Lecture Notes in Computer Science 2991

Edited by G. Goos, J. Hartmanis, and J. van Leeuwen

Springer
*Berlin*
*Heidelberg*
*New York*
*Hong Kong*
*London*
*Milan*
*Paris*
*Tokyo*

René Alt  Andreas Frommer
R. Baker Kearfott  Wolfram Luther (Eds.)

# Numerical Software with Result Verification

International Dagstuhl Seminar
Dagstuhl Castle, Germany, January 19-24, 2003
Revised Papers

Springer

Volume Editors

René Alt
Université Pierre et Marie Curie, Laboratoire LIP6
4 Place Jussieu, 75252 Paris Cedex 05, France
E-mail: Rene.Alt@lip6.fr

Andreas Frommer
Bergische Universität Wuppertal, Fachbereich C, Mathematik und Naturwissenschaften
Gauß-Straße 20, 42097 Wuppertal, Germany
E-mail: frommer@math.uni-wuppertal.de

R. Baker Kearfott
University of Louisiana at Lafayette, Department of Mathematics
Box 4-1010, Lafayette, LA 70504, USA
E-mail: rbk@louisiana.edu

Wolfram Luther
Universität Duisburg-Essen, Institut für Informatik und Interaktive Systeme
Lotharstraße 65, 47048 Duisburg, Germany
E-mail: luther@informatik.uni-duisburg.de

Cataloging-in-Publication Data applied for

A catalog record for this book is available from the Library of Congress.

Bibliographic information published by Die Deutsche Bibliothek
Die Deutsche Bibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data is available in the Internet at <http://dnb.ddb.de>.

CR Subject Classification (1998): G.1, G.4, D.2, F.2.1, D.3

ISSN 0302-9743
ISBN 3-540-21260-4 Springer-Verlag Berlin Heidelberg New York



Typesetting: Camera-ready by author, data conversion by PTP-Berlin, Protago-TeX-Production GmbH
Printed on acid-free paper SPIN: 10993262 06/3142 5 4 3 2 1 0

# Preface

Reliable computing techniques are essential if the validity of the output of a numerical algorithm is to be guaranteed to be correct. Our society relies more and more on computer systems. Usually, our systems appear to work successfully, but there are sometimes serious, and often minor, errors. Validated computing is one essential technology to achieve increased software reliability. Formal rigor in the definition of data types, the computer arithmetic, in algorithm design, and in program execution allows us to guarantee that the stated problem has (or does not have) a solution in an enclosing interval we compute. If the enclosure is narrow, we are certain that the result can be used. Otherwise, we have a clear warning that the uncertainty of input values might be large and the algorithm and the model have to be improved. The use of interval data types and algorithms with controlled rounding and result verification capture uncertainty in modeling and problem formulation, in model parameter estimation, in algorithm truncation, in operation round-off, and in model interpretation.

The techniques of validated computing have proven their merits in many scientific and engineering applications. They are based on solid and interesting theoretical studies in mathematics and computer science. Contributions from fields including real, complex and functional analysis, semigroups, probability, statistics, fuzzy interval analysis, fuzzy logic, automatic differentiation, computer hardware, operating systems, compiler construction, programming languages, object-oriented modeling, parallel processing, and software engineering are all essential.

This book, which contains the proceedings of the Dagstuhl Seminar 03041 'Numerical Software with Result Verification' held from January 19 to 24, 2003, puts particular emphasis on the most recent developments in the area of validated computing in the important fields of software support and in applications.

We have arranged the contributions in five parts. The first part deals with languages supporting interval computations. The paper by Wolff von Gudenberg studies different object-oriented languages with respect to their abilities and possibilities to efficiently support interval computations. The contribution by Hofschuster and Krämer gives an overview of the C-XSC project, a C++ class library supporting intervals, the precise scalar product, standard functions with intervals, and various class abstractions useful for scientific computation.

The second part is devoted to software systems and tools. In a joint paper, Kearfott, Neher, Oishi and Rico present and compare four such systems: GlobSol, a Fortran-based library for the verified solution of nonlinear algebraic systems of equations and global optimization; ACETAF, an interactive tool for the verified computation of Taylor coefficients; Slab, a complete Matlab-style high-performance interval linear algebra package; and (Fixed) CADNA, a tool for assessing the accuracy and stability of algorithms for embedded systems relying on a fixed-point arithmetic. Whereas the first three software systems

use (machine) interval arithmetic, the latter is based on the CESTAC method and its stochastic arithmetic. Going beyond double precision in machine interval arithmetic is the topic of the paper by Grimmer, Petras and Revol. They describe `intPackX`, a Maple module which, among others, provides correctly rounded multiprecision evaluation of standard functions, and the two C/C++ based libraries GMP-XSC and MPFI. The authors include several examples where multiple precision interval arithmetic is of primary importance, for example to show the existence of Kronrod-Patterson rules for numerical integration or in the numerical solution of ODEs in Asian options pricing. The last paper in this part is by Corliss and Yu who report on their approach and their strategy and experience when testing a preliminary version of an interval software package for its correctness.

As software supporting interval and validated computation becomes more and more popular, we witness an increasing number of new modeling techniques using intervals. The third part of this volume contains five papers on these topics. Kieffer and Walter consider parameter and state estimation in dynamical systems involving uncertain quantities. For cooperative models, they use interval-based set inversion techniques to obtain tight bounds on the parameters and states under the given uncertainties. In an additional paper, together with Braems and Jaulin, they propose a new, interval computation-based technique as an alternative to computer algebra when testing models for identifiability. Auer, Kecskeméthy, Tändl and Traczinski show that interval analysis provides new opportunities to model multibody systems and they present an advanced software system MOBILE that includes such interval techniques. Bühler, Dyllong and Luther discuss reliable techniques in computational geometry. They focus on distance and intersection computations, an area where slightly wrong floating-point results may produce a completely wrong view of the geometry. The last paper by Alefeld and Mayer deals with the more fundamental issue of how interval arithmetic iterations behave when applied to solve linear systems with a singular coefficient matrix.

Part four considers various applications of validation techniques in science and engineering. It starts with a contribution by Beelitz, Bischof, Lang and Schulte Althoff on methods that guarantee the absence of singularities in certain models for the analysis and design of chemical processes. This is of primary importance, since otherwise multiple steady states may result in spontaneous fluctuations which may even damage the chemical reactor. Fausten and Haßlinger consider workload distributions of service systems in telecommunications under quality-of-service aspects. They develop a method to determine workload distributions involving a verification step based on interval arithmetic. Three important problems in geodesy are dealt with in the paper by Borovac and Heindl, who present verified methods for the direct and the inverse problem of geodetic surveying and the three-dimensional resection problem. Among others, enclosure methods for ODEs turn out to be very useful here. Schichl describes the COCONUT project, a large, European, modular software project for constrained global optimization. The paper explains the architecture of this software system,

which uses the FILIB++ library for its components based on interval arithmetic. Finally, the paper by Oussena, Henni and Alt describes an application from medical imaging in which verified computations would be of great help.

The last part is devoted to alternative approaches to the verification of numerical computations. The contribution by Lester shows how one can use the formal specification checker PVS to validate standard functions like arctan and some exact arithmetic algorithms. Granvilliers, Kreinovich and Müller present three alternative or complementary approaches to interval arithmetic in cases where uncertainty goes beyond having bounds on input data: interval consistency techniques, techniques using probabilistic information and techniques for processing exact real numbers. This part closes with the paper by Putot, Goubault and Martel, who propose the use of static code analysis to study the propagation of round-off. They also present a prototype implementation of their approach.

We would like to thank all authors for providing us with their excellent contributions and for their willingness to join in groups to present a coherent description of related research and software. We are also grateful to Springer-Verlag for the fruitful cooperation when preparing this volume and, last but not least, to the referees listed below.

January 2004

René Alt
Andreas Frommer
R. Baker Kearfott
Wolfram Luther

## Referees

R. Alt
G. Alefeld
J.-M. Chesneaux
G. Corliss
A. Csallner
T. Csendes
A. Frommer
J. Garloff
L. Granvilliers
G. Heindl
P. Hertling
C. Jansson
R.B. Kearfott
M. Kieffer
W. Krämer
V. Kreinovich
B. Lang
W. Luther
R. Martin
G. Mayer
N. Müller
N. Nedialkov
M. Neher
W. Otten
K. Petras
G. Plonka-Hoch
M. Plum
E. Reinhardt
N. Revol
S. Rump
L. Salinas
H. Traczinski
E. Walter
J. Wolff v. Gudenberg

# Table of Contents

## Languages

OOP and Interval Arithmetic – Language Support and Libraries . . . . . 1
*Jürgen Wolff von Gudenberg*

C-XSC 2.0: A C++ Library for Extended Scientific Computing . . . . . 15
*Werner Hofschuster, Walter Krämer*

## Software Systems and Tools

Libraries, Tools, and Interactive Systems for Verified Computations: Four Case Studies . . . . . 36
*R. Baker Kearfott, Markus Neher, Shin'ichi Oishi, Fabien Rico*

Multiple Precision Interval Packages: Comparing Different Approaches . . . . . 64
*Markus Grimmer, Knut Petras, Nathalie Revol*

Interval Testing Strategies Applied to COSY's Interval and Taylor Model Arithmetic . . . . . 91
*George F. Corliss, Jun Yu*

## New Verification Techniques Based on Interval Arithmetic

Nonlinear Parameter and State Estimation for Cooperative Systems in a Bounded-Error Context . . . . . 107
*Michel Kieffer, Eric Walter*

Guaranteed Numerical Computation as an Alternative to Computer Algebra for Testing Models for Identifiability . . . . . 124
*Eric Walter, Isabelle Braems, Luc Jaulin, Michel Kieffer*

Interval Algorithms in Modeling of Multibody Systems . . . . . 132
*Ekaterina Auer, Andrés Kecskeméthy, Martin Tändl, Holger Traczinski*

Reliable Distance and Intersection Computation Using Finite Precision Geometry . . . . . 160
*Katja Bühler, Eva Dyllong, Wolfram Luther*

On Singular Interval Systems . . . . . 191
*Götz Alefeld, Günter Mayer*

## Applications in Science and Engineering

Result-Verifying Solution of Nonlinear Systems in the Analysis of Chemical Processes ..... 198
*Thomas Beelitz, Christian Bischof, Bruno Lang, Klaus Schulte Althoff*

Verified Numerical Analysis of the Performance of Switching Systems in Telecommunication ..... 206
*Daniela Fausten, Gerhard Haßlinger*

Result Verification for Computational Problems in Geodesy ..... 226
*Stefan Borovac, Gerhard Heindl*

Global Optimization in the COCONUT Project ..... 243
*Hermann Schichl*

An Application of Wavelet Theory to Early Breast Cancer ..... 250
*Baya Oussena, Abderrezak Henni, René Alt*

## Novel Approaches to Verification

Using PVS to Validate the Inverse Trigonometric Functions of an Exact Arithmetic ..... 259
*David Lester*

Novel Approaches to Numerical Software with Result Verification ..... 274
*Laurent Granvilliers, Vladik Kreinovich, Norbert Müller*

Static Analysis-Based Validation of Floating-Point Computations ..... 306
*Sylvie Putot, Eric Goubault, Matthieu Martel*

**Author Index** ..... 315

# OOP and Interval Arithmetic – Language Support and Libraries

Jürgen Wolff von Gudenberg

Universität Würzburg
97074 Würzburg, Germany
wolff@informatik.uni-wuerzburg.de

**Abstract.** After a short presentation of the paradigms of object oriented programming and interval arithmetic the languages C++ and Java are treated in more detail. Language features are regarded with respect to their support for the definition or application of interval arithmetic. In the final section the 4 libraries Profil/BIAS, C-XSC, filib++ as well as Sun Forte C++ are compared with respect to functionality and efficiency.

## 1 Paradigms

### 1.1 Object Oriented Programming

An object oriented program simulates a part of the real or an imaginary world. Objects are constructed and communicate with each other via messages. Classes are defined to describe objects of the same kind. The class is the central and most important construct of object oriented programming languages. A class defines a type by giving attributes to describe a data structure and methods to specify the behavior of objects of that type. Using encapsulation details of the structure and implementation may be hidden, a class hence defines an abstract data type. Separation of interface and implementation is a commonly used pattern as well as hiding details of the representation or internal execution of the methods. Objects are instances of classes in the sense of data types, they have attributes determining their state and thus are elements of the domain. Objects control their own state, a method call usually stimulates an object to report or change its state. The standard data types like integers or floating-point numbers are available as primitive types, the elements are just values, not objects.

Object oriented languages usually provide several forms of polymorphism. Operator or function overloading, parameterized data types or inheritance are the main kinds of polymorphism. Templates parameterized by a data type may be instantiated to create a new data type. Homogeneous lists or matrices are a typical example. Inheritance based hierarchical programming, in particular, is often used as synonym for object oriented programming. It allows for the definition of containers with very general element types that then also can host specializations or derived types. Iterators are provided to pass through the container structure.

R. Alt et al. (Eds.): Num. Software with Result Verification, LNCS 2991, pp. 1–14, 2004.

Hierarchies of data types may be built where, usually, interfaces or abstract classes are near the root and their descendants, implementations or specializations follow towards the leaves. In contrast to these general structures arrays nearly play any role. Interfaces – explicitly known in Java and implemented as fully abstract classes in C++ – are used to define an abstract data type. An interface provides the signatures of methods of implementing classes. Common behavior for all descendants may be predefined in an abstract class by a call of abstract methods.

Given abstract `add` and `negate` methods of a class `Fp`, e.g., the `subtract` method can be defined for all descendants as

```
Fp subtract(Fp b) { return add(b.negate()) }
```

### 1.2 Interval Arithmetic

The main concern of interval arithmetic is to compute reliable bounds. The arithmetic interval operations, therefore, use directed rounding, interval versions of elementary functions and lattice or set operations are provided. Since many algorithms in scientific computing are not only declared for scalars, interval vectors and matrices are very important.

The most prominent applications of interval arithmetic are the global optimization [4,2] and the result verification using fixed point theorems [7,3].

Computation of the range of a function is one of the key problems in interval arithmetic. We will use it to investigate the degree of support of interval arithmetic by object oriented languages. There are many different algorithms to enclose the range. Surprisingly enough, even the most simplistic approach can be defined with two possible flavors of semantics, and no decision for one or the other seems to be convincing.

**Interval Evaluation**
$f(\mathbf{x}) = \{f(x) | x \in \mathbf{x}\}$ denotes the range of values of the function $f : D_f \subseteq \mathbb{R} \to \mathbb{R}$ over the interval $\mathbf{x} \subseteq D_f$.

An enclosure of the range can be implemented by interval evaluation of the formula expression for $f$.

**Definition 1** *The* ***interval evaluation*** $\mathbf{f} : \mathbb{IR} \to \mathbb{IR}$ *of* $f$ *is defined as the function that is obtained by replacing every occurrence of the variable* $x$ *by the interval variable* $\mathbf{x}$ *and by replacing every operator by its interval arithmetic counterpart and every elementary function by its range.*

We call this mode the normal or interval mode. Note that arithmetic operators and elementary functions are defined on their natural domain and produce an error, if the argument contains a point that is not in the domain. Hence, this definition only holds, if all operations are executable without exception.

**Containment Evaluation**
Alternatively in the containment or extended mode a range enclosure computes the topological closure over $\mathbb{R}^* = \mathbb{R} \cup \{-\infty\} \cup \{\infty\}$ by extending the domain

of real arithmetic operators to $\mathbb{R}^*$ and that of elementary functions to their topological closure, see [8]. No errors are invoked, but the resulting interval may be $\mathbb{R}^*$ or $\emptyset$. In the following definition $\wp$ denotes the power set.

**Definition 2** *Let $f : D_f \subseteq \mathbb{R} \to \mathbb{R}$, then the containment set $f^* : \wp\mathbb{R}^* \mapsto \wp\mathbb{R}^*$ defined by*
$f^*(\mathbf{x}) := \{f(x) | x \in \mathbf{x} \cap D_f\} \cup \{\lim_{D_f \ni x \to x^*} f(x) | x^* \in \mathbf{x}\} \subseteq \mathbb{R}^*$
*denotes the extended range of $f$.*

**Definition 3** *The **containment evaluation** $\mathbf{f}^* : \mathbb{IR}^* \to \mathbb{IR}^*$ of $f$ is defined as the function that is obtained by replacing every occurrence of the variable $x$ by the interval variable $\mathbf{x}$ and by replacing every operator or function by its extended interval arithmetic counterpart.*

**Theorem 1.**

$$f(\mathbf{x}) \subseteq \mathbf{f}(\mathbf{x}) \tag{1}$$

$$f(\mathbf{x}) \subseteq f^*(\mathbf{x}) \subseteq \mathbf{f}^*(\mathbf{x}) \tag{2}$$

The proof of (1) is well known, a similar step by step proof for (2) is carried out in [8].

**Discussion**
Since arithmetic operations as well as the elementary functions are continuous over their domain and since this continuity is lost by the extended operations, only the interval mode should be used, if continuity is a presupposition as for example in result verification algorithms [3] using Brouwer's fixed-point theorem. In the containment mode additional constraints have to be added to ensure continuity.

The normal mode, however, may be too restrictive in global optimization [2]. Here it is correct to intersect argument interval and domain in order to obtain a feasable set.

## 2 Requirements and Realisations

In this section we enumerate the requirements which are necessary, recommended, helpful, or at least nice to embed interval arithmetic in the object oriented languages C++ and Java.

### 2.1 Requirements for Interval Arithmetic

- A data type `interval` can be defined. (mandatory)
- Vectors and matrices are available. (mandatory)
- Floating-point arithmetic is clearly specified. (mandatory)
- Directed rounding is provided. (recommended)
- Intervals can be read and written. (mandatory)

- Interval literals are accessible. (helpful)
- Operators and functions can be overloaded. (recommended)
- Functions may be passed as parameters. (recommended)
- Evaluation of expressions may be redefined by the user. (helpful)
- Data types can be parameterized. (helpful)

Every programming language of interest supports the definition of data types, vectors and matrices.

Floating-point arithmetic is available in hardware. For the definition of interval arithmetic a clear specification of the performable operations, their accuracy and rounding mode is mandatory.

Even if we can assume that IEEE arithmetic is provided on every computer, we can not be sure that directed roundings are immediately accessible. Therefore we consider 7 different rounding procedures. $\nabla$ denotes the function that maps a real number to its greatest lower floating-point neighbour, $\triangle$ to the least upper, and $\bigcirc$ to the nearest floating-point neighbour. Usually the hardware rounding mode has to be switched explicitly. This switching may be an expensive operation.

For the operation $[\underline{z}, \overline{z}] = [\underline{x}, \overline{x}] + [\underline{y}, \overline{y}]$ the rounding procedures are

- native: set $\nabla; \underline{z} = \nabla(\underline{x} + \underline{y})$; set $\triangle; \overline{z} = \triangle(\overline{x} + \overline{y})$
- native-switch : set $\nabla; \underline{z} = \nabla(\underline{x} + \underline{y})$; set $\triangle; \overline{z} = \triangle(\overline{x} + \overline{y})$; set $\bigcirc$
- native-onesided : set $\nabla; \underline{z} = \nabla(\underline{x} + \underline{y}); \overline{z} = \nabla(-\nabla(-\overline{x} - \overline{y}))$
- native-onesided-switch: set $\nabla; \underline{z} = \overline{\nabla}(\underline{x} + \underline{y}); \overline{z} = \nabla(-\nabla(-\overline{x} - \overline{y}))$; set $\bigcirc$
- no switch: $\underline{z} = \nabla(\underline{x} + \underline{y})$; $\overline{z} = \triangle(\overline{x} + \overline{y})$
- multiplicative: $\underline{z} = (\underline{x} + \underline{y}) * pred(1.0)$; $\overline{z} = (\overline{x} + \overline{y}) * succ(1.0)$
- pred-succ: $\underline{z} = pred(\underline{x} + \underline{y})$; $\overline{z} = succ(\overline{x} + \overline{y})$

The first 4 procedures expect that directed rounding is available in hardware and can be selected via a switch, the onesided roundings need only one switch. If the switch back to round to nearest is omitted, the semantics of the floating-point arithmetic, that usually works with round to nearest, is changed.

The no-switch rounding procedure assumes that all 3 rounding modes are immediately accessible. Multiplicative rounding may be applied, if only round to nearest is provided by the hardware. The predecessor and successor of a floating-point number may be obtained by a hardware instruction or by bit manipulation.

Input and output as well as interval literals may be realized by an `interval`↔ `string` conversion.

For the realisation of algorithms like interval Newton method or range evaluation it is strongly recommended that functions may be passed as parameters. The definition of a particular non-standard evaluation of expressions is a further helpful ingredient (see # expressions in Pascal-XSC ([5]).

### 2.2 Realisation in Java

Java is one of the very few languages that specify the semantics of their floating-point arithmetic. There are even two modes to use IEEE arithmetic. In the

`strictfp` mode every intermediate result occuring in an evaluation of an expression has to be rounded to the nearest number of the corresponding primitive data type `double` or `float`, hence the same result is obtained on any computer. In the default mode, however, registers with a more precise floating-point format may be used as well as combined operations like the fused multiply and add operation. Exceptions for the IEEE traps overflow or division by zero, e.g., are never raised in any of the two modes.

Directed roundings have to be accessed by native, i.e. non-Java, methods. Those methods can be defined in a utility class `FPU`.

```
public final class FPU {
   public static final native double addDown(double x, double y);
   public static final native double mulUp(double x, double y);
...
```

Since there are no global functions in Java these utility classes are really necessary. The standard class `Math` provides the elementary functions.

An interval class may be defined as follows

```
public class Interval {
  // Constructor
  public Interval(double x, double y) {
    inf = x < y ? x : y;
    sup = x > y ? x : y;
  }


  // Access and Utility methods
  public double getInf() {
    return inf;
  }

  public double diam() {
    return FPU.subUp(sup, inf);
  }
  // ...

  // updating Arithmetic methods
  public Interval sub(Interval other) {
    double tmp = other.inf;
    inf = FPU.subDown(inf, other.sup);
    sup = FPU.subUp(sup, tmp);
    return this;
  }
  // ...
```

```
  //  Arithmetic methods or functions
  public Interval fsub(Interval other) {
    Interval that = new Interval();
    that.inf = FPU.subDown(this.inf, other.sup);
    that.sup = FPU.subUp(this.sup, other.inf);
    return that;
  }
  // ...

  protected double inf, sup;
}
```

Access to the protected or private attributes is only possible by explicitly provided public methods like `getInf()`. Thus the construction of an illegal interval with `inf>sup` is prohibited.

Overloading of operators is not allowed in Java, but functions (methods) and constructors can be overloaded.

Whereas all primitive data types like `float` and `double` follow the usual value semantics, Java prescribes reference semantics for all class types like `Interval`. Unexpected side effects due to aliasing may occur. The updating versions of operations change the state of the object for which the method was called. For the expression

`x=x-x/z`

`x.sub(x.div(z))` overwrites the value of `x` before the quotient is subtracted, whereas `x.fsub(x.fdiv(z))` yields the expected result.

General containers with corresponding iterators are available, but they cannot be used for primitive types without wrapping those into classes.

Vectors and matrices can be built, but a drawback of Java for scientific computing is certainly the fact that matrices do not allocate contiguous memory.

Because of these deficiencies of Java three major issues have been raised by the Java Grande forum[1].

1. The extension of floating-point arithmetic that led to the second (default) mode for IEEE arithmetic.
2. The MultiArray proposal which is still under consideration.
3. Java Generics which will come. Although explicit wrapping of primitive types into classes will no longer be necessary, we think it will take some time until efficient instantiation of the parameterized containers will have been achieved.

Nevertheless progress for Java compilers like semantic inlining or light weight objects has been made [1] in order to increase the performance.

Functions may be represented as objects and hence passed as parameters as in the following example.

[1] http://www.javagrande.org/

```
public class IntNewton {
  public UnaryFunction f;
  public IntNewton(UnaryFunction fun) {
    f  = fun;
  }

  public Interval enclZero(Interval x, double eps) {

    Interval Mid = new Interval();
    Interval fx, dfx;

    do {
      Mid.assign(x.mid());
      fx = f.evalRange(x.mid());
      dfx = f.evalDerivRange(x);
      x.intersect(Mid.sub(fx.div(dfx)));
    } while (x.diam() > eps);

    return x;
  }
}
```

An object of the class `IntNewton` is constructed with an appropriate implementation of the sample interface `UnaryFunction`.

```
public interface UnaryFunction {
  public double eval(double x);
  public double evalDeriv(double x);
  public Interval evalRange(double x);
  public Interval evalDerivRange(double x);
  public Interval evalRange(Interval x);
  public Interval evalDerivRange(Interval x);
}
```

### 2.3 Realisation in C++

In C++ floating-point data types and their operations are implementation defined, the template `numeric_limits<T>` gives information about the properties like representation of infinities etc. The rounding mode has to be switched by assembler statements. This often causes problems with optimizing compilers which do not see the dependence of floating-point operations on those assembler statements.

Overloading of operators and the existence of global functions allow for a smooth implementation of interval arithmetic. Type parameters can be used in templates to define interval arithmetic for different base types. All operations needed to instantiate the templates are imported via traits templates, in general.

These traits collections map base type specific operations to common names used in the arithmetic class. Pre-instantiated classes for the standard base types `double` or `float` realize this mapping during compile time.

```
template <typename basetype> class interval
{
interval<basetype> & interval<basetype>::operator -=
(interval<basetype> const & o)
  {
    basetype tmp = o.INF;
    INF=fp_traits<basetype>::downward_minus(INF,o.SUP);
    SUP=fp_traits<basetype>::upward_minus(SUP,tmp);
    fp_traits<basetype>::reset();
    return *this;
  }
  // ...
friend interval<basetype> & interval<basetype>::operator -
        (interval<basetype> const & a, interval<basetype> const & b);
}
  // ...
template <typename basetype>
interval<basetype> & interval<basetype>::operator -
        (interval<basetype> const & a, interval<basetype> const & b)
{
  interval<basetype> that;
  that.INF=fp_traits<basetype>::downward_minus(a.INF,b.SUP);
  that.SUP=fp_traits<basetype>::upward_minus(a.SUP,b.INF);
  fp_traits<basetype>::reset();
  return that;
}
```

As a friend the globally defined binary operator - has access to the internal structure of the interval data type. Parameters of class type can be passed by value, by reference or preferably by const reference. Hence, the expression

```
x=x-x/z
```

is exactly written like this.

In C++ containers defined in the STL (Standard Template Library) are in general parameterized by their contents' type. Efficient instantiation with primitive types is possible. In generic computing (using the STL) iterators combine containers with algorithms. Matrices are stored row-wise in a contiguous area. The matrix template library (MTL)[2] includes a large number of data formats and algorithms, including most popular sparse and dense matrix formats and functionality equivalent to Level 3 BLAS[3]. An appropriate instantiation with intervals is possible but not straightforward.

[2] http://www.osl.iu.edu/research/mtl/
[3] http://www.netlib.org/blas/

There are mainly 4 ways to pass a function as a parameter

- by a function pointer. `double (*f)(double)`
- as virtual function with late binding as in Java
- via a function object that overloads the function call operator `()` via template parameter
- using expression templates

Expression templates [10] represent complete expressions 'symbolically' by recursive templates and allow for user defined evaluation strategies via instantiation. Since this is done during compilation time, efficiency is not lost. We have applied this concept for the accurate evaluation of dotproducts [6].

The definition of a function object is not only elegant but also most efficient, since the first two methods rely on a runtime dereference.

It is possible in C++ to overload the function call operator `()`. A call of this postfix operator for an object then exactly looks like a call of a function with the object's name.

Here is the C++ example of the interval Newton method.

```
template<class T_fun, class T_der>
interval enclZero(interval x, double eps,
       T_fun const & f, T_der const & df) {
    interval fx;
    do {
      fx = x.mid() - f(x.mid())/df(x);
       x.intersect(fx);
    } while (x.diam() > eps);
    return x;
}



// example use
y = enclZero(interval(0.0,10.0),1e-8,
    MyFunction(),MyDerivative());


class MyFunction
{
  public:
    interval operator()(double x) const
    { interval X(x);
      return cos(X) + sin(X*X); }
};
class MyDerivative
{
  public:
```

```
    interval operator()(interval X) const
    {  return -sin(X) + 2*X*cos(X*X); }
};
```

Here we selected a very simple interface, a more sophisticated implementation using expression templates and automatic differentiation is possible.

## 3 Interval Libraries

All considered libraries are written in C++ . We do not know any publicly available, widely used Java interval library.

All four libraries contain the arithmetic operators as global functions and the updating operators as methods. They provide a set of elementary functions, lattice or set operations like intersection or interval hull and a set of relational operations.

Differences are in the definition of the data type and rounding mode as well as in some further features.

### 3.1 C-XSC

C-XSC[4] is a comprehensive library. It supports intervals of base type `double`, and complex intervals. There is a version with software floating-point arithmetic and pred-succ rounding procedure whereas the new version relies on hardware arithmetic. The normal interval evaluation of ranges is supported.

The set of elementary functions includes exponential, logarithmic, trigonometric and hyperbolic functions as well as their inverses. All functions are implemented with 1 or 2 ulp accuracy.

C-XSC provides global operators for the set operations.

| | |
|---|---|
| `\|` | `hull` |
| `&` | `intersect` |
| `<=, >=` | membership tests |
| `<, >` | means interior |
| `!` | zero included |

The only relational operations are equality and non-equality.

Input/output with proper rounding is possible with streams or strings.

C-XSC further provides

- vectors and matrices
- datatype dotprecision
- dynamic multiple precision arithmetic and standard functions
- problem solving routines

[4] http://www.math.uni-wuppertal.de/org/WRST/xsc/cxsc.html

### 3.2 Profil/BIAS

Profil/BIAS[5] provides intervals for base type double in normal mode.

There are functions delivering an enclosure of the result of an arithmetic operation with two floating-point numbers. The set of lattice, relational and elementary functions is similar to C-XSC.

It has a sophisticated vector and matrix package and supports multiple precision interval arithmetic.

The current version is from 1994 and, hence, has some problems with newer compilers.

### 3.3 Sun Forte

The interval arithmetic library from Sun[6] features the extended mode and offers some compiler support. The interval class is given as a template, specializations for `float, double, longdouble` exist. The rounding mode is native-onesided.

There are convenient input/output features which manipulate the decimal string representation of binary floating-point numbers. There is, of course, a constructor with a string, input as well as output values are properly rounded, the latter in the decimal external format.

Single number input/output are provided, the number represents the midpoint, the radius of the interval is one decimal digit in the last place of the midpoint representation. E.g. output of [2.34499, 2.34501] yields 2.34500. During input to a program, [0.1,0.1] = [0.1] represents the point, 0.1, while using single-number input/output, 0.1 represents [0, 0.2].

The membership tests are implemented by functions, the operators are used for the set relational operations. Additionally possibly and certainly relational operations are provided. Possibly means that there are points in either interval for which the relation holds, certain relations hold for all points.

### 3.4 filib++

filib++ [7] is the newest of the libraries, the interface is similar to the Sun library whereas the implementation of the elementary functions is an accelerated, slightly less accurate, but rigorous version based on the C-XSC functions.

The interval class is given as a template with 3 parameters, the base type, the rounding mode and the evaluation mode.

Operators for the different base types are imported via traits templates. Specializations for `float, double` exist.

The rounding mode may be set to all 7 procedures, listed in section 2.1.

There are three possible choices for the evaluation mode parameter. The default is the normal mode, the extended mode can be chosen or a combination

[5] http://www.ti3.tu-harburg.de/Software/PROFILEnglisch.html
[6] http://docs.sun.com/source/816-2465/index.html
[7] http://www.math.uni-wuppertal.de/org/WRST/software/filib.html

of both modes that computes in the extended mode but sets a flag, whenever the normal mode would have raised an exception. This continuity flag informs the user, whether a continuity condition has been violated.

Input and output facilities are somewhat restricted in filib++ . The string constructor relies on the fact that the decimal to binary conversion is accurate. (The shipped default conversion routine has this property.) Output prints the two bounds using the default binary to decimal conversion. Additionally the bit image of the bounds can be output.

### 3.5 Timings

In the last section we want to present some performance tests of the arithmetic operators in each library or with different rounding procedure. Note that these results heavily depend on the underlying hardware, operating system, and compiler. Individual checks should be done to determine the most efficient version.

We tested the arithmetic operations in a loop, the numbers (`double`) were randomly generated into vectors of different lengths. The processor was a 2GHz Pentium 4 running under Linux. For filib++ we used the gcc 3.2.1 compiler with optimization level O3, for Profil/BIAS and C-XSC we had to choose gcc 2.95.3 optimization level O3 or O1, respectively.

A newer version of C-XSC that exploits the hardware arithmetic is in preparation. The performance will grow by a factor of 10, approximately.

**Comparison of Libraries**
The figures in the following tables denote MIOPs (million interval operations per second).

| Library | + | - | * | / |
|---|---|---|---|---|
| filib++ traits | 22.4 | 22.2 | 11.4 | 8.9 |
| filib++ macro | 17.7 | 17.6 | 10.9 | 8.0.97525 |
| profil | 11.6 | 11.3 | 7.6 | 9.8 |
| cxsc-1 | 1.8 | 1.5 | 1.3 | 0.7 |

The fastest traits version of filib++ was tested against an older version using no templates but macros, the Profil/BIAS library and the old version of C-XSC. The table shows that the new compiler technology makes the macro version obsolete.

**Timings Rounding Mode**
The dependence on the rounding mode is tested in the next table, where all rounding procedures in filib++ were compared. Note that in this case no-switch means no rounding, since the processor needs a switch to change the rounding mode. This mode does not deliver reliable bounds, it is only tested for comparison.

| Rounding mode | + | - | * | / |
|---|---|---|---|---|
| native | 22.4 | 22.2 | 11.4 | 8.8 |
| native-switch | 3.9 | 3.9 | 3.5 | 3.0 |
| native-onesided | 20.9 | 21.2 | 13.9 | 8.2 |
| native-onesided-switch | 19.2 | 19.3 | 8.9 | 6.3 |
| no-switch | 24.7 | 24.6 | 16.4 | 9.2 |
| multiplicative | 8.8 | 8.9 | 6.1 | 6.2 |
| pred-succ | 7.5 | 7.8 | 1.5 | 1.7 |

We think that the bad performance of native-switch is caused by the architecture of the processor that can handle two but not three switches effectively.

**Timings Extended Mode**
The next table displays the results for the extended mode.

| Rounding mode | + | - | * | / |
|---|---|---|---|---|
| native | 18.7 | 18.9 | 4.5 | 8.5 |
| native-switch | 3.6 | 3.6 | 2.5 | 2.8 |
| native-onesided | 11.9 | 11.9 | 7.9 | 6.3 |
| native-onesided-switch | 10.5 | 10.6 | 4.5 | 5.0 |
| no-switch | 22.0 | 22.1 | 10.6 | 9.1 |
| multiplicative | 8.5 | 8.5 | 4.6 | 5.6 |
| pred-succ | 6.8 | 7.0 | 0.5 | 0.9 |

**Comparison with Sun Forte**
Finally we compared filib++ with the Sun Forte library. These benchmarks were performed on a Sun Ultra 60 with 2 processors running at 360 MHz. filib++ with rounding native-onesided-switch in extended mode was tested against Sun's interval arithmetic. The filib++ benchmark was compiled by the gcc 3.2 compiler optimization level O3, the same program using Sun's intervals was compiled by Sun's CC compiler in default mode and with optimization level O5, respectively.

| Library | + | - | * | / |
|---|---|---|---|---|
| filib++ traits | 3.2 | 3.2 | 1.9 | 2.0 |
| Sun | 1.3 | 1.3 | 1.0 | 1.1 |
| Sun (O5) | 2.8 | 2.7 | 2.1 | 1.8 |

It turns out that the filib++ – gnu combination outperforms Sun.

## 4 Conclusion

Object orientation and interval arithmetic are complementary paradigms which well fit together. In our opinion the support of interval arithmetic in C++ is superior to that in Java. That is also evident by the fact that some C++ libraries are available and commonly used. Comparing the libraries shows that there are not so much differences, but some of them have really grown old and would benefit from a new updated release.

**Acknowledgements.** We like to thank German Tischler who performed the benchmarks and Werner Hofschuster who helped us with the Sun implementation in Wuppertal.

## References

1. Artigas, P.V. et al.: *High Performance Numerical Computing in Java: Language and Compiler Issues*, Proceedings of the 12'th Workshop on Language and Compilers for Parallel Computers, Aug. 4-6, 1999, San Diego, CA
2. Bliek, C. et al.: *Algorithms for Solving Nonlinear Constrained and Optimization Problems: The State of the Art*, `http://solon.cma.univie.ac.at/~neum/glopt/coconut/StArt.html`
3. Hammer, R. et al.: *C++ Toolbox for Verified Computing*, Springer, Berlin, 1995
4. Kearfott, R.B.: *Rigorous Global Search: Continuous Problems*, Kluwer Academic Publishers, Dordrecht, Netherlands, 1996
5. Klatte, R., et al.: *PASCAL– XSC — Language Reference with Examples* Springer-Verlag, Berlin /Heidelberg/New York, 1992.
6. Lerch, M.; Wolff v. Gudenberg, J.: *Expression Templates for Dot Product Expressions*, Proceedings of Interval 98, Reliable Computing 5 (1), p. 69-80, 1999
7. Rump, S.M. *Self-validating methods*, Linear Algebra and its Applications (LAA), 324:3-13, 2001.
8. Walster, G.W. et al.: *Practical Exception-free Interval Arithmetic on the Extended Reals*, Sun Microsystems, white paper, August 2001.
9. Walster, G.W. et al.: *Extended Real Intervals and the Topological Closure of Extended Real Relations*, Sun Microsystems, white paper, March 2002, `http://wwws.sun.com/software/sundev/whitepapers/index.html`
10. Veldhuizen, T.: *Expression Templates*, C++ Report, Vol. 7, No. 5, 1995

# C-XSC 2.0
# A C++ Library for Extended Scientific Computing

Werner Hofschuster and Walter Krämer

Bergische Universität Wuppertal
Scientific Computing/Software Engineering
42097 Wuppertal, Germany
{Hofschuster, Kraemer}@math.uni-wuppertal.de
http://www.math.uni-wuppertal.de/~xsc

**Abstract.** In this note the main features and newer developments of the C++ class library for extended scientific computing C-XSC 2.0 will be discussed.
The original version of the C-XSC library is about ten years old. But in the last decade the underlying programming language C++ has been developed significantly. Since November 1998 the C++ standard is available and more and more compilers support (most of) the features of this standard. The new version C-XSC 2.0 conforms to this standard. Application programs written for older C-XSC versions have to be modified to run with C-XSC 2.0. Several examples will help the user to see which changes have to be done. Note, that all sample codes given in [6] have to be modified to work properly with C-XSC 2.0.
All sample codes listed in this note will be made available on the web page http://www.math.uni-wuppertal.de/~xsc/cxsc/examples.

## 1 Introduction

For those who are not so familiar with C-XSC let us first motivate the library by quoting essential parts (with slight modifications) from the preface of the book [6]:

The programming environment C-XSC (C++ for eXtended Scientific Computing) is a powerful and easy to use programming tool, especially for scientific and engineering applications. C-XSC is particularly suited for the development of numerical algorithms that deliver highly accurate and automatically verified results. It provides a large number of predefined numerical data types and operators of maximum accuracy. The most important features of C-XSC are real, complex, interval, and complex interval arithmetic with mathematically defined properties; dynamic vectors and matrices; dotprecision data types (accurate dot products); predefined arithmetic operators with highest accuracy; standard functions of high accuracy; dynamic multiple-precision arithmetic and rounding control for the input and output of data.

