Interval Computations
No 2, 1993

An Experimental Interval Arithmetic Package in Maple

Amanda E. Connell and Robert M. Corless*

We describe an experimental arbitrary-precision interval arithmetic package written for the computer algebra system Maple. The functions implemented are those of the proposed Basic Interval Arithmetic Subroutine (BIAS) library. We give here an overview of the package design, some examples of the usage and code, and a report on our experiences. The package will be made available under the name INTPAK from the Maple share library (anonymous ftp to `daisy.uwaterloo.ca` in a subdirectory of the directory `maple`) at a future date.

Экспериментальный пакет интервальной арифметики для системы Maple

А. Е. Коннелл, Р. М. Корлесс

Описан экспериментальный пакет интервальной арифметики произвольной точности, написанный для системы компьютерной алгебры Maple. Функции реализованы согласно спецификациям библиотеки базовых подпрограмм интервальной арифметики (BIAS). Статья содержит обзор структуры пакета, несколько примеров использования и программирования, а также описывает опыт его применения авторами. В будущем пакет будет доступен всем желающим под именем INTPAK в составе библиотеки разделенного пользования Maple по anonymous ftp, по адресу `daisy.uwaterloo.ca`, в одном из подкаталогов каталога `maple`.

*This work was supported by NSERC. Amanda Connell's summer work term was financed by an NSERC Undergraduate scholarship, and her presentation of this work at the Numerical Analysis with Automatic Result Verification conference in Lafayette, LA in 1993 was supported by NSERC, by NSF, and by Waterloo Maple Software, Inc. We would also like to thank George Corliss, Stephen Merrill, and Jim Phillips for many informative discussions, pointers, and bug reports.

1 Introduction

The computer algebra system Maple [3] has high-quality arbitrary-precision 'point' arithmetic. If the 'environment variable' `Digits` is set to a value high enough that the hardware or C floating-point arithmetic cannot be used, then Maple's software arbitrary-precision arithmetic is used. In this case, arithmetic operations and single function evaluations are claimed to be accurate to within 0.6 ulp (units in the last place). [The basis of this claim is that functions are evaluated using as many guard digits as necessary: for example, the routine `evalf/W` uses a complicated formula involving Digits and the magnitude of $1 + e \cdot x$ to conservatively determine how many guard digits are required to guarantee accuracy of $W(x)$ to 0.6 ulp. See [4] for a discussion of $W(x)$.] However, this level of accuracy is claimed only for single operations, and no facilities exist in Maple (as distributed) for studying the effects of the *propagation* of errors, either of roundoff errors or of data uncertainties.

Maple also has very efficient facilities for exact integer and rational arithmetic, which avoids roundoff error altogether, but this comes with the inherent 'storage' penalty of possibly rapid growth of the integers involved as the computation proceeds. Maple allows very large integers (limited essentially only by the amount of memory on the machine), but some problems exhibit exponential growth in the storage requirements, which makes this approach impractical in these cases. In other cases, though, where the growth of the integers is not too severe, Maple's existing arithmetic is already satisfactory: one can do a long computation over the rationals and then convert to a decimal expansion as the final step and incur no more error than 0.6 ulp. Examples include solving linear systems of equations with rational coefficients or evaluating a determinant of a matrix of rational numbers.

However, no computation with expressions containing transcendental functions or requiring root extraction can be done in this way, and doing the computation symbolically instead can incur a tremendous penalty in the form of 'intermediate expression swell', which can cause super-exponential growth in the storage requirements as the computation proceeds. Further, the numerical stability of the final expression may be in doubt (Maple makes no claim to evaluate arbitrary *expressions* to 0.6 ulp accuracy).

For example, though Maple can quite easily exactly invert rational matrices of moderately large size, such as Hilbert matrices, asking Maple to

compute the (well-conditioned) *eigenvalues* of the Hilbert matrix of, say, order 25 by

1) computing the characteristic polynomial exactly and then
2) attempting to find the roots of this polynomial by using a numerical method such as is implemented in the Maple command `fsolve`

is a *bad idea* and leads to unnecessary expense and large errors. The roots of this polynomial are ill-conditioned, as many are, and this leads to trouble. Maple *can* successfully use this approach to find the eigenvalues of the Hilbert matrix of order 15, using the very precise routine `realroot`, written by Bruce Char, to isolate all fifteen positive roots of the characteristic polynomial (which, incidentally, needs 125-digit arithmetic to represent the coefficients accurately). The routine `realroot`, which comes packaged with Maple, provides exact dyadic rational intervals of arbitrary tightness containing each real root of the input univariate polynomial. For this family of examples, however, the extremely large size of the polynomial coefficients for higher-order Hilbert matrices soon defeats Maple — by exhausting our patience. Arithmetic with large rational numbers not only takes storage space, it takes time. It is better not to introduce the ill-conditioning in the first place and instead work directly with the eigenvalue problem.

