, pp. 301–321, 1993.

[4] H.-C. Fischer, *Schnelle automatische Differentiation, Einschließungsmethoden und Anwendungen,* Dissertation, Universität Karlsrhe, 1990.

[5] A. Griewank, “On Automatic Differentiation”, In: M. Iri and K. Tanabe (Eds.), *Mathematical Programming: Recent Developments and Applications,* Kluwer Academic Publishers, pp. 83–108,1989.

[6] A. Griewank and G. Corliss, Eds., *Automatic Differentiation of Algorithms: Theory, Implementation, and Applications,* Proceedings of Workshop on Automatic Differentiation at Breckenridge, SIAM, Philadelphia, 1991.

[7] H. Ratscheck and J. Rokne, *New Computer Methods for Global Optimization.* Ellis Horwood Limited, Chichester, 1988.

[8] E. Hansen, *Global Optimization Using Interval Analysis,* Marcel Dekker, New York, 1992.

[9] N. Revol and F. Rouillier, “Motivations for an arbitrary precision interval arithmetic and the MPFI library”, *Reliable Computing*, vol. 11, no. 4, pp. 275–290, 2005.

[10] M. Grimmer, K. Petras, and N. Revol, *Multiple Precision Interval Packages: Comparing Different Approaches*, École Normale Supérieure, Lyon, 2003.

[11] M. Grimmer, K. Petras, and N. Revol, “Multiple Precision Interval Packages. In Numerical Software with Result Verification”, In: Springer Lecture Notes in Computer Science, vol. 2991, pp. 64–90, 2004.

[12] W. Krämer, “Introduction to the Maple Power Tool intpakX”, *Serdica Journal of Computing*, Bulgarian Academy of Sciences, vol. 1, no. 4, pp. 467–504, 2007.

[13] W. Walter, *Analysis 1*, Reihe: Springer-Lehrbuch, 5. Auflage, 267 pp.

[14] U. Kulisch, *Computer Arithmetic and Validity*, Studies in Mathematics 33, de Gruyter, 2008.