R. Alt et al. (Eds.): Num. Software with Result Verification, LNCS 2991, pp. 15–35, 2004.

Accumulation of numbers is the most sensitive operation in floating-point arithmetic. By that operation scalar products of floating-point vectors, matrix products etc. can be computed without any error in infinite precision arithmetic, making an error analysis for those operations superfluous. Many algorithms applying that operation systematically have been developed. For others the limits of applicability are extended by using this additional operation. Furthermore, the optimal dot product speeds up the convergence of iterative methods (cited from [10,11]). C-XSC provides accurate dot products via software simulation (hardware support should increase the computation speed by 2 orders of magnitude, again, see [11]). Computing $x*y$ for floating point vectors $x$, and $y$ in C-XSC results in the best possible floating point result (exact mathematical result rounded to the nearest floating point number). Using the new C-XSC data type `dotprecision` the user can even store the result of dot products of floating point vectors with even millions of components without any error. The so called staggered format allows multiple-precision computations. The realization of arithmetic operations for variables of this data type use extensively the accurate dot product. With appropriate hardware support for dot product operations the staggered arithmetic would be very fast.

C-XSC consists of a run time system written in ANSI C and C++ including an optimal dot product and many predefined data types for elements of the most commonly used vector spaces such as real and complex numbers, vectors, and matrices. Operators for elements of these types are predefined and can be called by their usual operator symbols. Thus, arithmetic expressions and numerical algorithms are expressed in a notation that is very close to the usual mathematical notation.

Additionally, many problem-solving routines with automatic result verification (e.g. C++ Toolbox for Verified Computing with one- and multi-dimensional solvers for systems of linear equations, linear optimization, automatic differentiation, nonlinear systems of equations, global optimization and further packages like slope and taylor arithmetic or quadrature and cubature of singular integrals) have been developed in C-XSC for several standard problems of numerical analysis. All software is freely available.

## 2 Overview on the New Version C-XSC 2.0

Due to the following observations older C-XSC programs have to be modified slightly to run with C-XSC 2.0 (for details please refer to paragraph 4):

- All C-XSC routines are now in the namespace `cxsc`. So you have to fully qualify names of C-XSC routines (e. g. cxsc::sin(cxsc::intval(3.0)) ) or you have to include the line `using namespace cxsc;` in your source code.
- Now typecast constructors are available
- Constant values formerly passed by reference are now passed by `const` references
- Modifications in the field of subvectors and submatrices have been done

- The error handling is now done using the C++ exception handling mechanism (using `try`, `catch`, and appropriate exception classes)
- The new version of the library uses templates extensively

The source code of C-XSC 2.0 is freely available from `http://www.math.uni-wuppertal.de/~xsc/xsc/download.html` and the source code of a new version of the C++ Toolbox for Verified Computing [1] which works with C-XSC 2.0 is also freely available from the same web site.

## 3 Freely Available Software Based on C-XSC 2.0

Here we list (additional) software based on C-XSC 2.0 which is freely available from our web-site:

a) (Modified) Toolbox for Verified Computing (see [1]). This toolbox comprises a couple of verification algorithms for one- and multi-dimensional numerical problems:

a1) The available one-dimensional problem solving routines are:

- Accurate polynomial evaluation
- Automatic differentiation
- Nonlinear equations in one variable
- Selfverifying global optimization
- Accurate arithmetical expressions
- Zeros of complex polynomials

a2) The available multi-dimensional problem solving routines are:

- Systems of linear equations
- Linear optimization
- Automatic differentiation (gradient, Jacobi-, Hesse matrix)
- Nonlinear systems of equations
- Global optimization

b) Further available software packages are:

- Interval slope arithmetic (Breuer)
- Interval Taylor arithmetic (Breuer)
- Mathematical functions for complex rectangular intervals (Westphal)
- Verified quadrature and cubature of nonsingular and singular integrals (Wedner, see [8,20])
- Verified estimates for Taylor coefficients of analytic functions (Neher [16])
- Routines to compute rigorous worst case a priori bounds for absolute and/or relative errors of floating point algorithms (Bantle [7])
- Solvers for under- and overdetermined systems of linear equations (Hölbig [3])
- Verified solutions of ordinary differential equations (Lohner [13])

You can download the source code of all software packages from `http://www.math.uni-wuppertal.de/~xsc`.
There, you also find more specific information on the packages as well as some preprints.

## 4 Which Modifications in Source Codes Are Required?

In this section we try to answer the most frequently asked questions of C-XSC users concerning the migration of older C-XSC application programs to the new C-XSC 2.0 version. For those who are familiar with the C++ standard [5] the source code modifications should be rather obvious (see e.g. Stroustrup [19], Meyers [14,15]).

To make available the advanced input and output facilities (stream concept) of C++ you must include the headerfile `iostream` using the source line `#include <iostream>`. Note, the name of the header is **not** `iostream.h`. In general, the names of system header files coming with C++ do not have an extension.

To perform conversions of interval constants given as strings C-XSC uses the header file `#include <string>`. This header introduces (dynamic) C++ strings with predefined operators.

C-XSC delivers several header files. The extension of these files is `.hpp`. The header files correspond to the additional numerical data types available in C-XSC (like `interval`, `imatrix`, `cmatrix`, ...). The name of the header files are

```
cdot.hpp        dot.hpp         l_complex.hpp  lrvector.hpp
cidot.hpp       idot.hpp        l_imath.hpp    real.hpp
cimatrix.hpp    imath.hpp       l_interv.hpp   rmath.hpp
cinterval.hpp   imatrix.hpp     l_real.hpp     rmatrix.hpp
civector.hpp    interval.hpp    l_rmath.hpp    rvector.hpp
cmatrix.hpp     intmatrix.hpp   limatrix.hpp
complex.hpp     intvector.hpp   livector.hpp
cvector.hpp     ivector.hpp     lrmatrix.hpp
```

The leading `l` in the name of a header file indicates a long precision (staggered) data type, `dot` indicates dotprecision data types able to store dot products without errors (long accumulators). In contrast to system header files which are included in the form `#include <header>` C-XSC header files are included using double quotes `#include "cxscheader.hpp"`.

The result type of the routine `main()` should be `int`.

Newer C++ compiler implement the namespace concept more strictly. The standard namespace of C++ is called `std`. All C-XSC routines are defined in the namespace `cxsc`. If you don't want to fully qualify the names of such routines (e. g. `std::cout`, or `cxsc::interval`) you should include the two source lines

```
using namespace std;  //make available names like cout, endl, ...
using namespace cxsc; //make available names of C-XSC routines
```

in your application code.

The following simple example program demonstrates most of the points from above. It checks whether the number 0.1 is representable as a point interval in C-XSC. If this is not the case, the decimal number 0.1 is not exactly representable as a double number.

```
#include <iostream>     //C++ stream concept for input and output
#include <string>       //ANSI C strings
#include "interval.hpp" //C-XSC header file for data type interval

using namespace std;    //make available names like cout, endl, ...
using namespace cxsc;   //make available names of C-XSC routines

int main()
{
   interval x; //x is an interval variable

   string("[0.1,0.1]") >> x;  //convert the interval constant to its
                              //internal binary representation
                 //(using directed roundings)
   if (Inf(x) != Sup(x))
         cout << "Number x has no exact binary representation!";
      else
         cout << "Number x has an exact binary representation!";

   cout << endl << "x = " << x << endl; //decimal output using
                                        //C++ streams
   cout << Hex  << "x = " << x << endl; //hexadecimal output

   return 0;
}

/* ----------------------- Output -------------------------
Number x has no exact binary representation!
x = [  0.099999,  0.100001]
x = [+19999999999999e3FB,+1999999999999Ae3FB]
------------------------------------------------------- */
```

If your (older) application code contains calls to conversion functions like `_interval(...)` you should now use constructor calls like `interval(...)` instead. The C-XSC conversion functions (starting with an underscore) are obsolete.

Several function signatures of C-XSC routines have been changed from reference parameters (`T& x`) to `const` reference parameters (`const T& x`). The following C++ sample program demonstrates some consequences.

```
#include <iostream>
using namespace std;

void f(const double& x) { cout << "Formal argument with const" << endl; }

void f(double& x) { cout << "No const qualifier" << endl; }

int main()
{
   double x=2;
```

```
   f(1.0);    //1, actual argument is not an lvalue
   f(x);      //2, x is an lvalue
   f(1.0+x);  //3, actual argument is not an lvalue
   f(x+x);    //4, actual argument is not an lvalue
   return 0;
}
/*
Formal argument with const
No const qualifier
Formal argument with const
Formal argument with const
*/
```

Note, due to the `const` qualifier the signatures in the two definitions of f() are different in C++! If we remove the first definition of f(), the function calls in the lines indicated by 1, 3, and 4 produce errors during the compilation process. In these cases the actual arguments are not lvalues whereas the formal argument of type `double&` (see the second definition of f) requires an lvalue.

Note, that the two definitions

```
void g(const double x) {cout << "Formal argument with const" << endl;}
void g(double x) {cout << "No const qualifier" << endl;}
```

are not allowed simultaneously in a C++ program unit. Here, the formal arguments are not declared as references. This implies that in both cases the actual argument in a function call is passed by value (the values of the actual arguments can not be changed in the body of the function). So an additional `const` qualification does not make sense.

Operators like `[]` as member function of a class may be overloaded differently for objects and `const`-objects. This is demonstrated by the following C++ sample code (the `const` between the parameterlist and the body of the operator definition indicates that in the body of the function the attributes of the left hand side object in a corresponding operator call are not modifiable):

```
#include<iostream>
using namespace std;

typedef double T;

struct vector {
   vector(int k) //constructor
   {
      start= new T[k];
      for (int i=0; i<k; i++) start[i]= i;
   }

   //operator [] may be applied to vectors
   //elements are readable and writable (result type is T&)
   T& operator[](int k)
   {
```

```
      cout << "[] without const ... " << endl;
      return  start[k];
   }

   //operator [] may be applied to const vectors
   //elements are only readable (result type is const T&)
   const T& operator[](int k) const
   {
      cout << "[] with const ... " << endl;
      return  start[k];
   }

   ~vector() { delete[] start; } //destructor
 private:
   T* start;
};

int main() {
   vector x(3);
   cout << "x[2]: " << x[2] << endl;
   x[2]= 5; //Note, calling operator[] creates output (see below)
   cout << "x[2]: " << x[2] << endl;
   const vector y(3);  //the same as vector const y(3);
   cout << "y[2]: " << y[2] << endl;
// y[2]= 5; //would lead to a compile time error:
            //The left operand cannot be assigned to
   return 0;
}
/* Output:

x[2]: [] without const ...
2
[] without const ...
x[2]: [] without const ...
5
y[2]: [] with const ...
2
*/
```

In contrast to the older C-XSC versions C-XSC 2.0 uses additional helper classes `intvector_slice`, `rvector_slice`, `ivector_slice`, `cvector_slice`, `civector_slice`, `l_rvector_slice`, `l_ivector_slice`, `intmatrix_slice`, `intmatrix_subv`, `rmatrix_slice`, `rmatrix_subv`, `imatrix_slice`, `imatrix_subv`, `cmatrix_slice`, `cmatrix_subv`, `cimatrix_slice`, `cimatrix_subv`, `l_rmatrix_slice`, `l_rmatrix_subv`, `l_imatrix_slice`, `l_imatrix_subv` to implement subvectors and subarrays.

The following program shows how the first row and the first column of a real matrix may be modified calling a function called `testfct`. The formal parameter of this function must be of data type `rmatrix_subv`.

```
#include <iostream>
#include "rmatrix.hpp" //C-XSC header for real matrices
                       //header for real vectors is included
                                                        automatically

using namespace std;
using namespace cxsc;

void testfct(const rmatrix_subv& y) //pay attention to the data type of y
//void testfct(const rvector& y) an error message or a warning would be
//                               generated by actual compilers
{
    for (int i=Lb(y); i<=Ub(y); i++) y[i]= i;
}

int main(void)
{
   rmatrix M;          //M is a real matrix
   int dim;
   cout << "Dimension = "; cin >> dim;

   Resize(M,dim,dim);  //create M with dim rows and dim columns
   M= 1;               //set all elements of M to 1

   cout << "Matrix M:" << endl << M << endl;
   testfct(M[1]);      //M[1] means the first row of M
   cout << "Matrix M:" << endl << M << endl;

   testfct(M[Col(1)]); //M[Col(1)] means the first column of M
   cout << "Matrix M:" << endl << M << endl;

   M[Col(1)]= 9;       //set all elements of column 1 to 9
   cout << "Matrix M:" << endl << M << endl;

   return 0;
}

/* Output

Dimension = 3
Matrix M:
  1.000000   1.000000   1.000000
  1.000000   1.000000   1.000000
  1.000000   1.000000   1.000000

Matrix M:
  1.000000   2.000000   3.000000
  1.000000   1.000000   1.000000
  1.000000   1.000000   1.000000
```

```
Matrix M:
  1.000000   2.000000   3.000000
  2.000000   1.000000   1.000000
  3.000000   1.000000   1.000000

Matrix M:
  9.000000   2.000000   3.000000
  9.000000   1.000000   1.000000
  9.000000   1.000000   1.000000
*/
```

# 5 Examples

In this section we give a couple of complete sample codes to demonstrate the usage and several features of C-XSC 2.0.

## 5.1 Example: Accurate Summation of Floating-Point Numbers

Let us start with a very simple demonstration of how the accurate dot product feature may be used to get accurate results when summing up floating-point numbers of very different orders of magnitude. The C-XSC routine `accumulate(a,x,y)` computes `a+x*y` without any error. Here `x` and `y` are floating-point numbers and `a` is a variable of type `dotprecision` (a so called long accumulator):

```
//Severe cancellation when computing the sum of three numbers
//Using a dotprecision variable results in the correct result

#include <iostream> //C++ input and output
#include "dot.hpp"  //make available C-XSC's accurate dot product feature
using namespace std;
using namespace cxsc; //make available C-XSC names without cxsc::

int main() {
   const real large(1.23e35);    //create a large number

   dotprecision a(0); //a is a dot precision variable initialized by 0
   accumulate(a,  1.0, large);  //a = 1.0*large = 1.23e35
   cout << a << endl;

   accumulate(a,  1.0, 1.5);    //a = 1.0*large + 1.0*1.5
                                //  = 1.2300...015e35
   accumulate(a, -1.0, large);  //a=  large + 1.5 - large = 1.5
   cout << "Final correct result is" << a << endl;

   cout << "Naive floating point evaluation gives" << endl
        << "  the totally wrong result"
        << large + 1.5 - large << endl;
   return 0;
```

```
}
/*  output:
1.2300000000E+0035
Final correct resultis is   1.5000000000
Naive floating point evaluation gives
  the totally wrong result  0.000000
*/
```

The possibility to compute dot products of floating point vectors accurately is the key for the implementation of matrix/vector operations of maximum accuracy in C-XSC. This feature is also used extensively in defect correction steps of iterative schemes. The operations for the staggered data type (multiple-precision) available in C-XSC [9] are heavily based on accurate dot product computations.

### 5.2 Example: Accurate Evaluation of Arithmetical Expressions

The following arithmetical expression has been used by Loh and Walster [12] as an example in which numerical evaluations using IEEE 754 arithmetic gave a misleading result, even though use of increasing arithmetic precision suggested reliable computation (the expression is a rearrangement of Rump's original example given in [17]). Evaluating

$$f(a,b) = (333.75 - a^2)b^6 + a^2(11a^2b^2 - 121b^4 - 2) + 5.5b^8 + \frac{a}{2b} \qquad (1)$$

for $a = 77617$ and $b = 33096$ using 32-bit, 64-bit, and 128-bit round-to-nearest IEEE-754 arithmetic produces:

32-bit: $f = 1.172604$
64-bit: $f = 1.1726039400531786$
128-bit: $f = 1.172603940053178631858834904520183 8$

However, the correct result is -0.8273960...

To compute a sharp enclosure of $f(a,b)$ we use the staggered interval arithmetic available in C-XSC.

```
#include <iostream>
#include "l_interval.hpp" //staggered intervals (multi-precision
                                                          intervals)

using namespace cxsc;     //make available routines from namespace cxsc
using namespace std;

l_interval f ( const l_interval& a, const l_interval& b )
{
  l_interval z;           //multi-precision interval

  z = (333.75 - power(a,2))*power(b,6) + power(a,2)*(11.0*power(a,2)
      *power(b,2) - 121.0*power(b,4) - 2.0) + 5.5*power(b,8) + a/(2.0*b);
```

```
  return(z);
}

int main( )
{
  l_real     a, b; //multi-precision reals
  l_interval res;  //multi-precision interval
  real       Eps;

  cout << "Enter the arguments:" << endl;
  cout << "   a = " ; cin >> a;  //read a multi-precision real
  cout << "   b = ";  cin >> b;
  cout << endl;

  cout << "Desired accuracy: Eps = "; cin >> Eps;
  cout << endl;

  cout << "Evaluation of (333.75 -a^2)b^6+a^2(11a^2b^2-121b^4-2)
           +5.5b^8+a/(2b)"
       << endl << endl;

  stagprec=0;
  do {
      stagprec++;
      res = f(l_interval(a),l_interval(b));
      //Output format via dotprecision
      cout << SetDotPrecision(16*stagprec, 16*stagprec-3);
      cout << "Interval enclosure: " << res << endl;
      cout << SetDotPrecision(5,2);
      cout << "Diameter:           " << diam(res) << endl;
  } while (diam(res)>Eps);

  return 0;
}

/* ---------------------- Output -------------------------
Enter the arguments:
   a = 77617
   b = 33096

Desired accuracy:
   Eps = 1e-100

Evaluation of (333.75 -a^2)b^6+a^2(11a^2b^2-121b^4-2)+5.5b^8+a/(2b)

Interval enclosure: [-3.5417748621523E+0021,
                      3.5417748621523E+0021]
Diameter:           7.08E+0021

Interval enclosure: [-6.55348273960599472047761082650E+0004,
```

```
                      1.17260394005317869492444060598]
Diameter:           6.55E+0004

Interval enclosure: [-0.82739605994682136814116509547981629199903311 6,
                     -0.82739605994682136814116509547981629199903311 5]
Diameter:           2.74E-0048

Interval enclosure: [-0.82739605994682136814116509547981629199903311 5
                        7843848199178149,
                     -0.82739605994682136814116509547981629199903311 5
                        7843848199178148]
Diameter:           1.52E-0064

Interval enclosure: [-0.82739605994682136814116509547981629199903311 5
                        7843848199178148416727096930142615421803239 06
            213,
                     -0.82739605994682136814116509547981629199903311 5
                        7843848199178148416727096930142615421803239 06
            212]
Diameter:           1.87E-0096

Interval enclosure: [-0.82739605994682136814116509547981629199903311 5
                        7843848199178148416727096930142615421803239 06
                        2122310853275320281,
                     -0.82739605994682136814116509547981629199903311 5
                        7843848199178148416727096930142615421803239 06
                        2122310853275320280]
Diameter:           2.08E-0112
```

The last enclosure is accurate to more than 110 digits (that is to all digits printed).

Let us now solve the same problem (1) (example from Rump/Loh & Walster) with the toolbox algorithm for the accurate evaluation of arithmetical expressions:

```
#include <expreval.hpp>     //Expression evaluation

using namespace cxsc;
using namespace std;

Staggered f ( StaggArray& v )
{
  Staggered a, b;
```

```
  a = v[1];
  b = v[2];

  return((333.75 - Power(a,2))*Power(b,6) + Power(a,2)*(11.0*Power(a,2)
   *Power(b,2) - 121.0*Power(b,4) - 2.0) + 5.5 * Power(b,8) + a/(2.0*b));
}

int main ( )
{
  real     Eps, Approx;
  int      StaggPrec, Err;
  rvector  Arg(2);
  interval Encl;

  cout << SetPrecision(23,15) << Scientific;   //Output format

  cout << "Evaluation of (333.75 -a^2)b^6+a^2(11a^2b^2-121b^4-2)
           +5.5b^8+a/(2b)"
       << endl << endl;

  cout << "Enter the arguments:" << endl;
  cout << "   a = " ; cin >> Arg[1];
  cout << "   b = ";  cin >> Arg[2];
  cout << endl;

  cout << "Desired accuracy:  Eps = " ; cin >> Eps;
  cout << endl;

  Eval(f, Arg, Eps, Approx, Encl, StaggPrec, Err);

  if (!Err) {
    cout << "Floating-point evaluation:  " << Approx << endl;
    cout << "Interval enclosure:         " << Encl << endl;
    cout << "Defect corrections needed:  " << StaggPrec << endl;
  }
  else
    cout << EvalErrMsg(Err) << endl;

  return 0;
}

/* ---------------------- Output -------------------------

Evaluation of (333.75 -a^2)b^6+a^2(11a^2b^2-121b^4-2)+5.5b^8+a/(2b)

Enter the arguments:
   a = 77617
   b = 33096

Desired accuracy:  Eps = 1e-15
```

```
Floating-point evaluation:  1.172603940053179E+000
Interval enclosure:       [-8.273960599468215E-001,
                                    -8.273960599468213E-001]
Defect corrections needed:  2
------------------------------------------------------------ */
```

Again, the computed interval enclosure is sharp.

### 5.3 Example: Linear System of Equation

We want to solve the (ill-conditioned) system of linear equations $Ax = b$ with

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} = \begin{pmatrix} 64919121 & -159018721 \\ 41869520.5 & -102558961 \end{pmatrix},\ b = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix},\ x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

The correct solution is $x_1 = 205117922, \quad x_2 = 83739041.$

To solve this $2 \times 2$ system numerically we first use the wellknown formulas

$$x_1 = \frac{a_{22}}{a_{11}a_{22} - a_{12}a_{21}}, \quad x_2 = \frac{-a_{21}}{a_{11}a_{22} - a_{12}a_{21}} \tag{2}$$

The following ANSI-C program

```
#include <stdio.h>

int main(void)
{
   double a11= 64919121.0, a12= -159018721.0,
          a21= 41869520.5, a22= -102558961.0,
          h1, h2, x1, x2;

   h1= a11*a22;
   h2= a12*a21;
   x1=  a22/(h1-h2);
   x2= -a21/(h1-h2);
   printf("x1= %15f   x2= %15f\n", x1, x2);
   return 0;
}
```

produces the totally wrong result

$$x_1 = 102558961, \quad x_2 = 41869520.5.$$

I. e. using IEEE double-arithmetic to evaluate the formulas (2) shown above give meaningless numerical results.

We now try to solve the linear system using Matlab.
Here we compute the inverse matrix (theoretically, the first column of the inverse is the solution of the linear system)

```
>> inv(A)
Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 1.651447e-17.
ans =
          106018308.007132         -164382474.017831
      43281793.0017831                 -67108864
>> A*inv(A)
ans =
     0     2
    -1     2
>> inv(A)*A
ans =
     1     2
     0     1
```

A*inv(A) as well as inv(A)*A should give the identity matrix. Obviously, the computed results are again not reliable. But this time we get at least a warning from Matlab.

If we try to compute an enclosure of the solution vector $x$ using Rump's IntLab package [18]

```
x = verifylss(A,b)
```

we get the same warning as in Matlab (indeed it is the Matlab warning) and the output

```
No inclusion achieved.
x =
   NaN
   NaN
```

IntLab is not able to solve the system. No meaningless numerical values are produced.

Let us now try to solve our ill-conditioned problem using C-XSC. Calling the solver for systems of linear equations from the Toolbox library [2] (using the interactive toolbox example program `lss_ex`) we get the following enclosure of the solution:

```
Enter the dimension of the system: 2

Enter matrix A:
64919121 -159018721
41869520.5 -102558961

Enter vector b:
1 0

Naive floating-point approximation:
```

```
 2.051179220000000E+008
 8.373904100000000E+007

Verified solution found in:
[ 2.051179220000000E+008, 2.051179220000000E+008]
[ 8.373904100000000E+007, 8.373904100000000E+007]

Condition estimate:  1.2E+017
```

The computed result is the correct solution (internally the toolbox routine makes use of the accurate dot product evaluation available in C-XSC).

### 5.4 Example: Cauchy Principal Value Integral

The freely available package CLAVIS (**Cla**sses for **v**erified **I**ntegration over **S**ingularities) has been developed and implemented using C-XSC by Wedner as part of his thesis [20]. This package allows the computation of enclosures for definite integrals of several kinds (Riemann, Cauchy principal values, ...).
Let us start with two definitions:

The Cauchy principal value integral $I(f;\lambda)$ is defined as follows

$$I(f;\lambda) := \int_a^b \frac{f(x)}{x-\lambda}\,dx := \lim_{\varepsilon\to 0+}\left(\int_a^{\lambda-\varepsilon}\frac{f(x)}{x-\lambda}\,dx + \int_{\lambda+\varepsilon}^b \frac{f(x)}{x-\lambda}\,dx\right),\quad \lambda\in(a,b)$$

and $f \in C^{2n+1}[a,b]$.

The nested integral $I(f;\lambda,\mu)$ is defined in the following way:

$$I(f;\lambda,\mu) = \int_a^b\!\!\!\!\!\!-\int_c^d \frac{f(x,y)}{(x-\lambda)(y-\mu)}\,dy\,dx\ ,\quad \lambda\in(a,b),\ \mu\in(c,d).$$

We now compute an enclosure of the nested integral

$$I(f;\lambda,\mu) = \int_1^2\!\!\!\!\!\!-\int_1^2 \frac{\sin(e^{x^2})\sin(e^{y^2})e^{x^2+y^2}}{(x-\lambda)(y-\mu)}\,dy\,dx$$

with $\lambda = 1.25$ and $\mu = 1.5$ using the CLAVIS library. The header file `"cubature.h"` belongs to the CLAVIS library. To be able to link the program `cubature.o` must be linked. The following program also demonstrates how exceptions may be handled.

```
#include <iostream>
#include "cubature.h" //don't forget to link cubature.o
//source code of this program is assumed to be in the clavis directory

using namespace std;
using namespace cxsc;

// cauchy x cauchy integral (using cauchy x cauchy formula)
//
// f(x,y) = sin(exp(y*y)) * exp(y*y) * sin(exp(x*x)) * exp(x*x)
```

```
//
// complete integrand of I(f; lambda, mu):  f(x,y) / ((x-lambda)*(y-mu))
//
// --------------------------------------------------------------------

int main() {

   try {
      operand r( exp(sqr(y)) ), s( exp(sqr(x)) );
      integrand f = sin(r) * r * sin(s) * s;

      double lambda=1.25;  //singularity in x direction
      double mu=1.5;       //singularity in y direction
      double xlb=1, xub=2; //x-range of integration
      double ylb=1, yub=2; //y-range of integration
      double eps= 1e-6;    //required accuracy

      cauchy_integral example(f, lambda, mu);
      //compute an enclosure of I(f; lambda, mu):
      example.integrate(xlb, xub, ylb, yub, eps);
      cout << SetPrecision(8,2) << Scientific
              << "Required max. diameter of remainder:  " << eps << endl
              << SetPrecision(16,12) << example << endl;
   }//try
   catch(integrand::error e)
   { cout << " formelgen. " << e.i << endl; }

   return 0;
}//main

/* Output:

Required max. diameter of remainder:  1e-06
 number of intervals : 109 (44)
 #f                  : 17233

 approximationsum    : [-7.6237054671070354E+001,-7.6237054670795458E+001]
 d(approximationsum) : 2.7489477361086756E-010

 remainder           : [-4.9415981455851922E-007,4.9416704156171493E-007]
 d(remainder)        : 9.8832685612023414E-007

 enclosure           : [-7.6237055165230175E+001,-7.6237054176628404E+001]
 d(enclosure)        : 9.8860176933612820E-007

*/
```

The output shows, that

$$\int_1^2 \int_1^2 \frac{\sin(e^{x^2})\sin(e^{y^2})e^{x^2+y^2}}{(x-1.25)(y-1.5)}\,dy\,dx \in [-76.2370552, -76.2370541].$$

This result is guaranteed by the algorithm itself.

### 5.5 Example: Time Measurements

We are frequently asked for timings. Here we give a frame for time measurements. The source code can be modified in an obvious way to do timings for other operations and functions.

```
//Simple frame for time measurements
```

```
#include <iostream>
#include <ctime>        //clock()
#include "interval.hpp" //interval operations
#include "imath.hpp"    //elementary functions for interval arguments

using namespace std;
using namespace cxsc;

void start_clock(clock_t& t1);  //function to start the timer
void print_time_used(clock_t t1);

int main()
{
   long iMax= 100000;
   cout << "Number of repetitions: "<< iMax << endl;
   interval x(200.0,200.001);
   clock_t t; //defined in <ctime>

   cout << "Elementary function calls ..." << endl;
   start_clock(t);
   for(long i=0; i<iMax;)
   {
      x= ln(exp(atan(sin(cos(x)))));
      i++; //avoid compiler optimization
   }
   print_time_used(t);
}

void start_clock(clock_t& t1)
{
   t1= clock();
   if (t1 == clock_t(-1))  //terminate if timer does not work properly
   {
      cerr << "Sorry, no clock\n";
      exit(1);
   }
}

void print_time_used(clock_t t1)
{
   clock_t t2= clock();
   if (t2 == clock_t(-1))
   {
     cerr<< "Sorry, clock overflow\n";
     exit(2);
   }
   cout << "Time used: " << 1000*double(t2-t1)/CLOCKS_PER_SEC
        << " msec" << endl;
}
/*
```

```
Results computed on a SUN Ultra 60 Workstation running Solaris 7
using GNU C++ Compiler Version 3.2 without any optimization:

Number of repetitions: 100000
Elementary function calls ...
Time used: 1370 msec
*/
```

Note that the given frame for time measurements is not so appropriate to measure very short or very long execution times. For further timing results we refer to [21].

## 6 Current Work on C-XSC

- Finish the final version C-XSC 2.0 (the actual version is: Betarelease 2 from December 2002)
- Modify the sources in such a way that C-XSC will run with more C++ compilers (e.g. with SUN Forte, Compaq, other compilers available for Windows systems; up to now C-XSC 2.0 only runs with GNU C++ compilers from version gcc 2.95.2 to version gcc 3.2.)
- Adaptation and completion of the C-XSC test suite to more C++ compiler versions (most C++ compiler do not conform completely to the C++ standard. This still causes problems when using the already existing rudimentary test suite. Meanwhile the installation of the C-XSC library is checked in the following way: Install also the numerical toolbox and look whether the toolbox programs deliver correct results. If the computed results are equal to the prestored correct values it is assumed that the C-XSC installation was successful.)
- Improve performance: due to the extensive use of the C++ exception handling, the extensive use of template classes, and the extensive use of function inlining it is (up to now) not possible to compile C-XSC with the GNU Compiler using e. g. the compiler option -O3 as optimization level
- For historical reasons C-XSC is build on emulations for several basic floating point operations. This makes the actual C-XSC run time system portable but slow compared to the speed of hardware operations. Nowadays most processors conform to the IEEE 754 standard. So, fast hardware operations are available for all rounding modes. These operations will be used in forthcoming C-XSC versions (at least for special processors like Intel and SUN)
- A thorough documentation of the routines available in C-XSC will be prepared. This is important because due to significant modifications concerning C++ most available documentation is no longer up to date
- Simplification and redesign of the runtime system (RTS). The RTS comprises rounding control, reliable input/output routines, routines to compute accurate dot products for data types real, complex, interval, and complex interval, ...
- Development and implementation of parallel versions of selfverifying solvers based on C-XSC and MPI on cluster computers

**Acknowledgements.** Many colleagues and scientists (see [4] Paragraph 1) have directly and indirectly contributed to the realization of C-XSC and C-XSC 2.0. The authors would like to thank each of them for his or her cooperation.

Thanks to the referees for valuable comments and suggestions.

## References

1. Cuyt, A.; Verdonk, B.; Becuwe, S; Kuterna, P.: A Remarkable Example of Catastrophic Cancellation Unraveled. Computing 66, 309-320 (2001).
2. Hammer, R.; Hocks, M.; Kulisch, U.; Ratz, D.: C++ Toolbox for Verified Computing. Basic Numerical Problems. Springer-Verlag, Berlin (1995).
3. Hölbig, C.; Krämer, W.: Selfverifying Solvers for Dense Systems of Linear Equations Realized in C-XSC. Preprint BUW-WRSWT 2003/1, Universität Wuppertal (2003).
4. Hofschuster, W.; Krämer, W.; Wedner, S.; Wiethoff, A.: C-XSC 2.0: A C++ Class Library for Extended Scientific Computing, Preprint BUGHW-WRSWT 2001/1, University of Wuppertal, pp. 1-24 (2001).
5. ISO/IEC 14882: Standard for the C++ Programming Language (1998).
6. Klatte, R.; Kulisch, U.; Lawo, C.; Rauch, M.; Wiethoff, A.: C-XSC – A C++ Class Library for Scientific Computing. Springer-Verlag, Berlin (1993).
7. Krämer, W.; Bantle, A.: Automatic Forward Error Analysis for Floating Point Algorithms. Reliable Computing, Vol. 7, No. 4, pp 321-340 (2001).
8. Krämer, W.; Wedner, S.: Two adaptive Gauss-Legendre type algorithms for the verified computation of definite integrals. Reliable Computing Vol. 2, No. 3, pp. 241-253 (1996).
9. Krämer, W.; Kulisch, U., Lohner, R.: Numerical Toolbox for Verified Computing II. Advanced Numerical Problems. Draft version available: `http://www.uni-karlsruhe.de/~Rudolf.Lohner/papers/tb2.ps.gz`.
10. Kulisch, U.: The Fifth Floating-Point Operation for Top-Performance Computers or Accumulation of Floating-Point Numbers and Products in Fixed-Point Arithmetic. Bericht 4/1997 des Forschungsschwerpunkts Computerarithmetik, Intervallrechnung und Numerische Algorithmen mit Ergebnisverifikation, Universität Karlsruhe (1997).
11. Kulisch, U.: Advanced Arithmetic for the Digital Computer. Design of Arithmetic Units. Springer Verlag, Wien (2002).
12. Loh, Eugene and Walster, G. William: Rump's Example Revisited. Reliable Computing, Vol. 8, No. 3, pp. 245-248 (2002).
13. Lohner, R.: Einschließung der Lösung gewöhnlicher Anfangs- und Randwertaufgaben und Anwendungen. Dissertation, Universität Karlsruhe (1988).
14. Meyers, Scott: Effective C++, 50 Specific Ways to Improve Your Programs and Designs. Addision-Wesley (1998).
15. Meyers, Scott: More Effective C++, 35 New Ways to Improve Your Programs and Designs. Addision-Wesley (1997).
16. Neher, M.: Validated bounds for Taylor coefficients of analytic functions. Reliable Computing 7, pp. 307-319 (2001).
17. Rump, S. M.: Algorithms for verified inclusions – theory and practice. In: Moore, R. E. (ed.): Reliability in Computing, pp. 109-126, Academic Press, New York (1988).

18. Rump, S. M.: INTLAB - INTerval LABoratory. In Tibor Csendes (editor): Developments in Reliable Computing, pages 77-104. Kluwer Academic Publishers, Dordrecht (1999).
19. Stroustrup, B.: The C++ Programming Language. Special Edition, Addison-Wesley, Reading, Mass. (2000).
20. Wedner, S.: Verifizierte Bestimmung singulärer Integrale - Quadratur und Kubatur. Thesis, Univ. Karlsruhe (2000).
21. Wolff von Gudenberg, J.: OOP and Interval Arithmetic – Language Support and Libraries, this volume, pp. 1-14 (2004).

# Libraries, Tools, and Interactive Systems for Verified Computations
# Four Case Studies

R. Baker Kearfott[1], Markus Neher[2], Shin'ichi Oishi[3], and Fabien Rico[4]

[1] University of Louisiana, Lafayette, Louisiana 70504-1010 USA,
rbk@louisiana.edu
http://interval.louisiana.edu/kearfott.html
[2] Universität Karlsruhe, D-76128 Karlsruhe, Germany,
markus.neher@math.uni-karlsruhe.de
http://www.uni-karlsruhe.de/~Markus.Neher/
[3] Waseda University, 169-8555 Shinjuku/Tokyo, Japan,
oishi@oishi.info.waseda.ac.jp
http://www.oishi.info.waseda.ac.jp/
[4] Université Paris VI, F-75015 Paris, France,
fabien.rico@lip6.fr

**Abstract.** As interval analysis-based reliable computations find wider application, more software is becoming available. Simultaneously, the applications for which this software is designed are becoming more diverse. Because of this, the software itself takes diverse forms, ranging from libraries for application development to fully interactive systems. The target applications range from fairly general to specialized.
Here, we describe the design of four freely available software systems providing validated computations. Oishi provides Slab, a complete, high-performance system for validated linear algebra whose user interface mimics both Matlab's M-files and a large subset of Matlab's command-line functions. In contrast, CADNA (Fabien Rico) is a C++ library designed to give developers of embedded systems access to validated numeric computations. Addressing global constrained optimization and validated solution of nonlinear algebraic systems, Kearfott's GlobSol focuses on providing the most practical such system possible without specifying non-general problem structure; Kearfott's system has a Fortran-90 interface. Finally, Neher provides a mathematically sound stand-alone package ACETAF with an intuitive graphical user interface for computing complex Taylor coefficients and their bounds, radii of convergence, etc.
Overviews of each package's capabilities, use, and instructions for obtaining and installing appear.

**Keywords:** Validated computations, numerical linear algebra, embedded systems, Taylor series, interval arithmetic, stochastic arithmetic, global optimization, interactive software systems, software libraries

R. Alt et al. (Eds.): Num. Software with Result Verification, LNCS 2991, pp. 36–63, 2004.

# 1 Introduction

This work describes four diverse but well-developed validated computing packages packages: Slab (Shin'ichi Oishi), CADNA (Fabien Rico), GlobSol (R. Baker Kearfott), and ACETAF (Markus Neher). Slab, based on Matlab syntax, provides, in validated and interval form, many of the matrix operations and functions available in Matlab; Slab implements a novel, well-thought out scheme of directed rounding to efficiently achieve this result. CADNA implements both interval arithmetic and a type of stochastic arithmetic. GlobSol, containing a traditional but portable implementation of interval arithmetic, is meant for validated solution of general unconstrained and constrained global optimization problems. Finally, ACETAF focuses on computation of error bounds for Taylor coefficients. Slab provides a user interface that is identical, with some purposeful exceptions, to the familiar Matlab syntax, while ACETAF provides a convenient graphical user interface. CADNA consists of a C++ library for programmers, while GlobSol, although containing Fortran 90 libraries that are separately usable, can be used as a stand-alone system in which the users input problems with standard Fortran syntax.

Details for Slab appear in §2 below, while details for CADNA appear in §3, details for GlobSol appear in §4, and details for ACETAF appear in §5. We give a short overall summary in §6.

# 2 Slab (Shin'ichi Oishi)

## 2.1 Introduction and Overview

S. Oishi and S. M. Rump have developed a new verification method called *rounding mode controlled verification*, and have applied this method to simultaneous linear equations [41]. It has been shown in [41] that the total cost of calculating an approximate solution of a system of $n$-dimensional simultaneous linear equations and of calculating a rigorous error bound is $4/3\ n^3$ flops. Let us consider a computing system which conforms to the IEEE 754 floating point standard. Let $A$ and $B$ be $n \times n$ matrices whose elements are IEEE 754 double precision numbers. Then, we have shown [41] that an inclusion of a product of $A$ and $B$ can be calculated by

```
setround(down);
L = A * B;
setround(up);
U = A * B;
```

Here, MATLAB-like notation is used, and the instructions `setround(down)` and `setround(up)` mean to change the IEEE 754 rounding mode to `-Inf` and `+Inf`, respectively. Since we can use the optimized BLAS functions in this calculation to calculate a matrix product `A*B`, in practice this inclusion procedure can be executed with just twice as much time as that for calculating `A*B` using the

optimized BLAS. This is the fact we used in [41] to develop our fast algorithm to include a solution of a system of $n$-dimensional simultaneous linear equations.

Then, in [40], we have shown that verified enclosure of all eigenvalues of matrices can be computed with less additional time than that required to initially compute all approximate eigenvalues and eigenvectors. The method proposed in [41] is also based on the rounding mode controlled verification method. Moreover, it has been shown in the book [39] that the rounding mode controlled verification method has wide applicability to a variety of problems of numerical linear algebra.

