To be published by the International Journal of Interval Computations in the Proceedings of the International Workshop on Applications of Interval Computations, El Paso, Texas, 1995.

InC++ Library Family for Interval Computations

Eero Hyvönen
(eero.hyvonen@vtt.fi)

Stefano De Pascale
(stefano.depascale@vtt.fi)

Technical Research Centre of Finland (VTT)
Information Technology, Information Systems
P.O.Box 1201, 02044 VTT
(Tekniikantie 4B, Espoo)
FINLAND

Abstract

This paper presents a series of C++ libraries for interval function evaluation and constraint satisfaction. Classical interval arithmetic (IA) (Moore, 1966) is extended by open ended intervals, the notion of infinity and by "complement" and discontinuous intervals. Both algebraic and numerical IA techniques are combined for obtaining the actual range of interval functions efficiently and for determining better than local solutions for interval constraint satisfaction problems. Our practical goal is a set of portable C++ libraries that can be used in applications without deep understanding of interval analysis.

1. InC++ libraries

This paper presents main features of our InC++ family of C++ libraries for applications of interval arithmetic (IA). The libraries with their purposes are listed below.

Name	Purpose
LIA InC++	Extended (local) interval arithmetic
GIA InC++	Evaluate the actual range of an interval function
CIF InC++	Evaluate a set of mutually dependent interval functions
ICE InC++	Solve interval constraint satisfaction problems

LIA, GIA and ICE are new enhanced and renamed versions of the libraries in (Hyvönen et al., 1993, 1994).

2. LIA: Local Interval Arithmetic

Arithmetic expressions are evaluated in classical IA by generalizing arithmetic primitive operators for intervals. We call this evaluation technique *local*, because it does not take into account of the global dependencies of the different variable instances in the expression. Several local IA packages for standard programming languages have been implemented, such as Triplex (Cole, Morrison, 1982), Pascal-SC (Bohlender et al., 1987), Fortran-SC (Kulisch, 1987), IP++ (Alander, 1987), PBasic (Aberth, 1988), VPI (Ely, 1990). LIA InC++ (Hyvönen, De Pascale, 1994a) is a follower in this tradition with following new features:

Extensions to IA

1. *Open ended intervals*

Traditional IA packages consider only closed intervals. This is probably due to following problems that make the meaning of interval openness somewhat unclear from the mathematical view point.

(1) Rounding errors in the machine arithmetic used typically corrupt the precise interval limits (usually only integer limits can be computed accurately). By outward rounding techniques only a closed outer bound of the actual result can be computed in the general case.

(2) Multiple variable instances in the interval expression usually accumulate extra width in the result and information on limit openness becomes less useful.

However, from the practical viewpoint, information on whether a limit is closed or open is often important because the limits are typically the most important points of an interval. LIA InC++ supports open and half open interval types (x,y), (x,y], and [x,y). Problem 1 above is solved (at practical level) by representing the result interval at a precision level looser than the one used for computations. For example, it is reasonable to say

(1,2.1]*[0,3)=[0,6.3)

even if the upper limit could not be represented precisely in the machine. Problem 2 is simply accepted as a fact of life: usefulness of open interval ends lessens if the interval expression has multiple variable instances with non-zero width. This is of course the case with local closed IA as well.

2. *The notion of infinity*

The notion of infinity has been implemented: Overflowed intervals with ±∞ as the min or max are treated properly by extending IA rules for such cases. For example:

(0,∞)*(-∞,-1)=(-∞,0)

3. *Complement intervals*

LIA implements the idea of "complement" intervals and generalizes IA for them. A complement interval means "any value *not* within a given range" and is represented by switched interval delimiters. For example, half open complement interval types are defined below:

]a,b(= {x|x≤a or x>b}, e.g.,]2,3(= (-∞,2] ∪ (3,∞)
)a,b[= {x|x<a or x≥b}

By using complements, IA rules can be extended for some discontinuous situations not defined in classical IA, such as division with an interval containing zero:

