Multiple Precision Interval Packages: Comparing Different Approaches

Markus Grimmer, Knut Petras, Nathalie Revol

Laboratoire de l'Informatique du Parallélisme

École Normale Supérieure de Lyon
Unité Mixte de Recherche CNRS-INRIA-ENS LYON n° 5668

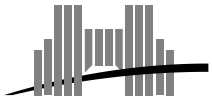
CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE

Multiple Precision Interval Packages: Comparing Different Approaches

M. Grimmer
K. Petras
N. Revol

June 2003

Research Report N° 2003-32



École Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : +33(0)4.72.72.80.37
Télécopieur : +33(0)4.72.72.80.80
Adresse électronique : `lip@ens-lyon.fr`

INRIA

Multiple Precision Interval Packages: Comparing Different Approaches

M. Grimmer
K. Petras
N. Revol

June 2003

Abstract

We give a survey on packages for multiple precision interval arithmetic, with the main focus on three specific packages. One is within a Maple environment, `intpakX`, and two are C/C++ libraries, GMP-XSC and MPFI. We discuss their different features, present timing results and show several applications from various fields, where high precision intervals are fundamental.

Keywords: Multiple precision interval arithmetic, packages, ease of use, efficiency, reliability.

Résumé

Dans cet article est effectué un tour d'horizon des outils implémentant l'arithmétique par intervalles en précision multiple. Un coup de projecteur est donné sur trois de ces outils. Le premier est un paquetage développé pour Maple: `intpakX`, les deux autres sont des bibliothèques C/C++: GMP-XSC et MPFI. Leurs spécificités sont présentées, puis leurs performances sont données. Enfin, sont développées quelques applications, issues de domaines d'application divers, pour lesquelles l'arithmétique par intervalles à grande précision est fondamentale.

Mots-clés: Arithmétique par intervalles en précision multiple, paquetages et bibliothèques, facilité d'emploi, efficacité, fiabilité.

Multiple Precision Interval Packages: Comparing Different Approaches

M. Grimmer Markus.Grimmer@math.uni-wuppertal.de
Universität Wuppertal, Fachbereich Mathematik
Wissenschaftliches Rechnen / Softwaretechnologie
K. Petras K.Petras@tu-bs.de
TU Braunschweig, Institut für Angewandte Mathematik
N. Revol Nathalie.Revol@ens-lyon.fr
INRIA, project Arenaire, LIP, École Normale Supérieure de Lyon

1 Why develop Multiple Precision Interval Packages?

1.1 Need for Arbitrary Precision Interval Softwares

Multiple precision is a floating-point arithmetic, where the number of digits of the mantissa can be any fixed or variable value. It is usually applied to problems where it is important to have a high accuracy (e.g., many digits of π). However, for such, mostly numerical, algorithms where extra computing precision is required, the loss of accuracy can be predictable or unpredictable. If this loss is predictable, then multiple precision arithmetic perfectly fulfils the application's needs. When it is unpredictable, interval arithmetic can reveal useful to bound this loss of accuracy. Of course, this interval arithmetic must also be based on a multiple precision arithmetic. Hence, we are particularly interested in
numerical problems, with a large and unpredictable loss of accuracy.

Although multiple precision interval arithmetic might help, one should be aware of the fact that this often means an increase in the computational time and memory usage, cf. section 3.

Before classifying softwares that implement multiple precision interval arithmetic, let us first point out the possibility to vary or not the computing precision. Sometimes *multiple precision* refers only to extended and fixed precision, whereas *arbitrary precision* is used for variable precision. In this paper, *multiple precision* refers to extended precision, whether it is variable or not. Arbitrary precision arithmetic offers the possibility to set precision to an arbitrary value as needed in the computations; this can be done either statically or dynamically, *i.e.* during the computations. Interval packages based on GNU multiple precision arithmetic or Maple arithmetic are such. But there are also approaches offering multiple precision arithmetic without the possibility to vary the precision, as the staggered multiple precision arithmetic in the XSC languages [24, 23].

1.2 Organization of the paper

The motivations and needs for multiple precision interval arithmetic packages are developed in this first part; they can be different if the package is developed for a software for scientific computing such as Maple or MatLab or as a library in a given programming language. The second part consists of a survey of various packages, and in particular the packages developed by the authors are presented: intpakX for Maple, MPFI in C and GMP-XSC in C++. In the third part, a comparison in terms of performance is conducted. In the last part, various applications are presented: interval Newton, range enclosure, linear algebra, quadrature, application to mathematical finance, global optimization.

1.3 Interval Arithmetic in Softwares for Scientific Computing

The reasons for the implementation of an interval package for softwares dedicated to scientific computing, such as MatLab, Maple or Mathematica, are different from those motivating interval libraries for standard programming languages like C++ (developed in §1.4). These software environments are not only powerful tools for various kinds of computations, but, in contrast to programming languages, they primarily aim at usability and convenience as well as at visualization of data. Moreover, they serve as means of education in schools and universities. Thus, an interval package for one of these software environments serves two different sets of purposes.

On the one hand, it offers the possibility to incorporate interval arithmetic into a tool for scientific computing:

- perform interval computations in an environment that did not offer (enough) interval support beforehand;
- incorporate verification into existing applications;
- combine symbolical computations with interval evaluation for computer algebra systems (Maple or Mathematica).

On the other hand, an interval package also benefits from the specificities of these computing environments:

- check results computed by this software or results from different environments by graphically displaying them;
- learn or teach interval arithmetic;
- use interval arithmetic without the need of being fully familiar with the concepts of a programming language.

Apart from these reasons, there are also reasons having their origin in the need for multiple precision computing. It gives the user the chance to get arbitrarily accurate approximations of numbers not representable in IEEE-754 arithmetic.

Since round-off errors are usually not expected to occur in computations of a Computer Algebra System (which is capable to do symbolical computations to avoid this kind of problem), you are apt to mistake rounded results as exact results. This especially applies to Maple or Mathematica as softwares for symbolical computing where numerical errors are not considered to happen. The combination of multiple precision and interval arithmetic is thus welcome to justify this feeling of safety.

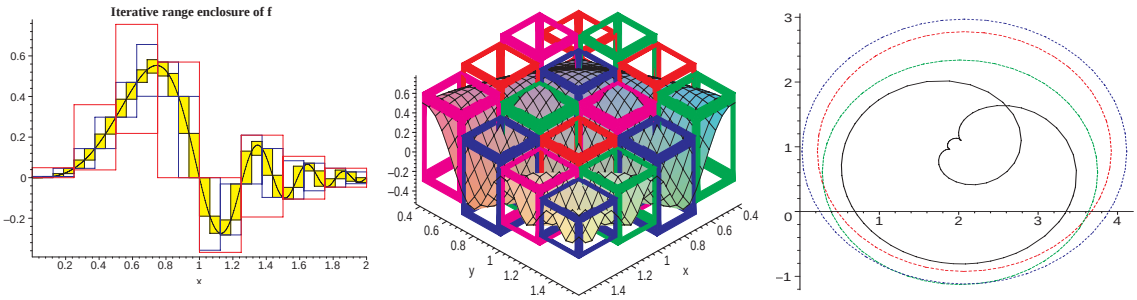
Moreover, random precision is a much more natural way to deal with numbers than the standardized floating-point arithmetic. This point has to be particularly mentioned regarding the convenience and the fact that an environment like Maple (especially with a GUI) has also teaching purposes.

Of course there are numerous other scientific reasons to work with multiple precision in different kinds of mathematical, engineering or other applications, as you can see from the applications mentioned in this article. The combination of multiple precision arithmetic and verification can be called natural in some way since both techniques aim at getting better results.

As for `intpakX`, the considered Maple package described in section 2.1, the main reasons for the recent implementation were the following. One of the intentions was to offer some algorithms and extended operations using the existing `intpak` framework [11] which had been part of the Maple Share Library. At the same time, the visualization of these interval applications should be

possible, also as a means to easily confirm the computed data. Examples of this can be found in [14]; here, we only want to give two examples of the enhanced or more convenient graphical output possibilities (see illustration).

The other specific intention was the fact that intervals can be defined in Maple without using `intpakX`, and expressions can be evaluated, but that this evaluation does not behave according to all expected mathematical properties. Proper rounding is not provided (see section 2) and there are a number of other effects (like the simplification of terms prior to their evaluation, e.g. simplification of $[1,2]-[1,2]$ into 0). Facing this, there was a need for an interval arithmetic which would have the expected mathematical properties and correct operators.



1.4 Libraries for Arbitrary Precision Interval Arithmetic: Efficiency Issues

Another point of view prevails to the implementation of multiple precision interval arithmetic libraries into programming languages. Here, the main issue is efficiency rather than ease of use or educational purposes. Indeed, the intended user is expected to be already familiar with a programming language and willing to incorporate interval computations into his/her programs. However, few programming languages or compilers have interval datatypes and operations as native ones (cf. §2.2). Thus, to allow interval computations in environments that do not support intervals, the solution consists in developing libraries.