We developed Slab, a MATLAB-like numerical tool, as a test for these algorithms. We considered the suitability of several development environments for the design of Slab. In particular, to solve functional equations, one should have a tool having the following properties:

- Support for operator overloading (for programming clarity and convenience) to handle various objects, such as intervals and automatic-differentiation, needed for verification.
- Access to instructions for changing the rounding mode.
- Availability of optimized BLAS routines for solving large problems.

We have examined various numerical tools with regard to these criteria. MATLAB 6.x satisfies all the requirements listed above. In fact, Rump has implemented the MATLAB toolbox INTLAB (`http://www.ti3.tu-harburg.de/~rump/intlab/`), which has interval arithmetic, validated elementary functions, and rounding mode controlled computation. One minor defect when using MATLAB is that part of the source code is not open. However, it is known that MATLAB uses LAPACK with the optimized BLAS generated by ATLAS (an open-source project for Automatically Tuned Linear Algebra Software, see `http://math-atlas.sourceforge.net/`).

Scilab (`http://www-rocq.inria.fr/scilab/`) is an another choice. In Scilab versions 2.6 and earlier, Scilab uses mainly LINPACK. Thus the level three BLAS routines cannot be accelerated, even if one uses an optimized BLAS. However, from Scilab version 2.7, Scilab uses LAPACK. Thus, one can use the optimized BLAS generated by ATLAS. Moreover, Scilab provides the function of operator overloading through the t-list. The instruction of changing the rounding mode can be implemented in Scilab using its "link" and "call" functions of C object files. Thus, Scilab 2.7 satisfies all requirements mentioned above.

Octave (`http://www.octave.org/`) also is a candidate. Although it uses LAPACK almost optimally, Octave does not have operator overloading. The instruction of changing the rounding mode can be implemented through an octfile.

RLAB (`http://rlab.sourceforge.net/`) is also a good choice. It uses LAPACK. Its grammar is similar to that of C-language. It seems that RLAB has not yet implemented user defined instructions. However, one can introduce easily a rounding-mode-changing instruction by directly rewriting its source code to add such an instruction.

Based on these observations, we think that it is useful to introduce a new small language designed for verification. For these reasons, we have developed

Slab, a new MATLAB-like interpreter. Slab's grammar is mixture of MATLAB and RLAB. It uses LAPACK, so it can be accelerated by an optimized BLAS. Slab has a unique feature, a verification mode. Namely, based on a recent result of the author, it provides verified results for solutions of simultaneous linear equations, eigenvalue problems of matrices and many standard problems in numerical linear algebra. Slab's instructions and operator overloading function are implemented by directly rewriting its source code. Slab is free software based on the GNU-license, and is downloadable from the site

```
http://www.oishi.info.waseda.ac.jp/~oishi/index.html
```

Slab can be installed on a Redhat 7.2 based Linux PC with a Pentium CPU. Moreover, with a little modification, it can be installed on Windows using Cygwin or on a Macintosh with OS X.

### 2.2 Overview of Slab

In this section, we give an overview of Slab.

1. Slab is a MATLAB-like numerical tool designed for verified numerical computation.
2. Slab has many new features suitable for verified numerical computation. For example:
   a) Rounding instructions: up(), down(), and near() are defined for rounding toward infinity, toward -infinity and to nearest, respectively.
   b) There is a validation mode. To enter the validation mode, type "!".
   c) In validation mode, the solution of $Ax = b$ can be obtained with error bound by typing `x=A\b`.
3. Slab has the built-in functions:

   ```
   sin, cos, log10, ln (log), exp, abs (fabs),
   tan, atan, acos, asi sinh, tanh, sqrt
   ```

   In approximation mode, they are coincide with C's built-in functions. In verification mode, although they return values calculated by multiple precision routines, their return values are still not verified.
4. The user can define functions by

   ```
   function func_name(a,b,...,z) {expr;expr;...;expr},
   ```

   where the `expr` represent expressions.
5. The instructions `for`, `while` and `if` can be used.
6. Matrices can be treated.
7. The imaginary unit should be introduced with `i=sqrt(-1)`.
8. The interval $[a, b]$ can be entered with `interval(a,b)`.

We now explain several Slab instructions in more detail:

**function.** The instruction "function" makes a user defined function: As an example, we present a program here for inclusion of a solution to

$$Ax = b.$$

Here, $A$ is an $n \times n$ real matrix and $b$ is a real $n$-vector.

**Algorithm 1: Inclusion algorithm for matrix equations**

```
function f(A,b,n) {
    R=inv(A);
    x=R*b;
    down();
    U=R*A-eye(n);
    s=A*x-b;
    up();
    V=R*A-eye(n);
    t=A*x-b;
    up();
    r=int(s,t);
    T=int(U,V);
    d=abs(T);
    Ar=R*r;
    ar=abs(Ar);
    dd=norm(d);
    arr=norm(ar);
    e=arr/(1-dd);
 }
```

**eig.** The function eig(A) returns all the eigenvalues and eigenvectors of an $n$ by $n$ point matrix $A$:

```
A> A=rand(3);
A> sol=eig(A)
ans.val =
         | * * * |
         | * * * |
         | * * * |
ans.vec =
         | * * * |
         | * * * |
         | * * * |
```

In this example, sol.val gives a diagonal matrix whose diagonal elements consist of all eigenvalues of $A$. On the other hand, the $n$-th column of sol.vec is an eigenvector of $A$ corresponding to the $n$-th diagonal element of sol.val. This function uses CLAPACK functions with optimized BLAS functions:

– For a real symmetric $A$, `dsyev_` is used.

- For a real general $A$, `dgeev_` is used.
- For a Hermitian $A$, `zheev_` is used.
- For a general complex $A$, `zgeev_` is used.

**interval.** The `interval` instruction is used to make an interval. The object $a$ and $b$ can be doubles or matrices.

```
A> a=interval(3,5)
ans =
      [ 3 , 5 ]
A> A = rand(2);
A> Z = [A,A+0.1]
ans =
      | [0.3 , 0.4 ]  [ -0.1 ,   0 ] |
      | [0.2 , 0.3 ]  [  0.1 , 0.2 ] |
```

Addition, subtraction, multiplication and division are overloaded.

**read.** In Slab, files having a name like "filename.s" are called s-files. If Slab commands are written in s-files, then such an s-file can be read as

```
A> read filename.s
```

The following is an example:

```
shell> cat test.s
a = 3
b = 5
c = a + b
d = a * b;
shell> Slab
Welcome to Slab!
A> read test.s
ans =
      3
ans =
      5
ans =
      8
A> d
ans =
      15
```

**solve.** The function `solve(func,x)` is a one-dimensional nonlinear equation solver based on Newton's method. Here, `func` is the name of function defined by `func=name(f)`, where a user-defined function $f(x)$ is defined separately. One then types `solve(func,x)` to solve the nonlinear equation

$$f(x) = 0.$$

Here, $x$ is an initial guess of a solution. The following is an example:

```
A> # Since function 'solve' is defined in s-file,
A> # one should first read s-file 'fsolve.s' by
A> read fsolve.s
A> # Then, define a nonlinear function.
A> function f(x) {
A>     a_= sin(sin(x))-0.5;
A> }
A> # Then, solve f(x)=0.
A> x=[1];
A> a=name(f);
A> y=solve(a,x)
ans =
      0.55106958309945
```

**linpro.** The instruction `linpro(c,C,b)` is an interface to the GNU routine `lp_solve` and solves the following linear programming problem:

```
max: c'x;
subject to
     Cx <= b;
     x >= 0;
```

Here, $c$ is an $n$-dimensional objective vector, $C$ an $m \times n$ matrix, and $b$ is an $m$-dimensional right hand side vector. Here is an example:

```
A> c = [-1,2];
A> C = [2,1;-4,4];
A> b = [5,5];
A> linpro(c,C,b)
Value of objective function: 3.75
x0                    1.25
x1                    2.5
```

The function linpro is an interface to the GNU routine `lp_solve`.

In addition to the instructions listed above, the functions

```
chol, do-while, eval, fft, for, getbits, if, ifft,
inv, linspace, lread, lu, max, name, ode, plot,
print, qr, quad, save, schur, svd, while, who
```

and others are implemented in Slab.

Finally, we shall describe a bit about Slab's verification mode. Slab has three operation modes: help mode, approximation mode and verification mode. Slab's prompts `H>`, `A>` and `V>` are assigned for help mode, approximation mode and verification mode, respectively. In each mode, if we type `help`, `$` or `!`, then Slab's mode changes to help mode, approximation mode and verification mode, respectively.

The instructions `A\b` and `eig(A)` behave differently according to the mode. Here, $A$ is an $n \times n$ matrix and $b$ is an $n$-vector. In approximation mode, the instructions `A\b` and `eig(A)` have the same meaning as those in MATLAB. In verification mode, they also return error bounds, if possible.

## 3 CADNA (Fabien Rico)

FIXED CADNA[1] is a C++ library designed to give to developers tools for estimating the quality of numerical results of embedded codes. Embedded architectures are low cost solutions that can be found everywhere, such as in cars, planes or cellular phones. These architectures are generally based on a simple processor that performs computations with fixed point numbers. They have to manage increasingly complex programs. Because many embedded systems are critical, there is a growing need for numerical validation of the results produced by such systems.

The FIXED CADNA library has been developed to help the designer of embedded code to define the fixed point arithmetic that fits the problem to be solved. More precisely, Fixed CADNA helps the designer seek the optimum dynamical range of the variables of the program (from the memory size point of view), to seek the numerical instabilities of his algorithm and to validate the result produced. This library is composed of a set of classes which can be substituted for the float and double type, and a graphical user interface. Thus, the embedded program designer can run code at the same time on several fixed arithmetics. The core of FIXED CADNA is stochastic arithmetic and the CESTAC method, which has been successfully used in the CADNA library for validating floating point code [4].

Section 3.1 presents our models of fixed point, interval, and stochastic arithmetic. Section 3.2 describes the library and the facilities offered to the programmer. Our library produces a log file that shows the numerical instabilities produced by the embedded code. This log file format is shown in section 3.3. Finally, the graphical user interface is described in section 3.4.

### 3.1 Arithmetic Models

We characterize fixed point representations with three values that define the dynamical range and the precision:

- The precision $s \in \{0, 1, \ldots, 31\}$ represents the number of bits of the number.
- The position $p \in \{0, 1, \ldots, p\}$ indicates the number of digits after the point.
- The scale $e \in \mathbb{Z}$ is for scaled fixed point representations

[1] Acknowledgment: This section is the description of joint work from the ANP team of the LIP6 laboratory at Université Pierre et Marie Curie. Special thanks go to Jean-Marie Chesneaux and Laurent-Stéphane Didier with whom this project as been developed.

Next, each number in the fixed point representation is defined by two values:

- the sign $\varepsilon \in \{-1, 1\}$,
- the integer mantissa $m \in \{0, 1, \ldots, 2^s\}$.

Thus, the value of a number $X$ is given by the following formula:

$$X = \varepsilon \times m \times 2^{e-p}. \tag{1}$$

This formula is similar to the formula that gives the value of floating point numbers, but in equation (1), the exponent $e - p$ is fixed.

Building on the fixed point representation, the FIXED CADNA library gives two additional representations:

- the interval fixed point representation, consisting of an interval composed of two fixed point numbers. It is adapted from classical interval arithmetic [27].
- the stochastic fixed point representation, using the CESTAC method for estimating the accuracy of a number.

The aim of the CESTAC [48,49] method, based on the probabilistic approach to round-off errors, is to estimate the effect of propagation of round-off errors on every computed result obtained with a finite arithmetic. It consists of making the round-off errors propagate in different ways to distinguish between a stable part of the mantissa[2], considered the significant part, and an unstable part[3], considered non-significant.

The first basic idea of the CESTAC method is to replace the usual finite arithmetic by a random arithmetic. The random arithmetic is obtained from the usual finite arithmetic by randomly perturbing the lowest-weight bit of the mantissa of the result of each arithmetic operation. The second basic idea is to run a code several times with this new arithmetic to obtain different results for each run.

In practice, the use of the CESTAC method consists of:

1. running the same program $N$ times in parallel with the random arithmetic; consequently, for each intermediate result $R$ of any finite arithmetic operation, a set of $N$ different computed results $R_i$, $i = 1, .., N$ is obtained,
2. taking the mean value $\overline{R} = \frac{\sum_{i=1}^{N} R_i}{N}$ as the computed result,
3. using Student's distribution to estimate a confidence interval for $R$, and then computing the number $C_{\overline{R}}$ of significant bits of $\overline{R}$ (i.e. the common bits between $\overline{R}$ and the exact result $r$) defined by

$$C_{\overline{R}} = \log_2 \left( \frac{\sqrt{N}.\left|\overline{R}\right|}{\tau_\beta.s} \right) \quad \text{with} \quad s = \sqrt{\frac{\sum_{i=1}^{N} \left(R_i - \overline{R}\right)^2}{N-1}},$$

$\tau_\beta$ is the value of Student's variable $t$ for $N-1$ degrees of freedom and probability $\beta$. The major interest of this method comes from the small available

[2] the part of result that doesn't change with different propagations
[3] the part of the result that depends on the round-off error

values of $N$. In practice, $N = 2$ or 3 are sufficient to obtain an accurate enough confidence interval.

The validity of this method has been proved under hypotheses which generally hold in real-life problems [3]. The hypotheses can be controlled during the run.

The primary application of the CESTAC method is to compute the number of exact significant bits of computed results, but the capability of knowing the accuracy of results leads to a new arithmetic: *stochastic arithmetic* [5,6, 49]. Stochastic arithmetic may also be seen as a model of a finite arithmetic with accuracy control. In stochastic arithmetic, order relations and the notion of equality are redefined to take into account the accuracy of operands.

For instance, two values will be stochastically equal if their difference is only due to round-off error propagation. For the order relation, a value will be strictly greater than another value if it is *significantly* greater than the other. On the other hand, a value will be greater or equal to another value if it is greater than the other or if their difference is only due to round-off error propagation.

*Discrete Stochastic Arithmetic* (DSA) is the joint use on a computer of the synchronous implementation of the CESTAC method and the stochastic definitions of order and equality relations. DSA enables one to estimate the impact of round-off errors on any result of a scientific code and also to check that no numerical instability occurred during the run, especially in branching statements. Moreover, the ability to estimate the numerical quality of any intermediate result leads to a true dynamical numerical debugging by detecting all numerical instabilities while running the code.

### 3.2 Using the Library

The goal of this library is to allow the developer to execute existing `C` code with new types without completely rewriting it. A simple mechanism for easily substituting types is to consider these types as objects having the same interface. Thus, our `C++` library is composed of a set of classes defining new types that can be substituted for `float` and `double` `C` types.

In practice, a generic type `REAL` is used for every variable whose type is changed. Next, it is necessary to include the header file corresponding to the chosen representation and compile the code. This inclusion associates the chosen fixed number representation to the generic type `REAL`. All computations on the `REAL` variables are performed with the selected representation.

Because all the types defined in our library are parameterized by the size `s`, the number of digits in the fractional part `p`, and the scale `e`, each variable with the generic type `REAL` must be declared with at least these parameters. This declaration constructs an object that has the properties of the chosen representation. Moreover, the `REAL` constructor may take a value `v` as an extra parameter, that is a `double` number with which the variable is initialized.

The arithmetic operations are allowed only between numbers in the same representation. This means that operations between two numbers expressed in

**Table 1.** Summary of the C++ library

| Type | Fixed | Interval Fixed | Stochastic Fixed |
|---|---|---|---|
| **Header file** | `fixed.h` | `interval.h` | `cadna_fixed.h` |
| **Defined type** | `REAL` | | |
| **Initialization** | `REAL::init(int nb, char *log_file)` | `REAL::init(int nb, int div, int test, int mul, int lost int threshold, char *log_file)` | |
| **Constructor** | `REAL(int s, int p, int e)` `REAL(int s, int p, int e, double v)` | | |
| **Operators** | `=, +=, -=, *=, /=, +, ++, --,*,/, ==, !=, <, >, <=, >=, <<` | | |

a different fixed representation are not permitted. An explicit conversion has to be made by the developer by reallocating the value with the operator `=`. Thus, precision losses due to implicit conversion are avoided. Moreover, to manipulate constant values, computations with `double` are allowed.

At execution time, a log file containing all the numerical instabilities (see section 3.3) detected by our library is produced. The programmer can specify the name of this log file and the maximum number of instabilities that will be detected and noted. Furthermore, it is possible to select the kind of instabilities to be logged for the stochastic fixed and interval fixed representations. Thus the library is initialized by the function `REAL::init`. A full description of this function is developed in section 3.3.

Table 1 summarizes the different functionalities of the new representation available in Fixed CADNA.

The example in Table 2 illustrates the use of our library on a FIR filter `C` code. In this code, the FIR filter computations are performed with a fixed point representation that is parameterized by the user. It has `size` bits and has `nb` bits in the fractional part. Including the file `fixed.h` permits the generic type `REAL` to represent fixed point numbers. The log file is initialized with `REAL::init(-1)`. It will contain all the instabilities detected and will be named `instability_fixed.log`. Note that the declaration of single variables and arrays is different. For instance, a simple variable `output` is declared with `REAL output(size,nb,0)`, and the array `h` is declared with `REAL *h=new REAL[NTAPS](size,nb,0)`.

### 3.3 The Log File

Using a finite arithmetic may lead to several instabilities. Our library has been designed to detect the following problems:

- Overflow,
- Division by a stochastic zero (only for stochastic representation),
- Division by an interval containing zero (only for interval representation),

**Table 2.** FIR filter example

```
#include <iostream>
#include "fixed.h"

void clear(int ntaps, REAL z[])
{
    int ii;
    for (ii = 0; ii < ntaps; ii++) {
        z[ii] = 0;
    }
}

REAL fir_basic(REAL input, int ntaps,
     const REAL h[], REAL z[])
{
  int ii;
  REAL accum(input);

  z[0] = input;
  accum = 0;
  for (ii = 0; ii < ntaps; ii++){
      accum += h[ii] * z[ii];
  }
  for (ii = ntaps - 2; ii >= 0; ii--){
      z[ii + 1] = z[ii];
  }
  return accum;
}

int main(int argc, char *argv[])
{
  #define NTAPS 6
  #define IMP_SIZE (3 * NTAPS)
```

```
static const double h_base[NTAPS] =
     { 1.1, 2.2, 3.3, 4.4, 5.5, 6.6 };
int size, nb;

REAL::init(-1);

size=atoi(argv[1]);
nb=atoi(argv[2]);
static REAL *h=new
            REAL[NTAPS](size,nb,0);
static REAL *h2=new
            REAL[2 * NTAPS](size,nb,0);
static REAL *imp=new
            REAL[IMP_SIZE](size,nb,0);
REAL output(size,nb,0);
int ii, state, i;

clear(IMP_SIZE, imp);
imp[5] = 1.0;
imp[6] = 1.5001;
imp[7] = 1.2;
for (ii = 0; ii < NTAPS; ii++){
   h2[ii] = h2[ii + NTAPS] = h[ii];
}

clear(NTAPS, z);
for (ii = 0; ii < IMP_SIZE; ii++){
   output = fir_basic(imp[ii],
                      NTAPS, h, z);
   cout << output << " ";
}
}
```

- Insignificant comparison of two stochastic numbers too close according to the accuracy (only for stochastic representation).
- Comparison between overlapping intervals (only for interval representation),
- Multiplication of two stochastic zero numbers,

Our choice is to never stop the computation and just log the instabilities into a file created by the init function. The CESTAC method detects cancellation that may occur in an addition or a subtraction. This information is logged in the same file. (See section 3.2.) The init function permits specification of the log file name and choosing which inconsistencies will be logged:

**static void** fixed_st::init(**int** my_trace_cadna = 10000,
**int** div = TRUE,
**int** test = TRUE,
**int** mul = TRUE,
**int** lost = TRUE,
**int** threshold = 4,
**char** *log_file = `"instability_fixed_st.log"`);

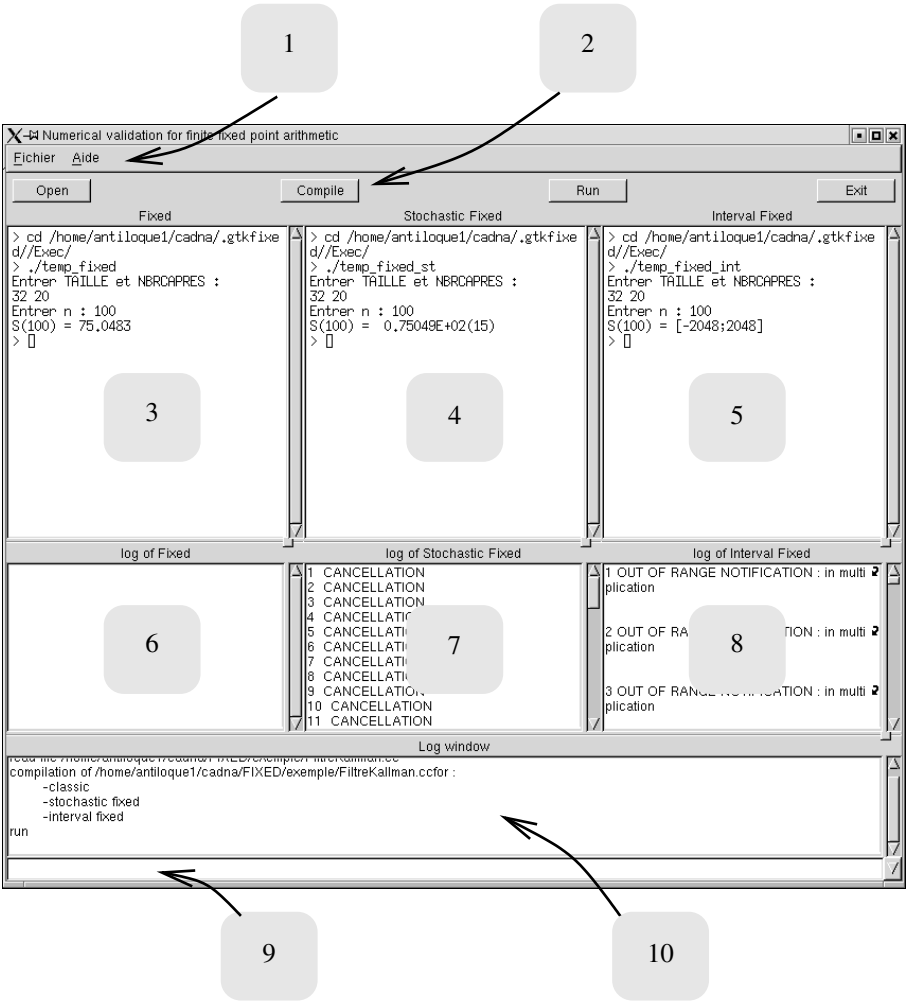



**Fig. 1.** Snapshot of the GUI for FIXED CADNA

For each type of arithmetic used, it is possible to select what is logged:

- stochastic fixed: the arguments `div`, `test`, `mul`, and `lost` are boolean, activating the log for division by stochastic zero, inconsistent comparison, multiplication of stochastic zero, and cancellation, respectively. The integer `threshold` is the threshold used for detecting cancellation, and `my_trace_cadna` is the maximum number of messages displayed. (If `my_trace_cadna`$= -1$, then all messages are displayed.)
- interval fixed: `mul`, `lost`, and `threshold` are ignored. They are only present to obtain the same interface.
- initial fixed: only the first and the second arguments are used.

### 3.4 The GUI for Parallel Execution

When using the library, it is necessary to choose a special arithmetic (stochastic or interval arithmetic). However, developers may need to test a program with all the arithmetics to take advantage of each of them. For this purpose, it is interesting to run the same program with different arithmetics in parallel. The aim of the Graphical User Interface is to do it easily.

The main window (see Figure 1) is split into three execution windows (1.3, 1.4, 1.5). Each of these presents the program result with a different representation. Below them, there are the log windows (1.6, 1.7, 1.8) presenting the content of the corresponding log file. (See section 3.3.) Those execution and log windows may be deleted using the `Property` entry of the menu 1.1.

A program is chosen through the menu `File>Open` or the button `Open` 1.2. The `Compile` and `Run` commands enable execution of the program with the different representations. If the running program needs keyboard input, the entry 1.9 enables one to dispatch it on a different process. Finally, the principal log window 1.10 keeps a trace of the former commands.

### 3.5 Summary

In this section, we have introduced new method to perform validated numerical calculations for embedded applications.

Numerical validation tools have existed before, but none of those are specifically designed for embedded applications, because they lack support for fixed point representation. Our library tries to fill this gap. It is based on the use of a new library that applies various known validation methods to fixed point numbers.

This library is just the first piece of work towards a complete toolbox dedicated to numerical validation of embedded applications.

## 4 GlobSol (R. Baker Kearfott)

### 4.1 Introduction

GlobSol began as a research code to study algorithms for verified Global Optimization. GlobSol grew out of `INTBIS` [23], a relatively simple FORTRAN-77 code and *ACM Transactions on Mathematical Software* algorithm for finding all solutions, with validation, to nonlinear algebraic systems. For ease of experimentation, simple automatic differentiation, consistent with the relatively small problems originally envisioned, was added, and a special technique for bound constraints (originally tried in [16]) was implemented. We also provided extensive capability for a technique we described in [15], a technique (discovered independently and probably earlier by others) that has developed into the field of "constraint propagation." One of the first projects done within this environment was development of techniques for avoiding the "cluster" problem ([7], [22], [46]) that occurs in exhaustive search algorithms when the system

is ill-conditioned or singular near the global optimum. We also implemented a technique for verifying feasible points [19] and thus included a capability for handling general equality-constrained problems. (We added separate handling of inequality-constrained problems later.) We studied and implemented extensions to the idea of interval slopes and slope arithmetic (perhaps first appearing [25]) to non-smooth functions, as we explained in [17] and [18, Ch. 6].

During this development (roughly from 1993 to 1998), we referred to GlobSol as `INTOPT-90`. A collected review of these and other techniques, some new theoretical analyses, and a description of the structure of `INTOPT-90` appears in [18].

GlobSol took on its present form (and its present name) as part of a cooperative research and development contract funded by Sun Microsystems and directed by G. W. Walster (and with extensive participation of George Corliss). The most significant advances during this phase of GlobSol's development are perhaps

- extensive testing and bug-removal (extremely important for software that purports to validate),
- polishing of the user interface,
- experimentation with GlobSol on a variety of practical problems, and
- polishing of the packaging, distribution, and installation process.

Although at first glance these advances may seem mundane, they are both a significant part of the total effort and absolutely indispensable for widely-used, lasting software.

We have recently provided some details of the above in the succinct review [20]. Here, we very briefly review requirements for installation and use of GlobSol, then focus on present weaknesses in GlobSol and how we are eliminating these weaknesses.

### 4.2 Statement of the Problem GlobSol Treats

For reference below, we now formally state the type of problem GlobSol solves. The general optimization problem is

$$\boxed{\begin{array}{l}\text{minimize } \phi(x) \\ \text{subject to } c_i(x) = 0,\ i = 1, \ldots, m_1, \\ \qquad\qquad g_i(x) \le 0,\ i = 1, \ldots, m_2, \\ \text{where } \phi : \mathbb{R}^n \to \mathbb{R} \text{ and } c_i, g_i : \mathbb{R}^n \to \mathbb{R}.\end{array}} \tag{2}$$

The sense in which GlobSol will solve problem (2) is

$$\boxed{\begin{array}{l}\text{Given a box } \boldsymbol{x} = ([\underline{x}_1, \overline{x}_1], \ldots [\underline{x}_n, \overline{x}_n]), \text{ find small boxes} \\ \boldsymbol{x}^* = ([\underline{x}^*_1, \overline{x}^*_1], \ldots [\underline{x}^*_n, \overline{x}_n]^*) \text{ such that any solutions of} \\ \text{minimize } \phi(x) \\ \text{subject to } c_i(x) = 0,\ i = 1, \ldots, m_1, \\ \qquad\qquad g_i(x) \le 0,\ i = 1, \ldots, m_2, \\ \text{where } \phi : \mathbb{R}^n \to \mathbb{R} \text{ and } c_i, g_i : \mathbb{R}^n \to \mathbb{R} \\ \text{are guaranteed to be within one of the } \boldsymbol{x}^* \text{ that has been found.}\end{array}} \tag{3}$$

### 4.3 Installation and Use of GlobSol

The main requirements for GlobSol are

1. a standard-conforming Fortran 90 or Fortran 95 compiler, and
2. a "make" utility.

A Fortran compiler is required because the user defines the optimization problem as a Fortran program. Even though GlobSol is compiled and linked only once (and the user's program is compiled and linked separately), the same version of the same compiler must nonetheless be used for both building GlobSol and compiling the user's input.

GlobSol can be obtained as a "zip" file from
`http://interval.louisiana.edu/GlobSol/download_globsol.html`
From there, one downloads a compressed file and an "unpack" script appropriate to the particular operating system and compiler. The scripts are for compilers on various Unix/Linux and Microsoft systems. However, the makefile that builds GlobSol has extensive in-line documentation, and can be changed as appropriate for new compilers and systems.

Succinct instructions for installing GlobSol appear in
`http://interval.louisiana.edu/GlobSol/install.html`.

GlobSol has extensive configuration options, accessible by editing a configuration file. GlobSol is run by supplying a command-line script. A simple example is accessible by following the installation instructions. For more details, see [20], or examine the various preprints related to GlobSol at
`http://interval.louisiana.edu/preprints.html`.

### 4.4 Improvements to GlobSol in Progress

GlobSol works relatively well for unconstrained problems, but performs weakly when there are many equality constraints. There are several reasons for this. We give these reasons, along with present work to overcome these problems, in the following paragraphs.

**Obtaining Upper Bounds on the Global Optimum.** First, GlobSol is weak at finding an upper bound on the global optimum, when constrained optimization is used. For unconstrained optimization, GlobSol (and other interval branch and bound algorithms) can obtain an upper bound on the global optimum by evaluating the objective function at any point $\check{x}$ (and using outwardly rounded interval arithmetic in the evaluation, for mathematical rigor); the closer $\check{x}$ is to an actual global optimizing point $x^*$, the sharper the upper bound on the global optimum. For constrained problems, there is a complication as outlined in [18, §5.2.4]: the interval evaluation needs to be taken over a small box in which a feasible point has been proven to lie. However, the same principle holds for constrained problems.

For unconstrained problems, GlobSol uses a simple steepest descent procedure followed by the MINPACK-1 routine `HYBRJ1` [33] to find a critical point of the Fritz–John equations, to increase the chances that $\check{x}$ is near a global optimizer. The MINPACK routines are freely available through NETLIB (`http://www.netlib.org/`), and can thus be distributed with GlobSol. In contrast, until recently, good routines that find approximations $\check{x}$ to local optimizers of constrained problems have been proprietary, and cannot be distributed with GlobSol. Since GlobSol is meant to be self-contained, we have instead provided our own routine that employs a generalized-inverse-based Newton method to project onto the feasible set [21]. As a consequence, in the constrained case, GlobSol finds rigorous upper bounds for the global optimizer, but may not find a reasonably sharp upper bound until late in the search process. For some applications, this is not a problem, but it can have a disastrous effect on efficiency in others.

Recently, Wächter's quality Fortran code Ipopt for constrained optimization (see `http://www-124.ibm.com/developerworks/opensource/coin/` and [50]) has become available under the Common Public License. (See `http://www.opensource.org/licenses/cpl.php`.) This code should provide approximate feasible points $\check{x}$ that are highly likely to be near global optimizers, thus enabling GlobSol to compute sharp upper bounds on global optimizers in the constrained case. We have recently interfaced Ipopt with GlobSol, and we are formulating experiments to analyze performance improvements.

**Obtaining Lower Bounds on the Range over Large Regions.** A good upper bound on the global optimum is generally combined in global search algorithms with good lower bounds on the range of the objective function over subregions $\boldsymbol{x}$ of the search space. If the lower bound of the objective over $\boldsymbol{x}$ is larger than the upper bound on the global optimum, then the subregion $\boldsymbol{x}$ can be rejected as not containing any global optima. In principle, a simple interval evaluation (occasionally replaced by a mean value extension) of the objective over $\boldsymbol{x}$ provides the required lower bound. Such a simple interval evaluation is what is currently implemented in GlobSol.

However, since such an evaluation does not take account of the constraints, it can have an enormous overestimation. As an example, consider the nonlinear minimax problem:

$$\min_x \max_{1\le i\le m} |f_i(x)|, \quad f_i : \mathbb{R}^n \to \mathbb{R}, \quad x \in \mathbb{R}^n, \quad m \ge n. \tag{4}$$

To date, we have had limited success in solving realistic problems of this type directly using GlobSol's non-smooth slope extensions. Alternately, we can convert the problem to a to a smooth problem with Lemaréchal's technique [29] as follows:

$$\begin{aligned} &\min_{x\in\mathbb{R}^n} v \\ &\text{such that } \left\{ \begin{array}{r} f_i(x) \le v \\ -f_i(x) \le v \end{array} \right\}, \quad 1 \le i \le m. \end{aligned} \tag{5}$$

In (5), we have introduced a single additional slack variable $v$, which becomes the value of the objective function. If $v$ is treated as an arbitrary additional independent variable, then GlobSol presently employs constraint propagation to narrow the range of $v$ when a subset of the region for the variables $x$ is given. However, this process does not take account of the coupling between the constraints, and has not enabled GlobSol to solve minimax problems efficiently. Furthermore, interval Newton methods applied to the Lagrange multiplier (or Fritz–John) system associated with (5) over large regions have not adequately accelerated the search process within GlobSol for realistic minimax problems.

In contrast, Floudas [10], Sahinidis [47] and their respective groups have used convex or linear relaxations of problem (2) to significant advantage in non-verified global optimization software. For example, to obtain a lower bound on an objective function over a region $\boldsymbol{x}$, the objective $\phi$ in problem (2) is replaced by a convex (or linear) objective that is known to be less than or equal to the actual objective over $\boldsymbol{x}$. Each left member $g_i$ of the inequality constraints is similarly replaced by a convex (or linear) underestimator. Likewise, each equality constraint $c_i(x) = 0$ is replaced by the two inequality constraints $c_i(x) \leq 0$ and $-c_i(x) \leq 0$, and then underestimated. The optimum of the resulting convex (or linear) program then is less than or equal to the global optimum of the original problem (2).

Experimenting with Sahinidis' BARON [44] software, we have been able to successfully find global optima of minimax problems of the form (4). Apparently, the reasons these techniques are successful where the others are not are because

1. they take account of the coupling between the constraints, and
2. the resulting relaxations (i.e. the derived simpler problems) have solutions, and these solutions are easy to obtain.

The computations with convex linear underestimators can be made rigorous with the following procedure:

1. Compute a relaxed (simplified) convex or linear problem over $\boldsymbol{x}$.
2. Compute the solution to the convex or linear problem with a floating-point solver that gives an approximate solution $\check{x}$.
3. Use $\check{x}$ with the validation technique in [13] to provide a rigorous lower bound on the solution to the relaxed (and hence on the solution to the original) problem over $\boldsymbol{x}$.

We are presently experimenting with this procedure, and will eventually incorporate it into GlobSol for minimax procedures.

Although each group develops different techniques, both Floudas [10] and Sahinidis [47] develop methods by which these underestimators can be computed automatically (with automatic-differentiation-like technology; see [10] and [47]); such techniques (and others) could eventually be incorporated into GlobSol.

**Efficiency of GlobSol's Automatic Differentiation, List Processing, etc.** As outlined in [18, §1.4 and §2.2], GlobSol interprets an internal representation of

the objective and constraints, termed a "code list", to compute point and interval values of the objective, constraint residuals, Jacobi and Hessian matrices, etc. This internal representation was designed with simplicity in mind, under the assumption that problems GlobSol would solve are relatively small and would not be limited by inefficiencies in function evaluation. However, for a number of problems, evaluation of the code list could speed computation.

Experiments by Corliss et al. under the Sun project have indicated that, for some problems, converting the code list to Fortran code then compiling it gave a noticeable performance improvement, but did not make a difference in the practicality of solving particular problems. On the other hand, operations for evaluating every constraint and the objective are included in a single code list, and all of these operations are performed whenever a particular objective or constraint value is needed at a new point (or interval) of evaluation. Separating the operations could benefit particular problems.

Another area of possible efficiency gains in GlobSol is in its list processing. In the global search, regions $\boldsymbol{x}$ are repeatedly bisected into $\boldsymbol{x}^{(1)}$ and $\boldsymbol{x}^{(2)}$; $\boldsymbol{x}^{(1)}$ is processed further, while $\boldsymbol{x}^{(2)}$ is stored in a linked list structure. Memory is allocated whenever a box is stored on the list, and is freed whenever a box is removed. For some problems, a more sophisticated allocation / deallocation scheme would greatly improve performance.

Although, with time, we intend to implement these GlobSol improvements, we do not place them at as high a priority as algorithmic improvements, such as use of convex underestimators. In our view, fundamental algorithmic improvements will advance both the practicality of GlobSol and the fundamental state of the art in verified global optimization more.

### 4.5 Simplification of GlobSol

At present, there are many optional algorithm paths in GlobSol, some of which are not used. This is a result of the original research nature of GlobSol. Eventually, some of these paths (along with supporting code) can be eliminated.

Other improvements in this general category include updating GlobSol's installation scripts.

### 4.6 Summary

In this section, we have described GlobSol, validated global optimization software for Fortran. GlobSol represents a little over a decade of work on algorithms and implementations. GlobSol is unusual among such packages in being openly available and self-contained. Although GlobSol has weaknesses for certain kinds of constrained problems, we are excited about alternate algorithms, as yet untried in a validated context, that promise to remove many of these weaknesses.

# 5 ACETAF (Markus Neher)

## 5.1 Introduction

The software package ACETAF has been developed by Ingo Eble and Markus Neher. It is a C++ program for the accurate computation of error bounds for Taylor coefficients of analytic functions. ACETAF originated from a subroutine in a program for the validated solutions of ODEs [34] and has evolved over three years to its present state, which includes additional features besides the computation of bounds for Taylor coefficients. For a user-defined complex function $f$, the following problems are solved with ACETAF. (We list the problems in the order in which they rely on each other).

- Rigorous computation of leading Taylor coefficients.
- Check of analyticity in a user-defined disc.
- Rigorous computation of bounds for Taylor coefficients with arbitrary order.
- Rigorous computation of bounds for Taylor remainder series.

In section 5.2, we report on the scope of ACETAF and the mathematical backgrounds for the problems solved by ACETAF. Section 5.3 deals with the availability of ACETAF. In the last section, we present a numerical example.

## 5.2 Range of Use of ACETAF

**Admissible Functions.** For all features of the program, the user may enter an expression for a function $f$ that must belong to the following set of admissible functions:

- Polynomials and rational functions,
- the exponential function, the sine, the cosine and the tangent function,
- the principal branch of the logarithm,
- the principal branches of the square root and of other roots with rational or (floating-point) real exponents,
- the principal branches of the inverse trigonometric functions, and
- finite compositions of these functions, such as $\exp(z^2)$ or $\tanh(\ln(z^2+1)/3)$.

Loops and branches are not allowed in the expression for $f$. For roots, logarithms, or inverse functions, principal branches are always assumed by the program. For example, $\ln z$ is interpreted as the principal branch $\ln|z| + \imath \operatorname{Arg} z$ of the logarithm (with $\operatorname{Arg} z \in (-\pi, \pi)$). As a special consequence, $\ln z$ is not defined if $z$ is a negative real number.

Furthermore, the underlying mathematical theory of the algorithms in ACETAF requires that $f$ be analytic in a user-defined disc in the complex plane. On request of the user, the program checks whether the user-defined function $f$ is analytic on the given disc.

**Rigorous Computation of Values and Ranges of Functions.** The algorithms that are employed in ACETAF rely on function values and ranges of functions. The validated determination of these is accomplished with interval computations [1,14,32,38]. Floating-point interval arithmetic [26,28] is used in the practical calculations to handle all roundoff errors. We assume that the reader is familiar with interval computations. We only introduce some notation, and we recall the definition of an inclusion function.