6/[-1,3]=(-∞,-6] ∪ [2,∞) =]-6,2[
[3,6]/[-1,3]=]-3,1[
[-2,6]/[-1,3]= (-∞,∞)

4. *Discontinuous intervals*

LIA library also extends IA for discontinuous intervals (d-intervals) (Hyvönen, 1992), i.e., for ranges of discontinuous values. According to (Hansen, 1992) the idea of discontinuous or multi-intervals can be traced back to Hanson (1968) and Kahan (1968). However, in spite of its intuitive nature the idea seems to be seldom used and LIA is to our knowledge the only implementation around.

Discontinuous intervals in LIA are discontinuous ranges of values and are represented as sets of intervals and complement intervals. For example:

{(1,2),]-4,7(} = {(-∞,-4], (1,2), (7,∞)}

Numbers, intervals and complements are useful special cases of the general discontinuous interval.

Programming and application

The library is implemented as an extension to C++ arithmetic and the standard ANSI C mathematical library. It overloads ordinary C++ arithmetic operators (+, -, ...), functions (pow, exp, sin,...) and other operators (=, ==, ...) for a generalized interval type DInterval. A programming example is given below:

```
#include "lia.h" // LIA InC++ header file
DInterval x ="{(1,2), [4, 4.2)}";
DInterval y(1.2, 1.9);    // y=[1.2, 1.9]
DInterval z=exp(x)+3*pow(y,2)-5.3;
printf("%s",z.bFormat(3)); // In 3 digits
```

This code would print the d-interval below in three digit precision:

{(1.74,12.9),[53.6,72.2)}

The traditional application of local IA implementations has been management of rounding errors due to finite machine arithmetic (Kulisch, Miranker, 1986). For example, the following C++ code

```
double p=10864.0, q=18817.0;
printf("%g",9*pow(p,4)-pow(q,4)+2*pow(q,2));
```

prints value 2 (in our HP workstation) although the correct value is 1 (!) (Rump, 1983). By using LIA InC++ and its outward rounding arithmetic the resultant interval is a reliable outer bound [-46,50] of the actual result. By linking LIA InC++ with a higher precision (than the standard double precision) math library stricter bounds for 1 can be obtained.

Rounding error management is often important, but the principal application area intended for LIA is rather to support versatile interval computations in practical applications dealing with imprecise data and in our more advanced InC++ libraries. LIA supports a more general notion of interval than other IA implementations we know of. In order to make extended IA available to a wider audience than only programmers, the library has been integrated as an Add-In library for Microsoft Excel spreadsheet program (De Pascale, Hyvönen, 1994).

3. GIA:Global Interval Arithmetic

The main limitation of classical IA and LIA InC++ is that a function value may have excess width if the function expression contains multiple variable instances (like in X*Y+X*Z). In order to determine the actual function range in such cases, GIA InC++ library (Hyvönen, De Pascale, 1994b) should be used instead.

Main features

GIA library has following innovative features:

1. *Algebraic optimization*

An algebraic optimization module is used for deriving an easy to evaluate algebraic representation for the function. After this a generalized numerical branch-and-bound algorithm in the tradition of (Skelboe, 1974, Asaithambi et al., 1982; Ratschek, Rokne, 1984, 1988; Moore, 1992; Hansen, 1992) is applied for determining the range of function values. The algebraic module simplifies function expressions and eliminates multiple variable instances by computer algebraic techniques (Davenport, 1992) (e.g., by canonizing polynomials and by using simplification rules) and generates quickly converging extensional forms (Ratschek, Rokne, 1984) for the function (e.g., Taylor form, interval Newton form, monotonicity test form, ...).

2. *Extended interval arithmetic*

Arguments of the function may be any interval types of LIA InC++ (see previous chapter). The result is an outer and inner bound of the actual value range. Sometimes, but not always, also subranges of the result are eliminated and the outer bound is a discontinuous interval.