Libraries developed into an existing programming language are compiled, *i.e.* interval operations are executed faster than within an interpreted package, which is the case in the previous section. Furthermore, the memory management is tailor-made by the programmer of the library, which implies that this memory management can be made more efficient than a general one, since it is dedicated to a specific kind of applications. A last source of efficiency lies in the use or not of the processor's arithmetic unit: in Maple, all computations are done with radix-10 digits and all operations are thus software ones; with XSC languages (cf. §2.2), operations are based on floating-point ones, with GMP-based libraries (cf. §2.3 and §2.4), they are based on machine integers.

However, the programming of a multiple precision interval arithmetic library does not necessarily involve a tremendous amount of work: efficient libraries for multiple precision floating-point arithmetic can be used as a basis; much of the work is then already done, in particular memory management issues are already handled by GMP.

Finally, if the chosen programming language offers operator overloading, such as object-oriented languages, then existing applications can be verified at almost no cost (modification of the datatypes). This feature is common to packages developed for scientific computing softwares as well as libraries developed in C++ for instance (cf. §2.3 and §2.4).

2 Survey of Various Implementations

2.1 Packages for Scientific Computing Softwares

IntLab for MatLab

IntLab [41, 40] is an interval arithmetic package for MatLab. The main goal of its author, S. Rump, is to compute verified results and to keep the performances of MatLab in terms of ease of use and of execution time. Thus, a clever way to perform interval matrix operations has been developed, which takes benefit of MatLab highly optimized routines. Procedures have been developed for automatic differentiation and for reliable solving of linear and nonlinear systems of equations. Since standard functions are not reliable in MatLab, S. Rump has also implemented guaranteed standard functions; a critical point is reliable and accurate argument reduction, and to implement it, so-called "long " arithmetic has been developed. Up to version 4.1.1, the procedures which have been developed are mainly the ones required for argument reduction: arithmetic operations, the π constant and the exponential function. This long arithmetic is "rudimentary, slow but correct" according to its author. Few standard functions are available and matrices with long components are not yet possible.

Package for Mathematica

Interval is a datatype in Mathematica. J. Keiper [22] justifies its introduction with arguments similar to the ones given in §1.3: education of a large number of potential users to interval arithmetic, ease of use, graphical possibilities and some examples to demonstrate the power of this arithmetic.

Since Mathematica offers high precision floating-point arithmetic, it was quite natural that intervals can have as endpoints exact numbers or floating-point numbers with arbitrary precision. However, J. Keiper warns against two unpleasant phenomena with Mathematica intervals. The first one is that outward rounding is done by the software, since setting rounding modes at a low level is non portable; this implies some excess in the width of computed intervals. This leads for instance to a width of 4.44089×10^{-16} for the following interval: `Interval [1.]` with Mathematica version 4.2, even with 1.0 being exactly representable, *i.e.* the width should be 0.

The second unpleasant phenomenon is illustrated by the following sequence:

```
In[1]:= e=15-39Sin[EulerGamma]-2Pi;

In[2]:= N[Interval[{e,e}],16]
Out[2]= Interval[{-12.5652, -12.5652}]

In[3]:=  N[Interval[{e,e}],17]
Out[3]= Interval[{-12.565205412135305, -12.565205412135305}]
```

i.e. the intersection of the two resulting intervals, which should each contain the exact value, is empty. Here is Keiper's explanation [22]. *Also, an assumption is made that is known to be false: library functions for the elementary functions are assumed to be correct to within one ulp and directed rounding by one ulp is used to "ensure" that the resulting interval contains the image of the argument. There are no known examples for which the elementary functions are in error by more than an ulp for high-precision arithmetic.* The previous computation is precisely such an example. Indeed, the accuracy of standard functions is not known with Mathematica: no precise specification is given for these functions.

In Mathematica, LU-related procedures and nonlinear systems solver can have intervals as arguments and return guaranteed results. Some extensions or applications based on this package are to be found in [7] and [31].

intpakX for Maple

`intpakX` is a Maple package for interval arithmetic. It contains data types, basic arithmetic and standard functions for real interval arithmetic and complex disc arithmetic. Moreover, it implements a handful of algorithms for validated numerical computing and graphical output functions for the visualization of results. The package `intpakX` thus gives the user the opportunity to do validated computing with a Computer Algebra System.

History and Implementation.

The first `intpak` version was created in 1993 by R. Corless and A. Connell [11] as an effort to incorporate real intervals into Maple. In 1999, `intpakX` was released by I. Geulig and W. Krämer [14, 15] as an extension to `intpak` incorporating important changes as well as a range of applications and an additional part for complex numbers. The current release `intpakX v1.0` (June 2002) is a redesigned package combining the formerly separate packages in one new version. In December 2002, it has been released by Waterloo Maple as *Maple PowerTool Interval Arithmetic* [1]. The package is implemented as a Maple module (a feature Maple offers since version 6, which allows information hiding as in the major object oriented programming languages).

The most important feature of the package is the introduction of new data types into Maple for

- real intervals and
- complex disc intervals.

A range of operators and applications for these data types (see below) have been implemented separately (with names differing from the standard operators' names), so that the new interval types do not rely on the (rough) notion of an interval Maple already has. So, `intpakX` intervals can be used safely with the implemented operators.

Also rounding is done separately, since there are examples where the rounding included in Maple is not done correctly. Namely, the expression $x - \varepsilon$ ($x > 0$ Maple floating-point number with n decimal digits, $\varepsilon < 10^{-n}$) yields x when Rounding is set to 0 or $-\infty$, although it should yield the largest n-digit number smaller than x. As needed in interval arithmetic, rounding is done outwardly in computations with `intpakX`.

`intpakX` functions, though being separately implemented, use standard Maple operators and functions (`intpakX` interval *sin* uses the Maple *sin* implementation for example). Thus, errors in Maple arithmetic being greater than $1ulp$ will affect `intpakX` results.

The graphical functions included in `intpakX` make it more convenient to use Maple graphics for interval computations. They use Maple graphics features to offer special output for the visualization of the intervals resulting from the concerned `intpakX` functions.

Functional Range and Applications.

As mentioned above, `intpakX` defines Maple types for real intervals and complex disc intervals.

Here is a survey of the operators, functions and algorithms that `intpakX` includes. First, functions and operators for real intervals are given followed by the incorporated numerical algorithms. After that, the functions for complex intervals are specified.

- On the level of basic operations, `intpakX` includes the four basic arithmetic operators denoted as &+, &-, &*, &/. It also includes extended interval division as an extra function.
- Furthermore there are power, square, square root, logarithm and exponential functions (note that square is implemented separately from general multiplication as needed for intervals) as well as union and intersection.

- A set of standard functions has been implemented (sin, cos, tan as well as their inverse and hyperbolic versions).
- Reimplementations of the Maple construction, conversion and unapplication functions are added.

The following numerical algorithms are implemented to work with the foregoing functions:

- verified computation of zeros (Interval Newton Method) with the possibility to find all zeros of a function on a specified interval;
- range enclosure for real-valued functions of one or two variables, which uses either interval evaluation or evaluation via the mean value form and adaptive subdivision of intervals.

Using the above algorithms, the user can choose between a non-graphical and a graphical version displaying the resulting intervals of each iteration step.

As for real intervals, there is a range of operators for complex disc arithmetic:

- in addition to the basic arithmetic operators, there are area-optimal multiplication and division as an alternative to carry out these operations;
- as a further function, the complex exponential function has been implemented.

Range enclosure for complex polynomials serves as an application for complex interval arithmetic.

2.2 Languages and libraries

Few languages and compilers include a support for interval arithmetic; let us quote the XSC languages [3] (Pascal [24], C/C++ [23]) and the Sun Forte compilers for Fortran and C/C++ [43]. However, times are changing and for instance the introduction of interval arithmetic in the BLAS library is being discussed (cf. `http:///www.netlib.org/blas/blast-forum/`).

XSC (eXtended Scientific Computing) languages

Multiple precision interval arithmetic is even more rare. Besides interval arithmetic, the XSC languages offer a "staggered" arithmetic, which is a multiple, fixed, precision. The chosen precision enables the exact computation of the dot product of two vectors of reasonable size with "double" floating-point components. This multiple precision type can be used for floating-point and interval values, it is called "dotprecision", and the corresponding arithmetic "staggered". This type of multiple-precision numbers consists of a vector $(x_1, ..., x_n)$ of double precision numbers whose sum yields the represented number $x = \sum_i x_i$. Such vectors can contain up to 39 entries. Indeed, it is limited to the dot product of double precision vectors, whose range of exponents is $\{-1022, \cdots, 1023\}$, plus extra positions to take into account the vectors' length.