The range of a function $f$ on a domain $D$ is denoted by $f(D)$, i.e. $f(D) := \{f(z) \mid z \in D\}$. An *inclusion function* $F$ of a given function $f$ on $D \subseteq \mathbb{C}$ is an interval function (an expression that can be evaluated according to the rules of interval arithmetic) that encloses the range of $f$ on all intervals $\boldsymbol{z} \subseteq D$:

$$F(\boldsymbol{z}) \supseteq f(\boldsymbol{z}) \quad \text{for all} \quad \boldsymbol{z} \subseteq D.$$

Real and complex floating-point interval arithmetic has been implemented in a number of programming languages and libraries, such as C-XSC, [24], filib++ [30,31], or INTLAB [43]. ACETAF runs with both the C-XSC or the filib++ interval library. Since neither of these libraries includes routines for complex standard functions, the complex standard functions library CoStLy [9] has also been included in ACETAF.

**Computation of Complex Taylor Coefficients.** ACETAF offers the computation of some leading Taylor coefficients of a user-defined function. These Taylor coefficients are computed via automatic differentiation [12,42].

The complex Taylor arithmetic is based on a well-known property of admissible functions: their real and imaginary parts can be expressed as compositions of real standard functions. For example, if $z = x + \imath y$, then $e^z = e^x \cos y + \imath e^x \cos z$. Braune and Krämer [2] used such decompositions for constructing inclusion functions; in ACETAF they are used for computing complex derivatives from real derivatives.

In general, if $f(z) = u(x, y) + \imath v(x, y)$ then we have $f'(z) = u_x(x, y) + \imath v_x(x, y)$. Similarly, specific Taylor coefficients of $f$ are calculated by applying the well known formulas of automatic differentiation to the real and the imaginary parts of $f$, respectively.

**Check of Analyticity.** The error bounds on the Taylor coefficients that will be presented in the next subsection require that $f$ be analytic on the disc $B$. Multi-valued analytic standard functions are all interpreted as being principal values with strict domain restrictions.

To detect violations of the analyticity of a user-defined function $f$ on a given disc, the analyticity of $f$ can be checked before computation of the bounds. If the proof of analyticity fails on the user-defined disc, then ACETAF computes a validated lower bound of the maximum radius to the given midpoint, such that $f$ is analytic on the full disc. This is done by a heuristic algorithm which uses bisection of the radius of the given disc.

Because the regions of analyticity are hard to detect for composite functions, the analyticity check is always recommended before the computation of the bounds for the Taylor coefficients.

**Bounds for Taylor Coefficients with Arbitrary Order.** The rigorous computation of bounds for Taylor coefficients with arbitrary order is the main feature of ACETAF. Such bounds are used for error analyses in numerical computations. For example, they are used in the well-known Taylor series method for the solution of ODEs [35]. Geometric series bounds for Taylor coefficients of analytic functions are also used for finding multiple zeros or clusters of zeros. In [45], the availability of such bounds is assumed, but no method for their computation is mentioned.

In ACETAF, four methods for calculating such bounds are implemented. Method I is Cauchy's estimate(6). For a function

$$f(z) = \sum_{j=0}^{\infty} a_j z^j, \quad |z| \leq r$$

that is analytic on a disc $B := \{z : |z| < r\}$ with positive radius $r$ and bounded on the circle $C := \{z : |z| = r\}$, it holds that

$$|a_j| \leq \frac{M(r)}{r^j}, \quad j \in \mathbb{N}_0, \tag{6}$$

where $M(r) := \max_{|z|=r} |f(z)|$.

The calculation of $M(r)$ poses a simple global optimization problem (cf. section 4 of this paper). In ACETAF, the following branch and bound algorithm is employed to compute a validated upper bound for Cauchy's estimate for an analytic function $f$ and a given circle $C$ with radius $r$:

1. For some $k_{\max} \in \mathbb{N}$, $C$ is split into segments $S_k$, $k = 1, \ldots, k_{\max}$, which are gathered in a list $L$.
2. Each segment is covered by a rectangular complex interval $\boldsymbol{z_k}$. With an inclusion function $F$ of $f$, a set $\boldsymbol{w} = [\underline{w_k}, \overline{w_k}] \supseteq |f|(\boldsymbol{z_k})$ is computed.
3. $\overline{M} := \max_k \overline{w_k}$ and $\underline{M} := \max_k \underline{w_k}$ are guaranteed upper and lower bounds for $M(r)$, respectively.
4. If $\overline{M} - \underline{M}$ is sufficiently small, then the algorithm is terminated. Otherwise, elements $S_k$ that cannot contain a maximum of $|f|$ are eliminated from the list $L$. The remaining segments are bisected and gathered in a new list $L$. The algorithm is then continued with step 2.

The three other methods that are implemented in ACETAF are variants of Cauchy's estimate, which have been developed in [36]. In method II, Cauchy's estimate is applied to the defect of some Taylor polynomial approximation of $f$; in method III, Cauchy's estimate is applied to some derivative of $f$. The most

general method IV is a generalization of the other three methods. Instead of $M(r)$, the number

$$V(r,m,l) := \max_{|z|=r} |f^{(m)}(z) - s_l(z)|$$

is used in the estimation of the Taylor coefficients of $f$, where $m$ and $l$ are integers, $f^{(m)}$ is the $m$th derivative of $f$, and $s_l$ is the $l$th Taylor polynomial (expanded at the origin) of $f^{(m)}$. Instead of (6), we obtain [36,37]

$$|a_j| \;\leq\; \frac{(j-m)!\,V(r,m,l)}{j!\,r^{j-m}} \quad \text{for } j > m+l. \tag{7}$$

Letting $s_{-1} = 0$, the four methods correspond to the following choices of $m$ and $l$:

- Method I (Cauchy's estimate) is obtained for $m = 0$, $l = -1$,
- method II consists of the choice $m = 0$, $l \geq 0$,
- method III consists of the choice $m > 0$, $l = -1$,
- method IV uses $m > 0$, $l \geq 0$.

For $m > 0$ in (7), the remainder series of $f$ is bounded by a series that converges faster than any geometric series, for all $z \in B$. Thus, the estimate (7) is a considerable improvement over Cauchy's estimate.

Extensive numerical testing has shown that the above optimization algorithm with recursive splittings is not optimal, neither with respect to the accuracy of the computed bound $V(r,m,l)$, nor with respect to the computation time. As an alternative to adaptive bisection, the user of the program can invoke a fixed uniform partitioning of $C$ with some $k_{\max}$ segments $S_k$. Based on user-defined values for $m$ and $k_{\max}$, the program determines the order $l$ of the Taylor polynomial such that $l$ is sufficiently large for a good approximation of $f$ by $t_l$, but reasonably small with respect to the overall computation time [8].

**Bounds for Taylor Remainder Series.** In addition to bounds for the Taylor coefficients of $f$, ACETAF also computes bounds for the Taylor remainder series $R_p(z) := \sum_{j=p+1}^{\infty} a_j z^j$ of $f$, for some $z$ with $|z| < r$. Bounds for $R_p$ are obtained from summing up the respective estimates for the Taylor coefficients in the remainder series. For the methods I and II, the remainder series is estimated by a geometric series. For example, in method I we obtain the estimate

$$R_p(z) \leq \frac{M(r)\left(\frac{|z|}{r}\right)^{p+1}}{1-\frac{|z|}{r}}.$$

A closed expression for the majorizing remainder series in methods III and IV is given in [37].

### 5.3 Availability of ACETAF

Our program is available in two versions, depending on the interval library that is used: C–XSC [24] or filib++ [30,31]. The C–XSC library is more comprehensive than filib++, but the latter is much faster than the former. The libraries C–XSC and filib++ are distributed under the terms of the GNU Lesser General Public License (formerly called GNU Library General Public License) [11].

ACETAF is distributed under the terms of the GNU General Public License [11]. The software is currently available at the following sites:

C–XSC and filib++: `http://www.xsc.de` and

ACETAF: `http://www.uni-karlsruhe.de/~Markus.Neher/acetaf.html`

At the moment, C–XSC supports the following platforms:

GNU C++ compilers gcc 2.95.2 or higher, PC with Linux,
GNU C++ compilers gcc 2.95.2 or higher, Sun Solaris workstation.

filib++ requires one of the GNU C++ compilers gcc 2.95.2 or higher, or the KAI C++ compiler. The filib++ macro library (which is used by ACETAF) is only supported on x86 systems and requires the use of GNU make.

ACETAF has been extensively tested and has been found to be reliable and robust. Of course, even though it is software for validated computations, it is subject to the same possible errors as conventional software. The program is distributed in the hope that it will be useful, but without any warranty.

**Graphical User Interface.** All input data (such as the order $l$ of the Taylor polynomial in the computation of $V(r, m, l)$, the maximal number of intervals in the list of the branch and bound algorithm, etc.) can be entered via a self-explanatory graphical user interface. The values are stored in an output file of the computation, and this file can be reused in other calculations.

The user of the program can enter four parameter values, which control the termination of the branch and bound algorithm:

- $t_{\max}$, the maximum computation time;
- $\varepsilon_{\text{abs}}$, the tolerated absolute error of the interval enclosure for the respective bound of each method;
- $\varepsilon_{\text{rel}}$, the tolerated relative error; and
- $k_{\max}$, the maximum number of subintervals.

The computation is terminated when at least one of these termination criteria is fulfilled.

**Symbolic Expression Handler.** ACETAF includes a symbolic expression handler, so that arbitrary user-defined compositions of the supported library functions can be used. The functions may be defined on arbitrary discs in the complex plane. Functions are entered as strings in the usual mathematical notation. The independent complex variable is represented by the literal “z”. The

literal "i" is used for complex unity. A function expression may contain constants in the scientific number format (such as 1.234E-05), the arithmetic operators +, -, *, /, the functions sqr, sqrt, exp, ln, sin, cos, tan, cot, asin, acos, atan, acot, sinh, cosh, tanh, coth, asinh, acoth, atanh, acoth, and the following functions with two arguments: power (integer powers), pow (real powers), and root (integer roots).

### 5.4 Numerical Example

Numerical examples were presented in [8,35,37]. Here, we only give one example for illustration. We show a table of upper bounds for $M(r)$ and $V(r, m, l)$ for different choices of $m$ and $l$, for several radii. The termination parameters are set to $\varepsilon_{\text{rel}} = 0.1$, $\varepsilon_{\text{abs}} = 0$, and $t_{\max} = 3600$ seconds (to avoid abortion of the program based on excess time). We used $k_{\max} = 1024$ for the computation of $M(r)$ and $k_{\max} = 8192$ for the computation of $V(r, m, l)$.

The table includes bounds for some of the Taylor coefficients and for some remainder sums of the respective functions, that were computed with ACETAF 2.8 and the filib++ interval library. The computation times (in seconds) were obtained on a PC with a 1200 MHz Athlon processor.

**Table 3.** Bounds for Taylor coefficients of $f(z) = (\cos z)/(z^2 + 101)$.

| $r$ | $l$ | $m$ | $M/V$ | $a_{100}$ | $a_{1000}$ | $R_{50}(0.95\,r)$ | Time |
|---|---|---|---|---|---|---|---|
| 1 | — | — | $M$ = 1.6E−02 | 1.6E−02 | 1.6E−02 | 2.3E−02 | < 1 |
| 1 | 10 | 0 | $V$ = 7.2E−09 | 7.2E−09 | 7.2E−09 | 1.1E−08 | 1.7 |
| 1 | −1 | 50 | $V$ = 3.4E+18 | 1.1E−75 | 1.2E−131 | 1.7E−49 | 1.7 |
| 1 | 19 | 31 | $V$ = 7.6E+00 | 1.4E−59 | 1.2E−92 | 1.5E−48 | 77 |
| 5 | — | — | $M$ = 1.2E+00 | 1.5E−70 | 1.2E−699 | 1.7E+00 | < 1 |
| 5 | 28 | 0 | $V$ = 1.3E−05 | 1.6E−75 | 1.4E−704 | 1.9E−05 | 2.4 |
| 5 | −1 | 50 | $V$ = 7.6E+31 | 2.8E−97 | 2.5E−782 | 3.3E−01 | 49 |
| 5 | 41 | 9 | $V$ = 9.4E−04 | 3.4E−85 | 2.1E−723 | 7.9E−13 | 12 |
| 10 | — | — | $M$ = 1.3E+04 | 1.3E−96 | 1.3E−996 | 1.9E+04 | < 1 |
| 10 | 56 | 0 | $V$ = 9.2E+03 | 9.2E−97 | 9.2E−997 | — | 3.6 |
| 10 | −1 | 10 | $V$ = 5.3E+23 | 8.4E−87 | 5.5E−997 | 4.6E+16 | < 1 |
| 10 | −1 | 30 | $V$ = 4.4E+75 | 5.6E−53 | 6.7E−985 | 1.8E+58 | 7.5 |
| 10 | −1 | 50 | $V$ = 2.3E+134 | 7.5E−10 | 8.0E−966 | 1.2E+117 | 49 |

**Example: Bounds for Taylor Coefficients of $f(z) = (\cos z)/(z^2 + 101)$.** $f$ has a singularity at $z = \sqrt{101}i$, and the circle with radius 10 is very close to

this point. Nevertheless, the computation of $M$ and $V$ is feasible, but the bounds obtained for $V(10, -1, m)$ are rapidly increasing with $m$.

For small radii, both methods II and III improve the bounds for the Taylor coefficients $a_j$ and for the remainder series $R_p$ by several orders of magnitude compared to the bounds that result from Cauchy's estimate.

## 6 Summary

We have presented four different software tools for verified computations. It appears that these packages are as diverse as the applications for which they were developed.

Each of the four packages is written so the user can define a particular problem in the same way it would be defined for non-rigorous software for the same purpose. The careful bounding of truncation errors is hidden in the code, where an unexperienced user may not even spot the difference. This is also true for the treatment of roundoff errors, which some of the packages in this paper do not handle themselves, but employ other well known interval libraries for this task. Hence, even a user knowing nothing about interval arithmetic or roundoff errors can use the software in the same way as conventional software.

As the development of computers continues, the question of computation times will become less important. Whether a computer program needs only five milliseconds or a full second to solve a particular problem is often irrelevant. Hence, if rigorous software is as simple to use as non-validated software, more users will be willing to use it to get validated results for their problems, even if the computation may take longer.

## References

1. G. Alefeld and J. Herzberger. *Introduction to interval computations.* Academic Press, New York, 1983.
2. K. Braune and W. Krämer. High-accuracy standard functions for real and complex intervals. In E. Kaucher, U. Kulisch, and Ch. Ullrich, editors, *Computerarithmetic: Scientific computation and programming languages*, pages 81–114. Teubner, Stuttgart, 1987.
3. J. M. Chesneaux. Study of the computing accuracy by using probabilistic approach. In C. Ulrich, editor, *Contributions to Computer Arithmetic and Self-Validating Numerical Methods*, pages 19–30. Baltzer, 1990.
4. J.-M. Chesneaux. Descriptif d'utilisation du logiciel CADNA-F. Technical Report 92-31, MASI Report, 1992.
5. J.-M. Chesneaux. Stochastic arithmetic properties. In C. Brezinski and U. Kulisch, editors, *Computational and Applied Mathematics, I-Algorithms and Theory*, pages 81–91. North-Holland, 1992.
6. Jean-Marie Chesneaux and Jean Vignes. Les fondements de l'arithmétique stochastique. *Comptes Rendus de l'Académie des Sciences, Série 1*, 315:1435–1440, 1992.
7. K. Du and R. B. Kearfott. The cluster problem in global optimization: The univariate case. *Computing (Suppl.)*, 9:117–127, 1992.

8. I. Eble and M. Neher. ACETAF: A software package for computing validated bounds for Taylor coefficients of analytic functions. *To appear in ACM TOMS*.
9. I. Eble and M. Neher. CoStLy: Complex standard functions library. `http://www.uni-karlsruhe.de/~Markus.Neher/CoStLy.html`, 2002.
10. C. A. Floudas. *Deterministic Global Optimization: Theory, Algorithms and Applications*. Kluwer, Dordrecht, Netherlands, 2000.
11. Free Software Foundation. GNU General Public License, Version 2. `http://www.gnu.org/licenses/licenses.html`, 1991.
12. A. Griewank. *Evaluating derivatives: Principles and techniques of algorithmic differentiation*. SIAM, Philadelphia, 2000.
13. C. Jansson. Rigorous lower and upper bounds in linear programming. Technical report, TU Hamburg-Harburg, 2002. `http://www.ti3.tu-harburg.de/paper/jansson/verification.ps`.
14. L. Jaulin, M. Kieffer, O. Didrit, and E. Walter. *Applied interval analysis*. Springer, London, 2001.
15. R. B. Kearfott. Decomposition of arithmetic expressions to improve the behavior of interval iteration for nonlinear systems. *Computing*, 47(2):169–191, 1991.
16. R. B. Kearfott. An interval branch and bound algorithm for bound constrained optimization problems. *Journal of Global Optimization*, 2:259–280, 1992.
17. R. B. Kearfott. Interval extensions of non-smooth functions for global optimization and nonlinear systems solvers. *Computing*, 57(2):149–162, 1996.
18. R. B. Kearfott. Treating non-smooth functions as smooth functions in global optimization and nonlinear systems solvers. In *Scientific Computing and Validated Numerics*, Mathematical Research, volume 90, pages 160–172, Berlin, 1996. Akademie Verlag.
19. R. B. Kearfott. On proving existence of feasible points in equality constrained optimization problems. *Math. Prog.*, 83(1):89–100, September 1998.
20. R. B. Kearfott. Globsol: History, composition, and advice on use. to appear in the proceedings of COCOS'02, held in Sophia-Antipolis, October, 2002.
21. R. B. Kearfott and J. Dian. An iterative method for finding approximate feasible points, 1998. preprint, `http://interval.louisiana.edu/GlobSol/Dian-approximate-optimizer.pdf`.
22. R. B. Kearfott and K. Du. The cluster problem in multivariate global optimization. *Journal of Global Optimization*, 5:253–265, 1994.
23. R. B. Kearfott and M. Novoa. Algorithm 681: INTBIS, a portable interval Newton/bisection package. *ACM Trans. Math. Software*, 16(2):152–157, June 1990.
24. R. Klatte, U. Kulisch, Ch. Lawo, M. Rauch, and A. Wiethoff. *C-XSC: A C++ class library for extended scientific computing*. Springer, Berlin, 1993.
25. R. Krawczyk and A. Neumaier. Interval slopes for rational functions and associated centered forms. *SIAM J. Numer. Anal.*, 22(3):604–616, June 1985.
26. U. Kulisch. *Advanced arithmetic for the digital computer*. Springer, Wien, 2002.
27. U. Kulisch and W. Miranker, editors. *A new approach to scientific computation: Proceedings of the Symposium on a New Approach to Scientific Computation (1982: IBM Thomas J. Watson Research Center)*, New York, NY, USA, 1983. Academic Press.
28. U. Kulisch and W. L. Miranker. *Computer arithmetic in theory and practice*. Academic Press, New York, 1981.
29. C. Lemaréchal. Nondifferentiable optimization. In M. J. D. Powell, editor, *Nonlinear Optimization 1981*, pages 85–89, New York, 1982. Academic Press.

30. M. Lerch, G. Tischler, and J. Wolff von Gudenberg. filib++ - Interval library specification and reference manual. Technical Report 279, Universität Würzburg, 2001.
31. M. Lerch, G. Tischler, J. Wolff von Gudenberg, W. Hofschuster, and W. Krämer. The interval library filib++ 2.0. Design, features and sample programs. Preprint 2001/4, Universität Wuppertal, Wissenschaftliches Rechnen/Softwaretechnologie, 2001.
32. R. E. Moore. *Interval analysis*. Prentice Hall, Englewood Cliffs, N.J., 1966.
33. J. J. Moré, B. S. Garbow, and K. E. Hillstrom. User guide for MINPACK-1. Technical Report ANL-80-74, Argonne National Laboratories, 1980.
34. M. Neher. LIVP: A Pascal–XSC program for the validated solution of IVPs for $n$th order linear ODEs with analytic coefficient functions. `http://www.uni-karlsruhe.de/~Markus.Neher/livptayp.html`, February 2000.
35. M. Neher. Geometric series bounds for the local errors of Taylor methods for linear $n$-th order ODEs. In G. Alefeld, J. Rohn, S. Rump, and T. Yamamoto, editors, *Symbolic Algebraic Methods and Verification Methods*, pages 183–193. Springer, Wien, 2001.
36. M. Neher. Validated bounds for Taylor coefficients of analytic functions. *Reliable Computing*, 7:307–319, 2001.
37. M. Neher. Improved validated bounds for Taylor coefficients and for Taylor remainder series. *J. Comput. Appl. Math.*, 152:393–404, 2003.
38. A. Neumaier. *Interval methods for systems of equations*. Cambridge University Press, Cambridge, 1990.
39. S. Oishi. *Numerical computation with result verification (in Japanese)*. Corona-Sha, Tokyo, 2000.
40. S. Oishi. Fast enclosure of matrix eigenvalues and singular values via rounding mode controlled computation. *Linear Algebra and its Applications*, 324:133–146, 2001.
41. S. Oishi and S. M. Rump. Fast verification of solutions of matrix equations. *Numerische Mathematik*, 90:755–773, 2002.
42. L. B. Rall. *Automatic differentiation: Techniques and applications, Lecture Notes in Computer Science, Vol. 120*. Springer, Berlin, 1981.
43. S. Rump. INTLAB – INTerval LABoratory. In T. Csendes, editor, *Developments in reliable computing*, pages 77–104. Kluwer, Dordrecht, 1999.
44. N. Sahinidis. Baron, 2003. `http://archimedes.scs.uiuc.edu/baron/baron.html`.
45. T. Sakurai and H. Sugiura. On factorization of analytic functions and its verification. *Reliable Computing*, 6:459–470, 2000.
46. H. Schichl and A. Neumaier. Exclusion regions for systems of equations, 2003. preprint, `http://www.mat.univie.ac.at/ neum/ms/excl.pdf`.
47. M. Tawarmalani and N. V. Sahinidis. *Convexification and Global Optimization in Continuous and Mixed-Integer Nonlinear Programming: Theory, Algorithms, Software, and Applications*. Kluwer, Dordrecht, Netherlands, 2002.
48. J. Vignes. New methods for evaluating the validity of mathematical software. *Math. Comp. Simul. IMACS*, 20:227–249, 1978.
49. J. Vignes. A stochastic arithmetic for reliable scientific computation. *Mathematics and Computers in Simulation*, 35(3):233–261, September 1993.
50. A. Wächter. *An Interior Point Algorithm for Large-Scale Nonlinear Optimization with Applications in Process Engineering*. PhD thesis, Carnegie Mellon University, 2002. `http://dynopt.cheme.cmu.edu/andreasw/thesis.pdf`.

# Multiple Precision Interval Packages: Comparing Different Approaches

Markus Grimmer[1], Knut Petras[2], and Nathalie Revol[3]

[1] Universität Wuppertal
Wissenschaftliches Rechnen / Softwaretechnologie
42097 Wuppertal, Germany
Markus.Grimmer@math.uni-wuppertal.de,
http://www.math.uni-wuppertal.de/wrswt/
[2] TU Braunschweig
Institut für Angewandte Mathematik
38106 Braunschweig, Germany
K.Petras@tu-bs.de,
http://www.tu-bs.de/~petras/
[3] INRIA, LIP
École Normale Supérieure de Lyon
69364 Lyon Cedex 07, France
Nathalie.Revol@ens-lyon.fr,
http://perso.ens-lyon.fr/nathalie.revol/

**Abstract.** We give a survey on packages for multiple precision interval arithmetic, with the main focus on three specific packages. One is a Maple package, `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.

## 1 Why Develop Multiple Precision Interval Packages?

### 1.1 Need for Arbitrary Precision Interval Arithmetic

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 $\pi$). However, for algorithms where extra computing precision is required (these are mostly numerical algorithms) it is important to distinguish between predictable and unpredictable loss of accuracy. If this loss is predictable, then multiple precision arithmetic perfectly fulfils the application's needs. When it is unpredictable, interval arithmetic can prove 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.**

R. Alt et al. (Eds.): Num. Software with Result Verification, LNCS 2991, pp. 64–90, 2004.

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.

The literature is inconsistent about the exact meaning of the term multiple 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 GMP (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, for example the staggered multiple precision arithmetic in the XSC (eXtended Scientific Computing) languages [25,26].

### 1.2 Organization of the Paper

The motivations and needs for multiple precision interval arithmetic packages are discussed in this first part. 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 Software Packages for Scientific Computing

The reasons for the implementation of an interval package for scientific computing software, such as MatLab, Maple or Mathematica, are different from those motivating interval libraries for standard programming languages like C++ (see Section 1.4).

These software environments are powerful tools for various kinds of computations, but, in contrast to programming languages, they primarily aim at usability, convenience and visualization of data. Moreover, they serve as means of education in schools and universities.

In addition to the general reasons for the implementation of an interval package, these packages serve the following purposes:

- combine symbolic computation with interval evaluation for computer algebra systems (Maple or Mathematica).
- 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.

One further reason especially applies to environments offering symbolic computation and multiple precision at the same time:

– In a computer algebra environment, the inexperienced user is apt to mistake rounded results for exact results, since symbolic computations are free of round-off errors, and he might expect that this will hold for the rest of his computations as well.

The combination of multiple precision and interval arithmetic is a way to fulfil this expectation.

Moreover, arbitrary 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 fact that an environment like Maple (especially with a GUI) serves teaching purposes.

### 1.4 Libraries for Arbitrary Precision Interval Arithmetic: Efficiency Issues

Other considerations apply to the implementation of multiple precision interval arithmetic libraries for programming languages. Here, the main issue is efficiency rather than ease of use and suitability for 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 native interval datatypes and operations (cf. Section 2.2). Thus, to allow interval computations in environments that do not support intervals, the solution consists in developing libraries.

Libraries developed for an existing programming language are compiled, *i.e.* interval operations are executed faster than within an interpreted package, which was detailed 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 application. A last source of efficiency lies in the use of the processor's arithmetic unit: with XSC (eXtended Scientific Computing) languages (cf. Section 2.2), operations are based on floating-point ones; with GMP-based (GNU Multiple Precision) libraries (cf. Section 2.3 and Section 2.4), they are based on machine integers. By contrast, in Maple all computations are done with radix-10 digits and all operations are thus software ones.

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 may already be handled, for instance it is performed by GMP.

Finally, if the chosen programming language offers operator overloading – as most object-oriented languages do – then modification of existing applications is very easy: indeed, only data types have to be changed. This feature is common to most packages developed for scientific computing software environments as well as libraries developed in C++ for instance (cf. Section 2.3 and Section 2.4).