3. *Recursive interval functions*

The algorithms are enhanced for evaluating recursive interval functions $x_i=f(x_1,...,x_i,...,x_n)$. Here GIA bounds the point set

$$X_i=\{x_i \mid x'_i=f(x_1,\ldots,x_i,\ldots,x_n),\ x'_i\approx x_i,\ x_j\in X_j,\ j=1,\ldots,i-1,i+1,\ldots,n\}$$

where situation $x'_i\approx f(x_1,\ldots,x_i,\ldots,x_n)$ is considered a *precision fixed point*, if $x'_i\approx x_i$ by using the user given absolute or relative precision criteria.

4. *Computational parameters*

The computational complexity of global interval function evaluation is unpredictable. As a result, versatile control facilities are necessary in practical applications. Major parameters settable in GIA include:

Precision. GIA bounds the actual range of values within a user-given relative or absolute precision criteria. The system also evaluates an inner approximation of the result. Outer and inner bounds together tell how precise the evaluation actually was.

Time limit. Computation can be terminated forcibly after a user given time limit, in which case the current outer and inner bounds are returned as the result. This possibility is quite useful in trading precision for time in applications such as real time systems.

Split limit. The numerical branch-and-bound algorithm can be controlled also by bounding the size of the search space.

Algebraic mode. Several algebraic optimization modes are available. This makes it possible to balance time and space requirements between algebraic optimization (definition time) and numerical evaluation (evaluation time). More algebraic optimization usually shortens execution time and vice versa.

Programming

As an example, evaluation of the Goldstein-Price function (Goldstein, Price, 1971) is presented below:

```
// Construct a function object fn
#include gia.h // Library header file
IFunction fn("z=(1+(x+y+z)^2*(19-14*x"
  "+3*x^2-14*y+6*x*y+3*y^2))*(30+(2*x-"
  "3*y)^2*(18-32*x+12*x^2+48*y-36*x*y"
  "+27*y^2))");
fn.SetValue("x",-2,2); // Set x=[-2,2]
fn.SetValue("y",-2,2); // Set y=[-2,2]
// Set desired precisions and time limit
fn.SetRelativePrecision(1e-6);
fn.SetAbsolutePrecision(1e-6);
fn.SetTimeLimit(-1); // = no time limit
fn.Evaluate();
DInterval z = fn.GetValue(); // Read result
printf("The value is: "); z.Display();
```

The code prints value [2.999.., 1015691.2..]. If the function is evaluated by ordinary local interval arithmetic, the result would be wider: [-87881320, 147125080].

4. CIF: Cascaded Interval Functions

CIF (Hyvönen, De Pascale, 1994c) generalizes GIA by dealing not only with one function but with a set of them:

$$S=\{Y_i=F_i(...)\}$$

Functions F_i may be recursive or mutually recursive (circular). There is only one function for each value variable Y_i. The task of CIF is *cascaded evaluation* of S, i.e., to evaluate the mutually dependent variable values Y_i globally in S. Cascaded function evaluation is important especially because its non-recursive form with exact input values is the basic computational paradigm used in current spreadsheet programs such as MS Excel, Lotus 1-2-3, Quattro Pro. The idea of CIF is to generalize this paradigm into the interval case.

When using exact value arithmetic, cascaded evaluation can be performed easily by sorting the functions properly before evaluation. When using IA this is not enough even if the functions were evaluated globally (by GIA) due to the local-global dependency problem at function set level. For example, when evaluating the functions

y=a+b-50*x, a=x^4+24, b=-10*x^3+35*x^2 (3.1)
x=[0,4]

we get value y=[-816,840] although the actual range is y=[-1,24].

Techniques and algorithms for solving the dependency problem in cascaded interval function evaluation were developed in (Hyvönen, 1994). The idea is to use a combination of algebraic and numerical interval techniques as follows:

1. *Algebraic globalization*. First, the user-given (local) functions are expanded (by chaining them) into minimally large *global functions* in order to collect dependent variable instances into single expressions.