Another example is the evaluation of functions defined by expressions automatically produced by Maple. For example, consider the expressions for the roots of a quadratic generated by the quadratic formula:

```
>solve(x^2 + 2*b*x + c, x);

                    2     1/2           2     1/2
          - b + (b  - c)   , - b - (b  - c)

>r := unapply("[1],b,c);

                                  2     1/2
            r := (b,c) -> - b + (b  - c)
```

In this example, the evaluation of $r(b, c)$ is well-known (to numerical analysts) to be numerically unstable for $b/c \gg 1$: but Maple does *not* know that, and indeed neither would an inexperienced user. More realistically, the algebraic solutions to cubics and quartics are also known to be unstable for certain values of the coefficients, but this is even less widely understood. Indeed, the expressions Maple can generate as the solution to more complicated problems can be much too large for human analysis and can also be of very dubious numerical stability. Some numerical analysts of our acquaintance refer to such output as "wallpaper expressions — because wallpaper

is about all they're good for"[1]. It would be useful if Maple could warn the user when the result of evaluating the expression is *unreliable*.

Fixed-precision interval arithmetic does *not* do this — instead, it provides the complementary tool of telling the user that when the interval bounds are tight, the computation was *reliable*. Of course, sometimes the bounds are not tight, which may be due to an unstable calculation or perhaps due to pessimism on the part of interval arithmetic, which is caused by insufficient use of the correlation of different parts of the expression. This problem might be curable in two ways: add more intelligence to the interval expression evaluation, as is done in [5], or we could instead apply more brute force in the form of increasing the number of digits carried in the calculation from the beginning. Both approaches seem very feasible in a high-level computational environment such as Maple. Thus the construction of an arbitrary-precision interval arithmetic package in Maple seems of interest.

Other reasons for the construction of this package include possible applications to 'honest plotting' [7], to code development (to provide numerical analyses of proposed code for e.g. sequence acceleration or for evaluation of the Jacobian elliptic functions), and to education — we will see an example later, of numerical integration, that can profitably be used in either a first-year calculus class or in a numerical analysis class.

2 Summary of the proposed BIAS standard

George Corliss has a draft proposal [6] for a standard common core interval arithmetic subroutine library, which will help enable users to write portable and efficient programs using interval arithmetic. We implemented our package using this proposed standard for two reasons: one, we were novices in the field of interval arithmetic and were grateful for such a simple way to take advantage of the many years' experience of others, and, two, by adhering to a standard we could expect that others would be able to quickly use the package. At this writing the only users of this package are a small seminar group run by George Corliss at Marquette University. From the comments of that group, who implemented such routines as the solution of

[1]This expression was originally used by Morven Gentleman, to our knowledge. It graphically and memorably describes the very real problem of 'expression swell', which is to symbolic computation what exponential amplification of roundoff error is to floating-point computation.

linear systems by Gaussian elimination and by the preconditioned interval Gauss-Seidel algorithm of [8] using the package described here, it is clear that the extent of our adhering to the proposed standard made their work easier.

The BIAS specifications are presented in [6] in a language independent form along with Fortran, Ada, and C bindings. The language-independent specifications address these areas:

- Declare a variable to be of interval type
- Arithmetic Operations
- Elementary functions
- Utility functions
- Membership functions
- Commonly used constants
- Error handling
- Rudimentary I/O

We have implemented the type declarations, the arithmetic operations, the elementary functions, some of the utility functions and membership functions, and some of the rudimentary I/O. We have not strictly adhered to the standard, using our own names for some of the functions (strict adherence would be better, and in the next version of the package we will attempt to ensure true compliance).