The details of this type of multiple precision arithmetic and its implementation can be found in [27] or [23]. Apart from computing accurate dot product, it has also been used for Horner evaluation of a polynomial in the interval Newton algorithm [26].

The range arithmetic

Other works are libraries rather than languages or compilers, they are developed in a given programming language. For instance, the "range" library has been developed by Aberth et al. as early as 1992 [4]: C++ has been chosen for its operator overloading facility and the library is thus easy to use; indeed, formulas involving "range" operands can be written exactly as formulas with usual floating-point operands. It has to be mentioned that the C++ language has evolved and the "range" library is now difficult to compile because its C++ is too old for most compilers. The

"range" type is an arbitrary precision floating-point type coupled with a "range", which controls the accuracy of the represented number: only relevant digits are stored, these digits being more relevant than the range which can be seen as an absolute error. For instance, when a cancellation occurs, the result has a small number of digits.

Aberth has developed numerical algorithms using this automatic accuracy control and presented them in [5]. This range arithmetic can be seen as a form of interval arithmetic, as long as no large intervals are used, since they cannot be represented as range objects: the range has to be smaller (in absolute value) than the corresponding number.

Brent's MP, Augment and a multiple precision interval package by Yohe

The oldest library implementing multiple precision interval arithmetic may well be the one developed in Fortran by Yohe in 1980 [44]. It is based on the one hand on the Augment preprocessor, which replaced arithmetic operators by calls to the appropriate functions, as operator overloading was not available, and on the other hand on Brent's MP package for multiple precision floating-point arithmetic [10]. However, Brent himself recommends to use a more recent package than MP: "MP is now obsolescent. Very few changes to the code or documentation have been made since 1981! [...] In general, we recommend the use of a more modern package, for example David Bayley's MPP package or MPFR" (cf. `http://web.comlab.ox.ac.uk/oucl/work/richard.brent/pub/pub043.html`).

Other works

The two packages which will be introduced now are based either on MPFR, following Brent's recommendation: the MPFI package, or on the floating-point type of the GMP package [2]: the GMP-XSC package. MPFI is presented first because it contains more "basic" functionalities, whereas GMP-XSC provides more elaborated things such as special functions.

2.3 MPFI

In order to implement an arbitrary precision interval arithmetic, a multiple precision floating-point library was needed. MPFR (*Multiple Precision Floating-point Reliable arithmetic library*) was chosen because it is a library for arbitrary precision floating-point arithmetic that is compliant with the IEEE-754 standard [18] and even more. It provides exact outward rounding facility for the arithmetic and algebraic operations, for conversions between different data types and also for the standard functions. Furthermore, it is portable and efficient: MPFR is based on GMP and efficiency is a motto for its developers, and the source code is available. MPFR is developed by the Spaces team, INRIA, France [13].

The MPFI library implements interval arithmetic on top of MPFR. MPFI stands for *Multiple Precision Floating-point Interval arithmetic library*, it is a portable library written in C and its source code and documentation can be freely downloaded [36].

Intervals are implemented using their endpoints, which are MPFR floating-point numbers. The specifications used for the implementation are based on the IEEE-754 standard:

- an interval is a connected closed subset of $\mathbb{R}$;
- if *op* is an n-ary operation and $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ are intervals, the result of $op(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$, the operation *op* performed with interval arguments, is an interval such that: $\{op(x_1, \ldots, x_n), x_i \in \boldsymbol{x}_i\} \subset op(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$;
- in case $op(x_1, \ldots, x_n)$ is not defined, then a `NaN` ("Not a Number", which stands for an invalid operation) is generated, *i.e.* the intersection with the domain of *op* is not taken prior to the operation;

- each endpoint carries its own precision (set at initialization or modified during the computations).

The arithmetic operations are implemented and all functions provided by MPFR are included as well (trigonometric and hyperbolic trigonometric functions and their inverses). Conversions to and from usual and GMP data types are available as well as rudimentary input/output functions. The code is written according to GMP standards (functions and arguments names, memory management).

The largest achievable computing precision is determined by MPFR and depends in practice on the computer memory. The only theoretical limitation is that the exponent must fit in a machine integer. It suffices to say that it is possible to compute with numbers of several millions of binary digits if needed. The computing precision is dynamically adjustable in response to the accuracy needed.

2.4 GMP-XSC

GMP-XSC was intended as a fast multiple precision package that might supplement the well-known package C-XSC. The name indicates that it is also based on the GNU multiple precision subroutines. The need for GMP-XSC came from Application 4.5 described below. The problem was to evaluate an integral over the real half axis. The integrand is oscillatory and thus, the cancellations are huge. This calls for a high precision arithmetic. Furthermore, the integrand contains special functions. One of them as well as elementary functions had to be evaluated in the complex plane. Finally, huge high order derivatives had to be estimated on intervals by using interval arithmetic. Multiple precision is not necessary but we need an arithmetic that deals with large exponents.

GMP-XSC contains all features that are necessary to solve the problem that was just described briefly and that will be described in more details below. It has some extra functions and its completion will go on. GMP-XSC is essentially a C++-wrapper for the C-program GMP-SC. This GMP-SC does the main work. It contains GMP-like routines including arithmetic operations, most elementary functions and some special functions for floating-point numbers (`mpf_t`, the original GMP data type), complex numbers (`mpc_t`), intervals (`mpi_t`), rectangular complex intervals (`mpci_t`), "large doubles" (`large_d`, which is a structure consisting of a double and an integer meaning the exponent) and "large intervals" (`large_i`, which is an interval between two `large_d`-s).

For real intervals, e.g., we now discuss the available functions. Arithmetic operations, square roots, squares, exponential function, logarithm, sine, cosine and arc tangent are incorporated. Furthermore, there are procedures `mpi_sico` and `mpi_sicoh`. They compute sine and cosine (or the corresponding hyperbolic functions) simultaneously. The reason is that they are often required together. One example is automatic generation of derivatives of $\sin f(x)$ (or $\cos f(x)$) if these derivatives are known for f. We get an arbitrary number of derivatives with only arithmetic operations if we know $\sin f(x)$ and $\cos f(x)$. For details on this automatic differentiation technique, see the book of Rall [34]. Moreover, with our method, we obtain the cosine almost without extra cost if we have the sine. The method of computing the sine consists of argument reductions by the factor $1/2$ in each step. We therefore compute $\sin(x/2)$ and $\cos(x/2)$. This readily yields $\cos x$. Special functions that were needed for the above-mentioned project are also implemented. This includes the Gamma function, the complementary error function and Hermite functions (see Abramowitz and Stegun [6] or Lebedev [30]).

Now, we sketch how the special functions are computed. The Gamma function is computed by producing a large argument via

$$\Gamma(x) = \frac{\Gamma(x+n)}{x(x+1)\cdots(x+n-1)}. \tag{1}$$

Then, we apply Stirling's formula

$$\ln\Gamma(z) \sim (z-\frac{1}{2})\ln z - z + \frac{1}{2}\ln(2\pi) + \sum_{\nu=1}^{\infty} \frac{\mathcal{B}_{2n}}{2n(2n-1)z^{2n-1}}. \tag{2}$$

For $z > 0$, the true value is enclosed by two consecutive partial sums.

$$B_s'(x) = B_{s-1}(x), \qquad \int_0^1 B_s(x)\,dx = 0, \qquad B_0(x) = 1.$$

The formal asymptotic expansion is not convergent, since

$$\mathcal{B}_{2s} = 2(-1)^{s+1}\frac{(2s)!}{(2\pi)^{2s}}(1+\theta_n) \qquad \text{with } 2^{-2s} < \theta_n < (2^{2s-1}-1)^{-1}.$$

The computation of the (rational) Bernoulli numbers is costly. Hence, numerator and denominator are stored for $\mathcal{B}_0, \ldots \mathcal{B}_{100}$ and can therefore be divided with given precision. In order to accelerate computation of the Gamma function for very high precision, storing more Bernoulli numbers would help. We could also calculate them online using the Fourier expansion of Bernoulli monosplines. This might be topic of a further version. Finally, it is tried to balance the number of factors in (1) and of summands in (2) in order to optimize the amount of work. Details are given in the documentation (see `http://www.tu-bs.de/~petras/software.html`).