2. *Algebraic optimization*. Efficient extensional forms are then derived for the global functions (like in GIA above).

3. *Numerical evaluation*. Finally,numerical interval algorithms are applied to the global functions (like in GIA) by a value propagation algorithm.

As an illustration of using CIF library, example (3.1) is programmed below

```
#include cif.h // Library header file
Cif s(2, "y=a+b-50*x", "a=x^4+24");
s.InsertFunction("b=-10*x^3+35*x^2");
s.SetValue("x","[0,4]");
s.GetValue("y")->Display();
```

After its construction, new functions can be inserted into the function set and old ones be modified or removed by special member functions. The Cif-object automatically maintains internal function definitions correct.

5. ICE: Interval Constraint Equations

Our most general library ICE (Hyvönen, De Pascale, 1994d) deals with *interval constraint satisfaction problems* (ICSP) (Davis, 1987; Hyvönen, 1992). An ICSP is a set of equations/inequations E with initial interval values S for its variables. For example:

a+b*c+3=0, a*c=3, sin(a)>=exp(c) (5.1)
a=[0,2π], b=(0,100), c=[-10,0.45]

The task is—roughly spoken—to constrain (filter) the intervals S as much as possible but without removing such values that could be extended to (exact) solutions of E within S. Numerical constraint satisfaction problems are encountered in one or another form in application fields such as scheduling and planning (Stefik, 1981; Fox, 1983), qualitative reasoning (Kuipers, Berleant, 1988), designing (Elias, 1986; Steinberg, 1987; Murtagh, Shimura, 1990), constraint programming (Cohen, 1990), spreadsheets (Konopasek, Jayaraman, 1984; Hyvönen, 1991, 1994a), mathematical problem solving (Gosling, 1983; Babitchev, 1993).

Extensions for solving ICSPs have been designed and implemented especially in Prolog (Cleary, 1987; Older, Vellino, 1993) and constraint programming languages such as CHIP (Lee, van Emden, 1991a) and CLP(R) (Lee, van Emden, 1991b). In contrast to traditional IA implementations, interval constraint systems treat arithmetical expressions declaratively as relational constraints, not as ordinary "directed" interval functions. During computation, any value used in a constraint can be modified depending dynamically on the situation.

ICE can be seen as an interval constraint extension for C++ programming (Hyvönen et al., 1994) based on tolerance propagation techniques (Hyvönen, 1992). The library is in contrast with the (constraint) logic programming systems above in two major ways:

1. ICE does not decompose constraints into primitive functional constraints. By this way, individual constraints can be solved globally. Systems using decomposition determine only local solutions with more excess width in the result intervals.
2. From the practical viewpoint, a major problem of constraint logic programming languages is that — at least for the time being — they do not integrate well with main stream computing. ICE InC++ is implemented as portable C++ libraries to overcome problems of integration, portability, and efficiency.

Programming with ICE is performed in a similar way as with CIF above: An Ice object is constructed, equational contraints are inserted (or removed), initial variable values and computational parametrers are set, the system is evaluated, and finally changed variable values are read as the result.

As an application, ICE library has been integrated with MS Excel (Hyvönen, De Pascale, 1994e) in order to demonstrate the possibilities of the next generation interval constraint spreadsheet concept (Hyvönen, 1991,1994a).

6. Conclusions

This paper overviewed a series of new C++ libraries for interval computations. The libraries generalize and are based on each other in the order of presentation. The main technical contributions of the work are the extended interval arithmetic used in the libraries, and the combined application of interval analysis, computer algebra, and tolerance propagation techniques for determining global interval values in an efficient way.

A major problem one has to live with when dealing with (global) IA is that the complexity of numerical interval computations is to a large extent unpredictable: It heavily depends on the functions evaluated and initial values used. Our solution to difficult situations is—in addition to trying to make the implementation as efficient as possible—the use of versatile computational parameters (precision level, time limit, algebraic mode, etc.) and the possibility of getting inner and outer bounds for the result of global interval function evaluation.