Arithmetic operations (required)

The binary operations implemented are `&+`, `&-`, `&*`, `&/`, and `&inv` which all take interval or numeric inputs and return intervals on output. The interval $[-\infty, \infty]$ may be returned by `&inv` or `&/` if a division by zero occurs.

Elementary functions (optional)

The unary elementary functions implemented are `&sqr`, `&sqrt`, `&ln`, `&exp`, `&** = &^`, `&intpower`, `&sin`, `&cos`, `&tan`, `&arcsin`, `&arccos`, `&arctan`, `&sinh`, `&cosh`, and `&tanh`. These functions all compute inclusions of the

range of these functions for interval arguments, including degenerate interval arguments.

Utility functions (required)

The functions `construct`, `ru`, and `rd`, to construct an interval (with optional outward rounding), round up by one ulp, and round down by one ulp, are basic to the package. (The real name of, for example, `ru` is `Interval_round_up` but the short 'alias' is used throughout the package for convenience).

Utility functions (optional)

The functions `&midpoint`, `&width`, `&intersect`, `&union`, and `&is_in` are also present. George Corliss contributed another utility in this class, the routine `is_disjoint`. All of these routines perform the natural functions indicated by their names.

Remark

The names of the arithmetic operations *had* to be different from that of the proposed standard since Maple's built-in operations $+$, $-$, etc., cannot be overloaded and indeed suffer from some very undesirable automatic simplifications that cannot be turned off (for example, $[0,1]-[0,1]$ is automatically simplified to 0 before any user routine would have a chance to even look at it). So we were forced to use Maple's so-called 'inert' operators `&+`, `&-`, `&*`, etc. These have two drawbacks: one is that they are slightly ugly (and take more than twice as long to type because `&` is not often used!) and, more seriously, Maple has the precedence rules for these operators hard-wired in, *incorrectly*. This bug is a historical one — the Maple group implemented inert operators, then realized that `&*` could be used for matrix (non-commutative) multiplication (the ordinary `*` operator has hardwired commutativity built-in), and so the precedence for the inert operator `&*` was lowered from that of all the other inert operators to that of ordinary multiplication. This leaves us with a situation where the precedence of these operators is exactly the opposite of what you would expect: a `&+` b `&*` c is parsed as $(a+b)*c$. The only cure for this is to explicitly use parentheses to force correct evaluation. For convenience, we implemented a routine to

convert from the use of ordinary Maple operators to the use of the interval operators here, but it would be more convenient still not to have to use it. A request that this precedence bug be fixed has been forwarded to Maple.

3 Sample code and usage

We exhibit the Maple code for the exponential function as an example.

```
#   PROCEDURE &EXP
#  expinfinity is only called from &exp.
#  It deals with FAIL and +/- infinity.
#  Like most of the other subroutines &exp
#  takes floating point intervals
#  or numerics (which are converted into intervals).
expinfinity:=proc(x);
  if x=FAIL then FAIL
  elif x=infinity then infinity
  elif x=-infinity then 0
  else evalf(exp(x))
  fi;
end:
Interval_exp:=proc(x);
  if type(x,interval) then
    if x=[] then []
    elif x[1]=FAIL or x[2]=FAIL then [FAIL,FAIL]
    else [Interval_Round_Down(expinfinity(x[1])),
          Interval_Round_Up(expinfinity(x[2]))]
    fi;
  elif type(x,num_or_FAIL) then
    Interval_exp(construct(x))
  else
    # Want to return unevaluated here.
    'Interval_exp(x)'
  fi;
end:
 alias(`&exp`=Interval_exp):
```

This routine uses the built-in Maple exponential function to provide the basic results and uses monotonicity of the exponential function to allow exponentials of intervals. We also allow arithmetic with the symbols `FAIL` and `infinity` which are the Maple equivalents of NaN (Not a Number); though of course they indicate also that the result is not a symbolic answer either. If `&exp` is called with a symbolic argument, it is simply returned 'unevaluated', that is as a sequence of symbols which can be processed and perhaps evaluated later. For example, if `x` has no interval or numerical value, then the following sequence results:

```
>&exp(x);
                        &exp(x)
```