One possible definition of the ν-th Hermite function is

$$H_\nu(z) = \frac{1}{\Gamma(-\nu)}\int_0^\infty e^{-t^2-2tz}t^{-\nu-1}, \qquad \Re\nu < 0$$

Our purpose is to compute the Hermite function for $\nu < 0$ and $z \geq 0$. However, for $\nu \geq 0$, it can be obtained from

$$H_{\nu+1}(z) = 2zH_\nu(z) - 2\nu H_{\nu-1}(z).$$

We have the series expansion

$$\begin{aligned} H_\nu(z) &= \frac{1}{2\Gamma(-\nu)}\sum_{k=0}^{\infty}\Gamma\left(\frac{k-\nu}{2}\right)\frac{(-2z)^k}{k!} \\ &= 2^\nu\sqrt{\pi}\Bigg\{\frac{1}{\Gamma\left(\frac{1-\nu}{2}\right)}\sum_{k=0}^{\infty}\frac{(-\nu)(2-\nu)\ldots(2k-2-\nu)\left(2z^2\right)^k}{(2k)!} \\ &\qquad -\frac{2z}{\Gamma\left(-\frac{\nu}{2}\right)}\sum_{k=0}^{\infty}\frac{(1-\nu)(3-\nu)\ldots(2k-1-\nu)\left(2z^2\right)^k}{(2k+1)!}\Bigg\} \end{aligned}$$

and the formal asymptotic expansion

$$H_\nu(z) \sim (2z)^\nu\sum_{k=0}^{\infty}\frac{(-\nu)_{2k}}{k!(-4z^2)^k} \qquad \text{for } |\arg z| \leq \tfrac{3}{4}-\delta,\ \delta > 0,$$

where we denote by

$$(a)_m = a\cdot(a+1)\cdots(a+m-1)$$

the Pochhammer symbol. In both cases and our parameters and arguments, two consecutive partial sums enclose the true function value if all further summands alternate in sign. For the series, this holds only if the moduli of the summands decrease.

First, we try the (relatively cheap) asymptotic expansion in order to get the prescribed accuracy. If this is not sufficient, the series expansion is used.

The summation has to be done with sufficiently many digits. The problem can be the increase of the modulus of the summands in case of the series expansion. A heuristic estimate is done with a double precision computation of the logarithm of the summands, where Gamma functions are replaced by Stirling's formula. During the computation of the sum, we determine the maximum modulus of the contributing summands in order to determine the loss of precision rigorously. If the heuristically estimated precision is not sufficient, it is corrected and summation is restarted. In our application [32], we need H_ν and its derivative, $H'_\nu = 2\nu H_{\nu-1}$ simultaneously. It is unnecessarily expensive to compute the two values separately, since they have many terms in common. These considerations are realized in

```
void mpf_hermite01_eps(mpf_t h, mpf_t dh, mpf_t x, mpf_t nu,
     mpf_t eps )
```

The values $h = H_\nu(x)$ and $H'_\nu(x)$ are computed with absolute accuracy ε.

The computation of the complementary error function is discussed in depth in [33], where the most interesting case is for complex arguments.

2.5 Final Remarks

MPFI and GMP-XSC have been developed at the same time. The authors did not know about the projects of each other. It is intended to produce one library that contains the advantages of both products.

3 Comparison and Results

From now on, the focus will be on three packages, one for Maple: `intpakX`, and two C/C++ libraries: MPFI and GMP-XSC. These packages are recent and they offer arbitrary precision and the usual set of standard functions.

They are compared using the following criteria: ease of use, accuracy and timings. Before detailing the comparison, let us insist on `intpakX` features.

3.1 intpakX specificities

Before presenting numbers, we want to recall the framework for `intpakX` as a package for a Computer Algebra System.

As a Maple package, the environment for the package is different from the ones that programming language libraries are working in. The reason for using Maple is not primarily to simply do numerical computations but to connect these to symbolical computation (e.g. to numerically evaluate terms obtained by symbolical computations). Furthermore, a Computer Algebra System (abbreviated as CAS in the following) has to be easy to use to serve its purpose in teaching and as a means of confirmation and visualization in attendance of other computing environments.

A greater ease of use often comes at the expense of less efficiency, so the expectation is that a CAS package might be efficient for the CAS in question, but usually slower than a programming library. Also, results obtained using the package in a graphical user interface (or GUI) will look different from those you get using a command line version of the CAS.

Apart from these considerations, the architecture of the multiple precision arithmetic and data type still plays an important role.

3.2 Ease of Use?

Since ease of use is a quality which is hardly measurable, we will only mention that a Computer Algebra System such as Maple does of course take into account ease of use much more than common programming languages, such as C or C++, do. Particularly for small to medium size projects, as well as for the definition of most kinds of mathematical notions, Maple is much faster

to program than a programming language that is not designed to offer especially easy ways for mathematical definitions.

3.3 Accuracy

In a multiple precision environment, you like to get especially tight enclosures of all results. In Maple, you have the possibility to set precision via an environment variable `Digits`. This variable is used in `intpakX` functions to calculate the necessary number of decimal digits for any calculation. In C/C++ libraries, variable and arbitrary computing precision is also possible: this is achieved through dynamic memory allocation to store the numbers.

The tightness of the results is governed by the way outward rounding is performed. With MPFR and thus MPFI, exact directed rounding is done, *ie.e* the resulting intervals are the tightest guaranteed enclosures of the exact results. In `intpakX`, the resulting intervals are rounded outwardly by 1 ulp, yielding an interval with a width of 2 ulps in a single calculation. In any case, the accuracy of the result thus only depends on the precision used and on the number of calculations done. In the implemented interval methods, the precision is adjusted to yield a result with the desired accuracy, and the user can specify the relative diameter of the intervals to be computed (or the number of iteration steps to be done). Thus, it depends on the settings how tight the resulting intervals are.

While the quality of results is a feature immanent to high precision arithmetic, the question of memory and speed determines to what degree a package can be used in practice. There are indeed limitations toward the number of digits from a practical point of view, since on the one hand the memory required to store the intermediate variables is limited and on the other hand execution times are growing with the number of digits used and can become prohibitive. From the times given in the subsection on test results below, you can get an impression on the time needed for computations with a certain number of digits. This includes some tests of implemented algorithms regarding time and number of iterations to reach a predefined accuracy.

There is also a maximum number of decimal digits predefined in the Maple kernel options which is set to 268435448. This is only a theoretical limit to the computations done since the tests were done with smaller numbers of digits. The limits with MPFI and GMP-XSC are that the exponents must fit into a machine integer (this limitation should be soon removed from GMP/MPFR) and that the mantissas cannot exceed the available memory.

3.4 Performances in terms of time

The following tests were executed with different packages to compare the speed of

- standard Maple arithmetic and interval arithmetic using `intpakX`;
- `intpakX` as a CAS package and programming languages/libraries;
- MPFR and MPFI;
- C-XSC and GMP-XSC.

3.4.1 Test Arrangements

- In Maple, `intpakX` results have been compared to non-interval Maple results, both with different numbers of decimal digits.
- The same calculations have been done in C-XSC using real floating-point numbers, real intervals and multiple precision intervals (staggered arithmetic) with different lengths.
- They have also been performed using GMP-XSC.

- Finally, the same set of tests has been done using MPFR and MPFI.

Two particular tests have been executed:

1. to test the speed of basic operators, matrix multiplications of different sizes and with varying computing precision have been done in the environments mentioned;
2. standard functions have been tested in expressions with single or multiple occurrences of different standard functions.

Furthermore, the section on applications contains tests on the applications included in the `intpakX` package and various applications either solved by GMP-XSV or MPFI or which were the starting motivation for their development.

More details on the performed tests are presented together with the corresponding results.
The results have been measured on a Sun Ultra 10 440MHz computer, except the MPFI experiments which have been conducted on a Sun Ultra 5 330MHz, and for which a correcting multiplying factor of 330/440 has been applied. The software versions used for the computations are Maple8 with `intpakX v.1.0`, C-XSC 2.0 beta2 with GNU `g++-3.2`, GMP 3.2 with `gcc-3.2`, and MPFI 1.1, based on GMP-4.1.2, with gcc-3.0.3 or g++-3.0.3. All times are displayed in seconds.

3.4.2 Results

Matrix multiplications (Maple)
The following times have resulted from a multiplication of matrices by hand, since the absence of overloaded operators in `intpakX` does not allow a direct multiplication of matrices. Different (full) matrices have been tested, including the Hilbert Matrix. This implies that the times below are not strictly valid for all examples, but show the ratio between non-interval and `intpakX` interval computations.

The numbers of digits given (15, 30, 90) are related to the corresponding lengths fo C-XSC real intervals and staggered intervals with 2 or 6 reals (a real variable has about 15 decimal digits accuracy).

Data Type/Matrix Size	15 Digits	90 Digits	540 Digits
Maple float			
10×10	0.08	0.21	0.78
20×20	0.86	1.85	6.86
30×30	2.59	5.75	25.94
intpakX interval			
10×10	2.65	2.78	6.72
20×20	20.16	23.38	63.59
30×30	72.46	81.84	237.28

The ratio between interval computations and their floating-point counterparts is given in the following table:

Matrix Size	15 Digits	90 Digits	540 Digits
10×10	33	13	8.6
20×20	23	13	9.3
30×30	28	14	9.1

It can be seen that the ratios for the different numbers of digits stay in the same range for growing matrix sizes while decreasing with growing numbers of digits.