Acknowledgements

This work has been supported financially by Technology Development Centre of Finland (TEKES). We thank Aarno Lehtola and Eero Peltola for discussions and support to our research. Thanks are also due to Trema Inc. and ViSolutions Inc. for co-operation.

References

O. Aberth, *Precise Numerical Analysis*, Wm. C. Brown, Dubuque, 1988.

J. Alander, On interval arithmetic range approximation methods of polynomials and rational functions, *Computers & Graphics* **9** (4) (1985) 365-372.

J. Alander, Programmer's Manual of Interval Package IP++, Report TKO B62 (Helsinki University of Technology, Faculty of Information Technology, Espoo, Finland, 1987).

N. Asaithambi, S. Zuhe, R. Moore, On computing the range of values, *Computing* **28** (1982) 225-237.

A. Babichev, O. Kadyrowa, T Kashevarove, A. Semenov, Unicalc — An intelligent solver for problems with imprecise and subdefinite data. International Conference on Numerical Analysis with Automatic Result Verification, Lafayette, USA, March 1993.

G. Bohlender, C. Ulrich, J. von Gudenberg, J. Rall, *Pascal-SC, a Computer Language for Scientific Computation* (Academic Press, New York, NY, 1987).

J.C. Cleary, Logical arithmetic, *Future Computing Systems* **2** (2) (1987) 125-149.

J. Cohen, Constraint logic programming languages, *Comm. ACM* **33** (7) (1990).

A. Cole, R. Morrison, Triplex: A system for interval arithmetic, *Software Practice and Experience* **12** (1982) 341-350.

J.H. Davenport, Y. Siret, E. Tournier, *Computer Algebra - Systems and Algorithms for Algebraic Computation* (Academic Press, New York, NY, 1993).

E. Davis, Constraint propagation with interval labels, *Artificial Intelligence* **32** (1987) 99-118.

S. De Pascale, E. Hyvönen, Extended Interval Arithmetic for Microsoft Excel (VTT, Technical Research Centre of Finland, Information Technology, Espoo, Finland, 1994).

J. Ely, *Prospects for Using Variable Precision Interval Software in C++ for Solving some Contemporary Scientific Problems*, dissertation, (The Ohio State University, Ohio, 1990).

M. Fox, *Constraint-Directed Search: A Case-Study of Job-Shop Scheduling*, dissertation, (Carnegie-Mellon University, Department of Computer Science, Pittsburgh, Pennsylvania, 1983).

A. Goldstein, J. Price, On descent from local minima. *Mathematics of computation* **25** (1971) 569-574.

J. Gosling, *Algebraic Constraints*, dissertation, (Carnegie-Mellon University, Department of Computer Science, Pittsburgh, Pennsylvania, 1983).

E. Hansen, *Global optimization using interval analysis* (Marcel Dekker, New York, 1992).

E. Hyvönen, Interval constraint spreadsheets for financial planning, *Proceedings of the 1st International Conference on Artificial Intelligence Applications on Wall Street* (IEEE Press, New York, 1991).

E. Hyvönen, Constraint reasoning based on interval arithmetic — The tolerance propagation approach, *Artificial Intelligence* **58** (1992) 71-112.

E. Hyvönen, Spreadsheets based on interval constraint satisfaction, *Artificial Intelligence in Engineering Design, Analysis and Manufactuiring* **8** (1994a) 27-35.

E. Hyvönen, Evaluation of Cascaded Interval Function Constraints. (VTT, Technical Research Centre of Finland, Information Technology, Espoo, Finland, 1994b).

E. Hyvönen, E., S. De Pascale, LIA InC++: A local interval arithmetic library, Research Report, (VTT, Technical Research Centre of Finland, Information Technology, Espoo, Finland, 1994a).