We have written a utility routine called `'convert/interval'` and another called `inapply` which together convert symbolic expressions to a form using the interval operators. The first utility recursively walks an expression, replacing the ordinary Maple operators with the appropriate interval ones. The second utility converts an expression into an operator (i.e. procedure) which evaluates its arguments using the interval operators. We give some examples of the usage of the interval arithmetic package in the following sample Maple session.

```
> load('intpak.mpl');
> f := sin(x)/(1+ cos(x))^2;
                                  sin(x)
                      f := -------------
                                        2
                           (1 + cos(x))

> f_interval := inapply(f,x);
     f_interval := x -> (&sin x) &*
                    inv((1 &+ (&cos x)) &intpower 2)

> Digits := 20;
                           Digits := 20

> f_interval( [0,0.5] );
                  [0, .13599504298972341743]

> # plotting shows the function is monotonic
> # and that above bound is tight.
> asympt(Ei(x),x);    # The exponential integral function.

        /        1      2      6      24         1  \
        |1/x + ---- + ---- + ---- + ---- + O(----)| exp(x)
        |        2      3      4      5         6  |
        \       x      x      x      x         x   /


# The following trick for removing the Landau O-symbol
# from a Maple expression is
# regrettably still occasionally necessary.

> subs(O=0,"):

> convert(",polynom);
            /        1      2      6     24 \
            |1/x + ---- + ---- + ---- + ----| exp(x)
            |        2      3      4      5 |
            \       x      x      x      x  /

> p := inapply(",x);
p := x -> (inv(x &intpower 1) &+ (inv(x &intpower 2)
          &+ ((2 &* inv(x &intpower 3))
          &+ ((6 &* inv(x &intpower 4))
          &+ (24 &* inv(x &intpower 5)))))) &* (&exp x)
```

```
> p([100.,200.]);
                                  42                                  85
     [.135084709489596021 06*10   , .729972379063694500 77*10   ]
>
```

4 Numerical integration: a didactic exercise

In the beautifully written article [9], professor Kahan exhibits several interesting integrals. He uses them to make several points, among which are that numerical integration is provably impossible, numerical integration is better than symbolic integration, and that the integration algorithm incorporated into the HP34C calculator (now inherited by the HP48 family) is a good one. Following a tradition exemplified by much of Kahan's own work, we will use one of his own examples to support exactly the opposite points.

We first examine his proof that numerical integration is impossible. Of course, he is correct, if point algorithms are used. His proof strongly uses the fact that point algorithms may not look at the program that evaluates the integrand: in a certain sense this disallows interval arithmetic, which in effect analyzes the integrand by replacing point arithmetic with interval arithmetic. But if we are permitted to 'cut the ground from under this proof' by modifying the interpreter or compiler so the program for the integrand uses interval arithmetic instead, then the conclusion becomes more doubtful. Certainly the rest of Kahan's proof no longer goes through, since it essentially consists of 'spying' on the numerical integrator by feeding a zero integrand to it and reporting back to the malicious user as to where the integrand was probed; then, one constructs a second function, zero precisely at those points but positive everywhere else, with a nonzero integral. This second, maliciously designed, function cannot be accurately evaluated with that quadrature method.

Of course, this does not apply to interval techniques: one cannot design a malicious function in that way as no intermediate information is overlooked. However, it may be that numerical integration is still impossible, due to some reason that we ourselves have not noticed. In particular, one might be able to defeat any interval techniques by using integrands with integrable singularities.

Second, Kahan uses the example

$$I(n,x) = \int_0^x \frac{dt}{1+t^n}$$

with $n = 64$ to show that numerical integration is superior to symbolic integration. We agree with Kahan that this example is "atypically modest out of consideration for the typesetter", because the symbolic answer has only 32 terms, and so the point we make below on the utility of symbolic techniques is not really general. However, we believe that point to be interesting and useful.

Kahan's numerical algorithm, implemented on the HP34C, makes short work of this problem for $n = 64$ and $x = 1$ and gives a much more satisfactory answer than the 32-term symbolic answer in terms of arctans and logs.

But what about larger n? The first such n we tried on the HP48SX was $n = 1024$. The calculator returns the answer $I(1024, 1) = 1$ in under a second, and claims this answer is correct to twelve decimal places.