Matrix multiplications (C-XSC)

Size	imatrix	l_imatrix (2 reals)	l_imatrix (6 reals)
20x20	0.07	0.15	0.68
100x100	7.92	16.18	83.19
200X200	63.70	132.38	663.07

Matrix multiplications (GMP-XSC)

Size	15	30	90	540 Digits
20x20	0.07	0.09	0.09	0.09
100x100	8.19	9.09	9.41	12.83
200X200	79.10	81.60	86.20	121.28

Matrix multiplication (MPFI)

Times using MPFR are not reported here. Previous experiments [37] report an overhead factor between 2 and 4 for matrix operations.

Size	15	30	90	540 Digits
20x20	0.01	0.01	0.02	0.03
100x100	1.89	2.16	3.88	5.71
200X200	15.78	18.59	23.99	47.97

GMP-XSC is slightly slower than MPFI because the focus was more on special functions with real or complex argument than on sophisticated rounding routines (see the remark in section 2.5).

If you consider the standard number of 15 digits, times using C-XSC or GMP-XSC are about ten times faster than with `intpakX`, and times using MPFI are more than 50 times faster than with `intpakX`. With growing numbers of digits, the increase of times is greater in C-XSC than in Maple or especially in GMP-XSC.

This effect becomes even more visible testing the standard functions.

Standard functions (Maple)

The standard functions were evaluated executing 1000 iterations with changing values for x. The computation times for the parameters is included in the numbers, but did not account for a major part of the times measured.

As an example, we give the Maple code for the performed operation (including the loading of the package):

```
restart;
libname:="/home/wmwr3/grimmer/maple/intpak/new/v1.0/lib",libname;
with(intpakX):

Digits:=90;
wid:=0.001;
imax:=1000;

expr1:=sin(x);
f:=inapply(expr1,x);     # convert to interval expression
sti:=time();
for i from 1 to imax do
  param:=i*0.01:
```

```
  param2:=param+wid:
  result[i]:=f([param,param2]):
od:
fti:=time();
dti:=fti-sti;
```

	Maple float (90 Digits)	intpakX int. (90 Digits)	ratio
$sin(x)$	4.63	19.42	4.1
$sinh(x)$	2.74	4.71	1.7
$exp(x)$	2.60	4.20	1.6

Standard functions (C-XSC)

	interval	l_interval (2 reals)	l_interval (6 reals)
$sin(x)$	0.0014	17.61	57.20
$sinh(x)$	0.0015	25.95	92.53
$exp(x)$	0.0012	17.74	78.78

Standard functions (single occurrence, GMP-XSC)

	15	30	90 Digits
$sin(x)$	0.22	0.30	0.74
$sinh(x)/cosh(x)$	0.25	0.35	0.68
$exp(x)$	0.16	0.23	0.52

The tables show that on the one hand, C-XSC times using staggered arithmetic are much higher even than Maple times and at the same time fast growing with increasing numbers of reals in one staggered variable. This shows that the C-XSC staggered arithmetic is not efficient being implemented as software only. The time amounts would be significantly smaller using a dotprecision arithmetic with a hardware accumulator (i.e. proposed by Kulisch [28]).

On the other hand, you can also see that standard IEEE arithmetic (as used in C-XSC real numbers) is still much faster than GMP multiple precision arithmetic with the same number of digits.

Computing expressions with multiple occurrences of standard functions yields similar results (roughly speaking, times add up if you do more than one evaluation of a standard function; times thus strongly depend on the expressions themselves).

In addition to the results above, here are some more results doing only a single evaluation of the standard functions with greater numbers of digits in `intpakX` and GMP-XSC.

Standard functions (Maple)

	10000 Digits	20000 Digits	40000 Digits	100000 Digits
$sin(x)$	14.62	57.25	196.95	1586.5
$sinh(x)$	2.92	10.79	41.04	234.03
$exp(x)$	3.28	12.21	46.59	249.05

Standard functions (GMP-XSC)

	10000 Digits	20000 Digits	40000 Digits	100000 Digits
sin	2.50	9.80	39.44	225.47
sicoh	1.18	4.83	18.51	104.12
exp	1.15	4.63	17.81	103.38

Since MPFR is slower than GMP, times are not reported here: it suffices to say they are longer. Indeed, the results returned by MPFR are eaxctly rounded results and this can explain the relatively high computing times. MPFI also returns the tightest enclosures of the exact results. It has been observed that MPFI times are much higher than MPFR times: a possible explanation for the trigonometric functions is that argument reduction is performed twice, once by MPFR and once by MPFI. But since this phenomenon is also observed for the other functions, it is a hint that programming improvements have to be done in MPFI.

Expecting a programming library to be faster, it strikes that the ratios comparing Maple, MPFR and GMP-XSC times are relatively small. The MPFI times are even higher.

Further Remarks

- Considering the comparison of Maple and `intpakX` times, we found decreasing ratios for greater numbers of digits. The decrease of the factors with increasing number of digits can be credited to the fact that the additional time for interval computations comprises time for arithmetic operations and some overhead time. The influence of the latter decreases the more time is used by arithmetic operations.
- For large numbers of digits, the computation time using the GUI version of Maple was significantly higher (up to twice the time) than using the command line version.
- For periodical functions (sin, cos, etc.) `intpakX` times are about 5-7 times larger than Maple floating-point operations due to a shift of the interval bounds and numerous case differentiations. For monotonous functions as the exponential function, the factor is approximately 2. The tests included the reading of the parameter and storage of the result which resulted in factors slightly smaller than 2.

Results of two of the implemented applications can be found in the following section.

4 Applications

In this section we give results of some applications for the interval packages.

4.1 intpakX for Maple

`intpakX` includes some applications of the defined interval types, functions and operators. In this subsection, we want to give some numbers to show to what extent and up to which level of accuracy the packages can be used conveniently.

The tested applications are the Interval Newton Method and Range Enclosure for functions of one real variable. For a theoretical introduction of the applications as such you can e.g. refer to [14].

The main criterion to be watched was the speed of the application executing the algorithms with growing numbers of iterations.

Here are times for the Interval Newton Method, first testing the computation of an interval containing 6 zeros with growing number of digits, then testing the computation of a growing number of zeros with constant number of digits (100) for $\sin \frac{1}{x-1}$ as an example.

Interval Newton Method

Digits	1000	2000	4000	10000
Time(secs.)	79.66	259.08	873.620	5072.57

Obviously the complexity of operations is quadratic with respect to number of digits used here, whereas it is linear in the number of zeros:

Zeros	Iteration steps	Time
31	247	26.78
318	2398	268.00
3183	23243	2666.71

Range Enclosure (2D)

Finally, some times for the range enclosure of a function of one real variable are given below, doing different numbers of subdivisions of the starting interval (here: evaluation of $f(x) = \exp(-x^2) * \sin(\pi * x^3)$ over the interval $X := [0.5, 2.]$).

Number of Subdiv.	5	10	15
Time	27.89	437.14	6834.20

4.2 Extended Interval Newton Algorithm

Interval Newton algorithm [17] has been adapted to arbitrary precision computations and implemented, cf. [35]. In this example, the algorithm is primarily intended for interval computations: it has been developed in order to enclose reliably every real zero of the function. Indeed, it uses an interval enclosure of the graph of the function and searches for zeroes in the intersection of this enclosure with the $Ox_1 \cdots x_n$ hyperplane, where $x_1, \cdots, x_n$ denote the variables.

With an interval arithmetic based on hardware floating-point numbers, the accuracy of the result is limited; in particular with a root of multiplicity $m > 1$ or a cluster of m zeroes, the accuracy on this zero is the computing precision divided by m. However, interval Newton algorithm is based either on a contracting scheme or, if the contraction is not efficient enough, on a bisection. This implies that arbitrary accuracy can be reached, if only enough computing precision is available. This remark led us to adapt and implement interval Newton algorithm in MPFI.

The adapted interval Newton algorithm exhibits the following features:

- arbitrary accuracy can be reached both on the enclosure of the zeros and on the range of the function on this enclosure, up to computer limits (time / memory);
- the computing precision is automatically adapted when needed; this happens when bisection is no more possible because the current interval contains only two floating-point numbers, or when the function evaluation does not narrow when the argument gets narrower.

Some experiments have been conducted on polynomials [35]. The first series concerns Chebyshev polynomials. They are known to be difficult to evaluate accurately even if they take their values in $[-1, 1]$, because their coefficients are large. A consequence is thus that it is quite difficult to get a small "residual" $F(X)$, smaller than the stopping threshold ε_Y. For instance, MatLab determines only 6 roots of C_{30}, the Chebyshev polynomial of degree 30 (it finds 24 complex roots for the 24 remaining ones), with 5 correct decimal digits. It finds only 8 roots of C_{26}, with 3 correct decimal digits. Yet the coefficients of C_{26} or of C_{30} are exactly representable by machine numbers and these results are not due to the approximation of the coefficients by double precision floating-point numbers. The proposed interval Newton algorithm gives very satisfactory results: every root is determined, no superfluous interval is returned as potentially containing a root and the existence and uniqueness of the roots in each enclosing interval is proven, for most of them.