E. Hyvönen, E., S. De Pascale, GIA InC++: A global interval arithmetic library, Research Report, (VTT, Technical Research Centre of Finland, Information Technology, Espoo, Finland, 1994b).

E. Hyvönen, E., S. De Pascale, CIF InC++: A library for cascaded interval function evaluation, Research Report, (VTT, Technical Research Centre of Finland, Information Technology, Espoo, Finland, 1994c).

E. Hyvönen, E., S. De Pascale, ICE InC++: A library for interval constraint equations, Research Report, (VTT, Technical Research Centre of Finland, Information Technology, Espoo, Finland, 1994d).

E. Hyvönen, E., S. De Pascale, Interval Constraint Excel, *Proceedings of STeP-94* (Finnish AI Society, Espoo, Finland, 1994e).

E. Hyvönen, S. De Pascale, A. Lehtola, Interval Constraint Satisfaction Tool INC++, Proceedings of 5th International Conference on Tools with Artificial Intelligence (IEEE Press, New York, 1993).

E. Hyvönen, S. De Pascale, A. Lehtola, Interval constraint programming in C++. In B. Mayoh, E. Tyugu, J. Penham (Eds.), *Constraint Programming,* NATO Advanced Science Institute, Series F (Springer-Verlag, Berlin, 1994, in press) .

M. Konopasek, S. Jayaraman, *The TK!Solver Book* (McGraw-Hill, Berkeley, CA, 1984).

B. Kuipers, D. Berleant, Using incomplete quantitative knowledge in qualitative reasoning, *Proceedings of AAAI-88* (Morgan Kaufmann Publishers, Los Altos, California, 1988) 324 - 329.

U. Kulisch, FORTRAN-SC, a study of a FORTRAN extension for engineering/scientific computation with access to ACRITH, IBM, 1987.

U. Kulisch, W. Miranker, The arithmetic of the digital computer: A new approach. *SIAM Review* **28** (1, March), 1986.

J. Lee, M. vanEmden, Numerical computation can be deduction in CHIP, working paper (Deparment of Computer Science, University of Victoria, Canada, 1991a).

J. Lee, M. vanEmden, Adapting CLP(R) to Floating Point Arithmetic, working paper (Deparment of Computer Science, University of Victoria, Canada, 1991b).

R. Moore, *Interval Arithmetic* (Prentice-Hall, Englewood Cliffs, NJ, 1966).

R. Moore, *Methods and Applications of Interval Analysis*, SIAM Studies in Applied Mathematics (SIAM, Philadelphia, 1979).

R. Moore, E. Hansen, A. Leclerc, Rigorous methods for global optimization, In: *Recent Advances in Global Optimization* (Princeton University Press, Princeton, USA, 1992) 321-342.

N. Murtagh, M. Shimura, Parametric engineering using constraint-based reasoning, *Proceedings of AAAI-90* (Morgan Kaufmann Publishers, Los Altos, California, 1990) 505-510.

W. Older, A. Vellino, Constraint arithmetic on real intervals, in: F. Benhamou and A. Colmerauer, eds., *Constraint Logic Programming, Collected Research* (MIT Press, Cambridge, MA, 1993).

H. Ratscheck, J. Rokne, *New Computer Methods for Global Optimization* (Ellis Horwood, Chichister, England, 1984).

H. Ratscheck, J. Rokne, *Computer Methods for the Range of Functions* (Ellis Horwood, Chichister, England, 1988).

S. Rump, How reliable are results of computers? *Jahrbuch Überblicke Mathematik 1983*, 163-168, translation by Bibliographisches Institut AG (1984).

S. Skelboe, Computation of rational interval functions, *BIT* **14** (1) (1974) 87-95.

M. Stefik, Planning with constraints (MOLGEN: Part 1). *Artificial Intelligence* **16** (1981) 111 - 140.

L. Steinberg, Design as refinement plus constraint propagation. The VEXED experience. *Proceedings of AAAI-87* (Morgan Kaufmann Publishers, Los Altos, California, 1987) 830-835.