Alas, that can't be right. We may show that $I(1024, 1) < 0.99952$ by computing the first three nonzero terms in the Taylor series for $1/(1+t^n) = 1 - t^n + t^{2n} - \cdots$ and integrating term by term (we will get tighter bounds later). So we may suspect that the algorithm of the HP34C (and hence the HP48SX) has fallen afoul of the impossibility proof mentioned earlier, and indeed this is the case. When we 'spy' on the integrator, we find that it never evaluates the function at any place near enough to the right-hand endpoint to see that the function is not identically 1. At least it returns the wrong answer quickly.

As an aside, the above series can be summed in Maple to find a very short symbolic answer, to get the answers below.

$$\begin{aligned} I(n,1) &= \frac{1}{2n}\left(\psi\Big(\frac{1}{2}+\frac{1}{2n}\Big) - \psi\Big(\frac{1}{2n}\Big)\right) \\ I(n,x) &= x\,F(1, 1/n; 1+1/n; -x^n) \end{aligned}$$

Maple can quickly and accurately evaluate these functions; for example, a simple call to `evalf` gives $I(1024, 1) = 0.99932388198340371$ accurate to 0.6 ulp (and this evaluation takes a fraction of a second). Thus, if we *extend our symbolic alphabet*, here including the hypergeometric function $F(a, b; c; x)$ and ψ, the logarithmic derivative of the Gamma function, we can get a very effective answer to this problem. After making this discovery, one anticlimactically finds this integral in Abramowitz & Stegun [1] in the

section on hypergeometric functions. So although of course we agree with Professor Kahan's point that numerical integration is more often useful than symbolic methods, it is also often useful to try hard to get a symbolic answer, as it may provide effective numerics in addition to insight.

Using interval arithmetic to evaluate right- and left-hand Riemann sums, to give lower and upper bounds on the integral of this monotonically decreasing function, we can get an answer of guaranteed accuracy. Using 10 panels, equally spaced in y, we find that

$$0.9494 < I(1024, 1) < 0.99945$$

and using 100 panels, that

$$0.99434122 < I(1024, 1) < 0.99934128$$

The code for evaluation of these Riemann sums is a straightforward application of the interval operators of the current package. Note, however, that we use panels equally-spaced in y and not x, for efficiency. We have thus taken advantage of knowledge of the graph of the integrand. Even so, the interval code to do this sum is very slow, taking over an hour on a 486-based IBM clone. One reason for the slowness is that the interval package is implemented in Maple code, which is *interpreted* and not compiled; Maple is an interpreted language because the usual tasks it is asked to do are 'one-off' calculations, rarely repeated, and in that context interpretation makes sense. Further, our package does a lot of type-checking, and was not written with efficiency in mind, since we knew that the end product would be slow in any case for the previous reason. This interval package was intended to be exploratory and not for 'production use'. The arbitrary-precision arithmetic used in this package is in fact the least expensive part of the computation, at least until the number of decimals asked for gets extremely large (perhaps, say, a few thousand digits).

One is tempted to use the midpoint rule and trapezoidal rule to get better answers for this particular problem. Unfortunately, there is a change in convexity of the integrand near $x = 1$ (detectable with interval arithmetic) and so the problem of getting bounds, rather than estimates, is complicated. In addition, note that higher-order techniques don't help on this integral as the derivatives are large, vitiating the expected efficiency gain from the higher order methods.

Finally, we emphasize that the points made by Professor Kahan are in fact valid: numerical integration *is* impossible (by point methods), the algorithm used by the HP34C (and now the HP48SX) *is* a good one (subject to the limitations of point methods), and the class of problems for which analytic techniques of integration are useful is really smaller than most first-year calculus teachers would have us believe. Indeed, his other examples in the paper really do support his conclusions, and we urge the reader of this present paper to examine [9]. However, the above example, used by Professor Kahan to support these points, actually supports the opposite conclusions — and makes a good case for interval techniques, which we are sure he would agree with.

5 Computation of π: a historic exercise

The oldest use of rounded interval arithmetic known to the authors is Archimedes' calculation of π. He calculated the lengths of the semiperimeters of inscribed and circumscribed polygons (from an idea of Antiphon) and used a formula for using known semiperimeters to calculate that of the semiperimeters of polygons with twice as many sides. Starting from six-sided polygons, he successively doubled up to polygons with 96 sides and achieved the first true bounds on the value of π. When using the formula, he rounded down (in rational arithmetic, finding rationals with smaller denominators which gave lower bounds) when dealing with the inscribed polygons and rounded up when dealing with the circumscribed polygons.