A second series presents quite the same conclusions obtained with the Wilkinson polynomial of degree 20: $W_{20}(x) = \prod_{i=1}^{20}(x-i)$ written in the expanded form. The initial precision is chosen large enough to enable the exact representation of the coefficients. This polynomial is difficult to evaluate accurately because its coefficients are large (their order of magnitude is 20!) and because it takes large values between its roots (their order of magnitude is 10^{16}). Consequently it is very difficult for our algorithm (essentially very time-consuming) to discard intervals not containing zero. The results are thus small enclosures for the roots along with a proof of their existence and uniqueness and a long list of other, not discarded, intervals, covering almost the whole interval $[1, n]$.

When the coefficient of X^{19} is perturbed by the interval $[-2^{-19}, 2^{-19}]$, every point between 8 and 20 is a root of a perturbed polynomial belonging to this interval polynomial; indeed, our algorithm returns small enclosures for the roots 1 to 7 and a covering of $[7.91, 22.11]$.

4.3 Numerical Linear Algebra

Nowadays, algorithms for solving systems of linear equations with result guarantee are very refined. If, however, the condition number of the involved matrix is large, the use of refined techniques but ordinary floating-point calculations usually does not help. One example is the Hilbert matrix:

$$H_n := \left(\frac{1}{\nu+\mu-1}\right)_{\nu,\mu=1,\ldots,n}.$$

Its condition number is about 3.5^n. Hence there is little hope to get the validated inverse for large n by using double precision numbers. A further problem is that we usually do not have to invert the Hilbert matrix but some other matrix with unknown, possibly large condition number. This calls for using multiple precision interval arithmetic. The user may choose the precision in advance but the inversion routine doubles the precision until it either produces the inverse matrix or reaches a user defined maximal precision.

The used algorithm is well-known (see Rump [39]). In case we want to solve a system of linear equations,

$$Ax = b, \qquad A \in \mathbb{R}^{n\times n}, \quad b \in \mathbb{R}^n,$$

we first compute an approximate inverse R by, say, the Gaussian algorithm and an approximate solution $\widetilde{x}$. If the entries of A are intervals, we take the respective midpoints and compute the approximate inverse of the resulting matrix. Introducing $y = x - \widetilde{x}$, we can rewrite the system as

$$y = R(b - A\widetilde{x}) + (I - RA)y =: f(y).$$

Thus, we can start a fixed point iteration for f. This converges if the spectral radius of $I - RA$ is smaller than 1. If R is close to the inverse of A, this spectral radius is close to zero and we have fast convergence.

Inversion is done in the same way. We just have to replace $b \in \mathbb{R}^n$ by the $n \times n$ identity matrix.

On a usual PC, the limits on n are not given by the increase of computation time but mainly by the size of the memory. In Table 1, we list the computation times t (in seconds on a 2.6 GHz Pentium) used for inversion of the $n \times n$ Hilbert matrix for certain values of n. The number of used binary digits in the computation was $32 \cdot \lfloor 11(n+2)/32 \rfloor$. The precision of the output is measured by $\mathrm{diam}([H_n^{-1}])$, the maximal diameter of an entry in the computed enclosure for H_n^{-1}.
The precision for $n \in \{128, 256\}$ can be relaxed slightly to gain some speed. $n = 256$, e.g., was also tested with $32 \cdot \lfloor 10(n+2)/32 \rfloor$ binary digits in the computation. Computation time was about 7402 seconds but the diameter was $> 10^{-6}$.

Remark 1 *There are benchmark competitions of supercomputers based on the inversion of very large matrices. It is, however, said explicitly that the produced matrices may have nothing to do with the true inverse. On the Dagstuhl conference, which underlies these proceedings, U. Kulisch*

n	time (s)	d	diam($[H_n^{-1}]$)
16	0.074	176	$0.37 \cdot 10^{-23}$
32	0.91	352	$0.64 \cdot 10^{-23}$
64	18.45	704	$0.17 \cdot 10^{-31}$
128	367.5	1408	$0.47 \cdot 10^{-48}$
256	8740	2816	$0.16 \cdot 10^{-80}$

Table 1: CPU time, number of used binary digits, diameter of the result.

proposed to introduce a benchmark test, which consists of the inversion of the 500×500 *Hilbert matrix with a certain number of guaranteed correct digits. Now, we know at least the correct result for* H_{256}^{-1} *up to absolute precision of 80 digits.*

4.4 Kronrod-Patterson Quadrature

Kronrod-Patterson quadrature formulae

$$Q_{n_k}^{KP,k}[f] = \sum_{\nu=1}^{n_k} a_\nu^{[k]} f(x_\nu^{[k]}), \qquad -1 \le x_\nu^{[k]} \le 1$$

for the determination of $I[f] = \int_{-1}^{1} f(x)\,dx$ are defined as follows. Let Q_n^G be the Gaussian quadrature formula with n nodes.

1. $Q_n^{KP,0} = Q_n^G$
2. $Q_{n_k}^{KP,k}$
   a) involves $n_k = 2^k(n+1) - 1$ nodes
   b) yields the correct integral value for all polynomials of degree $\le 3 \cdot 2^{k-1}(n+1) - 1$.

We call $Q_{n_{k+1}}^{KP,k+1}$ a Kronrod-Patterson extension of $Q_{n_k}^{KP,k}$. Not even the existence of Kronrod-Patterson extensions for $k > 1$ has been proved theoretically. Using interval arithmetic, it is possible to give an existence proof and to determine nodes $x_\nu^{[k]}$ and coefficients $a_\nu^{[k]}$. This is done as follows: define

$$p^{[k]}(x) = \prod_{\nu=1}^{n_k} (x - x_\nu^{[k]}).$$

Property 2b) is equivalent to

$$\int_{-1}^{1} p^{[k]}(x) q(x)\,dx = 0 \qquad \text{for all } q \in \mathbb{P}_{2^{k-1}(n+1)-1}$$

(see Brass [8, Theorem 55]). We set

$$p^{[k]} = \sum_{\nu=0}^{n_k} \alpha_\nu T_\nu,$$

where $T_\nu(x) = \cos \nu \arccos x$ are the Chebyshev polynomials of the first kind. The leading coefficient of $p^{[k]}$ is 1 and that of T_{n_k} is 2^{n_k-1}. Therefore, $\alpha_{n_k} = 2^{1-n_k}$. Since the nodes of $Q_{n_{k+1}}^{KP,k+1}$ contain those of $Q_{n_k}^{KP,k}$, $p^{[k+1]}/p^{[k]}$ is a polynomial. Therefore, we may set

$$\frac{p^{[k+1]}(x)}{p^{[k]}(x)} = \sum_{\nu=0}^{n_{k+1}-n_k} \beta_\nu T_\nu(x).$$

Here, we have $\beta_{n_k} = 2^{1+n_k-n_{k+1}}$. Knowing $p^{[k]}$, we have to compute $p^{[k+1]}/p^{[k]}$ such that

$$\int_{-1}^{1} p^{[k]}(x)\frac{p^{[k+1]}(x)}{p^{[k]}(x)}T_\lambda(x)\,dx = 0 \qquad \text{for } \lambda = 0, 1, \ldots, 2^{k-1}(n+1) - 1$$

This means that, knowing the α_ν, we have to compute the β_ν such that

$$\sum_{\nu=0}^{n_{k+1}-n_k} \beta_\nu \cdot \left(\sum_{\mu=0}^{n_k} \alpha_\mu \int_{-1}^{1} T_\nu(x)T_\mu(x)T_\lambda(x)\,dx \right) = 0 \quad \text{for } \lambda = 0, 1, \ldots, 2^{k-1}(n+1) - 1. \tag{3}$$

This looks like a homogeneous system (for the β_ν) but note that $\beta_{n_k} \neq 0$ and the corresponding terms can be shifted to the right-hand side. Up to now, everything seems solvable. The important condition $x_\nu^{[k]} \in [-1, 1]$, however may be hurt. Nodes could either stay real but be outside $[-1, 1]$ or even vanish to the complex plane. This would be a problem, since $Q_{n_k}^{KP,k}$ should replace the continuous linear functional $I : C[-1, 1] \to \mathbb{R}$. In the case that $x_\nu^{[k]} \notin [-1, 1]$, $Q_{n_k}^{KP,k}$ would not even be a functional on $C[-1, 1]$.