## 2 Survey of Various Implementations

### 2.1 Packages for Scientific Computing Software Environments

#### IntLab for MatLab

IntLab [43,44] is an interval arithmetic package for MatLab. The main objective of its author, S. Rump, is to compute verified results with similar capabilities as 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 $\pi$ 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 [24] justifies its introduction with arguments similar to the ones given in Section 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 and leads for instance to a width of $4.44089 \times 10^{-16}$ 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 Mathematica version 4.2):

```
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, each of which should contain the exact value, is empty. One possible explanation can be found in [24]: *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 wrong previous computation can also be attributed to unvalidated conversion from real to interval and to unvalidated binary-to-decimal conversion in input/output routines.

In Mathematica, LU-related procedures and nonlinear system solvers can have intervals as arguments and return guaranteed results. Some extensions or applications based on this package are to be found in [7] and [33].

### 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.

One motivation for the implementation of `intpakX` was to offer some algorithms and extended operations using the existing `intpak` framework [11] which used to be part of the now discontinued 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 [15]; here, we simply give three examples of the enhanced or more convenient graphical output possibilities (see illustration).

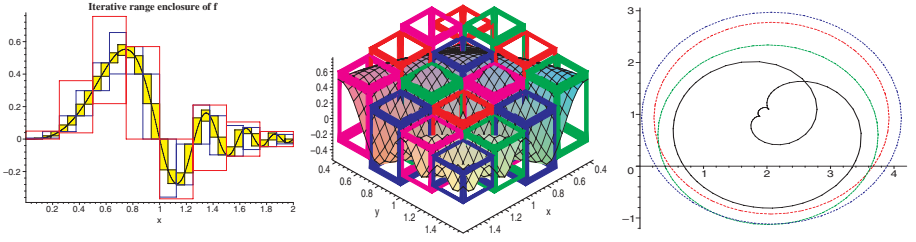



**Fig. 1.** Example output for the range enclosure of $f := x \to \exp(-x^2) \cdot sin(\pi x^3)$ (left), $g := (x, y) \to \exp(-xy) \cdot sin(\pi x^2 y^2)$ (center), and a complex polynomial with three different enclosures (right).

The other specific motivation was the fact that intervals can be defined in Maple without using `intpakX`, but that the evaluation of interval expressions does not behave according to all expected mathematical properties. Proper rounding is not provided (see below) 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 offer the expected mathematical properties and correct operators.

**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 [15,16] 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 was 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).

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 $x-\varepsilon$ ($x>0$ a 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 easier to use Maple graphics in conjunction with interval computations. They use Maple graphics features to offer special output for the visualization of the intervals resulting from the concerned `intpakX` functions.

**Scope of implemented functions 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 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 are provided.
- 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 (for short examples, see [17]):

- verified computation of zeros (Interval Newton Method) with the possibility to find enclosures of all zeros of a function on a specified (adequately small) interval; a branch and bound technique is used to display the resulting intervals in each step.
- 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.

Like 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. Let us denote by $Z := \langle c, r\rangle$ the complex disc centered at $c$ with radius $r$, with $c$ a complex number and $r$ a nonnegative real number. Interval operations are used to compute the complex disc

$$\begin{aligned}\exp(\langle c, r\rangle) &:= \left\langle \exp(c), max_{\Phi\in[0..2\pi)} \left|exp(c + r(\cos(\Phi) + i\sin(\Phi))) - \exp(c)\right|\right\rangle \\ &= \langle e^c, |e^c|\ (e^r - 1)\rangle \end{aligned} \tag{1}$$

with $e^c = e^{c_1}(\cos(c_2) + i\sin(c_2))$ (for $c = c_1 + ic_2$) (this is discussed more detailedly in [15]). The upper bound of the resulting interval for the radius is used as the radius of the new disc while the new center is defined by the midpoint of $e^c$ (interpreted as a rectangular complex interval). Formula (1) uses the fact that the maximum value of $|exp(z) - exp(c)|,\ z \in Z$, is reached for $z \in \partial Z$ (see, e.g., [14]).

Range enclosure for complex polynomials serves as an application for complex interval arithmetic. Three different versions are implemented: the first and second use a Horner scheme with centered and area-optimal multiplication, respectively, the third one uses a centered form.

### 2.2 Languages and Libraries

Few languages and compilers include a support for interval arithmetic; let us quote the XSC languages [3] (C/C++ [25], Pascal [26]) and the Sun Forte compilers for Fortran and C/C++ [47]. 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 [25] or [29]. Apart from computing accurate dot product, it has also been used for Horner evaluation of a polynomial in the interval Newton algorithm [28].

#### 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 [49]. 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 [20] 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 [39].

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: $\{op(x_1, \ldots, x_n), x_i \in \boldsymbol{x}_i\} \subset op(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$;
- furthermore, $op(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ or $f(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$, where $f$ is an elementary function, returns the tightest enclosing interval with floating-point endpoints;
- 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 (which will be removed in future versions) 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, many 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).

Those special functions that were needed for the above-mentioned project are implemented. These are the Gamma function, the complementary error function and Hermite functions (see [6] or [32]).

### 2.5 Final Remark

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 timing. Before presenting details, let us recall some `intpakX` features.

### 3.1 intpakX Specifics

The need for symbolic computing is a main reason for using a Maple package, while you don't necessarily use it if you want to do numerical computations only. 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.

Convenience is difficult to measure, but 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.

This has to be considered when you compare the times of the three packages mentioned before. Yet, the architecture of the multiple precision arithmetic and data type still plays an important role.

### 3.2 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, i.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.

### 3.3 Timing

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. The times presented in the tests subsection show how problem sizes and numbers of digits can be chosen to get results in reasonable time.

There is 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 mantissa cannot exceed the available memory.

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.

#### 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 -O2` or `g++-3.0.3 -O2`. All times are displayed in seconds.

**Results**

*Matrix Multiplications (Maple)*
The following times have resulted from a multiplication of matrices "by hand" (i.e. using 3 nested loops – 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 for 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 [40] 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 time 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.

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 exactly 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. This 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 when 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) 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 distinctions. 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. A theoretical foundation has been given in [15].

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 the 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 [19] has been adapted to arbitrary precision computations and implemented, cf. [38].

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 [38]. 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 $\varepsilon_Y$. 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:

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

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 [42]). 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 $H_n^{-1}$.

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

| $n$ | time $(s)$ | $d$ | $\mathrm{diam}([H_n^{-1}])$ |
|---|---|---|---|
| 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 | $0.16 \cdot 10^{-80}$ |

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 proposed to introduce a benchmark test, which consists of the inversion of the* $500 \times 500$ *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 including all those from $Q_{n_{k-1}}^{KP,k-1}$
   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. Nevertheless, it is one of the standard methods for numerical integration. Using interval arithmetic, it is possible to give an existence proof and to determine nodes $x_\nu^{[k]}$ and coefficients $a_\nu^{[k]}$. We sketch the method:

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

Property 2b) is equivalent to

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

(see [8, Theorem 55]). The initial quadrature formula is the Gaussian. The nodes are the zeros of a Legendre polynomial, which can be evaluated easily (for validation, we strongly recommend the use of its Chebyshev expansion and to use a stable evaluation of Chebyshev polynomials $T_n(x) = \cos(n \arccos x)$, see below). Now, given $p^{[k]}$, we want to determine $p^{[k+1]}$. Since $Q^{KP,k+1}_{n_{k+1}}$ uses the same nodes as $Q^{KP,k}_{n_k}$, $p^{[k+1]}/p^{[k]}$ is a polynomial. We therefore write (2) for $p^{[k+1]}$ as

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

Expanding $p^{[k]}$ and $p^{[k+1]}/p^{[k]}$ in terms of Chebyshev polynomials, we obtain a linear system for the Chebyshev coefficients of $p^{[k+1]}(x)/p^{[k]}(x)$, which can be solved with the methods described, e.g., in Section 4.3. Knowing these coefficients, we can use Newton's method to determine the nodes $x_\nu^{[k+1]}$. Finally, we determine the Chebyshev coefficients of $p^{[k+1]}$ in order to allow the next step and to determine the coefficients $a_\nu^{[k+1]}$.

Besides numerical linear algebra, the procedure requires the stable (and fast) evaluation of Chebyshev polynomials. Such a method can be based on $T_0(x) = 1$, $T_1(x) = 1$ and the recurrence relations

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

Chebyshev polynomials of the second kind are treated similarly.

Not only the existence, but also the positivity of a quadrature formula, i.e., the positivity of its coefficients $a_\nu$ (in our case $a_\nu^{[k]}$) is important. From theory, many nice properties follow from this positivity (see, e.g. [9]).

The presented iterative method is very sensitive with respect to perturbations in an early step. Numerical validation therefore requires high precision arithmetic.

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 corresponding linear systems. 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 [45]. The computationally complicated part is

$$\sum_{|b|\in\{\nu,\nu+2\}} \int_0^\infty H_{-\nu-4}\left(\frac{\cosh y}{\sqrt{2q}}\right) e^{yb} \Im\left\{e^{i\pi b}\,\mathrm{erfc}\left(\frac{y+bh+i\pi}{\sqrt{2h}}\right)\right\} dy \qquad (3)$$

where $\nu, q$ and $h$ are certain positive parameters. $H_\mu$ is a Hermite function, which is defined for negative $\mu$ by

$$H_\mu(z) = \frac{1}{\Gamma(-\mu)} \int_0^\infty e^{-t^2-2tz} t^{-\mu-1}\, dt \qquad (4)$$

(see, e.g. [32]). 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,

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

Properties of these two special functions are given, e.g, in [6] and [32].

The main difficulty is that, due to the oscillatory nature of the integrand, the complete integral is smaller than 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 [35].

### 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 [18,22], 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 2. With flat functions, using double floating-point precision, the optimum is very well approximated: $[0.00000, 1.00564E-08]$ for the Ratz 8 function (cf. [48]), but the optimizer is not accurately determined; a whole region containing points where the function takes values close to the optimal one is returned: $[0.93750, 1.09375]^9 \times [-10.0000, 10.0000]$ for the Ratz 8 function.

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 of figure 2) 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:

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

For such functions, the program usually runs out of memory: a huge list of intervals which are potential optimizers is kept; the program does a "best first" search and subdivides a lot of these candidates, but it does not manage to discard them.

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 [34,46]: the problem is to determine the localization of particles, through the minimization of the electrostatic energy of the system. More formally, the problem is to determine the global minimum of

$$\sum_{i=1}^{n}\sum_{j=i+1}^{n}\frac{1}{d(X_i, X_j)}$$

where $X_i$ and $X_j$ are the locations of particles $i$ and $j$, and $d$ is the Euclidean distance, subject to $X_i$ lies on the unit sphere. 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.

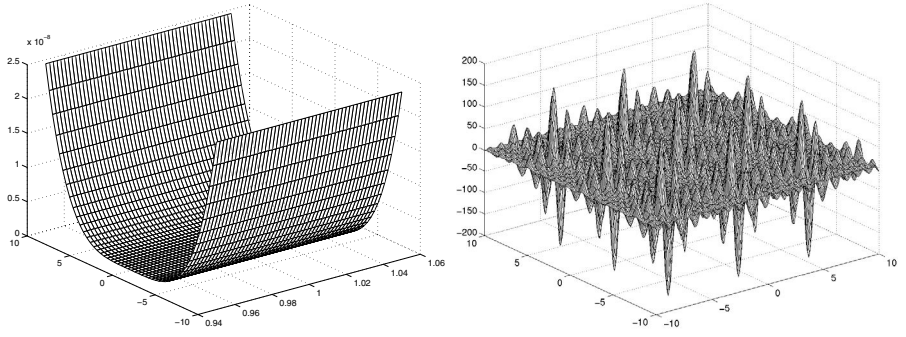



**Fig. 2.** Ratz 8 and Levy ($n^\circ$ 3) functions

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 Availability

The current software packages, corresponding documentations and application programs are available through the internet.

### 5.1 intpakX

This Maple package is available on
`http://www.math.uni-wuppertal.de/wrswt/software/intpakX/`
together with some documentation and examples.

It is also available as *"Research Powertool Interval Arithmetic"* from Waterloo Maple™ on
`http://www.mapleapps.com/powertools/ResearchApplication.shtml`.

### 5.2 MPFI

MPFI is a C package. It is available on `http://perso.ens-lyon.fr/nathalie.revol/software.html`: it includes a documentation, the source code and some rudimentary tests. This software requires a C compiler, GMP (which can be downloaded at `http://www.swox.com/gmp/`) and MPFR (available on `http://www.mpfr.org/`).

### 5.3 GMP-XSC

This package is available on `http://www.tu-bs.de/~petras/software.html`, where installation and usage is described. This software requires C, C++ and GMP. The latter is often part of LINUX distributions or may be obtained via, e.g., `http://www.swox.com/gmp/`

The applications mentioned in sections 4.3, 4.4 and 4.5 can also be found on `http://www.tu-bs.de/~petras/software.html`.

## 6 Conclusion

This paper presents a survey of existing packages for multiple precision interval arithmetic. Details are given for three packages: `intpakX` for Maple (which focuses on ease of use), and MPFI and GMP-XSC for C/C++ (which focus on efficiency and reliability through the use of a programming language). These three packages have been compared in Section 3.

The results show that getting tight and guaranteed results may sometimes take a lot of time, especially if a program is designed to be easy to use. This particularly applies to the standard functions which have to be further optimized. Yet, it is expected that multiple precision interval arithmetic will be more widely used in the future, since various complete, easy and rather efficient packages are now available. We hope that input from an increasing number of users 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. W. Fischer, I. Lieb, *Funktionentheorie.* Vieweg, 1992.
15. Geulig, I., Krämer, W. *Computeralgebra und Verifikationsalgorithmen.* University of Karlsruhe, 1998.
16. Geulig, I., Krämer, W. *Intervallrechnung in Maple - Die Erweiterung intpakX zum Paket intpak der Share-Library.* University of Karlsruhe, 1999.
17. Grimmer, M.: *Interval Arithmetic in Maple with intpakX.* PAMM Vol. 2, Nr. 1, p. 442-443, Wiley-InterScience, 2003.
18. E. Hansen. *Global optimization using interval analysis.* Marcel Dekker, 1992.
19. E. Hansen and R.I. Greenberg. An interval Newton method. *J. of Applied Math. and Computing*, 12:89–98, 1983.
20. T.-J. Hickey, Q. Ju and M.-H. Van Emden. Interval arithmetic: from principles to implementation *J. of ACM*, 2002.
21. N. Higham. *Accuracy and stability of numerical algorithms.* 2nd ed., SIAM, 2002.
22. R.B. Kearfott. *Rigorous global search: continuous problems.* Kluwer, 1996.
23. R.B. Kearfott, G.W. Walster. On stopping criteria in verified nonlinear systems or optimization algorithms. *ACM TOMS*, 26(3):373–389, 2000.
24. J. Keiper. Interval arithmetic in Mathematica. *Interval Computations*, (3), 1993.
25. R. Klatte, U. Kulisch, C. Lawo, M. Rauch, and A. Wiethoff. *C-XSC a C++ class library for extended scientific computing.* Springer Verlag, 1993.
26. Klatte, R., Kulisch, U. et al. *PASCAL-XSC.* Springer Verlag, 1991.
27. O. Knueppel. PROFIL/BIAS - a fast interval library. *Computing*, 53(3-4):277–287, 1994.
28. W. Krämer, U. Kulisch and R. Lohner. *Numerical toolbox for verified computing II – Advanced Numerical Problems* . To appear (1998).
29. Krämer, W. *Mehrfachgenaue reelle und intervallmässige Staggered-Correction Arithmetik mit zugehörigen Standardfunktionen.* Report of the Institut für Angewandte Mathematik, Karlsruhe, 1988.
30. Kulisch, U. *Advanced Arithmetic for the Digital Computer. Design of the Arithmetic Units.* von U. W. Kulisch Springer, Wien, 2002.
31. 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.
32. N.N. Lebedev. *Special functions and their applications.* Dover Publications, Inc., New York, 1972.
33. R. Maeder. The Mathematica Programmer: Interval Plotting and Global Optimization. *The Mathematica Journal*, 7(3):279–290, 1999.
34. K. J. Nurmela. *Constructing spherical codes by global optimization methods.* Research report no 32, Helsinki University of Technology, 1995. Available on `http://www.tcs.hut.fi/Publications/bibdb/HUT-TCS-A32.ps.gz`
35. K. Petras. *Numerical Computation of an Integral Representation for Arithmetic-Average Asian Options.* Preprint available on `http://www.tu-bs.de/~petras/publications.html`

36. 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`
37. L.B. Rall. *Automatic differentiation – Techniques and Applications*, Springer Lecture Notes in Computer Science, Vol. 120, Berlin, 1981.
38. 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`.
39. N. Revol and F. Rouillier. The MPFI library. `http://www.ens-lyon.fr/~nrevol`, 2001.
40. N. Revol and F. Rouillier. Motivations for an arbitrary precision interval arithmetic and the MPFI library. To appear in *Reliable Computing*, 2003.
41. 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`.
42. 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.)
43. S. Rump. *Developments in reliable computing, T. Csendes ed.*, chapter INTLAB - Interval Laboratory, pages 77–104. Kluwer, 1999.
44. S. Rump. Fast and parallel interval arithmetic. *BIT*, 39(3):534–554, 1999.
45. M. Schröder. *The Laplace transform approach to valuing exotic options: the case of the Asian option.* Mathematical finance, Trends Math., Birkhäuser, 2001, 328–338.
46. M. Shub and S. Smale. Complexity of Bezout's theorem III. Condition number and packing. *J. of Complexity*, 9:4–14, 1993.
47. Sun Microsystems, Inc. *C++ interval arithmetic programming reference.* 2000.
48. A. Wiethoff. *Verifizierte globale Optimierung auf Parallelrechern.* PhD thesis, Karlsruhe, Germany, 1997.
49. J. M. Yohe. Portable software for interval arithmetic. *Computing*, Suppl. 2, 211–229, 1980.

# Interval Testing Strategies Applied to COSY's Interval and Taylor Model Arithmetic

George F. Corliss and Jun Yu

Electrical and Computer Engineering
Marquette University
Milwaukee WI, USA

**Abstract.** The COSY Infinity software package by Berz et al. is widely used in the beam physics community. We report execution-based testing of its interval and Taylor model arithmetics. The testing strategy is careful to avoid contamination by inevitable rounding errors. Tests were ported to Sun's F95 and INTLAB. In each package, we uncovered violations of containment which have all been corrected by their authors. We encourage users of COSY and most other software packages to check author/vendor web sites regularly for possible updates and patches.

## 1 Testing COSY's Interval Arithmetic

During Spring 2002, the reliable computing email list `reliable_computing@interval.louisiana.edu` had an active discussion of COSY Infinity [1,9] (Berz et al., available from `http://cosy.pa.msu.edu` [2]). COSY Infinity is an arbitrary order package for multivariate automatic differentiation and interval and Taylor model arithmetic. It can be used in an interpreted version, which we tested, in a compiled version from Fortran 77 and C programs, or through objects in Fortran 90 and C++. The `reliable_computing` discussions raised concerns about the reliability of interval and Taylor model arithmetics, so Berz commissioned the execution-based testing of COSY interval arithmetic we report here. We also applied our tests to Sun Microsystems' Fortran 95 [10] and Rump's INTLAB for MATLAB [13,14,15].

Testing software is challenging. Myers summarizes testing philosophy, "The purpose of testing is to find errors" [11]. Kit [8], Kaner et al. [6], or Whittaker [16] offer best practice in industrial software quality assurance.

Authors of many packages for interval arithmetic have tested their work, but there is little literature describing those tests. In TOMS 737 [7], Kearfott et al. tested their Fortran 77 INTLIB arithmetic operations with a combination of specially constructed and randomly generated arguments. Corliss [4] gave a suite of programs for "testing" environments for interval arithmetic for usability and speed. Sun Microsystems says their Fortran 95 interval elementary function library has undergone exhaustive testing, which is confidential.

The focus of this paper is on the testing of COSY's interval and Taylor model arithmetic. Since we found little methodological discussion in the literature, we

R. Alt et al. (Eds.): Num. Software with Result Verification, LNCS 2991, pp. 91–106, 2004.

developed testing methods that could be applied more generally. Besides the testing of COSY, we applied our methods also to Sun's F95 and INTLAB primarily to validate our testing methods. The testing methods have wider utility, but our focus is execution-based testing of COSY.

## 2 What Is "Correct?"

The fundamental tenet of the interval community is, *"Thou shalt not lie!"* It is an error to i) violate containment or ii) assert a mathematical falsehood. Our testing exposed violations of containment for

1. COSY: power when the exponent is not an integer, but very close to it.
2. COSY: (with warning) tan when the interval argument crosses discontinuity.
3. INTLAB: sqrt for most arguments.
4. Sun F95: tanh for many negative arguments.
5. COSY Taylor models: sin, asin, and acos.

We give details of errors we found in Sects. 5 and 9. On the other hand, questions of appropriate domains for interval operations, tightness of enclosures, speed, and ease of use are not considered errors, but may represent opportunities for improved performance. We raise some of those issues in Sects. 6, 7, and 8.

## 3 Test Strategy

To complete the testing in a timely manner, we accepted a very narrow scope. We tested the arithmetic operations unary and binary addition and subtraction, multiplication, and division, and the intrinsic functions power, sin, cos, tan, asin, acos, atan, sinh, cosh, tanh, log, exp, sqrt. sqr, and isqrt. Our goal is to identify i) violations of containment or ii) assertions of mathematical falsehood. We developed a set of test cases consisting of an interval vector $[x]$ and an expression $f(x)$. Expected results are computed a posteriori in Maple. We did not attempt testing of other features of COSY including its linear dominated bounder, shrink-wrapping, or ODE solving.

We denote by $[\widehat{f([x])}]$ the result of challenging the interval arithmetic to evaluate $f$ on the interval $[x]$. We seek examples $x \in [x]$ for which $f(x)$ is not in $[\widehat{f([x])}]$. We do not need to know the true containment set of $f([x])$. Instead, we use Maple as the "referee" of containment. We

1. Read each test case into a COSY driver;
2. Construct COSY intervals for the arguments;
3. Evaluate the expression using COSY interval arithmetic;
4. Write binary values of the arguments and the COSY result;
5. Read the binary arguments and COSY results into Maple;
6. Perform many point evaluations $f(x)$ for $x \in [x]$;
7. Compare Maple's $f(x)$ with COSY enclosure.

The most challenging aspect of conducting the tests was to prevent inevitable roundoff errors from contaminating our results.

### 3.1 Roundoff Errors

Suppose we wish to test sin on [0.1, 0.6] on an IEEE arithmetic machine. Fundamentally, it is *impossible,* since 0.1 and 0.6 are not exactly representable. We cannot even express the question, "What is sin [0.1, 0.6]?"

Roundoff errors may be introduced into tests of interval software when we

1. Read test cases into the test driver;
2. Construct interval(s) for the arguments;
3. Extract interval bounds from arguments and results;
4. Write arguments and results to a file;
5. Read arguments and results into Maple;
6. Construct Maple variable precision representations;
7. Perform Maple operations;
8. Report from Maple.

Table 1 suggests the schematic flow of the testing process. It shows the communication from COSY to Maple of both the exact (binary) argument(s) used to challenge each arithmetic operation or intrinsic function and the exact (binary) result computed by COSY. We consider each potential source of roundoff error in turn. The issue is not with Maple. Issues 1 - 3 concern COSY. Issues 4 - 6 concern communication between any pair of dissimilar software packages.

**Table 1.** Schematic of the flow of testing

| COSY | | Maple referee |
|---|---|---|
| Enter [0.1, 0.6] | | |
| $\Downarrow$ round near | | |
| [ internal IEEE 754 ] | $\Longrightarrow$ | [ $\cdots$ ] |
| $\Downarrow$ INTV round out | | $\Downarrow$ multiprecision |
| INTV( $\cdots$ ) | | $x \in$ [ $\cdots$ ] |
| $\Downarrow f$ round out | | $\Downarrow f$ multiprecision |
| $\Downarrow$ | | $f(x)$ |
| funct($\cdots$) | $\Longrightarrow$ | enclosed in? |

**Read Test Cases into the Test Driver.** We must separate the testing of the input and output routines from the testing of the operations. Our goal is to test the operations of interval arithmetic. We read files of test cases into a test driver. We view the internal binary values as *truth,* while the ASCII values in the file are viewed as approximations. In the few cases where the difference matters, we use test arguments that are exactly representable in binary, and we check whether they are read exactly.

**Construct an Interval.** COSY's interval constructor INTV() by default adds one ULP outward to its arguments to compensate for assumed possible inward rounding in assigning their values. We tested COSY using the INTV() constructor to model usual use. Using INTV() prevented us from testing cases such as asin([1, 1]) because INTV(1.0, 1.0) contains points at which asin is not defined.

**Extract Interval Bounds.** After challenging COSY's interval arithmetic, we write the challenge arguments and the COSY results to a file. We use the COSY functions INL() and INU() to extract the lower and upper bounds, respectively, of the COSY intervals. We verified by both execution testing and by direct code inspection that INL() and INU() return their respective values with no rounding.

**Write Arguments and Results to a File.** To avoid roundoff in writing the challenge arguments and COSY's results to a file for reading by Maple, we write them in binary form, either big endian or little endian, depending on the host testing platform.

**Read Arguments and Results into Maple.** We read binary representations of the challenge arguments and the COSY results into Maple. The binary representation is system dependent, so we used different functions to read binary files written in big or little endian formats. We verified the correct transmission of values by comparing HEX dumps from COSY, Maple, and DOS's debug.

**Construct Maple Variable Precision Representations.** We read the binary file into Maple using 900 decimal digit arithmetic. Maple's binary read and convert to decimal is not accurate (previously known), so we wrote our own binary read in Maple, reading each byte as an integer and reassembling the IEEE representation using Maple's 900 decimal digit arithmetic.

**Perform Maple Operations.** We used 900 decimal digits to ensure that the full dynamic range of 53 bit mantissa IEEE double precision numbers is exactly representable in the decimal form used by Maple's variable precision arithmetic. Even if Maple's variable precision arithmetic were not accurate in the last few digits, we are safe, since we are detecting violation of containment errors in about the 14 - 17 th decimal digit.

Is 900 decimal digits "large?" No. In order for our logic to hold, we must ask Maple to evaluate the sin at *exactly* the same endpoints with which we challenged COSY. We have "exact" in the form of binary values. 53 binary digit numbers can be exactly representable in a finite number (56) of decimal digits (not the other way around). Representing the full range of IEEE double precision numbers, about $10^{-308}$ to $10^{308}$, requires another 617 decimal digits. To get Maple to evaluate sin at INF(X) and SUP(X) as evaluated by COSY, we must use at least 673 decimal digits in Maple. 900 gives a margin of error in case Maple's last few digits might be in error, of which we saw no evidence. In practice, we saw some incorrectly diagnosed "failures" using 100 decimal digits, but not with 200 digits.

Violations of containment are detected in Maple by comparing Maple's 900 digit evaluation of $f(x)$ with COSY's enclosure. If a violation of containment were due to a rounding error in Maple's evaluation, the failure of containment would be in the last few of the 900 digits, and increasing to 1000 or more digits would resolve them. In all violations of containment we observed, the failure was of approximately the accuracy of double precision computation, and increasing the number of digits had no effect.

In each violation of containment detected by Maple, careful human examination of the test case confirms that reported violations of containment truly represent a failure of the software under test. We used Maple for its arbitrary precision capabilities to detect the errors, but once found, errors are visible to the reader in this paper or in COSY execution with no need to rely on Maple.

**Report from Maple.** Values printed by Maple are subject to rounding error on output, but all of our conclusions have been drawn using internal Maple representations. Any Maple output rounding has no effect on our conclusions.

### 3.2 Test Cases

We tested 30 multi-operation expressions, but if an arithmetic package gets individual operations and intrinsic functions right, it will get complicated expressions right, too. Hence, we tested primarily 2,600+ expressions composed of a single operation or intrinsic function.

For elementary operations, no matter how wide the arguments, extrema occur at the endpoints, except for division by intervals containing zero. Similarly for intrinsic functions, extrema are always at the endpoints, except for a modest set of exceptions (e.g., sin and cos for arguments that span $\pi$ or $\pi/2$), which we enumerate and test. Hence, we are most likely to find violations of containment at endpoints of the challenge arguments.

Our Maple "referee" checks interior points, but we observed no failures at interior points. For each test case, we have Maple evaluate the expression under test at 11 points in the challenge argument interval using 900 decimal digit approximate arithmetic, as illustrated in the pseudo-code below. All errors we found would have been detected using only two points in the challenge interval. If $f(x)$ is not in COSY's result interval, we have a likely violation of containment, which we verify by human inspection of results as described in Sect. 5.

```
for (i = 0; i <= 10; i++) {
  y = INF(X) + (SUP(X) - INF(X)) * i/10.0
  fx = f(y)
  ERROR if fx is outside COSY result
}
```

We might look at extrema of the function, check at randomly chosen points, or at far more points. There are separate test cases to challenge evaluation within one ULP of extrema, so checking at extrema is already covered. Random tests are rarely as effective at uncovering errors as carefully constructed challenges; our test cases uncovered all the errors we found. None of our 500,000 random tests uncovered an error. Similarly, we had checked at 10,000 points (vs. 11) early in our testing, but all the errors we found at endpoints.

Most of our test cases came from TOMS 737 [7]. Kearfott et al. tested their Fortran 77 INTLIB interval arithmetic operations with a combination of specially constructed and randomly generated arguments. We added a few specially constructed arguments of our own and 30 multi-operation expressions taken from

tests of a validated quadrature package by Corliss and Rall [3]. In general, we expect interval arithmetic most likely to fail for very large or very small (in either absolute or relative terms) domain or range values, near boundaries of domains, or near underflow or overflow.

To increase the coverage of our tests of binary operations, each pair of arguments was used in several combinations. For example for addition and subtraction, argument intervals $[a]$ and $[b]$ give test cases

- $[a] + [b]$, $[a] - [b]$, $[-a] + [b]$, $[-a] - [b]$
- $[-a] + [-b]$, $[-a] - [-b]$, $[a] + [-b]$, $[a] - [-b]$
- $[b] + [a]$, $[b] - [a]$, $[-b] + [a]$, $[-b] - [a]$
- $[-b] + [-a]$, $[-b] - [-a]$, $[b] + [-a]$, $[b] - [-a]$

For multiplication, with $0 \leq [\underline{a}, \overline{a}]$ and $0 \leq \left[\underline{b}, \overline{b}\right]$, we test 16 combinations:

- $[\underline{a}, \overline{a}] \times \left[\underline{b}, \overline{b}\right]$, $[-\underline{a}, \overline{a}] \times \left[\underline{b}, \overline{b}\right]$, $[-\overline{a}, -\underline{a}] \times \left[\underline{b}, \overline{b}\right]$, $[-\overline{a}, \underline{a}] \times \left[\underline{b}, \overline{b}\right]$
- $[\underline{a}, \overline{a}] \times \left[-\underline{b}, \overline{b}\right]$, $[-\underline{a}, \overline{a}] \times \left[-\underline{b}, \overline{b}\right]$, $[-\overline{a}, -\underline{a}] \times \left[-\underline{b}, \overline{b}\right]$, $[-\overline{a}, \underline{a}] \times \left[-\underline{b}, \overline{b}\right]$
- $[\underline{a}, \overline{a}] \times \left[-\overline{b}, -\underline{b}\right]$, $[-\underline{a}, \overline{a}] \times \left[-\overline{b}, -\underline{b}\right]$, $[-\overline{a}, -\underline{a}] \times \left[-\overline{b}, -\underline{b}\right]$, $[-\overline{a}, \underline{a}] \times \left[-\overline{b}, -\underline{b}\right]$
- $[\underline{a}, \overline{a}] \times \left[-\overline{b}, \underline{b}\right]$, $[-\underline{a}, \overline{a}] \times \left[-\overline{b}, \underline{b}\right]$, $[-\overline{a}, -\underline{a}] \times \left[-\overline{b}, \underline{b}\right]$, $[-\overline{a}, \underline{a}] \times \left[-\overline{b}, \underline{b}\right]$

and similarly for division. In addition, we constructed more than 500,000 random tests that discovered no additional errors:

```
loops for i and j
  a = RAND(); b = RAND();
  x1 := +- 0.a * 2^+-i;
  x2 := +- 0.b * 2^+-j;
  [X] := [x1, x2];
  expr(X);
```

## 4 Test Environment

Our tests of COSY and INTPAK were executed on an HP notebook PC N5270 with a 700 MhZ Pentium III processor, 128 MB RAM, and a 20 GB hard disk under Microsoft Windows ME. The tests were replicated on a Toshiba Satellite 4090XDVD with an Intel Celeron at 400 Mhz, 128 MB RAM, running Windows 98. Our tests of Sun Workshop 6 were conducted on a Sun Enterprise 250, UltraSPARC 3 with one CPU at 450 Mhz with 512 Mb RAM. We tested

- COSY version 8.1 (updated June 8, 2002) downloaded from `www.cosy.pa.msu.edu` on June 25, 2002. The tests were repeated on a modified version of COSY provided on May 2, 2003.
- Sun WorkShop 6 update 1 Fortran 95 6.1 2000/09/11 (from f95 -V ...). The tests were repeated with a patched version released in September, 2002.
- INTPAK version 4.0, `www.ti3.tu-harburg.de/~rump/intlab` downloaded on January 15, 2003. The tests were repeated on Version 4.1.1 downloaded on January 22, 2003.

We used Maple 6 and MATLAB version 5.2. In Maple, we use little beyond the underlying variable precision arithmetic, so newer versions should have no effect on our tests. The error in INTPAK was traced to an anomaly in MATLAB which might be changed in a later version, although Rump observed the same anomaly in the current MATLAB version as of January, 2003.

## 5 Test Results

In this section, we report the results of our tests. In Sect. 3.2, we claimed to have verified suspected errors by human inspection. In this section, we offer the errors for inspection by the reader. Maple found the errors, but the reader can see them with no dependence on Maple.

### 5.1 COSY: POWER Near an Integer

**Test case (ASCII):** $[2.0, 2.0]^{1.00000000001}$
**As presented to COSY:** $2^{1.00000000001000000000827\ldots}$ (approximate decimal representation of binary value)
**COSY result:** $[1.999999999999999555\ldots, 2.000000000000000444\ldots]$ (approximate decimal representation)
**Maple's** $f(x)$**:** $2.0000000000138\ldots$ (approximate decimal representation), which violates containment by about $10^{-11}$.
**Cause:** The POWER operator was intended only for internal use by COSY for integer and half-integer exponents. Exponents within $10^{-10}$ of an integer or a half-integer are rounded to the nearby integer or a half-integer. Exponents further from an integer or a half-integer are rounded with a warning message.
**Solution:** COSY authors removed the POWER operator from the list of user callable operations.

### 5.2 COSY: TAN Crossing Discontinuity

**Test case (ASCII):** $\tan([1.0, 2.0])$ or $\tan([1.0, 1.0E + 30])$
**COSY result:** Print a warning and return $[-1.0E{+}35, 1.0E{+}35]$, which violates containment at points very close to $\pi/2$. This is a problem if the user ignores the warning, or if the warning scrolls off the screen.
**Cause:** COSY correctly recognized that the challenge argument includes a singularity, but it returned finite bounds.
**Solution:** COSY authors modified COSY so that after the warning is printed, execution halts.

### 5.3 COSY: ASIN or ACOS at ±1

**Test case (ASCII):** asin(1), asin([−1.0, 1.0]), or similarly for acos.
**COSY result:** Messages "asin(1) does not exist, " and "asin([-1, 1]) does not exist," respectively. These assert mathematical falsehoods.

**Cause:** COSY's interval constructor INTV() outwardly rounds the intervals [1, 1] and [-1, 1], even though their endpoints are exactly representable. Hence, COSY correctly detects that the challenge argument includes points outside the domain of asin. The default output routines in the test environment round endpoints as printed in the message, although other environments printed more digits, so the message was correct as printed.
**Solution:** COSY authors changed the formating of the message to read, "arcsin does not exist for the interval $[0.999999999999999, 1.000000000000001]$."

### 5.4 Sun F95: tanh (Negative)

To validate the testing methodology, we re-wrote the same test battery for Sun's F95 compiler. For challenge arguments less than about -4, e.g., tanh ([-4.879, -4.267]), containment fails by 1-2 ULP's.
**Cause:** There was a discrepancy between production and development versions.
**Solution:** Sun corrected the problem within one week, releasing an update.

### 5.5 INTLAB: sqrt

To further validate the testing methodology, we re-wrote the same test battery in Matlab for Rump's INTLAB. For the sqrt function, every degenerate interval fails by one ULP, and most thick intervals fail.
**Cause:** MATLAB's sqrt is not the IEEE sqrt. It uses round to nearest, rather than the current rounding mode.
**Solution:** Within a day, Rump posted a corrected version of INTLAB using its own rounding control for sqrt.

## 6 Domains: Opportunity for Improvement?

When a package for interval arithmetic encounters arguments outside the mathematical domain, it can respond by

1. Continue execution with empty, NAN, over/underflow, or other special value
2. Consider $f([x])$ as $f([x] \cap \text{domain of } f)$ (Sun's approach)
3. Halt execution, possibly with an error message (COSY and INTLAB)

As originally tested, COSY was not consistent in its handling of arguments outside the mathematical domain. Those inconsistencies have been corrected by the COSY authors.

COSY considers it a fatal error to evaluate outside the domain of an expression, e.g., asin(1) or sqrt(0). These examples are outside the domain because COSY enlarges the intervals on construction. Sun's F95 "handled" many cases COSY did not. For example, Sun considers sqrt ([-1, 1]) to be [0, 1].

We suggest handling of domains as an opportunity for improvement. We found no further violations of containment, and we understand why COSY treats

asin(1) or sqrt(0) as fatal errors. However, we would consider it an improvement if COSY were able to evaluate such cases correctly.

Sun's csets (containment sets) represent Sun's effort to handle domains. Csets are based on an elegant theory, but their implications are not well understood by the interval community. For example, Neher has given an example $f(x) = \sqrt{x} + 1/2 = 0$ on [-4, 4]. Naive cset evaluation gives $f([-4, 4]) = [1/2, 5/2] \subset [-4, 4]$, incorrectly suggesting the existence of a fixed point. Cset evaluation appears to require independent verification of continuity, which is done implicitly in some systems for interval arithmetic.

## 7 Tightness: Opportunity for Improvement?

COSY makes many compromises for efficiency over tightness of the intervals. For example, the COSY interval constructor INTV() rounds endpoints outward, while Sun's F95 and Rump's INTLAB provide interval constructors that accept strings and round outward only when necessary to guarantee containment.

We compared the excess widths of the COSY, Sun F95, and INTLAB results across our test cases. Table 2 shows the number of Units in the Last Place (ULP's) the interval result is wider than the Maple result, the interval computed by Maple in 900 decimal digit arithmetic. Compared with IEEE double precision computed by COSY, the Maple result is a very good approximation to the true result. We do not have exactly the correct number of ULP's in every case, but we do have a reliable measure of excess widths. Suppose (in pseudocode)

$t_U$ = Maple upper bound of the result
$c_U$ = upper bound computed by COSY ($t_U \le c_U$)
$r_U = c_U - t_U$
if $t_U = 0$ then $r_U = r_U * 2^{1022}$ else $r_U = r_U / |t_U| * 2^{52}$
Similarly for the ULP's at the lower bound $r_L$
Add $r_L + r_U$ ULP's at lower and upper bounds

For example, consider $[1, 2] + [3, 4] = [4, 6]$. The COSY result is

$$[\text{FC FF FF FF FF FF 0F 40 08},\ \text{04 00 00 00 00 00 18 40 08}]\ \text{(hex)}$$
$$= [3.999\ 999\ 999\ 999\ 998\ 223\ ...,\ 6.000\ 000\ 000\ 000\ 003\ 552\ ...]\ ,$$

which is eight excess ULP's because the constructors INTV(1.0, 2.0) and INTV (3.0, 4.0) round out, and the operator ADD rounds out further. Sun's F95 and INTLAB give excess widths of zero ULP's for this example. The excess widths in ULP's can be large when the true answer is near the underflow limit.

Table 2 shows the number of test cases for which the interval result had excess widths shown. Smaller excess widths are better, so it is better to have more test cases with excess widths of 0 - 2 and fewer test cases with larger excess widths. The first row in Table 2 shows that COSY computed the tightest possible enclosure (zero excess width) in 33 test cases, while F95 and INTLAB were as tight as possible in 1277 and 1201 test cases, respectively, from the total

**Table 2.** Excess width in ULP's

| | COSY June '02 | COSY May '03 | Sun F95 | INTLAB ver. 4 |
|---|---|---|---|---|
| 0 | 33 | 33 | 1277 | 1201 |
| 1 | 1 | 1 | 697 | 607 |
| 2 | 81 | 79 | 251 | 292 |
| 3-4 | 746 | 746 | 26 | 147 |
| 5-8 | 906 | 906 | 1 | 9 |
| 9-16 | 194 | 190 | 0 | 2 |
| 17-32 | 151 | 129 | 0 | 0 |
| 33-64 | 17 | 15 | 0 | 0 |
| 65-128 | 6 | 6 | 0 | 0 |
| 129-256 | 14 | 14 | 0 | 0 |
| 257-512 | 12 | 12 | 0 | 0 |
| Total | 2161 | 2130 | 2252 | 2259 |

of 2,600 test cases. Test cases with no finite true result, with true result zero, or with underflow or overflow are excluded, leading to different numbers of total test cases reported for each package.

Loss of tightness is not an error, but it is an opportunity for improvement, possibly at the expense of speed or portability. The Sun and INTLAB results in Table 2 show that increased tightness is achievable.

## 8 Speed: Opportunity for Improvement?

We prefer fast programs to slow ones, but unbiased, comprehensive speed testing is difficult and controversial. Speed is not in the scope of our tests, but we have run programs implementing the same test cases in different environments, and we suspect some readers might wonder, "How long did each take?" We make no claim of fair testing of speed. That could be the subject of another paper, but we report what we observed.

COSY and INTLAB timings were made on a Toshiba Satellite 4090XDVD with an Intel Celeron at 400 Mhz, 128 Mb RAM, running Windows 98, denoted by (Win 98) in Table 3. The versions of COSY and INTLAB we tested both run in an interpreted mode. The Sun F95 timings were made on a Sun Enterprise 250, UltraSPARC 3, 1 CPU at 450 Mhz with 512 Mb RAM, denoted by (SPARC) in Table 3. The F95 code was compiled, linked, and run. We have not reported compile and link times.

Table 3 reports CPU time for one million evaluations of the Shekel 5 function, commonly used to measure a Standard Time Unit (STU) [5]:

$$f(x) = -\sum_{i=1}^{m=5} \frac{1}{(x - A_i)(x - A_i)^T + c_i} \ ,$$

where $A_i$ denotes the $i$th row of a given $5 \times 5$ matrix A, and $c$ is a given vector of length 5. Evaluation of the Shekel 5 function reflects arithmetic operations, so we also report CPU time for the evaluation of

$$f(x) = \log_{10}(\text{asin } (\sin^2(x) + \cos^2(x) - \exp(\text{atan } (-x^2)))) \tag{1}$$

constructed to reflect executions for intrinsic function evaluations.

**Table 3.** CPU times in seconds

| | COSY (Win98) | INTLAB (Win98) | Sun F95 (SPARC) |
|---|---|---|---|
| 1 M evaluations of Shekel 5 | | | |
| Double precision | 92 | 410 | 25.4 |
| Interval | 157 | 23289 | 33.2 |
| 1 M evaluations Equation (1) | | | |
| Double precision | 7.3 | 142 | 2.89 |
| Interval | 25.4 | 41650 | 13.58 |
| 2,600 interval test cases | 6.0 | 19.1 | 0.3 |

INTLAB interval times were estimated by timing 10,000 evaluations and multiplying by 100. Execution of our interval test cases is dominated by disk I/O. In this environment, interpreted COSY is significantly faster than interpreted INTLAB, although recoding either one in a style more appropriate for its environment may yield significant improvements. We did not attempt to optimize the performance, preferring to keep the code for the tests as similar as possible in each environment. For example, the INTLAB code uses loops rather than much faster vector operations. The ratio of interval / real times for COSY are comparable with Sun's F95, and significantly smaller than INTLAB. Results in other environments may be markedly different.

Regarding tightness and speed, Martin Berz responds to the results of our tests,

"COSY is designed on the two premises of portability across platforms on the one hand, and use within the Taylor model framework on the other. The desired portability is achieved by building interval intrinsics based on F77 intrinsics, with the necessary safety factors of around four ULP's because of the inherent precision (or rather lack thereof) of the intrinsics. The use in the Taylor model framework entails that in practically relevant calculations, these slight overestimations usually do not matter since the Taylor model approach is used for large domain intervals where because of dependency, conventional validated methods usually have much larger overestimations in all but the simplest cases. Furthermore, since the vast majority of effort in the Taylor model arithmetic lies in the floating point coefficient arithmetic which is highly optimized in COSY, the efficiency of the interval implementation is of secondary significance."

We repeated our tests replacing the default safety factor in COSY for inflation of F77 intrinsics by an inflation of one ULP at each end. We observed reduced excess widths and no further violations of containment.

## 9 Testing COSY's Taylor Model Arithmetic

After testing COSY's interval arithmetic, we turned to its Taylor model arithmetic. Revol et al. [12] provide mathematical proofs that the algorithms in COSY for multiplying a Taylor model by a scalar and for adding or multiplying two Taylor models return Taylor models satisfying the containment property. We performed broader, execution-based testing. Revol's proof of the algorithm and our execution-based testing are complementary. The proof is more general than a (large) collection of test cases in the sense that test cases can demonstrate the existence of an error, but cannot demonstrate absence of errors. Our execution-based tests might discover implementation errors of a correct algorithm, and we covered operations and intrinsic functions Revol did not consider.

Given an interval vector $[x]$ and an expression $f(x)$, a Taylor model $TM_f$ is

1. $p(x)$, a polynomial in $x$ with floating-point coefficients, and
2. $[I]$, an interval

such that $f(x) \in TM_f(x) = p(x) + I$ for all $x \in [x]$. The goal of our execution-based testing was to find examples for which containment of *point* evaluation failed, i.e., $x \in [x]$ for which $f(x)$ is not in $TM_f(x)$. We did not consider the weaker range bound test: $f([x]) \in TM_f([x])$. By inclusion monotonicity, if $f(x) \in TM_f(x)$ for all $x \in [x]$, then $f([x]) \in TM_f([x])$. The point evaluation challenges might discover an error which could be masked by even slight interval overestimation in the interval evaluation challenge.

### 9.1 Verification Process

COSY's Taylor model arithmetic can be verified using COSY's interval arithmetic to verify COSY's Taylor model arithmetic. All the comparison is done inside COSY. Alternatively, we can use Maple as a referee. Both of the tests are rigorous. The second test might detect containment failures the first one does not, but it is difficult to communicate the required information to Maple. We would have to communicate sparse structure of the Taylor model and binary values of its coefficients. The first test is much faster, and it is the approach we used.

**Taylor Model Verification:**

1. Evaluate the function $f$ over the domain $[x]$.
   For example: $f = \cos(3.14 + 1.57 * x)$ on $[x] = [-1, 1]$.
2. Construct the Taylor model expression of $f$ (TM_EXPR) in COSY.
   TM_EXPR := COS(-3.14 * TM_ONE + (1.57 * TM_ONE) * TM_INDEP);

TM_ONE is Taylor model for the constant ONE. It is used to convert constants such as -3.14 and 1.57 into Taylor models.
TM_INDEP is a Taylor model for the independent variable.

3. Construct the interval expression of $f$ (INL_EXPR) in COSY.
   IVL_EXPR := COS(INTV(-3.14, -3.14) + INTV(1.57, 1.57) * VAR1);
   VAR1 is the interval independent variable.
4. Choose a point $z \in [x]$ and convert it to a tight interval $[z]$ using COSY's interval constructor.
5. Evaluate the polynomial part of the Taylor model expression (TM_EXPR) on the tight interval ($[z]$) and add the remainder bound.
6. Evaluate the interval expression (IVL_EXPR) on the tight interval ($[z]$).
7. Compare the results of 5) and 6).
   **If the intervals are disjoint, there is an error.**

### 9.2 Testing Scope

We designed test cases to evaluate the COSY operations of +, −, ×, sin, cos, tan, asin, acos, atan, sinh, cosh, tanh, log, exp, sqrt, sqr, isqrt, and unary + and −. Taylor model operations combine their operand polynomials and interval remainder bounds using floating point arithmetic to the extent possible and guaranteeing that the resulting Taylor model preserves containment. We tested Taylor models with both general domains for the independent variables and domains normalized to $[-1, 1]^n$ at dimension 1 (13 expressions): order 1, ..., 20; dimension 2 (20 expressions): order 1, ..., 18; and dimension 7 (21 expressions): order 1, 2, 3, and 4. "Dimension" denotes the number of independent variables, and "order" is the order of the Taylor model polynomial. The Taylor models were challenged at the corner points of $n$-dimensional boxes and at a few interior points. As for the interval tests, we expect errors to be most visible at the boundaries. Here is pseudo-code for these tests:

```
Loop for general and normalized domain
    Dimension = 1; Loop for order = 1, ..., 20
        Loop for 9 challenge points
            Loop for Taylor model 1 ... 13
                Pass to 149 unary operations
                Pass to 69 binary operations
    Dimension = 2; Loop for order = 1, ..., 18
        Loop for 25 challenge points
            Loop for Taylor model 1 ... 20
                Pass to 149 unary operations
                Pass to 69 binary operations
    Dimension = 7; Loop for order = 1, 2, 3, 4
        Loop for 256 challenge points
            Loop for Taylor model 1 ... 21
                Pass to 149 unary operations
                Pass to 69 binary operations
```

This represents more than 300,000 Taylor models challenged at a total of over 14 million points. That test suite required about eight hours on the 400 Mhz Intel Celeron machine described in Sect. 8. In constructing test cases, we considered order, dimension, normalization, domain, challenge points in the domain, sparsity, oscillation, simplicity, and special numbers to create at least one test case from each test case equivalence class. We adopted the same philosophy as in the interval tests that the test case is the internal binary form of the expression constructed from approximate ASCII representations.

A second test suite used 11 expressions such as

1. $\cos(-3.141592653590006 + 1.570796326794687\ x_1)$;
2. $\sin(-4.712388980384691 + 1.570796326794690\ x_1)$;
3. $\text{asin}\ (0.0009999999999999983\ x_1)$;
4. $\text{asin}\ (-0.4935 + 0.003499999999999997\ x_1)$;
5. $\text{asin}\ (0.0004999999999999989\ x_1 + 0.0004999999999999989\ x_2 x_5)$;

Loop for general and normalized domain
  Dimension = 1; Loop for order = 1, 7, 15, 17, 20
    Loop for 8 challenge points
      Loop for expression 1 ... 5
  Dimension = 2; Loop for order = 1, 7, 15, 17, 18
    Loop for 25 challenge points
      Loop for expression 1 ... 9
  Dimension = 7; Loop for order = 1, 2, 3, 4
    Loop for 256 challenge points
      Loop for expression 1 ... 11

This represents 228 Taylor models challenged at more than 25,000 points. This test required about 90 seconds and disclosed violations of containment in sin and cos and in asin and acos.

### 9.3 Containment Error in sin and cos

We found a violation of containment error in sin and cos (examples 1 and 2 above) in arguments of dimensions 1 and 2 with order 17 at $x_1$ near -1.
**Cause:** In the test environment, integer arithmetic used internally by COSY overflows and wraps from positive to negative with no alert, warning, or trap.
**Solution:** Replace some integer arithmetic in the sin and cos modules by double precision. The remaining COSY code was carefully scanned to be sure there were no similar use of integer arithmetic. The May 2, 2003, version of COSY runs the test cases as expected.

### 9.4 Containment Error in asin and acos

We found several violation of containment errors in asin (examples 3 - 5 above).
**Cause:** In one case in the asin module, some coefficients were multiplied by $[0, h]$ instead of $[-h, h]$.
**Solution:** Correct the coding error. The May 2, 2003, version of COSY runs the test cases as expected.

## 10 Conclusions and Extensions

Testing software of this complexity is itself a complex task. One needs to develop test cases that distinguish subtle errors. For interval packages, one must present to the software under test cases free from possible roundoff, and one similarly must guard against roundoff in specifying the expected result.

Effective testing of interval and Taylor model arithmetic in COSY is difficult because the conservative outward rounding of interval arithmetic can mask subtle errors. Simple test cases were successful (found errors) where more complicated tests had failed. For example, we found Taylor model errors in sin and in asin, although extensive $\sin(\text{asin}(x))$ and $\text{asin}(\sin(x))$ tests had passed. Similarly, asymmetric tests seemed to be more powerful. The error in sin and cos appeared only for order 17 because the remainder has the form $[0, \delta]$ rather than $[-\delta, \delta]$. The error is present in other orders, but it is hidden by slight excess widths introduced by repeated outward roundings.

Although our test suites for both interval and Taylor model arithmetics are large, they are neither comprehensive nor exhaustive. For example, one might port Gonnet's floating point tests from
`www.inf.ethz.ch/personal/gonnet/FPAccuracy/Analysis.html`. Gonnet's is a demanding test for the accuracy of double precision intrinsic functions. He uses challenge points known to be problematic or for which evaluation values are known to be problematic. Gonnet's additional values may disclose errors in interval or Taylor model evaluation.

Execution based testing cannot show the absence of errors, but can only demonstrate their presence. While we prefer to see no errors in our programs, especially in programs that claim to compute with guarantees, we think it speaks well of the authors of the COSY, Sun F95, and INTLAB packages we tested that we found relatively few errors. We cannot guarantee that they are now error-free, but our tests should appreciably raise the level of confidence in their reliability.

Complete software for the testing reported here is available from
`www.eng.mu.edu/corlissg/Pubs/COSYtest`.

**We encourage users of COSY and most other software packages to check author/vendor web sites regularly for possible updates and patches.**

**Acknowledgment.** This work was funded in part from Michigan State University. The testing could not have been completed without the assistance of Martin Berz and Kyoko Makino. The article is based on a talk presented at the Dagstuhl Seminar on Numerical Software with Result Verification, January 20, 2003. We appreciate the referees' many helpful comments.

## References


1. Martin Berz. COSY INFINITY Version 8 reference manual. Technical Report MSUCL–1088, National Superconducting Cyclotron Laboratory, Michigan State University, East Lansing, MI 48824, 1997.
2. Martin Berz. COSY INFINITY web page, 2000. `cosy.pa.msu.edu`.
3. George F. Corliss. Performance of self-validating quadrature. In Pat Keast and Graeme Fairweather, editors, *Proceedings of the NATO Advanced Workshop on Numerical Integration: Recent Developments, Software, and Applications*, pages 239–259. Reidel, Boston, 1987.
4. George F. Corliss. Comparing software packages for interval arithmetic, 1993. Presented at SCAN '93, September 1993, Vienna.
5. Laurence C. W. Dixon and G. P. Szegö. *Towards Global Optimization 2.* North-Holland, 1978.
6. Cem Kaner, Jack Falk, and Hung Quoc Nguyen. *Testing Computer Software, Second edition.* Wiley, New York, 1999.
7. R. Baker Kearfott, M. Dawande, K.-S. Du, and Chenyi Hu. INTLIB: A portable FORTRAN 77 interval standard function library. *ACM Transactions on Mathematical Software*, 1994.
8. Edward Kit. *Software Testing in the Real World: Improving the Process.* Addison Wesley, 1995.
9. Kyoko Makino and Martin Berz. Taylor models and other validated functional inclusion methods. *International Journal of Pure and Applied Mathematics*, 4(4):379–456, 2003. `bt.pa.msu.edu/pub/`.
10. Sun Microsystems. Sun ONE Studio 7 (formerly Forte Developer 7) Interval Arithmetic, 2002.
11. Glenford Myers. *The Art of Software Testing.* Wiley, New York, 1979.
12. Nathelie Revol, Kyoko Makino, and Martin Berz. Taylor models and floating-point arithmetic: Proof that arithmetic operations are validated in COSY. LIP report RR 2003-11, University of Lyon, France, 2003. MSU HEP report 30212, submitted, `bt.pa.msu.edu/pub/`.
13. Siegfried M. Rump. Fast and parallel interval arithmetic. *BIT*, 39(3):539–560, 1999.
14. Siegfried M. Rump. INTLAB - INTerval LABoratory. In Tibor Csendes, editor, *Developments in Reliable Computing*, pages 77–104. Kluwer Academic Publishers, Dordrecht, 1999. `www.ti3.tu-harburg.de/rump/intlab`.
15. Siegfried M. Rump. Rigorous and portable standard functions. *BIT*, 41(3):540–562, 2001.
16. James A. Whittaker. *How to Break Software: A Practical Guide to Testing.* Addison Wesley, Boston, 2003.

# Nonlinear Parameter and State Estimation for Cooperative Systems in a Bounded-Error Context

Michel Kieffer and Eric Walter

Laboratoire des Signaux et Systèmes
CNRS – Supélec – Université Paris-Sud
Plateau de Moulon, 91192 Gif-sur-Yvette, France
{kieffer, walter}@lss.supelec.fr

**Abstract.** This paper is about guaranteed nonlinear parameter and state estimation. *Sets* are computed that contain all possible values of the parameter (or state) vector given bounds on the acceptable errors. The main requirement is that the dynamical equations describing the evolution of the model can be bounded between cooperatives models, *i.e.*, models such that the off-diagonal entries of their Jacobian matrix remain positive. The performances and limitations of the techniques proposed are illustrated on a nonlinear compartmental model.

## 1 Introduction

Parameter and state estimation problems are encountered when modeling a process that involves uncertain quantities to be estimated from measurements.

Consider a system with known input vector $\mathbf{u}(t)$ and output vector $\mathbf{y}(t)$. Assume it is described by a model with the same input and consisting of a *dynamical state equation*

$$\mathbf{x}'(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{p}, \mathbf{w}(t), \mathbf{u}(t)), \tag{1}$$

with initial condition

$$\mathbf{x}(0) = \mathbf{x}_0(\mathbf{p}), \tag{2}$$

and an *observation equation*

$$\mathbf{y}_{\mathrm{m}}(\mathbf{x}(t), \mathbf{p}, t) = \mathbf{g}(\mathbf{x}(t), \mathbf{p}) + \mathbf{v}(t), \tag{3}$$

where the vector $\mathbf{x}$ is the state of the model, $\mathbf{x}'$ is its derivative with respect to time, $\mathbf{p}$ is a vector of unknown parameters and $\mathbf{w}$ and $\mathbf{v}$ are vectors of state perturbations and measurement noise. State perturbations account for the fact that (1) is only an approximation of reality. Measurement noise is introduced in (3) to represent the imperfection of the sensors measuring the outputs of the system.

R. Alt et al. (Eds.): Num. Software with Result Verification, LNCS 2991, pp. 107–123, 2004.

Finding an estimate $\widehat{\mathbf{p}}$ for $\mathbf{p}$ such that the output of the model $\mathbf{y}_{\mathrm{m}}(\mathbf{x}(t), \mathbf{p}, t)$ is an acceptable approximation of the output of the system $\mathbf{y}(t)$ is called *parameter estimation*. Similarly, finding an estimate $\widehat{\mathbf{x}}(t)$ for $\mathbf{x}(t)$ is called *state estimation*. When the two problems are solved simultaneously, one speaks of joint parameter and state estimation.

This paper focuses on bounded-error estimation. In this context, it is assumed that the perturbations and noise are bounded with known bounds and one looks for the *set* of all parameter (or state) vectors that are consistent with the experimental data and these bounds. Specific methods are available for the case where the model output is linear in the parameter vector (or the initial state vector), and we shall concentrate on the more difficult nonlinear case.

The first part of this paper deals with recursive state estimation of a continuous-time model assuming that the system output is measured at discrete time instants. An idealized algorithm is presented first. As the Kalman filter, it alternates prediction and correction. The prediction step computes the evolution of the set corresponding to the state estimate. The correction step takes place as soon as a measurement of the system output becomes available. It computes the intersection of the previously calculated set with the set of all state vectors that are consistent with this measurement and the bounds on the measurement error.

The second part of this paper deals with parameter estimation. It is much shorter since the same type of tools are used as in the correction step of state estimation.

The specific difficulty when estimating the parameter or state vector of a continuous-time state-space model is that most often there is no closed-form solution of the differential state equation, which complicates the required obtainment of an inclusion function for this solution. Guaranteed interval integration could in principle be used, but it becomes notoriously pessimistic as soon as the uncertainty in the parameters and initial conditions is large, as required here. We shall see that a much less pessimistic numerical inclusion function for the model output can be evaluated if the differential model can be enclosed between two cooperative systems.

State and parameter estimations are illustrated with compartmental models, widely used in biology.

## 2 Recursive State Estimation

### 2.1 Introduction

In this section, an estimate $\widehat{\mathbf{x}}(t)$ for $\mathbf{x}(t)$ is to be obtained such that $\mathbf{y}_{\mathrm{m}}(\mathbf{x}(t), \mathbf{p}, t)$, the output of the model $(1)-(3)$, is an acceptable approximation of the output $\mathbf{y}(t)$ of the system.

Note that the parameter vector $\mathbf{p}$ is not necessarily known. Two approaches may be considered to estimating $\mathbf{x}(t)$ when $\mathbf{p}$ is uncertain. A first method as-

sumes some prior knowledge about the evolution of $\mathbf{p}$, described by the differential equation

$$\mathbf{p}'(t) = \mathbf{f}_{\mathrm{p}}(\mathbf{x}(t), \mathbf{p}(t), \mathbf{w}_{\mathrm{p}}(t), \mathbf{u}(t)), \ \mathbf{p}(0) = \mathbf{p}_0, \tag{4}$$

where $\mathbf{w}_{\mathrm{p}}$ plays the same role as $\mathbf{w}$ for $\mathbf{x}$. (If the parameters are assumed to be constant, the differential equation in (4) boils down to $\mathbf{p}'(t) = \mathbf{0}$.) Defining an *extended state* vector

$$\mathbf{x}^{\mathrm{e}}(t) = \left(\mathbf{x}^{\mathrm{T}}(t), \mathbf{p}^{\mathrm{T}}(t)\right)^{\mathrm{T}},$$

makes it possible to obtain from (1) and (4) an *extended dynamical state equation*

$$\begin{pmatrix} \mathbf{x}(t) \\ \mathbf{p}(t) \end{pmatrix}' = \begin{pmatrix} \mathbf{f}(\mathbf{x}(t), \mathbf{p}, \mathbf{w}(t), \mathbf{u}(t)) \\ \mathbf{f}_{\mathrm{p}}(\mathbf{x}(t), \mathbf{p}(t), \mathbf{w}_{\mathrm{p}}(t), \mathbf{u}(t)) \end{pmatrix}, \ \begin{pmatrix} \mathbf{x}(0) \\ \mathbf{p}(0) \end{pmatrix} = \begin{pmatrix} \mathbf{x}_0(\mathbf{p}_0) \\ \mathbf{p}_0 \end{pmatrix}$$

or equivalently

$$(\mathbf{x}^{\mathrm{e}}(t))' = \mathbf{f}^{\mathrm{e}}(\mathbf{x}^{\mathrm{e}}(t), \mathbf{w}(t), \mathbf{w}_{\mathrm{p}}(t), \mathbf{u}(t)), \ \mathbf{x}^{\mathrm{e}}(0) = \mathbf{x}_0^{\mathrm{e}}(\mathbf{p}_0).$$

With this approach, which corresponds to joint parameter and state estimation, the distinction between state variables and parameters disappears, and the situation is formally equivalent to the case with no uncertain parameters.

A second method, which is the one to be employed in this paper, integrates the uncertainty about $\mathbf{p}$ in the state perturbations and measurement noise. No attempt will then be made at estimating $\mathbf{p}$, which will be considered as a *nuisance* parameter vector.

When $\mathbf{f}$ and $\mathbf{g}$ in (1) and (3) are linear functions of the state vector and when moreover the perturbations and noise are additive and receive a probabilistic description by their means and covariances, Kalman filtering [15] is the standard approach to state estimation. In the context of bounded errors, many tools are also available, see, *e.g.*, [2], [20] and [23].

In a nonlinear context, the methodology is far less developed. When uncertainty is explicitly taken into account, this is most often by using an extended Kalman filter [4] based on the linearization of (1) around the state trajectory. It is well known that this type of filter may fail to produce a useful estimate of the state vector, and that the characterization of the uncertainty in this estimate is not reliable.

Guaranteed state bounding is an attractive alternative, which has been considered in a discrete-time context in [11] and [18]. All state vectors consistent with the data, model and bounds are enclosed in a *subpaving*, consisting of a union of disconnected boxes. In a continuous-time context, a state estimator for models such as that described by (1) and (3) was proposed in [10] but with no state perturbation or parameter uncertainty taken into account. Techniques bounding the state of continuous-time systems with poorly known state equations and inputs are presented in [1] and [6], with applications in waste processing. Provided specific assumptions are satisfied by the signs of the entries of $\partial\mathbf{f}/\partial\mathbf{x}$, *interval observers* can be built. An interval observer is a pair of classical point observers

computing a *box enclosure* of the state $\mathbf{x}$ at any given time based on lower and upper bounds for each of the uncertain variables.

In this paper, interval observers and the recursive state estimation algorithm presented in [12] and [18] are combined to enclose the state $\mathbf{x}(t)$ of the model $(1)-(3)$ at any given instant of time $t$ in a subpaving. This is performed recursively and can thus be implemented in real time. Preliminary results have been presented in [19].

An idealized algorithm is first proposed in Section 2.2. An implementable counterpart of this algorithm is then described in Section 2.3. The advantages and limitations of the approach are illustrated on an example in Section 2.4.

### 2.2 Idealized Algorithm

Consider the model $(1)-(3)$ and a set of sampling instants $\mathcal{T}=\{t_i\}_{i\in\mathbb{N}^*}$, such that $t_{i+1}>t_i$, at which the measurements $\mathbf{y}(t_i)$ have been collected. Initially, $\mathbf{x}(0)$ is only known to belong to some box $[\mathbf{x}_0]$. The vector $\mathbf{p}$ of uncertain parameters is assumed to be constant and to belong to some known prior box $[\mathbf{p}_0]$. The state perturbation $\mathbf{w}(t)$ is assumed to satisfy $\underline{\mathbf{w}}(t)\leqslant\mathbf{w}(t)\leqslant\overline{\mathbf{w}}(t)$ at any $t\geqslant 0$, where $[\mathbf{w}(t)]=[\underline{\mathbf{w}}(t),\overline{\mathbf{w}}(t)]$ is known for all $t$ and the inequalities are to be understood componentwise. The measurement noise $\mathbf{v}(t_i)$ is similarly assumed to belong to $[\mathbf{v}(t_i)]=[\underline{\mathbf{v}}(t_i),\overline{\mathbf{v}}(t_i)]$, known at each $t_i$. The information $\mathcal{I}(t)$ available at time $t\geqslant 0$ is given by

$$\mathcal{I}(t)=\left\{[\mathbf{x}_0],[\mathbf{p}_0],\{[\mathbf{w}(\tau)],\mathbf{u}(\tau)\}_{\tau\in[0,t]},\{[\mathbf{v}(t_i)]\}_{i=1}^{M}\right\}, \tag{5}$$

where $t_M$ is such that $t_M\leqslant t<t_{M+1}$. In this context, causal state estimation is the caracterization of the *set* $\mathcal{X}(t)$ of all values of the state $\mathbf{x}(t)$ at any time $t\geqslant 0$ that are consistent with $\mathcal{I}(t)$.

As in the Kalman filter, the idealized recursive causal state estimator consists of two steps.

For the *prediction step*, assume that $\mathcal{X}(t_i)$ is some set guaranteed to contain $\mathbf{x}(t_i)$. For any given $\mathbf{x}\in\mathcal{X}(t_i)$, let $\varphi\left(\mathbf{x},t,t_i,\mathbf{p},\{\mathbf{w}(\tau),\mathbf{u}(\tau)\}_{\tau\in[t_i,t]}\right)$ be the value at time $t$ of the flow associated with (1) that coincides with $\mathbf{x}$ at time $t_i$. Define the *predicted* set $\mathcal{X}^+(t_{i+1})$ as

$$\begin{aligned}\mathcal{X}^+(t_{i+1})=\big\{&\varphi\left(\mathbf{x},t_{i+1},t_i,\mathbf{p},\{\mathbf{w}(\tau),\mathbf{u}(\tau)\}_{\tau\in[t_i,t_{i+1}]}\right)\\ &\mid\mathbf{p}\in[\mathbf{p}_0],\mathbf{w}(\tau)\in[\mathbf{w}(\tau)],\mathbf{x}\in\mathcal{X}(t_i),\tau\in[t_i,t_{i+1}]\big\}.\end{aligned} \tag{6}$$

By construction, $\mathbf{x}(t_{i+1})\in\mathcal{X}^+(t_{i+1})$.

Now, for the *correction step*, let $[\mathbf{y}(t_{i+1})]$ be the box containing all possible values of the noise-free output when the value of the measured output is $\mathbf{y}(t_{i+1})$

$$[\mathbf{y}(t_{i+1})]=\mathbf{y}(t_{i+1})-[\mathbf{v}(t_{i+1})], \tag{7}$$

and let $\mathcal{X}^{\mathrm{o}}(t_{i+1})$ be the set of all values of the state at time $t_{i+1}$ that could have led to an observation $\mathbf{y}$ in $[\mathbf{y}(t_{i+1})]$

$$\mathcal{X}^{\mathrm{o}}(t_{i+1})=\{\mathbf{x}\in\mathbb{R}^n\mid\mathbf{g}(\mathbf{x},\mathbf{p})\in[\mathbf{y}(t_{i+1})],\mathbf{p}\in[\mathbf{p}_0]\}. \tag{8}$$

Then, the *corrected* set

$$\mathcal{X}(t_{i+1}) = \mathcal{X}^{+}(t_{i+1}) \cap \mathcal{X}^{\mathrm{o}}(t_{i+1}) \tag{9}$$

is also guaranteed to contain $\mathbf{x}(t_{i+1})$ (see Figure 1).

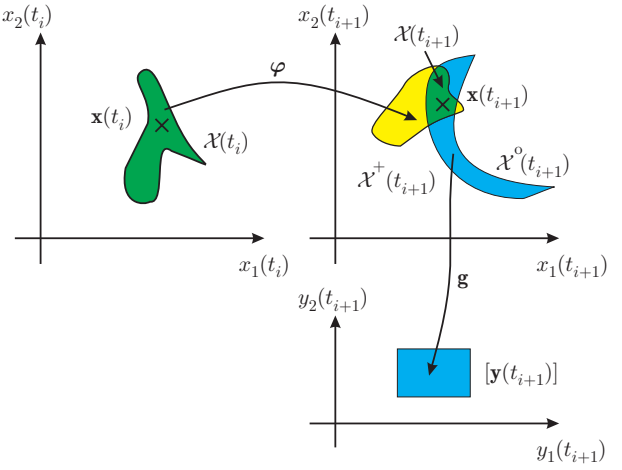



**Fig. 1.** Idealized state estimation

This is summarized in the following idealized algorithm.

**Algorithm 1**
*For* $i = 0$ *to* $N-1$, *do* {

1. *Prediction*: *evaluate* $\mathcal{X}^{+}(t_{i+1})$;
2. *Correction*: $\mathcal{X}(t_{i+1}) = \mathcal{X}^{+}(t_{i+1}) \cap \mathcal{X}^{o}(t_{i+1})$; } ■

It is easy to show [16] that $\mathcal{X}(t)$ as evaluated by Algorithm 1 is the smallest set guaranteed to contain $\mathbf{x}(t)$ that can be computed from $\mathcal{I}(t)$ and (1). The next section presents the basic tools required to obtain an implementable counterpart to Algorithm 1.

### 2.3 Implementation Issues

To obtain an implementable counterpart to Algorithm 1, three main problems have to be solved.

The first one is to represent the sets $\mathcal{X}(t_i)$, $\mathcal{X}^{\mathrm{o}}(t_i)$ and $\mathcal{X}^{+}(t_i)$ in computer memory. In this paper, the description of sets using subpavings presented in [17] is used.

The second problem is the evaluation of $\mathcal{X}^{\mathrm{o}}(t_{i+1})$ during the correction step (8). An outer approximation $\widehat{\mathcal{X}}^{\mathrm{o}}(t_{i+1})$ of $\mathcal{X}^{\mathrm{o}}(t_{i+1})$ by a subpaving can be obtained using the SIVIA algorithm (see below). The precision of this outer approximation is controlled by a precision factor $\varepsilon_{\mathrm{S}}$.

The remaining problem is the solution at the prediction step of the set of IVPs required to evaluate $\mathcal{X}^{+}(t_{i+1})$. Standard guaranteed tools are available to solve IVPs such as $\{\mathbf{x}' = \mathbf{f}(\mathbf{x}, t), \mathbf{x}(0) = \mathbf{x}\}$ or $\{\mathbf{x}' = \mathbf{f}(\mathbf{x}, t), \mathbf{x}(0) \in [\mathbf{x}]\}$, see, *e.g.*, AWA ([21], [22]), COSY ([8], [9]) or VNODE ([24]). These techniques use Brouwer's fixed-point theorem to show the existence of a solution and build a Taylor expansion of the solution while bounding the remainder. However, they become very inefficient in the presence of unknown parameters or bounded state perturbations because the bounds on the remainder soon become extremely large. We shall present a more efficient approach, based on cooperativity.

**Sivia.** Using interval analysis, it is possible to provide inner and outer approximations of the set $\mathcal{X}^{\mathrm{o}}(t_{i+1})$ defined by (8), using the algorithm SIVIA (for *Set Inverter Via Interval Analysis*, see [13] and [14]) briefly recalled here.

An initial bounded search set $\mathbb{X}^{\mathrm{o}}$ guaranteed to contain $\mathcal{X}^{\mathrm{o}}(t_{i+1})$ has to be provided first. SIVIA partitions $\mathbb{X}^{\mathrm{o}}$ into three subpavings, namely $\mathbb{X}_{\mathrm{in}}$ contained in $\mathcal{X}^{\mathrm{o}}(t_{i+1})$, $\mathbb{X}_{\mathrm{out}}$ such that its intersection with $\mathcal{X}^{\mathrm{o}}(t_{i+1})$ is empty and $\mathbb{X}_{\mathrm{bound}}$ for which no conclusion could be reached.

Consider a box $[\mathbf{x}] \subset \mathbb{X}^{\mathrm{o}}$ and let $[\mathbf{g}](.)$ be an inclusion function for $\mathbf{g}(.)$.

1. If $[\mathbf{g}]([\mathbf{x}], [\mathbf{p}_0]) \subset [\mathbf{y}(t_{i+1})]$, then for any $\mathbf{x} \in [\mathbf{x}]$ and $\mathbf{p} \in [\mathbf{p}_0]$, $\mathbf{g}(\mathbf{x}, \mathbf{p}) \in [\mathbf{y}(t_{i+1})]$ and $[\mathbf{x}]$ is entirely included in $\mathcal{X}^{\mathrm{o}}(t_{i+1})$; it is thus stored in $\mathbb{X}_{\mathrm{in}}$.
2. If $[\mathbf{g}]([\mathbf{x}], [\mathbf{p}_0]) \cap [\mathbf{y}(t_{i+1})] = \emptyset$, then $\mathbf{g}([\mathbf{x}], [\mathbf{p}_0]) \cap [\mathbf{y}(t_{i+1})] = \emptyset$ and $[\mathbf{x}]$, proved to have an empty intersection with $\mathcal{X}^{\mathrm{o}}(t_{i+1})$, can be stored in $\mathbb{X}_{\mathrm{out}}$.
3. If neither of the previous tests is satisfied, then $[\mathbf{x}]$ is undetermined. If the width of such an undetermined box is larger than the precision factor $\varepsilon_{\mathrm{S}}$, then it is bisected into two subboxes $[\mathbf{x}_1]$ and $[\mathbf{x}_2]$ to which the same tests are applied. Undetermined boxes that are too small to be bisected are stored into $\mathbb{X}_{\mathrm{bound}}$.

$\mathcal{X}^{\mathrm{o}}(t_{i+1})$ is thus bracketed (in the sense of inclusion) between $\mathbb{X}_{\mathrm{in}}$ and $\widehat{\mathcal{X}}^{\mathrm{o}}(t_{i+1}) = \mathbb{X}_{\mathrm{in}} \cup \mathbb{X}_{\mathrm{bound}}$. The volume of the uncertainty subpaving $\mathbb{X}_{\mathrm{bound}}$ may be reduced, at the cost of increasing computational effort. Note that there is actually no need to store $\mathbb{X}_{\mathrm{out}}$.

**Inclusion functions based on cooperativity.** This section aims at defining an implementable procedure for computing $\mathcal{X}^{+}(t_{i+1})$ defined by (6) based on the concept of cooperativity [26].

**Definition 1.** *A dynamical system*

$$\mathbf{x}' = \mathbf{f}(\mathbf{x}, t)$$

*is* cooperative *over a domain* $\mathcal{D}$ *if*

$$\frac{\partial f_i}{\partial x_j}(\mathbf{x},t) \geqslant 0, \text{ for all } i \neq j,\ t \geqslant 0 \text{ and } \mathbf{x} \in \mathcal{D},$$

i.e., *if all off-diagonal entries of the Jacobian matrix of* $\mathbf{f}$ *are non-negative for all* $t \geqslant 0$ *and* $\mathbf{x} \in \mathcal{D}$.

The following theorem, which is a reformulation of a result in [26], will be used to obtain an enclosure for $\mathbf{x}(t)$ in (1). This enclosure will be instrumental in the implementation of the prediction step.

**Theorem 1.** *If there exists a pair of cooperative systems*

$$\underline{\mathbf{x}}' = \underline{\mathbf{f}}\left(\underline{\mathbf{x}},\underline{\mathbf{p}},\overline{\mathbf{p}},t\right) \text{ and } \overline{\mathbf{x}}' = \overline{\mathbf{f}}\left(\overline{\mathbf{x}},\underline{\mathbf{p}},\overline{\mathbf{p}},t\right) \tag{10}$$

*satisfying*

$$\underline{\mathbf{x}}_0 \leqslant \mathbf{x}(0) \leqslant \overline{\mathbf{x}}_0$$

*and*

$$\underline{\mathbf{f}}\left(\underline{\mathbf{x}},\underline{\mathbf{p}},\overline{\mathbf{p}},t\right) \leqslant \mathbf{f}(\mathbf{x},\mathbf{p},\mathbf{w},\mathbf{u}) \leqslant \overline{\mathbf{f}}\left(\overline{\mathbf{x}},\underline{\mathbf{p}},\overline{\mathbf{p}},t\right),$$

*for all* $\mathbf{p} \in \left[\underline{\mathbf{p}},\overline{\mathbf{p}}\right]$, $\mathbf{w}(t) \in [\underline{\mathbf{w}}(t),\overline{\mathbf{w}}(t)]$, $t \geqslant 0$ *and* $\mathbf{x} \in \mathcal{D}$ *then the state of (1) satisfies*

$$\underline{\mathbf{x}}(t) \leqslant \mathbf{x}(t) \leqslant \overline{\mathbf{x}}(t), \text{ for all } t \geqslant 0,$$

*where* $\underline{\mathbf{x}}(t) = \underline{\phi}\left(\underline{\mathbf{x}}_0,\underline{\mathbf{p}},\overline{\mathbf{p}},t\right)$ *is the flow associated with* $\left\{\underline{\mathbf{x}}' = \underline{\mathbf{f}}\left(\underline{\mathbf{x}},\underline{\mathbf{p}},\overline{\mathbf{p}},t\right), \underline{\mathbf{x}}(0) = \underline{\mathbf{x}}_0\right\}$ *and* $\overline{\mathbf{x}}(t) = \overline{\phi}\left(\overline{\mathbf{x}}_0,\underline{\mathbf{p}},\overline{\mathbf{p}},t\right)$ *is the flow associated with* $\left\{\overline{\mathbf{x}}' = \overline{\mathbf{f}}\left(\overline{\mathbf{x}},\underline{\mathbf{p}},\overline{\mathbf{p}},t\right), \overline{\mathbf{x}}(0) = \overline{\mathbf{x}}_0\right\}$. ♦

For any $t \geqslant 0$, the box-valued function

$$[\phi]\left(\underline{\mathbf{x}}_0,\overline{\mathbf{x}}_0,\underline{\mathbf{p}},\overline{\mathbf{p}},t\right) = \left[\underline{\phi}\left(\underline{\mathbf{x}}_0,\underline{\mathbf{p}},\overline{\mathbf{p}},t\right), \overline{\phi}\left(\overline{\mathbf{x}}_0,\underline{\mathbf{p}},\overline{\mathbf{p}},t\right)\right]$$

is thus an inclusion function for $\mathbf{x}(t)$, the solution of (1). However, this function is difficult to evaluate, as usually no explicit expressions are available for $\underline{\phi}(.)$ and $\overline{\phi}(.)$. Interval analysis provides tools for computing guaranteed outer approximations of the solution of initial value problems, see, *e.g.*, [24]. Using these techniques, it becomes possible to compute tight enclosures of $\underline{\phi}\left(\underline{\mathbf{x}}_0,\underline{\mathbf{p}},\overline{\mathbf{p}},t\right)$ and $\overline{\phi}\left(\overline{\mathbf{x}}_0,\underline{\mathbf{p}},\overline{\mathbf{p}},t\right)$ as

$$\left[\underline{\phi}\left(\underline{\mathbf{x}}_0,\underline{\mathbf{p}},\overline{\mathbf{p}},t\right)\right] = \left[\underline{\underline{\phi}\left(\underline{\mathbf{x}}_0,\underline{\mathbf{p}},\overline{\mathbf{p}},t\right)}, \overline{\underline{\phi}\left(\underline{\mathbf{x}}_0,\underline{\mathbf{p}},\overline{\mathbf{p}},t\right)}\right]$$

and

$$\left[\overline{\phi}\left(\overline{\mathbf{x}}_0,\underline{\mathbf{p}},\overline{\mathbf{p}},t\right)\right] = \left[\underline{\overline{\phi}\left(\overline{\mathbf{x}}_0,\underline{\mathbf{p}},\overline{\mathbf{p}},t\right)}, \overline{\overline{\phi}\left(\overline{\mathbf{x}}_0,\underline{\mathbf{p}},\overline{\mathbf{p}},t\right)}\right].$$

The function

$$[[\phi]]\left([\mathbf{x}_0],[\mathbf{p}],t\right) = \left[\underline{\underline{\phi}\left(\underline{\mathbf{x}}_0,\underline{\mathbf{p}},\overline{\mathbf{p}},t\right)}, \overline{\overline{\phi}\left(\overline{\mathbf{x}}_0,\underline{\mathbf{p}},\overline{\mathbf{p}},t\right)}\right] \tag{11}$$

is thus such that

$$\varphi\left(\mathbf{x}, t, 0, \mathbf{p}, \{\mathbf{w}(\tau), \mathbf{u}(\tau)\}_{\tau\in[0,t]}\right) \in [[\phi]]\left([\mathbf{x}_0], [\mathbf{p}], t\right),$$

for $\mathbf{x}_0 \in [\mathbf{x}_0]$, $\mathbf{p} \in \left[\underline{\mathbf{p}}, \overline{\mathbf{p}}\right]$, $\mathbf{w}(t) \in [\underline{\mathbf{w}}(t), \overline{\mathbf{w}}(t)]$, $t \geqslant 0$, and is therefore an inclusion function for the solution $\mathbf{x}(t)$ of (1), which can be numerically evaluated for any $t \geqslant 0$.

Let $\Phi([\mathbf{x}_0], [\mathbf{p}_0], t)$ be the set of all $\mathbf{x}(t)$ that can be traced back to an initial condition in $[\mathbf{x}_0]$ according to (1) with a parameter vector $\mathbf{p} \in [\mathbf{p}_0]$. Then if the conditions of Theorem 1 are verified

$$\mathbf{x}(t) \in \Phi([\mathbf{x}_0], [\mathbf{p}_0], t) \subset [[\phi]]([\mathbf{x}_0], [\mathbf{p}_0], t) \text{ for any } t \geqslant 0.$$

Interval observers using $[[\phi]]([\mathbf{x}_0], [\mathbf{p}_0], t)$ are only able to provide a box containing $\Phi([\mathbf{x}_0], [\mathbf{p}_0], t)$. However, $\Phi([\mathbf{x}_0], [\mathbf{p}_0], t)$ is usually not a box, see Figure 2. Here, we propose to improve the accuracy of the approximation of $\Phi([\mathbf{x}_0], [\mathbf{p}_0], t)$ by enclosing it in a subpaving using the ImageSp algorithm presented in [17] and [18] and briefly recalled now.

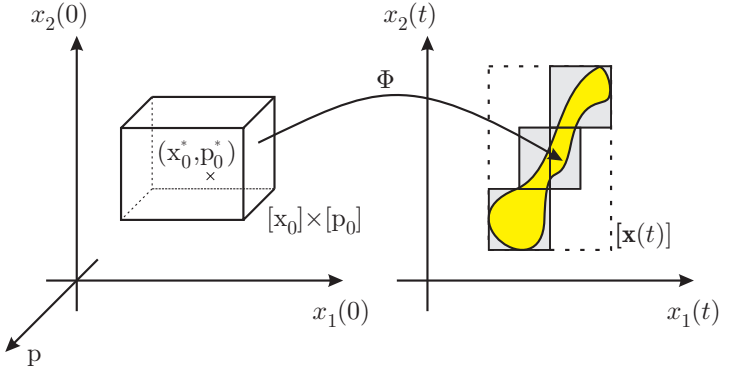



**Fig. 2.** State estimates obtained with an interval observer (box in dashed lines) and an approximate set observer (union of light grey boxes on the right)

The algorithm ImageSp consists of three steps. First, $[\mathbf{x}_0]$ is *minced*, *i.e.*, divided into boxes of width less than a given precision factor $\varepsilon_I$. Then, the images of all these boxes are evaluated using an inclusion function of $\Phi$ and stored into a list $\mathcal{L}$ of image boxes. Finally, all boxes in $\mathcal{L}$ are merged to obtain a subpaving guaranteed to contain $\Phi([\mathbf{x}_0], [\mathbf{p}_0], t)$. The time needed to obtain this subpaving and the precision of the description (measured, *e.g.*, using a Hausdorff distance to the approximated set) increase when the precision factor $\varepsilon_I$ decreases.

The only requirement for ImageSp is the availability of an inclusion function for $\Phi$, which is obtained using $[[\phi]]$.

*Remark 1.* In the previous presentation, only $[\mathbf{x}_0]$ has been minced, but if one considered the extended state vector $\mathbf{x}^{\mathrm{e}}(t) = (\mathbf{x}^{\mathrm{T}}(t), \mathbf{p}^{\mathrm{T}})^{\mathrm{T}}$, with initial condition $\mathbf{x}_0^{\mathrm{e}} \in [\mathbf{x}_0^{\mathrm{e}}] = ([\mathbf{x}_0]^{\mathrm{T}}, [\mathbf{p}_0]^{\mathrm{T}})^{\mathrm{T}}$, the mincing could have been performed on $[\mathbf{x}_0^{\mathrm{e}}]$. When $[\mathbf{p}_0]$ is a non-degenerate interval, the resulting enclosure is usually more precise, but obtained with an increased computational effort.

**Implementable algorithm.** Assume that $\mathcal{X}(t)$ has to be evaluated, with $t$ such that $t = t_N$ and that $\mathcal{X}(0) = [\mathbf{x}_0]$. The following algorithm is a counterpart to Algorithm 1.

**Algorithm 2**
*For* $i = 0$ *to* $N - 1$, *do* {

1. *Prediction: evaluate* $\widehat{\mathcal{X}}^{+}(t_{i+1})$ *using* ImageSp*;*
2. *Correction: evaluate* $\widehat{\mathcal{X}}(t_{i+1})$ *using* Sivia *with initial search domain* $\widehat{\mathcal{X}}^{+}(t_{i+1})$*;* } ■

Convergence properties have been established in [14] for Sivia and in [18] for ImageSp. The convergence of Algorithm 2 depends not only on $\varepsilon_{\mathrm{S}}$ and $\varepsilon_{\mathrm{I}}$, but also on the quality of the enclosure of (1) provided by the pair of cooperative systems.

### 2.4 Example

Compartment models are frequently used in pharmacokinetics, chemistry or biology. They consist of a collection of tanks containing material. These tanks, represented by circles exchange material between them and with the rest of the world, as materialized by arrows. Each tank is supposed to be homogeneous and the quantity of material in compartment $i$ is denoted by $x_i$. Many types of compartment models are available (see, *e.g.*, [5]), but all share the property that the evolution of the quantities of material in the compartments is governed by a state equation that may easily be enclosed between cooperative models.

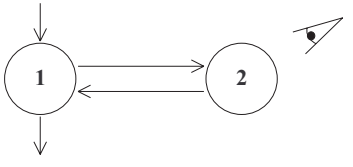



**Fig. 3.** Two-compartment model

To facilitate presentation, we shall consider a simple academic example, the structure of which is nevertheless typical of nonlinear compartmental models.

Assume that the evolution of the vector of quantities of material $\mathbf{x} = (x_1, x_2)^{\mathrm{T}}$ in the compartments of the model of Figure 3 is given by

$$\begin{cases} x_1' = -\dfrac{p_1 x_1}{1 + p_2 x_1} - p_3 x_1 + p_4 x_2 + u, \\ x_2' = \dfrac{p_1 x_1}{1 + p_2 x_1} - p_4 x_2, \end{cases} \tag{12}$$

and that only $x_2$ is measured, according to the measurement equation

$$y(t_i) = (1 + e_1)\, x_2(t_i), \tag{13}$$

where $e_1$ is bounded.

All parameters are supposed to be known except for $p_1 \in [p_1] = [0.9, 1.1]$ and the initial state of the system is only known to belong to $[\mathbf{x}_0] = [0, 1] \times [0, 1]$. Data have been simulated with the actual value of the parameter vector $\mathbf{p}^* = (1, 4/3, 1/2, 1/4)^{\mathrm{T}}$, $\mathbf{x}_0^* = (0, 0)^{\mathrm{T}}$ and

$$\begin{cases} u(t) = 1 \text{ when } 0 \leqslant t < 1 \text{ and } 2.5 \leqslant t < 3.5 \\ u(t) = 0 \text{ elsewhere.} \end{cases}$$

At 20 regularly-spaced time instants from 0.5s to 10s, a measurement of $x_2$ is taken and corrupted by a bounded relative noise $e_1 \in [-0.1, 0.1]$. The problem is to determine the set of all values of the state vector that are consistent with the model, the measurements and their uncertainty.

The dynamical model (12) can be bounded by the two models

$$\begin{cases} x_1' = -\dfrac{\overline{p}_1 x_1}{1 + p_2^* x_1} - p_3^* x_1 + p_4^* x_2 + u, \\ x_2' = \dfrac{\underline{p}_1 x_1}{1 + p_2^* x_1} - p_4^* x_2, \end{cases} \tag{14}$$

and

$$\begin{cases} x_1' = -\dfrac{\underline{p}_1 x_1}{1 + p_2^* x_1} - p_3^* x_1 + p_4^* x_2 + u, \\ x_2' = \dfrac{\overline{p}_1 x_1}{1 + p_2^* x_1} - p_4^* x_2, \end{cases} \tag{15}$$

which are easily proved to be cooperative, as the vector of quantities of material is positive. Moreover, as $[\mathbf{x}_0] = [0, 1] \times [0, 1]$, the conditions of Theorem 1 are satisfied. Thus, the prediction part of the recursive state estimation algorithm presented in Section 2.3 can be implemented using an inclusion function built from (14) and (15) and evaluated by guaranteed numerical integration.

The correction step involves the SIVIA algorithm presented in Section 2.3. The bounds for $v(t_i)$ in (3) are computed knowing that

$$y(t_i) = \mathbf{g}(\mathbf{x}^*(t_i), \mathbf{p}^*)\,(1 + e_1(t_i)),$$

where $e_1(t_i)$ is a realization of a random variable with support restricted to $[-0.1, 0.1]$ and

$$\mathbf{g}(\mathbf{x}, \mathbf{p}) = x_2.$$

Thus

$$y(t_i) \in \mathbf{g}(\mathbf{x}^*(t_i), \mathbf{p}^*)(1 + [-0.1, 0.1])$$

and

$$\begin{aligned}\mathbf{g}(\mathbf{x}^*(t_i), \mathbf{p}^*) &\in \frac{1}{1 + [-0.1, 0.1]} y(t_i) \\ &\in (1 + [-0.081, 0.112])\, y(t_i)\end{aligned}$$

At each measurement time, the measurement noise is thus known to belong to

$$[v(t_i)] = [-0.081, 0.112]\, y(t_i).$$

All resulting intervals guaranteed to contain the noise-free output of the system are represented on Figure 4.

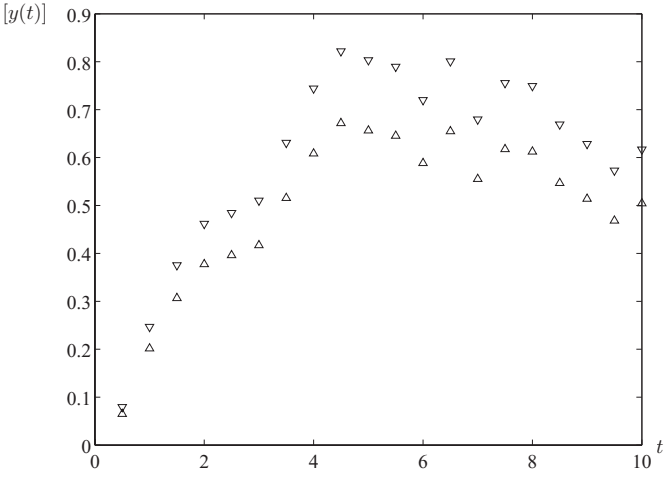



**Fig. 4.** Intervals guaranteed to contain the true values of $x_2(t_i)$

Two simulations have been performed, both with the algorithm described in Section 2.3, but with differing prediction step. The first one (Case *a*) is performed with a direct guaranteed integration of (12) taking into account the uncertain parameters. The second (Case *b*) involves an inclusion function built with (14) and (15). Guaranteed integration has been performed using the VNODE package, see [24]. The lower and upper bounds of the smallest boxes enclosing the predicted and corrected sets are represented for each measurement time on Figure 5

**Table 1.** Simulation results for recursive state estimation

| | Case $a$ | Case $b$ | |
|---|---|---|---|
| $\varepsilon_{\mathrm{S}} = \varepsilon_{\mathrm{I}}$ | 0.025 | 0.025 | 0.05 |
| Computing time ($s$) | 31 | 50 | 13 |
| Volume of $\widehat{\mathcal{X}}(10)$ | 0.0034 | 0.0021 | 0.0035 |

(for $x_1$, the lower bounds of the predicted sets coincide with the lower bounds of the corrected sets). All computations have been performed with $\varepsilon_{\mathrm{S}} = \varepsilon_{\mathrm{I}}$ on an Athlon 1800+ and the results are summarized in Table 1.

In both cases, 90% of the computing time is spent during the first prediction step, when the knowledge about the initial value of the state is poor; the last steps take less than 0.1 s each. The enclosures obtained in Case $b$ are more accurate than in Case $a$ for the same value of $\varepsilon_{\mathrm{S}}$ and $\varepsilon_{\mathrm{I}}$, and obtained much faster when a given final precision is required. These results illustrate the efficiency of the bounding approach using cooperative systems.

## 3 Bounded-Error Parameter Estimation

In this section, an estimate $\widehat{\mathbf{p}}$ for $\mathbf{p}$ is to be obtained such that the output $\mathbf{y}_{\mathrm{m}}(\mathbf{x}(t), \mathbf{p}, t)$ of the model (1) − (3) is an acceptable approximation of the output $\mathbf{y}(t)$ of the system. Here, the state vector $\mathbf{x}(t)$, if it is only known to belong to a given box $[\mathbf{x}(t)]$, plays the role of the nuisance uncertain quantity that is not estimated.

### 3.1 Introduction

Standard parameter estimation techniques (see, *e.g.*, [28] and the references therein) compute $\widehat{\mathbf{p}}$ as the argument of the minimum of a given cost function, *e.g.*,

$$j(\mathbf{p}) = (\mathbf{y} - \mathbf{y}_{\mathrm{m}}(\mathbf{p}))^{\mathrm{T}} (\mathbf{y} - \mathbf{y}_{\mathrm{m}}(\mathbf{p})),$$

where

$$\mathbf{y} = \left(\mathbf{y}^{\mathrm{T}}(t_1), \ldots, \mathbf{y}^{\mathrm{T}}(t_N)\right)^{\mathrm{T}}$$

and

$$\mathbf{y}_{\mathrm{m}}(\mathbf{p}) = \left(y_{\mathrm{m}}^{\mathrm{T}}(\mathbf{x}(t_1), \mathbf{p}, t_1), \ldots, y_{\mathrm{m}}^{\mathrm{T}}(\mathbf{x}(t_N), \mathbf{p}, t_N)\right)^{\mathrm{T}}$$

are the system and model outputs collected at given time instants $t_i$, $i = 1, \ldots, N$. This minimization can be performed by local-search algorithms such as Gauss-Newton or Levenberg-Marquardt, but there is no guarantee of convergence to a global minimizer of $j(\mathbf{p})$ and this minimizer may even not be unique. Random search, using, *e.g.*, simulated annealing or genetic algorithms cannot provide any guarantee either that the global minimum has been found

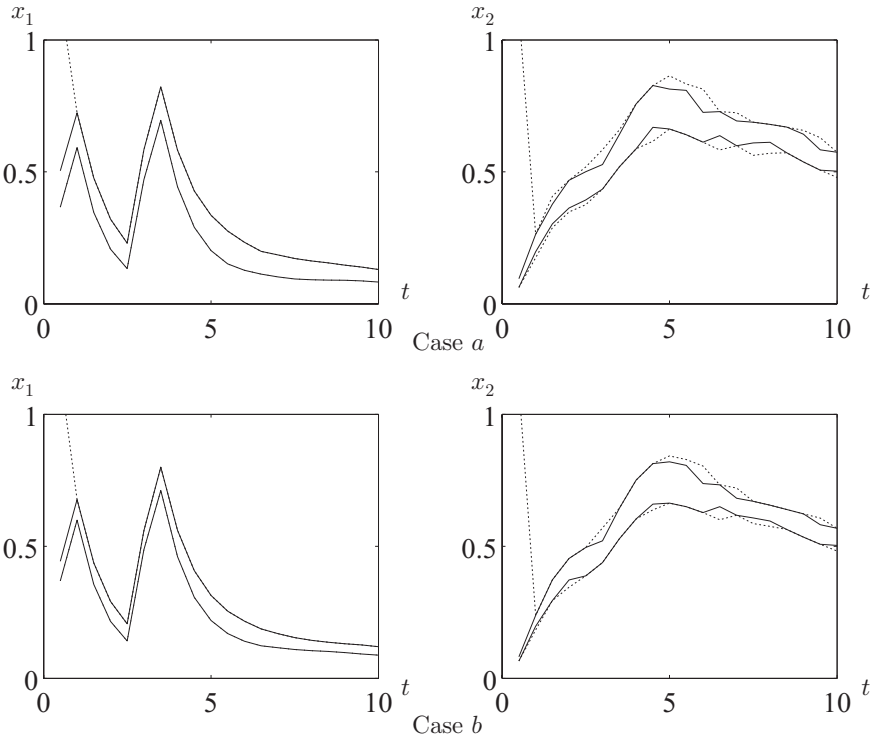



**Fig. 5.** Recursive bounded-error state estimation; lower and upper bounds of the smallest boxes enclosing the predicted set (dashed line) and corrected set (solid line); Case *a*: direct guaranteed integration of the model; Case *b*: guaranteed integration of the bounding cooperative systems

after finite computations. Only global guaranteed techniques, such as Hansen's algorithm [7], based on interval analysis, can obtain such guaranteed results.

Parameter bounding is an alternative approach searching for the *set* of all parameter vectors that are consistent with the experimental data, model structure and error bounds. It is similar to the correction step involved in the recursive state estimation algorithm presented in Section 2.2.

### 3.2 Principle

With the same hypotheses as in Section 2.2, the parameter vector $\mathbf{p} \in [\mathbf{p}_0]$ is deemed acceptable if the difference between the output $\mathbf{g}(\mathbf{x}(t_i), \mathbf{p})$ of the deterministic part of the model and the experimental datum $\mathbf{y}(t_i)$ remains in $[\underline{\mathbf{v}}_i, \overline{\mathbf{v}}_i]$ for all $i = 1, \ldots, N$. Parameter estimation then amounts to characterizing the set $\mathbb{P}$ of all acceptable $\mathbf{p} \in [\mathbf{p}_0]$

$$\mathbb{P} = \{\mathbf{p} \in [\mathbf{p}_0] \mid \mathbf{y}(t_i) - \mathbf{g}(\mathbf{x}(t_i), \mathbf{p}) \in [\underline{\mathbf{v}}_i, \overline{\mathbf{v}}_i],\ i = 1, \ldots, N\}. \qquad (16)$$

When the observation equation (3) reduces to

$$\mathbf{y}_{\mathrm{m}}(\mathbf{p}, t) = \mathbf{h}(\mathbf{p}, t) + \mathbf{v}(t), \tag{17}$$

with $\mathbf{h}(\mathbf{p}, t)$ some closed-form expression where $\mathbf{x}(t)$ does not appear, then the way $\mathbb{P}$ may be characterized depends mainly on whether $\mathbf{h}(\mathbf{p}, t)$ is linear in $\mathbf{p}$. If it is, $\mathbb{P}$ is a polytope that may be described exactly [27] or by an outer-approximation for instance using ellipsoids [3], [25]. When $\mathbf{h}(\mathbf{p}, t)$ is nonlinear in $\mathbf{p}$, $\mathbb{P}$ is no longer a polytope and may even be disconnected. One may nevertheless get a guaranteed enclosure of $\mathbb{P}$ using SIVIA.

When no closed-form solution of the model equations is available, again numerical integration has to be put at work to compute a box $[\mathbf{x}(t_i)]$ containing the state at each $t_i$ in order to enclose $\mathbf{g}(\mathbf{x}(t_i), \mathbf{p})$. The box $[\mathbf{x}(t_i)]$ is obtained efficiently when (1) can be bounded between two cooperative systems as in Section 2.3.

The characterization of $\mathbb{P}$ is then realized using SIVIA, as presented in Section 2.3. The main difference is that bisection is now performed in $\mathbf{p}$-space instead of $\mathbf{x}$-space.