Archimedes' method was not improved upon in principle for 1800 years, and the culmination of the use of this method was by Ludolph van Ceulen who used it to calculate 34 digits of π using polygons with 2^{62} sides. See [2] for details and further references. Below we use Archimedes' method to calculate bounds for π just slightly better than Ludolph van Ceulen's. Of course we use numerically stable versions of Archimedes' doubling formulas, to prevent unnecessary ultimate growth in the width of the bounding intervals.

```
#
# Let p[n] be the semiperimeter of an n-gon INSCRIBED
# in a unit circle.
# Let P[n] be the semiperimeter of an n-gon CIRCUMSCRIBED
# about a unit circle.
# Then
# p[2*n] = 2*p[n]/sqrt(2*(1+sqrt(1-(p[n]/n)^2))) and
# P[2*n] = 2*p[n]/(1 + sqrt(1-(p[n]/n)^2))
#
> Digits := 40:  ktimes := 61: start := time():
> p := table():  P := table():
> p[6] := construct(3);
                                    p[6] := [3., 3.]

> P[6] := construct(evalf(2*sqrt(3)),rounded):
> for k to ktimes do
>   n := 3*2^k;
>   den := [1.,1.] &+ &sqrt( [1.,1.]
                &- &sqr( p[n] &/ construct(n) ) );
>   P[2*n] := ([2.,2.] &* p[n]) &/ den;
>   p[2*n] := ([2.,2.] &* p[n]) &/ &sqrt( ([2.,2.] &* den) );
> od:
> for n in [seq(3*2^k,k=1..ktimes)] do
>   print(p[n][1],` < Pi < `,P[n][2]);
> od;
    3.,  < Pi < , 3.464101615137754587054892683011744733887

                < ... several lines of output omitted ... >

     3.141592653589793238462643383279502883794,  < Pi < ,
                                                 ^^^^

     3.141592653589793238462643383279502884704
                                                 ^^^^
                                    <added for emphasis>

# Number of sides on the polygon:
> 3*2^ktimes;
                        6917529027641081856
# Error:
> P[][2] - p[][1];
                                    -36
                             .910*10

> total_time := time() - start;
                       total_time := 85.950  <seconds>
```

6 Conclusions

While there are two main irritants in this package, namely, the necessity of using the clumsy inert operators and the very slow speed resulting from running interpreted code, the package has been informative and useful. We have recommended to the Maple group that they fix the precedence bug with

the inert operators, and work on a compiler so that arbitrary numerical tasks might be more readily attempted in Maple.

References

[1] Abramowitz, M. and Stegun, I. A. *Handbook of mathematical functions.* Dover, 1965.

[2] Borwein, J. M. and Borwein, P. B. *Ramanujan and Pi.* Scientific American **256** (2) (February 1988).

[3] Char, B. W., Geddes, K. O., Gonnet, G. H., Leong, B. L., Monagan, M. B., and Watt, S. M. *The Maple V language reference manual.* Springer-Verlag, New York, 1991.

[4] Corless, R. M., Gonnet, G. H., Hare, D. E. G., and Jeffrey, D. J. *On Lambert's W function.* Technical Report CS-93-03, University of Waterloo, 1993.

[5] Corliss, G. F. and Rall, L. B. *Computing the range of derivatives.* In: Kaucher, E., Markov, S. M., and Mayer, G. (eds.) "Computer Arithmetic, Scientific Computation and Mathematical Modelling", IMACS Annals on Computing and Applied Mathematics, Baltzer, Basel, 1991, pp. 195–215.

[6] Corliss, G. F. *Proposal for a basic interval arithmetic subroutines library (BIAS).* Preprint.

[7] Fateman, R. *Honest plotting, global extrema, and interval arithmetic.* Proceedings Int'l Symp. on Symbolic and Algebraic Computing, Berkeley, 1992, pp. 216–223.

[8] Hansen, E. *Global optimization using interval analysis.* Marcel Dekker, New York, 1992.

[9] Kahan, W. M. *Handheld calculator evaluates integrals.* Hewlett-Packard Journal (August 1980), pp. 23–32.

[10] Moore, R. E. *Methods and applications of interval analysis.* SIAM, Philadelphia, 1979.

Department of Applied Mathematics
University of Western Ontario
London, Ontario,
Canada N6A 5B7