Knowing enclosures for $x_\nu^{[k]}$, $\nu = 1, \ldots, n_k$, we get enclosures for the coefficients α_ν of the nodal polynomial. This yields a linear system of equations with interval coefficients. Its solution defines the polynomial $p^{[k+1]}/p^{[k]}$ with interval coefficients $[\beta_\nu]$ (being enclosures for the β_ν). Using the interval Newton method, we can compute guaranteed intervals $[x_\nu^{[k+1]}]$ for the zeros of all the polynomials represented by the interval coefficients. If the intervals $[x_\nu^{[k+1]}]$ are pairwise disjoint and contained in $[-1, 1]$, we have proved the existence of a Kronrod-Patterson extension (the uniqueness follows from the theory).

One question is how to evaluate a polynomial p and its derivative efficiently and in a stable manner, if we have the representation

$$p(x) = \sum_{\nu=0}^{n} c_\nu T_\nu(x).$$

First, note that

$$\frac{T'_{\nu+1}(x)}{\nu+1} = \frac{\sin(\nu+1)\arccos(x)}{\sqrt{1-x^2}} \equiv U_\nu(x).$$

U_ν is the Chebyshev polynomial of the second kind of degree ν. We have

$$T_0(x) = 1, \qquad T_1(x) = x, \qquad T_\nu(x) = 2xT_{\nu-1}(x) - T_{\nu-2}(x), \quad \nu = 2, 3, \ldots$$

and

$$U_0(x) = 1, \qquad U_1(x) = 2x, \qquad U_\nu(x) = 2xU_{\nu-1}(x) - U_{\nu-2}(x), \quad \nu = 2, 3, \ldots$$

Using this recursion is usually not a good idea. If $|x|$ is close to 1 (or larger), an error propagates with a factor of about 2 (or larger) with each application of the recursion formula. Alternatively, we deduce the recursions

$$T_{2\nu(x)} = 2T_\nu^2(x) - 1, \qquad T_{2\nu+1}(x) = 2T_{\nu+1}(x)T_\nu(x) - T_1(x)$$

and

$$U_{2\nu(x)} = 2T_\nu(x)U_\nu(x) - U_0(x), \qquad U_{2\nu+1}(x) = 2T_{\nu+1}(x)U_\nu(x).$$

Again, the error may be multiplied by about 2 in each application. In order to obtain the value of the Chebyshev polynomial of degree n, however, we have to apply the recursion only about $\log_2 n$ times.

An important property of a quadrature formula is its positivity, i.e., the positivity of its coefficients a_ν (in our case $a_\nu^{[k]}$). From theory, many nice properties follow from this positivity (see, e.g., Brass and Förster [9]). The coefficients are given by

$$a_\nu^{[k]} = \int_{-1}^{1} l_\nu(x)\,dx \qquad l_\nu(x) = \frac{p(x)}{(x - x_\nu^{[k]})p'(x_\nu^{[k]})}, \qquad p(x) = \prod_{\nu=1}^{n_k}(x - x_\nu^{[k]}).$$

In order to calculate the integral, we may apply one of the known quadrature formulas that are exact for all polynomials of degree n_k. We chose the Clenshaw-Curtis formula (see Brass [8]). This requires the calculation of function values of the integrand l_ν. The derivative of p in the denominator is calculated as above. The remaining factor is calculated as defined.

Existence and positivity are proved by computing the enclosures for nodes and coefficients. Non-existence may have different reasons. In our cases, it was proved by showing that $p^{[k]}$ and its first derivative have the same sign at -1. Hence, there must be a zero of $p^{[k]}$ or its first derivative on the left of the basic interval, which means that we do no longer have the full number of zeros in $[-1, 1]$.

We have tested the program for $n_k < 1024$. Again, the restrictions on n_k came from restrictions on the sizes of the matrices in the linear system (3). The results are

Theorem 1 *The Kronrod-Patterson extensions with $n_k < 1024$ for $n_0 \notin \{2, 4\}$ exist and are positive. If $n = n_0 = 2$ (or $n = n_0 = 4$) , we have existence and positivity for $n_k \leq 47$ (or $n_k \leq 319$) as well as non-existence for $n_k = 95$ (or $n_k \leq 637$, respectively).*

4.5 An Oscillating Integrand from Mathematical Finance

Starting point of GMP-XSC was the numerical computation of the price of an arithmetic-average Asian option according to Schröder's integral representation [42]. The computationally complicated part is

$$\sum_{|b|\in\{\nu,\nu+2\}} \int_0^\infty H_{-\nu-4}\left(\frac{\cosh y}{\sqrt{2q}}\right) e^{yb} \Im\left\{ e^{i\pi b} \operatorname{erfc}\left(\frac{y + bh + i\pi}{\sqrt{2h}}\right)\right\} dy \tag{4}$$

where ν, q and h are certain positive parameters. H_μ is a Hermite function, which is defined for negative μ by

$$H_\mu(z) = \frac{1}{\Gamma(-\mu)} \int_0^\infty e^{-t^2-2tz} t^{-\mu-1}\, dt \tag{5}$$

(see, e.g., Lebedev [30]). From this, we get all Hermite functions by applying

$$H_{\mu+1}(z) = 2zH_\mu(z) - 2\mu H_{\mu-1}(z).$$

$\Im$ denotes the imaginary part and erfc is the complementary error function,

$$\operatorname{erfc}(z) = \frac{2}{\sqrt{\pi}} \int_z^\infty e^{-t^2}\, dt.$$

Properties of these two special functions are given, e.g, in Abramowitz/Stegun [6] and Lebedev [30].

The main difficulty is that, due to the oscillatory nature of the integrand, the complete integral is smaller then the maximum of the integrand by a factor of 1/10 to the power of dozens or even hundreds. This required a validated error control with the help of automatic differentiation combined with interval computations or complex interval computations. Evaluation of the integrand requires the computation of special functions (partially or non-real arguments) with interval arithmetic. This lead to the features that are incorporated in GMP-XSC up to now.

Details are given in [32].

4.6 Global optimization: some difficult cases

For one of the authors, a motivation to work on multiple precision interval arithmetic came from difficulties encountered with the global optimization of some "nasty" functions.

Interval arithmetic is the arithmetic of choice to do global optimization of continuous functions which are not necessarily convex. Indeed, it provides global information on the function, such as an enclosure of its range over a whole (interval) set. On the opposite, deterministic classical

numerical algorithms provide an optimum which is guaranteed to be global only under some stringent conditions. As far as probabilistic methods are concerned, they return an optimum with prescribed probability to be close to the global optimum, but which is not guaranteed. Interval algorithms, such as Hansen's algorithm [16, 20], have been developed in order to determine the guaranteed global optimum of a function. These methods can be costly in terms of computational time and memory.

However, even interval arithmetic can fail to determine the global optimum of some functions. Indeed, the functions which are difficult to optimize can be roughly classified into two types. Some functions are extremely flat, cf. the Ratz 8 function represented on the left of figure 4.6. With flat functions, the optimum is very well approximated but the optimizer is not accurately determined; a whole region containing points where the function takes values close to the optimal one is returned.

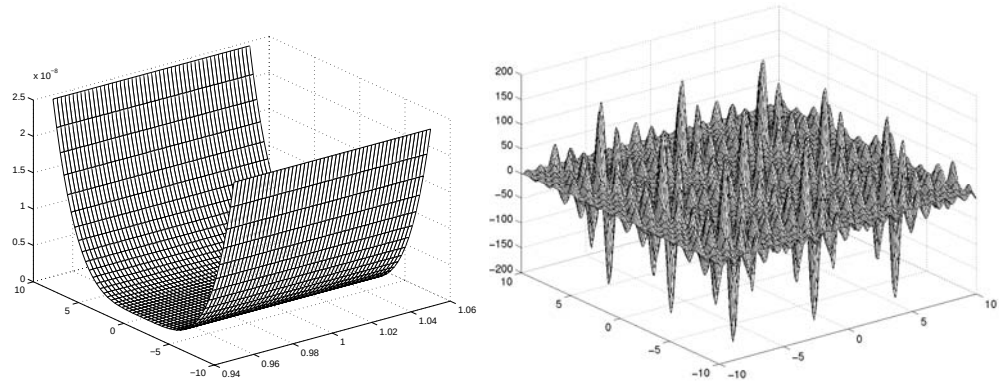
Other nasty functions are "egg-box" functions"; these functions have a huge number of local optimizers, such as the following functions: the Levy (n^o 3) function on $[-10, 10]^2$ (cf. right part, figure 4.6) defined as

$$f(x,y) = -\left(\sum_{i=1}^{5} i\cos[(i-1)x+i]\right) \times \left(\sum_{j=1}^{5} j\cos[(j+1)y+j]\right)$$

has 760 local minima, 18 global minima; with $n = 10$, the following function has 10^{10} local minima and only one global minimum:

$$f(x_1, \ldots x_n) = 10\sin(\pi x_1)^2 + (x_n - 1)^2 + \sum_{i=1}^{n-1}(x_i - 1)^2[1 + 10\sin(\pi x_{i+1})^2].$$

Furthermore, the local optima can be very close to the global one, which means that the interval algorithm cannot discard them. An example can be found in chemistry, with a problem of molecular conformation: the problem is to determine the localization of particles, through the minimization of the electrostatic energy of the system. This problem takes values ranging from the global minimum to the infinity (when two particles are located at the same place): this means that multiple precision can help to magnify the difference between local and global minima. Furthermore, the number of local minimizers is huge and it is impossible to gather them into a single region, since every local minimizer is isolated. The memory needed to store the list of potential optimizers is thus large. It is a modern challenge to determine and prove the optimality of configurations with over 120 particles.