### 3.3 Example

Consider the same example as in Section 2.4, and suppose now that the initial state is perfectly known $\mathbf{x}_0 = (0, 0)^{\mathrm{T}}$ and that only the last two components of the parameter vector are known, the first two $(p_1, p_2)$ being only known to belong to $[0, 5] \times [0, 5]$.

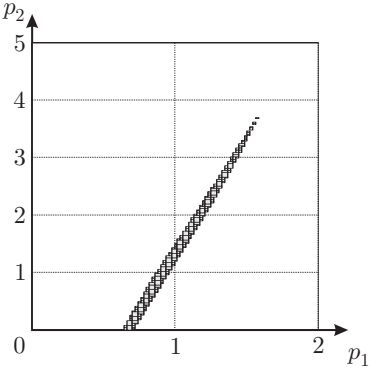



**Fig. 6.** Parameter estimation; solution subpaving in the $(p_1, p_2)$ −plane when $\varepsilon = 0.025$

Data have been obtained using the same simulation conditions as in Section 2.4. To evaluate an inclusion function for the state, the two bounding cooperative systems are now

$$\begin{cases} x_1' = -\dfrac{\overline{p}_1 x_1}{1+\underline{p}_2 x_1} - p_3^* x_1 + p_4^* x_2 + u, \\ x_2' = \dfrac{\underline{p}_1 x_1}{1+\overline{p}_2 x_1} - p_4^* x_2, \end{cases} \tag{18}$$

and

$$\begin{cases} x_1' = -\dfrac{\underline{p}_1 x_1}{1+\overline{p}_2 x_1} - p_3^* x_1 + p_4^* x_2 + u, \\ x_2' = \dfrac{\overline{p}_1 x_1}{1+\underline{p}_2 x_1} - p_4^* x_2, \end{cases} \tag{19}$$

The problem is now to evaluate the set of all parameter values $(p_1, p_2)$ that are compatible with the collected data and their associated error bounds (see Figure 4). The SIVIA algorithm has been used with initial search box $[0,5]\times[0,5]$ in parameter space. Guaranteed integration is again performed with the help of VNODE. With $\varepsilon_{\mathrm{S}} = 0.05$, the subpaving represented on Figure 6 has been obtained in 400 s on an Athlon 1800+. It contains the actual values of the parameters $(p_1^*, p_2^*) = (1, 4/3)$.

## 4 Conclusions

This paper presents an alternative and guaranteed approach for parameter and state estimation for continuous-time nonlinear differential models in a context of bounded errors with known bounds. An outer-approximation of the set of all parameter or state vectors that are consistent with the model structure and experimental data is obtained.

The only requirement is that the dynamical state equation of the system can be bounded between two cooperative systems. This is the case for all compartment models and for many other positive systems, *i.e.*, systems for which the state and parameters are constrained to remain positive.

The benefit of the enclosure between cooperative systems has been illustrated on an example. An ODE with uncertain parameters is replaced by two bounding ODEs with known parameters, the integration of which can be performed much more accurately, eliminating the wrapping effect.

## References

1. V. Alcaraz-González, A. Genovesi, J. Harmand, A. González, A. Rapaport, and J. Steyer. Robust exponential nonlinear interval observer for a class of lumped models useful in chemical and biochemical engineering. Application to a wastewater treatment process. In *Proc. MISC'99 Workshop on Applications of Interval Analysis to Systems and Control*, pages 225–235, Girona, February 24-26, 1999.
2. F. L. Chernousko. *State Estimation for Dynamic Systems.* CRC Press, Boca Raton, FL, 1994.

3. E. Fogel and Y. F. Huang. On the value of information in system identification - bounded noise case. *Automatica*, 18(2):229–238, 1982.
4. A. Gelb. *Applied Optimal Estimation*. MIT Press, Cambridge, MA, 1974.
5. K. Godfrey. *Compartimental Models and Their Application*. Academic Press, London, 1983.
6. J. L. Gouzé, A. Rapaport, and Z. M. Hadj-Sadok. Interval observers for uncertain biological systems. *Journal of Ecological Modelling*, (133):45–56, 2000.
7. E. R. Hansen. *Global Optimization Using Interval Analysis*. Marcel Dekker, New York, NY, 1992.
8. J. Hoefkens, M. Berz, and K. Makino. Efficient high-order methods for ODEs and DAEs. In G. Corliss, C. Faure, and A. Griewank, editors, *Automatic Differentiation : From Simulation to Optimization*, pages 341–351, New-York, NY, 2001. Springer-Verlag.
9. J. Hoefkens, M. Berz, and K. Makino. Verified high-order integration of DAEs and ODEs. In W. Kraemer and J. W. von Gudenberg, editors, *Scientific Computing, Validated Numerics, Interval Methods*, pages 281–292, Boston, 2001. Kluwer.
10. L. Jaulin. Nonlinear bounded-error state estimation of continuous-time systems. *Automatica*, 38:1079–1082, 2002.
11. L. Jaulin, M. Kieffer, I. Braems, and E. Walter. Guaranteed nonlinear estimation using constraint propagation on sets. *International Journal of Control*, 74(18):1772–1782, 2001.
12. L. Jaulin, M. Kieffer, O. Didrit, and E. Walter. *Applied Interval Analysis*. Springer-Verlag, London, 2001.
13. L. Jaulin and E. Walter. Guaranteed nonlinear parameter estimation from bounded-error data via interval analysis. *Mathematics and Computers in Simulation*, 35(2):123–137, 1993.
14. L. Jaulin and E. Walter. Set inversion via interval analysis for nonlinear bounded-error estimation. *Automatica*, 29(4):1053–1064, 1993.
15. R. E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the AMSE, Part D, Journal of Basic Engineering*, 82:35–45, 1960.
16. M. Kieffer. *Estimation ensembliste par analyse par intervalles, application à la localisation d'un véhicule*. PhD thesis, Université Paris-Sud, Orsay, France, 1999.
17. M. Kieffer, L. Jaulin, I. Braems, and E. Walter. Guaranteed set computation with subpavings. In W. Kraemer and J. W. von Gudenberg, editors, *Scientific Computing, Validated Numerics, Interval Methods*, pages 167–178, Boston, 2001.
18. M. Kieffer, L. Jaulin, and E. Walter. Guaranteed recursive nonlinear state bounding using interval analysis. *International Journal of Adaptative Control and Signal Processing*, 6(3):193–218, 2002.
19. M. Kieffer and E. Walter. Guaranteed nonlinear state estimator for cooperative systems. In *Proceedings of SCAN 2002*, Paris, 2002.
20. A. Kurzhanski and I. Valyi. *Ellipsoidal Calculus for Estimation and Control*. Birkhäuser, Boston, MA, 1997.
21. R. Lohner. Enclosing the solutions of ordinary initial and boundary value problems. In E. Kaucher, U. Kulisch, and C. Ullrich, editors, *Computer Arithmetic: Scientific Computation and Programming Languages*, pages 255–286. BG Teubner, Stuttgart, 1987.
22. R. Lohner. Computation of guaranteed enclosures for the solutions of ordinary initial and boundary value-problem. In J. R. Cash and I. Gladwell, editors, *Computational Ordinary Differential Equations*, pages 425–435, Oxford, 1992. Clarendon Press.

23. M. Milanese, J. Norton, H. Piet-Lahanier, and E. Walter, editors. *Bounding Approaches to System Identification*. Plenum Press, New York, NY, 1996.
24. N. S. Nedialkov and K. R. Jackson. Methods for initial value problems for ordinary differential equations. In U. Kulisch, R. Lohner, and A. Facius, editors, *Perspectives on Enclosure Methods*, pages 219–264, Vienna, 2001. Springer-Verlag.
25. F. C. Schweppe. *Uncertain Dynamic Systems*. Prentice-Hall, Englewood Cliffs, NJ, 1973.
26. H. L. Smith. *Monotone Dynamical Systems: An Introduction to the Theory of Competitive and Cooperative Systems*, volume 41 of *Mathematical Surveys and Monographs*. American Mathematical Society, Providence, RI, 1995.
27. E. Walter and H. Piet-Lahanier. Exact recursive polyhedral description of the feasible parameter set for bounded-error models. *IEEE Transactions on Automatic Control*, 34(8):911–915, 1989.
28. E. Walter and L. Pronzato. *Identification of Parametric Models from Experimental Data*. Springer-Verlag, London, 1997.

# Guaranteed Numerical Computation as an Alternative to Computer Algebra for Testing Models for Identifiability

Eric Walter[1], Isabelle Braems[1], Luc Jaulin[2], and Michel Kieffer[1]

[1] Laboratoire des Signaux et Systèmes
CNRS – Supélec – Université Paris-Sud
Plateau de Moulon, 91192 Gif-sur-Yvette, France
{walter, braems, kieffer}@lss.supelec.fr
[2] Laboratoire d'Ingénierie des Systèmes Automatisés
ISTIA, 62 avenue Notre Dame du Lac, 49000 Angers, France
luc.jaulin@univ-angers.fr

**Abstract.** Testing parametric models for identifiability is particularly important for knowledge-based models. If several values of the parameter vector lead to the same observed behavior, then one may try to modify the experimental set-up to eliminate this ambiguity (which corresponds to performing qualitative experiment design). The tediousness of the algebraic operations involved in such tests makes computer algebra particularly attractive. This paper describes some limitations of this classical approach and explores an alternative route based on new definitions of identifiability and numerical tests implemented in a guaranteed way. The new approach is illustrated in the context of compartmental modeling, widely used in biology.

## 1 Introduction

In many domains of pure and applied sciences, one would like to build a mathematical model from input-output experimental data. Sometimes, the only purpose of modeling is to mimic these observations, with no physical interpretation in mind. One then speaks of a *black-box model*. The situation considered in this paper is different. It is assumed that some prior knowledge is used to build a mathematical model that depends on a vector of parameters to be estimated from the data. If the model is entirely based on such a prior knowledge, one speaks of a *white-box model*. This is an idealized situation seldom encountered and the model is often a mixture of knowledge-based and black-box parts. One then speaks of a *gray-box model*. For white-box and gray-box models, all or some of the parameters receive a physical interpretation, and one would like to make sure that these parameters can be estimated meaningfully.

Let $\mathbf{u}$ be the (known) vector of the inputs of the system, which is usually a function of time $t$, and let $\mathbf{y}(t)$ be the corresponding vector of the outputs of the system at time $t$. A typical set-up for estimating the vector $\mathbf{p}$ of the parameters

R. Alt et al. (Eds.): Num. Software with Result Verification, LNCS 2991, pp. 124–131, 2004.

of the model of this system (see, for instance, [1], [2] or [3]) is to give the system and model the same input (one then speaks of a *parallel model*), and to look for the estimate $\widehat{\mathbf{p}}$ that minimizes the sum of the squares of the differences between the system and model outputs

$$\widehat{\mathbf{p}} = \arg\min_{\mathbf{p}} \sum_{i=1}^{f} (\mathbf{y}(t_i) - \mathbf{y}_{\mathrm{m}}(t_i, \mathbf{p}))^{\mathrm{T}} (\mathbf{y}(t_i) - \mathbf{y}_{\mathrm{m}}(t_i, \mathbf{p})).$$

In this equation, the $t_i$s are the instants of time at which the outputs of the system are measured and $\mathbf{y}_{\mathrm{m}}(t, \mathbf{p})$ is the vector of the outputs of the model at time $t$ when the parameter vector takes the value $\mathbf{p}$. The dependence of $\mathbf{y}$ and $\mathbf{y}_{\mathrm{m}}$ on the input $\mathbf{u}$ is omitted to simplify notation. When $\mathbf{p}$ has a physical meaning, one would like to know whether finding a numerical value for $\widehat{\mathbf{p}}$ gives any indication about the actual values of the physical parameters of the system under investigation. If not, one may try to modify the experimental set-up in order to remove the ambiguity. This is why it is desirable to reach a conclusion *as soon as possible* (if possible before performing any actual experimentation). A partial answer is found, under idealized conditions, with the concept of *identifiability*. We shall start by presenting the classical notion of identifiability before pointing out some of its limitations and proposing alternative definitions of identifiability and a guaranteed numerical method of test consistent with these new definitions.

## 2 Classical Approach to Identifiability Testing

Assume that there are no measurement noise or system perturbations, that the input and measurement times can be chosen in the most informative manner and that the system is actually described by a model with output $\mathbf{y}_{\mathrm{m}}(t_i, \mathbf{p}^*)$, where $\mathbf{p}^*$ is the (unknown) true value of the parameter vector. Under these idealized conditions, it is always possible to find at least one $\widehat{\mathbf{p}}$ such that the "system" with parameters $\mathbf{p}^*$ and the "model" with parameters $\widehat{\mathbf{p}}$ behave in *exactly* the same manner for all inputs and times, which we shall denote by

$$\mathbf{y}_{\mathrm{m}}(t, \widehat{\mathbf{p}}) \equiv \mathbf{y}_{\mathrm{m}}(t, \mathbf{p}^*). \tag{1}$$

It suffices to take the trivial solution $\widehat{\mathbf{p}} = \mathbf{p}^*$ for (1) to be satisfied. If this solution is unique, then the model is said to be *globally* (or *uniquely*) identifiable. This is of course desirable. Unfortunately, there may be parasitic solutions. If the number of solutions of (1) for $\widehat{\mathbf{p}}$ is greater than one, then we know that even under idealized conditions it will not be possible to estimate meaningfully all components of $\mathbf{p}^*$ with a single point estimate such as $\widehat{\mathbf{p}}$. As an illustrative example, consider the compartmental model described by Figure 1. Each circle represents a tank. The $i$th tank contains a quantity $x_i$ of material. These tanks exchange material between themselves and with the exterior as indicated by arrows. A usual assumption in linear compartmental modeling is that the flow of material leaving a compartment via an arrow is proportional to the quantity of material in this compartment. The constants of proportionality of these exchanges are then

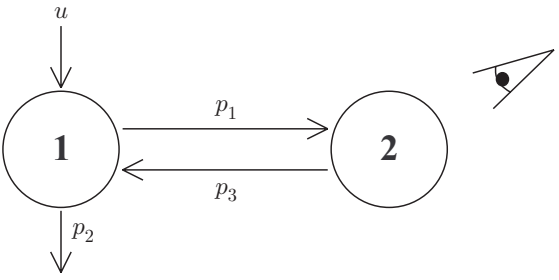



**Fig. 1.** Compartmental model

parameters to be estimated. Note that even when the compartmental model is linear, its output is nonlinear in these parameters, which significantly complicates their estimation. The dynamical state-space equations associated with a given compartmental model are very simple to obtain by writing down mass balances for each compartment. Such models, or variants of them, are widely used in biology and find applications in other experimental sciences such as pharmacokinetics, chemistry or ecology [4], [5]. For the model of Figure 1, mass balances in Compartments 1 and 2 lead to

$$\frac{\mathrm{d}x_1}{\mathrm{d}t} = -(p_1 + p_2)x_1 + p_3 x_2 + u$$

and

$$\frac{\mathrm{d}x_2}{\mathrm{d}t} = p_1 x_1 - p_3 x_2.$$

Assume that there is no material in the system at time 0, so $\mathbf{x}(0) = \mathbf{0}$, and that the quantity of material in Compartment 2 can be measured at any positive time, so

$$\mathbf{y}_\mathrm{m}(t, \mathbf{p}) = x_2(t, \mathbf{p}).$$

The question we are interested in is as follows: assuming that noise-free data are generated by a compartmental model with the structure described by Figure 1 and parameters $\mathbf{p}^*$, can the value of $\mathbf{p}^*$ be recovered from an analysis of the input-output data?

An obvious difficulty with this question is that the numerical value of $\mathbf{p}^*$ is unknown (since the very purpose of the exercise is to estimate it!), so we would like to reach a conclusion that would not depend on this value. Unfortunately, this is impossible in general, because there are usually atypical hypersurfaces in parameter space for which the conclusion is not the same as for all other values of the parameter vector. An example of such an atypical hypersurface is the plane defined by $p_1^* = 0$ for the model of Figure 1. Indeed, if there is no flow from Compartment 1 to Compartment 2 then no material ever reaches Compartment 2 and $y(t) = y_\mathrm{m}(t, \mathbf{p}^*) \equiv 0$, so there is no information in the

system output about $p_2^*$ and $p_3^*$. This is of course pathological and one would not use such a model if one had reasons to believe that there is no exchange from Compartment 1 to Compartment 2. The existence of such pathological situations led to the following usual definition of *structural* (or *generic*) *identifiability* [6]: a model is *structurally globally identifiable* (s.g.i. for short) if *for almost any value of* $\mathbf{p}^*$

$$\mathbf{y}_\mathrm{m}(t, \widehat{\mathbf{p}}) \equiv \mathbf{y}_\mathrm{m}(t, \mathbf{p}^*) \Rightarrow \widehat{\mathbf{p}} = \mathbf{p}^*.$$

If a model is not s.g.i., then there are several values of $\widehat{\mathbf{p}}$ for the same input-output behavior, and it is impossible to find out which one of them corresponds to $\mathbf{p}^*$ even in our idealized noise-free experimental set-up. The situation can only get worse in the presence of noise or perturbations. Moreover since there are several models with the same behavior, there are several ways of reconstructing non-measured state variables, *e.g.*, by Kalman filtering, with different results. So it is important to test models for identifiability whenever unknown parameters or state variables have a physical meaning or when decisions are to be taken on the basis of the numerical values of the estimates of these quantities.

A typical method of test consists of two steps. The first one is the obtention of algebraic equations that $\widehat{\mathbf{p}}$ and $\mathbf{p}^*$ must satisfy for (1) to hold true. For the model of Figure 1, it is easy to show that its transfer function is

$$\frac{Y(s)}{U(s)} = \frac{p_1}{s^2 + (p_1 + p_2 + p_3)s + p_2 p_3},$$

or equivalently that

$$\frac{\mathrm{d}^2 y}{\mathrm{d}t^2} + (p_1 + p_2 + p_3)\frac{\mathrm{d}y}{\mathrm{d}t} + p_2 p_3 y = p_1 u.$$

So, for almost any value of $\mathbf{p}^*$, (1) holds true if and only if

$$\begin{cases} \widehat{p}_1 = p_1^*, \\ \widehat{p}_1 + \widehat{p}_2 + \widehat{p}_3 = p_1^* + p_2^* + p_3^*, \\ \widehat{p}_2 \widehat{p}_3 = p_2^* p_3^*. \end{cases}$$

The second step is then the search for all solutions of these equations for $\widehat{\mathbf{p}}$. In the case of the model of Figure 1, these solutions are the trivial solution $\widehat{\mathbf{p}} = \mathbf{p}^*$ and

$$\begin{cases} \widehat{p}_1 = p_1^*, \\ \widehat{p}_2 = p_3^*, \\ \widehat{p}_3 = p_2^*. \end{cases}$$

The model of Figure 1 is therefore not s.g.i. The roles of $p_2$ and $p_3$ can be interchanged, and it is impossible to know which is which. Moreover, since there are two models with the same input-output behavior, there are two ways of reconstructing $x_1$ from measurements of $x_2$, even in a noise-free situation, leading to different values of $\widehat{x}_1$. Note that the parameter $p_1$, which takes the same values in the two solutions is s.g.i., and recall that most of this analysis becomes false if $p_1^* = 0$.

## 3 Limitations of This Classical Approach

Steps 1 and 2 of structural identifiability testing require algebraic manipulations that may become exceedingly complicated for models of a more realistic size. Both are facilitated by computer algebra [7], but these algebraic manipulations may become so complex that they are no longer feasible even on present-day computers. Moreover taking into account the fact that only real solutions are of interest is still a subject of research with computer algebra. Failing to detect that all solutions for $\widehat{\mathbf{p}}$ but one are complex would mean failing to detect that the parameters are actually globally identifiable.

Consider, for example, the (static) one-parameter model

$$y_{\mathrm{m}}(p) = p(p-1)(p+1).$$

Equation (1) translates into

$$\widehat{p}(\widehat{p}-1)(\widehat{p}+1) = p^*(p^*-1)(p^*+1),$$

and the set of real solutions for $\widehat{p}$ is a singleton, a pair or a triple depending on the value taken by $p^*$. So global identifiability is *not* a structural property for this model.

These shortcomings call for new definitions of identifiability, first presented in [8].

## 4 New Definitions and Method of Test

The parameter $p_i$ will be said to be *globally identifiable in* $\mathbb{P}$ (g.i.i.$\mathbb{P}$) if for all $(\mathbf{p}^*, \widehat{\mathbf{p}})$ in $\mathbb{P} \times \mathbb{P}$, $\mathbf{y}_{\mathrm{m}}(t, \widehat{\mathbf{p}}) \equiv \mathbf{y}_{\mathrm{m}}(t, \mathbf{p}^*)$ implies that $\widehat{p}_i = p_i^*$. The model will be g.i.i.$\mathbb{P}$ if all of its parameters are g.i.i.$\mathbb{P}$. With this new definition of identifiability, atypical hypersurfaces are no longer allowed in $\mathbb{P}$ and unique identifiability can be established even if the model is not structurally globally identifiable. It makes sense to study identifiability in a specific region $\mathbb{P}$ of parameter space, if only because some information is usually available on the sign and possible range for each physical parameter.

It does not suffice to have realistic new definitions of identifiability, methods of test are also needed. A model will be g.i.i.$\mathbb{P}$ if and only if

$$\nexists(\mathbf{p}^*, \widehat{\mathbf{p}}) \in \mathbb{P} \times \mathbb{P} \text{ such that } \mathbf{y}_{\mathrm{m}}(t, \widehat{\mathbf{p}}) \equiv \mathbf{y}_{\mathrm{m}}(t, \mathbf{p}^*) \text{ and } \|\widehat{\mathbf{p}} - \mathbf{p}^*\|_\infty > 0.$$

In practice, it will usually suffice to prove that

$$\nexists(\mathbf{p}^*, \widehat{\mathbf{p}}) \in \mathbb{P} \times \mathbb{P} \text{ such that } \mathbf{y}_{\mathrm{m}}(t, \widehat{\mathbf{p}}) \equiv \mathbf{y}_{\mathrm{m}}(t, \mathbf{p}^*) \text{ and } \|\mathbf{p}^* - \widehat{\mathbf{p}}\|_\infty > \delta,$$

where $\delta$ is some small positive number to be chosen by the user. The model will then be said to be $\delta$-g.i.i.$\mathbb{P}$. Testing whether a model is $\delta$-g.i.i.$\mathbb{P}$ boils down to a constraint satisfaction problem (CSP). The algorithm SIVIA, combined with a

forward-backward contractor, can be used to bracket the solution set $\mathbb{S}$ of the CSP

$$(\mathbf{p}^*, \widehat{\mathbf{p}}) \in \mathbb{P} \times \mathbb{P},\ \mathbf{y}_{\mathrm{m}}(t, \widehat{\mathbf{p}}) \equiv \mathbf{y}_{\mathrm{m}}(t, \mathbf{p}^*),\ \|\mathbf{p}^* - \widehat{\mathbf{p}}\|_\infty > \delta$$

between inner and outer approximations:

$$\underline{\mathbb{S}} \subset \mathbb{S} \subset \overline{\mathbb{S}}.$$

If $\overline{\mathbb{S}}$ is empty, then the model is $\delta$-g.i.i.$\mathbb{P}$. If $\underline{\mathbb{S}}$ is not empty, then the model is not $\delta$-g.i.i.$\mathbb{P}$. Details about SIVIA can be found in the paper by Kieffer and Walter in this volume and in [9], where forward-backward contractors are also presented.

## 5 Benchmark Example

The model of Figure 2 could serve as a benchmark example. It has been proposed to describe the distribution of drugs such as Glafenine in the body [10], [11] after oral administration. Compartment 1 corresponds to the drug in the gastro-

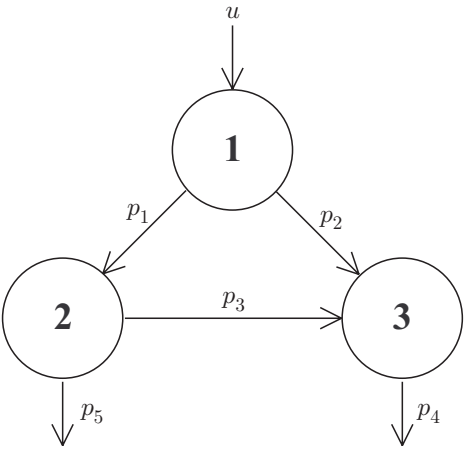



**Fig. 2.** Model of the distribution of Glafenine

intestinal tract, and Compartments 2 and 3 respectively correspond to the drug and its metabolite in the systemic circulation. The state equation of this model is

$$\begin{bmatrix} \frac{dx_1}{dt} \\ \frac{dx_2}{dt} \\ \frac{dx_3}{dt} \end{bmatrix} = \begin{bmatrix} -(p_1+p_2) & 0 & 0 \\ p_1 & -(p_3+p_5) & 0 \\ p_2 & p_3 & -p_4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} u.$$

By measuring the plasma concentration of the drug and its metabolite, the quantities of drug in Compartments 2 and 3 are determined up to unknown multiplicative constants, so

$$\mathbf{y}_{\mathrm{m}}(t,\mathbf{p}) = \begin{pmatrix} p_6 x_2(t) \\ p_7 x_3(t) \end{pmatrix},$$

where $p_6$ and $p_7$ are respectively the inverses of the volumes of Compartments 2 and 3. The dimension of the parameter vector $\mathbf{p}$ is thus seven. The corresponding transfer matrix is trivial to obtain by taking the Laplace transform of the state and observation equations and then eliminating the state variables. The same approach as in the introductory example of Section 2 can then be used to obtain a set of nonlinear equations in $\widehat{\mathbf{p}}$ and $\mathbf{p}^*$ that are equivalent to

$$\mathbf{y}_{\mathrm{m}}(t,\widehat{\mathbf{p}}) \equiv \mathbf{y}_{\mathrm{m}}(t,\mathbf{p}^*).$$

These equations can be written as

$$\widehat{p}_1\widehat{p}_6 = p_1^* p_6^*$$

$$\widehat{p}_2\widehat{p}_7 = p_2^* p_7^*$$

$$\widehat{p}_7\left(\widehat{p}_1\widehat{p}_3 + \widehat{p}_2\widehat{p}_3 + \widehat{p}_2\widehat{p}_5\right) = p_7^*\left(p_1^* p_3^* + p_2^* p_3^* + p_2^* p_5^*\right)$$

$$\widehat{p}_1 + \widehat{p}_2 + \widehat{p}_3 + \widehat{p}_5 = p_1^* + p_2^* + p_3^* + p_5^*$$

$$\widehat{p}_1\widehat{p}_3 + \widehat{p}_1\widehat{p}_5 + \widehat{p}_2\widehat{p}_3 + \widehat{p}_2\widehat{p}_5 = p_1^* p_3^* + p_1^* p_5^* + p_2^* p_3^* + p_2^* p_5^*$$

$$\widehat{p}_1 + \widehat{p}_2 + \widehat{p}_3 + \widehat{p}_4 + \widehat{p}_5 = p_1^* + p_2^* + p_3^* + p_4^* + p_5^*$$

$$\begin{aligned}\widehat{p}_1\widehat{p}_3 + \widehat{p}_1\widehat{p}_4 + \widehat{p}_1\widehat{p}_5 + \widehat{p}_2\widehat{p}_3 + \widehat{p}_2\widehat{p}_4 + \widehat{p}_2\widehat{p}_5 + \widehat{p}_3\widehat{p}_4 + \widehat{p}_4\widehat{p}_5 = \\ p_1^* p_3^* + p_1^* p_4^* + p_1^* p_5^* + p_2^* p_3^* + p_2^* p_4^* + p_2^* p_5^* + p_3^* p_4^* + p_4^* p_5^*\end{aligned}$$

$$\widehat{p}_4\left(\widehat{p}_1\widehat{p}_3 + \widehat{p}_1\widehat{p}_5 + \widehat{p}_2\widehat{p}_3 + \widehat{p}_2\widehat{p}_5\right) = p_4^*\left(p_1^* p_3^* + p_1^* p_5^* + p_2^* p_3^* + p_2^* p_5^*\right)$$

Their obtention is facilitated by the use of computer algebra.

We said in [8] that this model was $\delta$-g.i.i.$\mathbb{P}$ for $\mathbb{P} = [0.6, 1]^{\times 7}$ and $\delta = 10^{-9}$, but this remains to be confirmed, as this result may have been obtained with an incorrect software.

## 6 Conclusions

The concept of identifiability is important whenever physically meaningful parameters or state variables are to be estimated from experimental data. Testing models for structural global identifiability is not always possible, even with the help of computer algebra, and when a conclusion can be reached, it is not always relevant. This has led us to propose new definitions of global identifiability in a domain of parameter space. With this definition, it is possible to prove identifiability even in cases where the parameters are not structurally identifiable. The tests are performed via interval constraint satisfaction programming, with the use of contractors to avoid bisection as much as possible, thereby reducing the effect of the curse of dimensionality. We hope to have convinced the reader that identifiability testing is both a useful part of model building and an interesting challenge for interval analysts.

In this paper, it was assumed that there was a single model structure to be considered for the description of the data. When several model structures are in competition, a natural question to ask is whether it will be possible to select one that is more appropriate than the others. This question can be answered in the same type of idealized setting as considered for identifiability and corresponds then to the notion of *distinguishability*. The methodology advocated here for testing models for identifiability readily extends to the test of model structures for distinguishability.

## References

1. Norton, J.P.: An Introduction to Identification. Academic Press, London (1986)
2. Ljung, L.: System Identification, Theory for the User, 2nd Edition. Prentice Hall, Englewood Cliffs (1999)
3. Walter, E., Pronzato, L.: Identification of Parametric Models from Experimental Data. Springer-Verlag, London (1997)
4. Jacquez, J.A.: Compartmental Analysis in Biology and Medecine. Elsevier, Amsterdam (1972)
5. Godfrey, K.: Compartimental Models and Their Application. Academic Press, London (1983)
6. Walter, E.: Identifiability of State Space Models. Springer-Verlag, Berlin (1982)
7. Raksanyi, A., Lecourtier, Y., Walter, E., Venot, E.: Identifiability and distinguishability testing in computer algebra. Math. Biosci. **77(1-2)** (1985) 245–266
8. Braems, I., Jaulin, L., Kieffer, M., Walter, E.: Guaranteed numerical alternatives to structural identifiability testing. In: Proceedings of the 40th IEEE Conference on Decision and Control, Orlando (2001) 3122–3128
9. Jaulin, L., Kieffer, M., Didrit, O., Walter, E.: Applied Interval Analysis. Springer-Verlag, London (2001)
10. Balant, L.: Applicability of different types of models in health and disease. Drug Metab. Rev. **15** (1984) 75–102
11. Venot, A., Walter, E., Lecourtier, Y., Raksanyi, A., Chauvelot-Moachon, L.: Structural identifiability of "first-pass" models. Journal of Pharmacokinetics and Biopharmaceutics **15** (1987) 179–189

# Interval Algorithms in Modeling of Multibody Systems

Ekaterina Auer, Andrés Kecskeméthy, Martin Tändl, and Holger Traczinski

Universität Duisburg-Essen
Fakultät für Ingenieurwissenschaften
47048 Duisburg, Germany
{auer, traczinski}@informatik.uni-duisburg.de
{a.kecskemethy, m.taendl}@uni-duisburg.de

**Abstract.** We will show how a variety of interval algorithms have found their use in the multibody modeling program MOBILE. This paper acquaints the reader with the key features of this open source software, describes how interval arithmetic help to implement new transmission elements, and reports on interval modeling of dynamics, which is an inherent part of multibody simulations. In the latter case, the interval extension of MOBILE enhanced with an interval initial value problem solver (based on VNODE) is presented. The functionality of this application is shown with some examples. We provide insights into techniques used to enhance already existing modeling software with interval arithmetic concepts.

## 1 Interval Arithmetic in MOBILE: Areas of Application and Integration Strategies

Interval arithmetic is often criticized for its inapplicability to real life problems. This work claims the contrary by showing how it can be employed in multibody systems' modeling, an important area of applied physics, and in particular, in the program MOBILE. Interval arithmetic is used here to not only ensure the validity of the obtained results, but to also provide new modeling opportunities.

Mechanical interactions are usually modeled with the help of differential equations. It would be very time consuming to manually make up these equations each time. For that reason various types of modeling software have found a market in industry. Usually, such software produces the respective system of differential equations from the (formalized) description of an arbitrary mechanical system and is also capable of solving it thus characterizing the necessary system's properties.

In the present context, we employ the multibody library MOBILE described in [1,2]. It is able to model arbitrary mechanical systems and is characterized by its high computational speed (section 2 of this paper describes this program in more detail).

In the process of solving different problems with MOBILE, new tasks presented themselves, some of which proved to be most effectively dealt with by applying interval techniques. As a simple example of such a task, the incorporation

R. Alt et al. (Eds.): Num. Software with Result Verification, LNCS 2991, pp. 132–159, 2004.

of some external measurements as parameters into a model can be considered. Measurements are usually performed with a (small) error, the influence of which on the system's behavior is sometimes of interest. Moreover, the models always differ, if only slightly, from real life systems. Hence, it is useful to allow some uncertainty in the model and see how it affects the results.

Intervals offer an elegant way for solving the above tasks. Thus, it is appropriate to combine interval principles with modeling algorithms, which presupposes integration of interval methods into the already existing program MOBILE. This integration is performed in three layers. The basic layer is the interfacing between MOBILE and interval arithmetic; the package PROFIL/BIAS [3] was chosen to provide the appropriate data types and methods.

Based on this interface, additional structures need to be defined. For example, a simple replacement of floating-point arithmetic with interval arithmetic is insufficient because of its undesirable by-products such as the wrapping effect. Therefore, we have to improve the "naive" interval extension by exploiting knowledge about underlying MOBILE structures. Once this is done, the interval extension can be enhanced with more complicated algorithms, for example, for solving ordinary differential equations (ODEs) or computing validated distances, as well as design new MOBILE components, which allow, for example, uncertainty in measurements. All that constitutes the middle layer of integration, on top of which a connection to the outside world can be considered. Thus, the third and last integration layer require building interfaces to industrial modeling software [4].

Our goal is to implement an extension of MOBILE capable of interval calculus. To achieve this, we will proceed on two levels: implementation of interval kinematics and interval dynamics of mechanical systems. To develop the former, basic interval algorithms, such as addition, subtraction, etc. are required, as well as more complicated ones, such as solution of interval constraint equations, etc. The present state of this side of implementation is reflected in section 3 of this paper. To implement interval dynamics, one has to find an interface between interval initial value problem solvers (IIVPS) and MOBILE.

The task of integrating an IIVPS into certain types of modeling software is not completely free from difficulties. One of the major problems is obtaining derivatives.

On the one hand, there are several interval algorithms to solve initial value problems. Their common feature is the presence of several system function's derivatives. As a rule, the higher their order, the tighter the enclosure obtained. The well-known derivative free methods from numerics (Adams, Runge-Kutta, etc.) proved themselves hard to adapt to intervals.

On the other hand, most of the modeling software has no facilities to produce derivatives of arbitrary order. The usual methods of automatic differentiation, employed in many IIVPS, are impossible to make use of, because they require the right hand side of a given problem to be symbolically expressed, while in most cases this expression remains unknown. All the information given about the system function is its "numerical" values at some arbitrary points and its

algorithmic representation in a certain programming language. Therefore, the additional task to be solved on this implementation level is obtaining derivatives, which comply with the demands of validated algorithms, using only the above information. Possible ways of dealing with this problem as well as achievements towards modeling of systems' dynamics are described in section 4 of this paper.

A short summary of the most important results and a prospect on further work can be found at the conclusion of this paper.

## 2 The Multibody Modeling Library MOBILE

The modeling and simulation of the dynamical behavior of mechanical systems is a well-studied field in mechatronics. During the last 25 years, a large number of researchers have developed several formalisms for the automatic generation and resolution of the dynamical equations of multibody systems [5]. Some of these methods are still used today as universal engines for mechanics-based calculations in modern CAD systems, including for example ADAMS [6] in IDEAS and SD-FAST [7] in Pro/ENGINEER. Other formalisms have concentrated on specific areas of engineering, including for example recursive methods [8,9] for robotics, or symbolic computation methods for real time applications [10,11]. These approaches have the advantage of being comprehensive and provide comfortable user interfaces. However, due to their monolithic structure they lack the efficiency and capability of interaction with other simulation packages.

The present approach uses *object-oriented programming* for defining an open-architecture multibody library. The mechanical components are modeled as abstract mappings, termed *kinetostatic transmission elements*, which transmit motion and loads between sets of input and output variables called *state objects*. This results in an intuitive formulation, which allows the designer to put together the models of the parts of a mechatronic system in virtually the same way as they would actually be assembled in the real world. Moreover, by substituting other mathematical objects for real numbers, generic multibody formulations can be obtained which can be used for example for interval, stochastic, and fuzzy analysis. The multibody library MOBILE was implemented using the object-oriented programming language C++. Currently, only rigid bodies are modeled, but extensions to problems of structural mechanics, hydraulics and control theory can be incorporated into the general procedure.

Mathematically, the operations relating to the kinetostatic transmission elements correspond to well-known mappings of differential geometry: the transmission of position and velocity correspond to a nonlinear mapping between two smooth manifolds and the corresponding *push-forward* function for tangent vectors, while force mapping corresponds to the *pull-back* function being applied to cotangent vectors.

From a computational point of view, the applied method renders a *responsibility driven client/server* model [12] in which multibody operations are defined as "services" provided by an object at any time during program execution independently of its internal implementation according to a specific "contract".

In MOBILE, the basic "contract" of kinetostatic transmission elements consists of two main services: one for transmission of motion ("`doMotion`") and one for transmission of forces ("`doForce`"). More elaborated objects are defined at the following three levels of modeling complexity: (1) basic modeling, which involves only pure kinetostatic transmission elements, (2) sparse-Jacobian modeling, in which the interconnection structure and efficient methods for obtaining velocity transformations are considered, and (3) inertia-transmission modeling, in which the individual components are regarded as Riemannian manifolds able to generate and transmit mass properties. A description of the latter two levels of modeling complexity can be found under [1] and [13].

### 2.1 The Concept of Kinetostatic Transmission Elements

The central modeling element for mechanical systems is the *kinetostatic transmission element* (Fig. 1), which regards a mechanical component as an element `MoMap` that maps a set of $n$ scalar variables collected in the *input vector* $\underline{q}$ to a set of $m$ scalar variables collected in the *output vector* $\underline{q}'$.

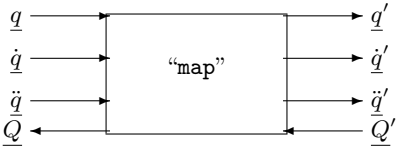



**Fig. 1.** Simple model of a kinetostatic transmission element

Associated with this mapping, there exist three kinematic functions and a force-associated function. The kinematic functions are the mapping itself and its first and second derivatives. These are collected in the *motion transmission functions*

$$\left.\begin{array}{ll} \underline{\text{position:}} & \underline{q}' = \underline{\varphi}\,(\,\underline{q}\,) \\ \underline{\text{velocity:}} & \dot{\underline{q}}' = \mathbf{J}_\phi\,\dot{\underline{q}} \\ \underline{\text{acceleration:}} & \ddot{\underline{q}}' = \mathbf{J}_\phi\,\ddot{\underline{q}} + \dot{\mathbf{J}}_\phi\,\dot{\underline{q}} \end{array}\right\} \quad . \tag{1}$$

Here, $\mathbf{J}_\phi = \partial\underline{\varphi}/\partial\underline{q}$ is the $m \times n$ *Jacobian* of the transmission element, which is not required explicitly by the clients of the `MoMap` element. For the *force transmission function*, one assumes that the transmission element is *ideal*, i. e. that it neither consumes nor produces power. Then, virtual work at the input and output are equal:

$$\delta\underline{q}^{\mathrm{T}}\,\underline{Q} = \delta\underline{q}'^{\mathrm{T}}\,\underline{Q}' \quad . \tag{2}$$

After substituting $\delta\underline{q}' = \mathbf{J}_\phi\,\delta\underline{q}$ and noting that this condition must hold for all virtual displacements $\delta\underline{q} \in \mathbb{R}^n$, one obtains

$$\underline{\text{force:}} \qquad \underline{Q} = \mathbf{J}_\phi^{\mathrm{T}}\,\underline{Q}' \quad , \tag{3}$$

where $\mathbf{J}_\phi^{\mathrm{T}}$ is the transpose of the Jacobian $\mathbf{J}_\phi$.

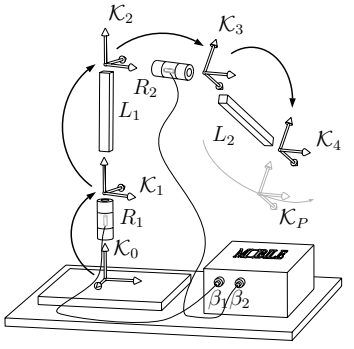
This transformation is directed from the (kinematical) output of the transmission element to its (kinematical) input. Note also that, in general, $\mathbf{J}_\phi$ need not be regular, in fact, not even square, so one cannot assume that (3) can be inverted. Thus force transmission is in general directed in the *opposite* direction to motion transmission.

In MOBILE, each transmission element "remembers" its once defined inputs and outputs for its lifetime. Hence, execution of the "`doMotion`" and "`doForce`" is possible by linking dynamically and without any arguments. Moreover, kinetostatic transmission elements can be concatenated by connecting the outputs of one element to the inputs of the other. The transmission functions of such a composite transmission element (termed `MoMapChain` in MOBILE) can be realized by concatenation of motion transmission in the order of the mechanical chain starting at the inertial system, and in reverse order for force transmission. In MOBILE, `MoMapChain` objects are simply ordered lists supporting the "`<<`"-operator for appending a `MoMap` object on the right to a `MoMapChain` on the left.

Inputs and outputs of transmission elements can be scalar or spatial quantities. They are regarded in MOBILE as *state objects*. Spatial motion is stored in *frames*, while scalar quantities are stored in objects termed *scalar variables*. Each of these objects embraces the complete information regarding position, velocity, acceleration, and load. In MOBILE, a frame is represented by an individual object of class `MoFrame` with members `R`, `r`, `ang_v`, `lin_v`, `ang_a`, `lin_a`, `t`, `f`, denoting the rotation matrix, the radius vector, the angular and linear velocity vectors, the angular and linear acceleration vectors, the torque and the force, respectively. As a convention, all vectors are assumed to be decomposed in the moving frames. Scalar state objects in MOBILE belong to the classes `MoLinearVariable` and `MoAngularVariable` for linear and rotational variables, respectively. Both classes have the members `q`, `qd`, `qdd`, `Q` denoting position, velocity, acceleration, and generalized force of the variable. However, for linear variables `q` is of type `MoReal`, whereas for angular variables `q` is of type `MoAngle`. The difference between a `MoAngle` and a `MoReal` is that the former stores its sine and cosine together with the value of the variable, so that these do not have to be repeatedly calculated.

State objects act as interface elements between which the kinetostatic transmission elements carry out their mappings: objects of the class `MoFrame` represent the junctions by which the mechanical components are connected together, while scalar state objects represent the actuator variables which are used to drive the joints or to move the bodies along a prescribed trajectory.

As an example, Fig. 2 shows a simple manipulator consisting of two revolute joints and its corresponding MOBILE modeling. Note that this is the entire source code of an executable program. Note also, that there is a one-to-one correspondence between the physical components and their program counterparts, and that at the end of the modeling, it is possible to treat the composite system as a simple transmission element by invoking the `doMotion` function. The places where the values of the state objects are prescribed and those where the




a) system model