The global optimization of such functions can greatly benefit from multiple precision interval arithmetic. The development of a dedicated software is an ongoing work.

5 Conclusion

This paper proposes a survey of existing packages for multiple precision interval arithmetic. Among the recent packages which offer arbitrary precision and every usual mathematical facilities, three have been more closely studied: `intpakX` for Maple, MPFI and GMP-XSC for C/C++. They are also representative of the different existing trends: either ease of use and educational purposes or efficiency and reliability through the use of a programming language.

These three packages have been compared in section 3 The results reinforce the *a priori* opinion that the price to pay for ease of use is speed. However, they have also put in evidence that getting tight and guaranteed results also takes time. The efficiency of the implementation of standard functions will be reworked; thanks to the elaboration of this common paper, authors are now aware of this point!

It is now expected that multiple precision interval arithmetic will be more and more widely used. Indeed, various packages, which are complete, easy to use and efficient, are now available. In particular, more applications will be developed using these packages. We hope to get a larger community of users and to get remarks from them that will help improving our packages.

References

[1] Maple PowerTool Interval Arithmetic.
`http://www.mapleapps.com/powertools/interval/Interval.shtml`

[2] GMP, GNU Multiple Precision library.
`http://www.swox.com/gmp/`

[3] XSC Languages.
`http://www.math.uni-wuppertal.de/wrswt/xsc-sprachen.html`

[4] O. Aberth and M.J. Schaefer. Precise computation using range arithmetic, via C++. *ACM TOMS*, 18(4):481–491, December 1992.

[5] O. Aberth. *Precise numerical methods using C++*. Academic Press, 1998.

[6] M. Abramowitz and I.A. Stegun. *Handbook of Mathematical Functions.* National Bureau of Standards, 1964.

[7] Y. Akyildiz, E. D. Popova and C. P. Ullrich. *Towards a more complete interval arithmetic in Mathematica.* Proceedings of the Second International Mathematica Symposium, 29–36, 1997.

[8] H. Brass. *Quadraturverfahren.* Vandenhoeck & Ruprecht, Göttingen 1977.

[9] H. Brass, and K.-J. Förster. *On the application of the Peano representation of linear functionals in numerical analysis.* In: Recent progress in inequalities (Niš, 1996, Math. Appl., 430, Kluwer Acad. Publ., Dordrecht, 1998), 175–202.

[10] R. P. Brent. A Fortran multiple-precision arithmetic package. *ACM TOMS*, 4:57–70, March 1978.

[11] A.E. Connell and R.M. Corless. An experimental interval arithmetic package in Maple. In *Num. Analysis with Automatic Result Verification*, 1993.

[12] Corliss, G. *intpak for Interval Arithmetic in Maple.* Journal of Symbolic Computation, 11, 1994.

[13] D. Daney, G. Hanrot, V. Lefèvre, F. Rouillier, and P. Zimmermann. The MPFR library. `http://www.mpfr.org`, 2001.

[14] Geulig, I., Krämer, W. *Computeralgebra und Verifikationsalgorithmen.* University of Karlsruhe, 1998.

[15] Geulig, I., Krämer, W. *Intervallrechnung in Maple - Die Erweiterung intpakX zum Paket intpak der Share-Library.* University of Karlsruhe, 1999.

[16] E. Hansen. *Global optimization using interval analysis.* Marcel Dekker, 1992.

[17] E. Hansen and R.I. Greenberg. An interval Newton method. *J. of Applied Math. and Computing*, 12:89–98, 1983.

[18] T.-J. Hickey, Q. Ju and M.-H. Van Emden. Interval arithmetic: from principles to implementation *J. of ACM*, 2002.

[19] N. Higham. *Accuracy and stability of numerical algorithms.* 2nd ed., SIAM, 2002.

[20] R.B. Kearfott. *Rigorous global search: continuous problems.* Kluwer, 1996.

[21] R.B. Kearfott, G.W. Walster. On stopping criteria in verified nonlinear systems or optimization algorithms. *ACM TOMS*, 26(3):373–389, 2000.

[22] J. Keiper. Interval arithmetic in Mathematica. *Interval Computations*, (3), 1993.

[23] R. Klatte, U. Kulisch, C. Lawo, M. Rauch, and A. Wiethoff. *C-XSC a C++ class library for extended scientific computing.* Springer Verlag, 1993.

[24] Klatte, R., Kulisch, U. et al. *PASCAL-XSC.* Springer Verlag, 1991.

[25] O. Knueppel. PROFIL/BIAS - a fast interval library. *Computing*, 53(3-4):277–287, 1994.

[26] W. Krämer, U. Kulisch and R. Lohner. *Numerical toolbox for verified computing II – Advanced Numerical Problems* . To appear (1998).

[27] Krämer, W. *Mehrfachgenaue reelle und intervallmässige Staggered-Correction Arithmetik mit zugehörigen Standardfunktionen.* Report of the Institut für Angewandte Mathematik, Karlsruhe, 1988.

[28] Kulisch, U. *Advanced Arithmetic for the Digital Computer. Design of the Arithmetic Units.* von U. W. Kulisch Springer, Wien, 2002.

[29] M. Lerch, G. Tischler, J. Wolff von Gudenberg, W. Hofschuster and W. Krämer. *The interval library filib++ 2.0.* `http://www.math.uni-wuppertal.fr/org/WRST/software/filib.html`, Preprint 2001/4, Universität Wuppertal, Germany 2001.

[30] N.N. Lebedev. *Special functions and their applications.* Dover Publications, Inc., New York, 1972.

[31] R. Maeder. The Mathematica Programmer: Interval Plotting and Global Optimization. *The Mathematica Journal*, 7(3):279–290, 1999.

[32] K. Petras. *Numerical Computation of an Integral Representation for Arithmetic-Average Asian Options.* Preprint available on `http://www.tu-bs.de/~petras/publications.html`

[33] K. Petras. *A Method for Calculating the Complex Complementary Error Function with Prescribed Accuracy.* Preprint available on `http://www.tu-bs.de/~petras/publications.html`

[34] L.B. Rall. *Automatic differentiation – Techniques and Applications*, Springer Lecture Notes in Computer Science, Vol. 120, Berlin, 1981.

[35] N. Revol. Newton's algorithm using multiple precision interval arithmetic. To appear in *Numerical Algorithms*, 2003. Research report 4334, INRIA, 2001, `http://www.inria.fr/rrrt/rr-4334.html`.

[36] N. Revol and F. Rouillier. The MPFI library. `http://www.ens-lyon.fr/~nrevol`, 2001.

[37] N. Revol and F. Rouillier. Motivations for an arbitrary precision interval arithmetic and the MPFI library. To appear in *Reliable Computing*, 2003.

[38] F. Rouillier and P. Zimmermann. Efficient isolation of polynomial real roots. To appear in *J. of Computational and Applied Math.*, 2003. Research report 4113, INRIA, 2001, `http://www.inria.fr/rrrt/rr-4113.html`.

[39] S.M. Rump. *Solving algebraic problems with high accuraccy.* Habilitationsschrift, Karlsruhe, 1983 (also contained in U. Kulisch, W.L. Miranker (eds.): *A new approach to scientific computation.* Proceedings of a symposium held at the IBM Research Center, Yorktown Heights, N.Y., 1982. Academic Press, New York, 1983, pp. 51–120.)

[40] S. Rump. *Developments in reliable computing, T. Csendes ed.*, chapter INTLAB - Interval Laboratory, pages 77–104. Kluwer, 1999.

[41] S. Rump. Fast and parallel interval arithmetic. *BIT*, 39(3):534–554, 1999.

[42] M. Schröder. *The Laplace transform approach to valuing exotic options: the case of the Asian option.* Mathematical finance, Trends Math., Birkhäuser, 2001, 328–338.

[43] Sun Microsystems, Inc. *C++ interval arithmetic programming reference.* 2000.

[44] J. M. Yohe. Portable software for interval arithmetic. *Computing*, Suppl. 2, 211–229, 1980.