```
#include <Mobile/MoElementaryJoint.h>
#include <Mobile/MoRigidLink.h>
#include <Mobile/MoMapChain.h>

int main()
{
// reference frames and actuator variables
   MoFrame K0, K1, K2, K3, K4;
   MoAngularVariable beta1, beta2;
// transmission elements
   MoVector l1(0,4,0), l2(0,3,0);
   MoRigidLink L1 ( K1 , K2 , l1 );
   MoRigidLink L2 ( K3 , K4 , l2 );
   MoElementaryJoint R1 ( K0 , K1 , beta1 , z_axis );
   MoElementaryJoint R2 ( K2 , K3 , beta2 , x_axis );
// complete system
   MoMapChain Manipulator;
   Manipulator << R1 << L1 << R2 << L2;
// definition of actuator positions
   beta1.q = 0.25 * MO_PI;
   beta2.q = -0.35 * MO_PI;
// transmission of motion
   Manipulator.doMotion(DO_ALL);
   return 0;
}
```

b) executable program

**Fig. 2.** Example: Programming of a simple manipulator

transmission functions are invoked may be apart. For example, one might set the kinematical inputs within a module modeling the behavior of a control unit, while the `doMotion` function is invoked in a module, where a sensor measuring the approach velocity of another object is modeled. This technique, also termed "*modeling by programming*" [1], leads to lean programming models that can be incorporated easily and efficiently into other environments.

For the generation of the equations of motion, a coordinate independent approach is employed in which the relevant terms of the equations of motion are reproduced from pure motion and force traversals of the system. The equations of motion of minimal order are denoted by

$$\mathbf{M}\,(\,\underline{q}\,;\,t\,)\,\ddot{\underline{q}} + \underline{b}\,(\,\underline{q}\,,\,\dot{\underline{q}}\,;\,t\,) = \underline{Q}\,(\,\underline{q}\,,\,\dot{\underline{q}}\,;\,t\,), \tag{4}$$

where $\underline{q} = [q_1, \ldots . q_f]^{\mathrm{T}}$ are the generalized coordinates, $\mathbf{M}$ is the $f \times f$ generalized mass matrix and $\underline{b}$ and $\underline{Q}$ are the vectors of generalized Coriolis and centrifugal force as well as generalized applied forces, respectively. Mass-inertia properties can be regarded by computing the corresponding d'Alembert forces and motions through corresponding elements termed `MoMassElement` in MOBILE. Applied forces are regarded through a further set of kinetostatic transmission elements denominated `MoForceElement`, which finally takes care of evaluating applied forces from the position and velocity state of particular objects and applying the resulting forces back to these state objects. As `MoMassElement`s and `MoForceElement`s do not further transmit motion, the overall structure of a multibody system takes the form depicted in Fig. 3.

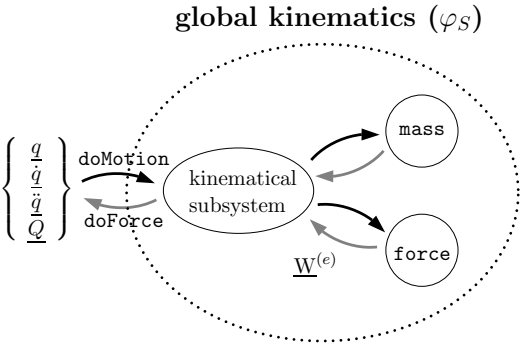



**Fig. 3.** Model of the inverse dynamics of a multibody system

If one regards the whole system, consisting of a kinematical subsystem and the attached mass and force elements, as one transmission element called *global kinematics*, one can prescribe the motion of the generalized coordinates $\underline{q}$, their first and second order time derivatives as well as the applied forces, collected in a vector $\underline{W}^{(e)}$ and perform the composition of the motion and force transmission function as a function $\varphi_S^{D^{-1}}$ which exactly implements the *inverse dynamics* ($D^{-1}$) of the system, i. e. the computation of the *residual forces*

$$\overline{\underline{Q}} = \varphi_S^{D^{-1}}(\,\underline{q}\,,\,\underline{\dot{q}}\,,\,\underline{\ddot{q}}\,;\,\underline{W}^{(e)}\,;\,t\,) = -\mathbf{M}\,(\underline{q};t)\;\underline{\ddot{q}} - \widehat{\underline{Q}}\,(\,\underline{q}\,,\,\underline{\dot{q}}\,;\,\underline{W}^{(e)}\,;\,t) \quad , \qquad (5)$$

as a function of $\underline{q}$, $\underline{\dot{q}}$, $\underline{\ddot{q}}$, and $\underline{W}^{(e)}$. With this function, it is easy to obtain the unknown quantities $\widehat{\underline{Q}}$ and $\mathbf{M}$ of the equations of motion: for $\widehat{\underline{Q}}$, one just evaluates $\varphi_S^{D^{-1}}$ after setting $\underline{\ddot{q}} = 0$, and for $\mathbf{M}$, one first eliminates $\widehat{\underline{Q}}$ from the calculations (by turning off the evaluation of gyroscopic and applied forces and of rheonomic terms) and then calculates $\varphi_S^{D^{-1}}$ for one acceleration equal to one, e. g. $\ddot{q}_\nu = 1$, while all others are equal zero, giving the $\nu$ th column of $\mathbf{M}$. Thus, it is possible to obtain the complete dynamics of a system just by using the kinetostatic transmission elements.

## 3 Integration of Interval Arithmetic in MOBILE

In this section we first give a brief introduction of interval arithmetic which is provided by the basic scalar data type `MoInterval` in MOBILE. It is possible to achieve guaranteed simulation results with this data type. Uncertain inputs, which, for example, arise from sensor measurements, can also be modeled. Moreover, the use of interval arithmetic increases the number of modeling options, for instance, by introducing a revolute joint with slackness. Hence, interval extension makes MOBILE more powerful.

To obtain not only reliable but also useful results, we have to tackle typical problems of interval arithmetic such as the wrapping effect. We show how to cope with the wrapping effect in MOBILE and modify kinetostatic transmission elements for this purpose.

### 3.1 Basics of Interval Arithmetic

Floating-point arithmetic can lead to erroneous results even if computations are made according to the IEEE 754 standard [14]. A remarkable example is given in [15]. To obtain a verified enclosure for the exact result, interval arithmetic can be used, in which all calculations are performed with two floating-point numbers, a lower and an upper bound for each of the inputs and results. In conjunction with directed roundings, we can obtain an enclosure of the exact result on a computer.

Another advantage of intervals is the handling of uncertain data, e.g. data resulting from measurements, which can also be represented as intervals.

Now we can give a brief survey of interval arithmetic. Consider a real interval $X = [\underline{x}, \overline{x}] := \{x \in \mathbb{R} \mid \underline{x} \le x \le \overline{x}\}$. For any arithmetic operation $\circ \in \{+, -, \cdot, /\}$ and intervals $X$, $Y$, we can define the corresponding interval arithmetic operations:

$$\begin{aligned} X \circ Y &:= \{x \circ y \mid x \in X, y \in Y\} \\ &= \{x \circ y \mid \underline{x} \le x \le \overline{x}, \underline{y} \le y \le \overline{y}\} \\ &= \{z \mid \min(\underline{x} \circ \underline{y}, \underline{x} \circ \overline{y}, \overline{x} \circ \underline{y}, \overline{x} \circ \overline{y}) \le z \le \max(\underline{x} \circ \underline{y}, \underline{x} \circ \overline{y}, \overline{x} \circ \underline{y}, \overline{x} \circ \overline{y})\} \; . \end{aligned}$$

With $\underline{z} := \min(\underline{x} \circ \underline{y}, \underline{x} \circ \overline{y}, \overline{x} \circ \underline{y}, \overline{x} \circ \overline{y})$ and $\overline{z} := \max(\underline{x} \circ \underline{y}, \underline{x} \circ \overline{y}, \overline{x} \circ \underline{y}, \overline{x} \circ \overline{y})$, we obtain

$$Z := [\underline{z}, \overline{z}] = X \circ Y \; .$$

For standard functions $f$, e.g. sin, cos, exp, ln, there are algorithms to compute an accurate interval containing the value set $V_f(X) := \{f(x) \mid x \in X\}$ of $f$, where $X$ is an interval [16].

Thus, we are able to compute an enclosure of the value set $V_f(X)$ for any function $f$ which is composed of fundamental operations and standard functions. We replace the variable $x$ by an interval $X$ with $x \in X$ and compute $f(X)$ using appropriate interval operations. Then,

$$f(x) \in V_f(X) \subseteq f(X) \quad \text{for all } x \in X \; .$$

Note that $f(X)$ depends on the representation of $f$, i.e. it is possible that $f(X) \neq g(X)$ for $f \equiv g$ [17].

We only have a finite number of machine numbers for calculations on a computer. A *machine interval* is represented by two machine numbers. For an interval $[\underline{x}, \overline{x}]$, we round $\underline{x}$ down to the largest machine number equal or less than $\underline{x}$, and $\overline{x}$ up to the smallest machine number equal or greater than $\overline{x}$.

### 3.2 The New Data Type `MoInterval`

The basic floating-point data type for real scalars in MOBILE is `MoReal`. Currently, it is the same as `double` in C++. A new data type has been introduced into MOBILE for interval calculations: the class `MoInterval`. Besides a variable of the type `INTERVAL` from the interval arithmetic package PROFIL/BIAS [3],

`MoInterval` also provides a `double` variable for bounding the absolute computation error. This bounding is computed automatically as described in [18].

A variable of the type `MoInterval` can be used in the same way as a `MoReal` variable because the arithmetic operations and the standard functions are overloaded. Besides, an interval given by its lower and upper bound can be assigned to a `MoInterval` variable.

In each case where a floating-point number is used to construct a `MoInterval` value, it is guaranteed that the (decimal) number represented by the corresponding string in the MOBILE program is completely enclosed by the interval value of `MoInterval`. For example, `MoInterval(0.1)` and `MoInterval(0.1,0.1)` both result in the same interval enclosing exactly three consecutive floating-point numbers $a < b < c$, where $b$ is the floating-point representation of the decimal value 0.1. Of course, this is an overestimation, because there is an interval enclosing 0.1 containing only two consecutive floating-point numbers for lower and upper bound. This can be achieved by using the constructor with string input: `MoInterval("0.1")`.

The output of a `MoInterval` variable is printed in the following way: first an enclosing interval is printed — because of the internal binary representation the lower bound may be rounded down and the upper bound may be rounded up for output — then a maximum absolute error is printed in parentheses.

A list of constructors, member operators, and member functions can be found under [19].

### 3.3 Extended MOBILE Objects

The main goal of the new extension package is "easy handling", which means that a user familiar with MOBILE does not have to learn a new language. Of course, there are new classes for mathematics and kinetostatic transmission elements, which provide interval arithmetic [19], but they can be used in the same way as the corresponding well-known MOBILE classes. Only the names are different. All names of MOBILE classes start with `Mo`, e.g. `MoRigidLink`, and the names of the corresponding interval classes begin with `MoInterval`, e.g. `MoIntervalRigidLink`.

Besides the names of classes, some of the names of constants have been changed for interval purpose. For example, `INTERVAL_PI` replaces the floating-point constant `MO_PI` and represents a `MoInterval` enclosing $\pi$.

The MOBILE program in Fig. 4 describes the model of the example from section 2.1 and calculates the position of its tip. It is assumed here that rigid links have certain masses and elementary joints rotate about the same $x$ axis. This program is derived from the classical MOBILE one just by placing the words `Interval` and `INTERVAL` in the correct positions. The resulting output of the classical MOBILEprogram is

```
Position = (0,5.6816,1.90138) ,
```

whereas the output of the interval version reads as follows:

```
#include <Mobile/MoIntervalElementaryJoint.h>
#include <Mobile/MoIntervalRigidLink.h>
#include <Mobile/MoIntervalMassElement.h>
#include <Mobile/MoIntervalMapChain.h>
int main () {
  MoIntervalFrame K0 , K1 , K2 , K3 , K4;
  MoIntervalAngularVariable beta1,beta2 ;
  MoIntervalVector l1 , l2 ;
  MoIntervalMass m1,m2 ;
  MoIntervalElementaryJoint R1 ( K0, K1, beta1, xAxis ) ;
  MoIntervalRigidLink     rod1 ( K1, K2, l1 ) ;
  MoIntervalElementaryJoint R2 ( K2, K3, beta2, xAxis ) ;
  MoIntervalRigidLink     rod2 ( K3, K4, l2 );
  MoIntervalMassElement Tip1 ( K2, m1 ) ;
  MoIntervalMassElement Tip2 ( K4, m2 );
  MoIntervalMapChain Manipulator ;
  Manipulator << R1 << rod1 << Tip1 << R2 << rod2  << Tip2 ;
  l1     = MoIntervalVector ( 0 , 4 ,  0 ) ;
  l2     = MoIntervalVector ( 0 , 3 ,  0 );
  m1     = 0.1 ;
  m2     = 0.1 ;
  beta1.q = 0.25*INTERVAL_PI ;
  beta2.q = -0.35*INTERVAL_PI ;
  Manipulator.doMotion(DO_INTERVAL_ALL) ;
  cout << "Position = " << K4.R*K4.r << endl ;
}
```

**Fig. 4.** The interval program for the manipulator (kinematics)

```
Position = ([0,0] ( 0.0000000000000000E+0),
5.6815966736316[161,819] ( 1.9739362421216854E-14),
1.9013761416213[121,817] ( 1.8214953086484989E-14)) .
```

The three intervals contain the exact solutions for the three components of the position vector. The value in parentheses after an interval indicates the corresponding maximum absolute computation error.

### 3.4 A Sloppy Joint

Interval arithmetic is not only a means of computing verified results or handling uncertain data, it also enhances the modeling opportunities of MOBILE.

In [20], a sloppy revolute joint is modeled to cope with real world revolute joints. The rotation axes of two connected bodies are no longer assumed to be exactly concentric, but the relative distance between these axes is within a specific (small) range.

Fig. 5 shows the CAD drawings of a sloppy joint. The corresponding model with the inserted sloppy link is displayed in Fig. 6.

The parameter $\varphi_i$ that describes the relative orientation between two connected bodies is the same for the sloppy joint and for an ideal one. Two more parameters are added for the sloppy joint to describe the unique positions of the two bodies.

Assume that the two bodies are not coupled directly, but connected by one additional rigid link, called a *sloppy link* $\mathbf{l}_i$. In MOBILE this sloppy link is

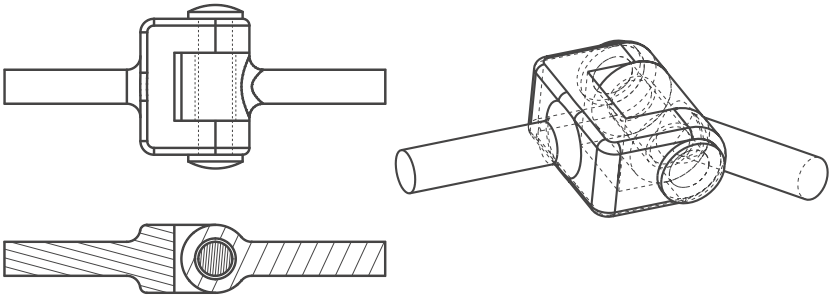


**Fig. 5.** CAD model of a sloppy joint

modeled as a regular rigid link that connects the two bodies. This connection is achieved by using two regular revolute joints. The two additional parameters are the length $l_i$ of the link and the relative orientation angle $\alpha_i$. The length $l_i$ can be chosen in the interval $[0, l_i^{\max}]$ and the orientation angle can be any angle $\alpha_i \in [0, 2\pi[$. These parameters are of the `MoInterval` data type.

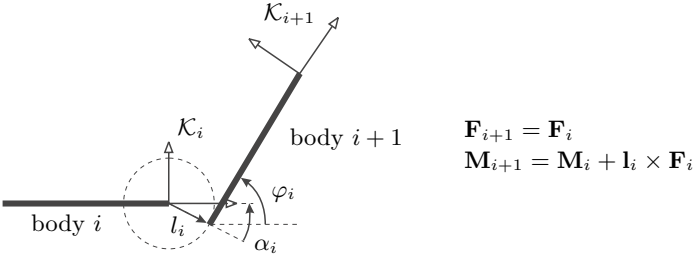



**Fig. 6.** Calculation of position, force, and torque in a sloppy joint

An example of a manipulator built with sloppy joints and a comparison of results obtained by interval arithmetic and Monte Carlo simulation are presented in [20].

### 3.5 Avoiding the Wrapping Effect

The change of data types in MOBILE from `MoReal` to `MoInterval` leads to verified simulations but not always to useful results because of the dramatic influence of the wrapping effect. This influence can be demonstrated in the example from section 3.3 with sloppy joints instead of `MoElementaryJoint` objects (we replace them with `MyIntervalSlacknessJoint` objects).

Consider a sloppiness in each joint of 0.02, i.e. the maximum distance between the reference points of the frames `K0` and `K1` is 0.02, the same holds for `K2` and

K3. The task is to determine the position $p$ of the tip, i.e. the position of the reference point of K4 with respect to K0.

The computed result in MOBILE with (naive) interval arithmetic is the interval vector

$$p = \begin{pmatrix} [0,0] \\ [5.6224993009825814, 5.7406940462807193] \\ [1.8422787689722757, 1.9604735142704182] \end{pmatrix}.$$

Here we have an overestimation in the second and third component, where the diameters of both intervals are about 48% larger than the exact ones. This is due to the fact that the disc (in the following we disregarded the first dimension because we dealt with a plane problem) for the position of the reference point of K0 is enclosed in a square interval which is then rotated by 45° and wrapped with an interval again. The same happens when the rotation in the second joint is made.

This overestimation can be avoided in several ways. Since we started with a disc, a rotation should not lead to a larger disc except for rounding error accumulation. Hence, we could use midpoint-radius calculations. However, as shown in [20], the shape for possible positions of an end-effector of a manipulator with several uncertain inputs can still be really unpredictable.

Consider mapping from the position $p_0$ of the reference point of frame K0 to the position $p_2$ of the reference point of frame K2

$$p_2 = p_0 + \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos r & -\sin r \\ 0 & \sin r & \cos r \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 4 \\ 0 \end{pmatrix}$$

with $r = \frac{1}{4}\pi$. If $p_0$ is an interval vector, it is translated without any rotation or deformation. The same holds for the mapping $p_2 \mapsto p$. This leads to another approach in avoiding the wrapping effect.

Unfortunately, a redesign of MOBILE classes is necessary: for example, the class `MoFrame`, which stores the kinematic and static state of an "oriented" point in space, and transmission elements like `MoRigidLink` or `MoElementaryJoint`, which map this information to other `MoFrame` objects. If we additionally store the input error of the position as an interval, we need to do this in global coordinates. In the future the described approach will also be used to avoid the wrapping effect in the velocity and acceleration.

For the above example with the modified transmission elements the resulting interval vector is then

$$p = \begin{pmatrix} [0,0] \\ [5.6415966736316107, 5.7215966736316970] \\ [1.8613761416213070, 1.9413761416213929] \end{pmatrix},$$

which is a tighter enclosure with respect to rectangular intervals.

## 4 Interval Modeling of Dynamics

This section describes an approach to validating dynamics in MOBILE and presents an IIVPS for "numerically" modeled mechanical systems. The performance of the solver is demonstrated in some examples.

### 4.1 An Approach to Interval Modeling of Dynamics in MOBILE

Before speaking about interval modeling of dynamics in MOBILE, let us digress for a moment and consider multibody modeling software in general. It can be roughly divided into two types, which are nominally denoted here as *"symbolic"* and *"numerical"*. They differ in how they represent the model of a given mechanical system. The former produces an explicit *symbolic* description of the resulting differential equations. For instance, if a model for the harmonic oscillator is sought, with parameters $m$ and $k$, this type of software delivers the equation $m\ddot{x}(t) + kx(t) = 0$. A drawback of this kind of software is its slowness; besides, the explicit representation is so intricate in most cases that it has to be manually simplified.

Packages of the "numerical" type dispense with the symbolic description of the model for the sake of higher computational speed. They provide values of the specified model parameters at some arbitrary points (hence the name *numerical*). These values are "exact" in the same manner as the values obtained by substitution of given points into the symbolic description are "exact". The only difference is that the general expression remains unknown in the former case.

It seems important here to point out two different meanings of the word "numerically". One meaning expresses the fact that no explicit "symbolic" description of the resulting entities is produced. Another indicates the presence of a certain approximation error and, consequently, the loss of accuracy. In our context, this meaning of the word "numerically" can be contrasted with that of the word "symbolically", which stands for differentiation without approximation errors with the help of the usual automatic differentiation techniques (e.g. [21]). These two meanings should not be confused: if we speak here about "numerical software", we mean it in the first sense.

The problem which appears on building an IIVPS into MOBILE consists in seeming impossibility of the "exact" or "symbolic" differentiation in case of "numerical" software. MOBILE belongs to the "numerical" type, but to model dynamics with interval methods, one has to compute many derivatives of a system function, which are not provided by the program.

There are few options to handle this problem. The use of divided differences or other methods of numerical differentiation is not allowed because it contradicts the general idea of verification, which is to guarantee that the solution lies within a certain interval. Nothing can be guaranteed if truncation errors in derivatives are not taken into account.

Therefore, the next option is to obtain derivatives by considering system's physics. There exist some theories that deal with the finding of Jacobians and

Hessians of a modeled function (e.g. [1]). Probably, it is possible to develop similar theories about higher-order time derivatives and their Jacobians, but not without thorough understanding of the underlying subject matter. A serious limitation of this approach lies in the necessity to reconsider the whole theory to find a derivative of order $n + 1$ even if an algorithm for the $n$-th one is known. Studies that deal with this kind of problems and attempt to automatize this process are unknown to the authors. In short, this approach might provide the necessary derivatives, but research on that topic must constitute a separate study, which would be best performed by a physicist.

On closer look at the problem, the third option, which we use in MOBILE, becomes evident. Numerical differentiation might be the answer if all what is known about a function is its values at some given points, but in our case the *algorithmic representation* of this function in some programming language is known as well. This piece of code can be treated as the "explicit" symbolical representation of the function and help to reach its derivatives with the usual techniques of automatic differentiation. This variety of "exact" differentiation, called *algorithmic differentiation* [22], has been applied to MOBILE. By summarizing the theoretic aspects of both algorithmic differentiation and IIVPS, the following section provides the understanding of what derivatives are necessary to model dynamics with interval methods and how to retrieve them.

### 4.2 Theory Overview: IIVPS and Algorithmic Differentiation

The idea of interval solution of initial value problems is not new. There heve been several studies in this area: see [23,24]. Most studies incorporate or develop the methods presented by R. Lohner [21].

The task is formulated as follows: The set of autonomous initial value problems

$$\begin{cases} y'(t) = f(y) \\ y(t_0) \in [y_0] \end{cases} \tag{6}$$

is considered, where $t \in [t_0, t_n] \subset \mathbb{R}$ for some $t_n > t_0$, $f \in C^{p-1}(\mathcal{D})$, $\mathcal{D} \subseteq \mathbb{R}^m$ is open, $f : \mathcal{D} \mapsto \mathbb{R}^m$, and $[y_0] \subset \mathcal{D}$. The problem is discretized on a grid $t_0 < t_1 < \ldots < t_n$ with $h_{j-1} = t_j - t_{j-1}$. The solution of (6) with an initial condition $y(t_{j-1}) = y_{j-1}$ is denoted by $y(t; t_{j-1}, y_{j-1})$ as well as the set of solutions $\{y(t; t_{j-1}, y_{j-1}) \mid y_{j-1} \in [y_{j-1}]\}$ by $y(t; t_{j-1}, [y_{j-1}])$. The goal is to compute interval vectors $[y_j], j = 1, \ldots, n$, that are guaranteed to contain the solution of (6) at $t_1, \ldots, t_n$. That is, $y(t; t_0, [y_0]) \subseteq [y_j]$, $j = 1, \ldots, n$.

The $j$th step of most validated methods consists of two stages [24]:

1. *Proof of existence and uniqueness.* Compute a stepsize $h_{j-1}$ and an a priori enclosure $[\tilde{y}_{j-1}]$ of the solution such that $y(t; t_{j-1}, y_{j-1})$ is guaranteed to exist for all $t \in [t_{j-1}, t_j]$ and all $y_{j-1} \in [y_{j-1}]$ and $y(t; t_{j-1}, [y_{j-1}]) \subseteq [\tilde{y}_{j-1}]$ for all $t \in [t_{j-1}, t_j]$.

*2. Computation of the solution.* Compute a tight enclosure $[y_j] \subseteq [\tilde{y}_{j-1}]$ such that $y(t_j; t_0, [y_0]) \subseteq [y_j]$. There are several algorithms to solve this problem, which follow roughly the same scheme.

*2.1. Choice of the underlying method.* One can choose a one-step method

$$y(t; t_j, y_j) = y(t; t_{j-1}, y_{j-1}) + h_{j-1}\varphi(y(t; t_{j-1}, y_{j-1})) + z_j \ ,$$

where $\varphi$ is some method function, and $z_j$ is the local error. The usual choice for $\varphi$ is Taylor series expansion.

*2.2. Enclosure of the local error.* Find an enclosure for the local error $z_j$. In case of the Taylor series expansion of order $p-1$, this enclosure is obtained as $[z_j] = h_{j-1}^p f^{[p]}([\tilde{y}_{j-1}])$, where $f^{[p]}([\tilde{y}_{j-1}])$ is an enclosure of the $p$-th Taylor coefficient over $[\tilde{y}_{j-1}]$.

*2.3. Enclosure of the global error.* Compute a tight enclosure of the solution. In the case of the Taylor series expansion of order $p-1$ the resulting formula is

$$\begin{aligned} [y_j] \quad = \quad & \overbrace{\hat{y}_{j-1} + \sum_{i=1}^{p-1} h_{j-1}^i f^{[i]}(\hat{y}_{j-1})}^{\text{approximate solution}} + [z_j] \\ & + \overbrace{\left( I + \sum_{i=1}^{p-1} h_{j-1}^i J(f^{[i]}, [y_{j-1}]) \right) ([y_{j-1}] - \hat{y}_{j-1})}^{\text{global error}} \ , \end{aligned} \tag{7}$$

where $\hat{y}_{j-1} \in [y_{j-1}]$, $J(f^{[i]}, [y_{j-1}])$ is the Jacobian of $f^{[i]}$ evaluated at $[y_{j-1}]$. These Jacobians are equal to the Taylor coefficients for the solution of the associated variational equation

$$Y' = \frac{\partial f}{\partial y} Y, \qquad Y(t_{j-1}) = I \ , \tag{8}$$

which leads to an algorithm for their computation [21]. This is the so-called *direct* Taylor series method for the global error propagation, which in most cases overestimates the enclosure due to the *wrapping effect* [21]. Some methods to reduce this effect use non-orthogonal ("parallelepiped") and orthogonal ("QR factorization") coordinate transformations, ellipsoids, zonotopes, and Taylor models [25].

This approach to validated integration requires computation of Taylor coefficients as well as of their Jacobians for the system function $f(y)$ ($f^{[i]}(\hat{y}_{j-1})$ and $J(f^{[i]}, [y_{j-1}])$, respectively). Those can be retrieved with the help of algorithmic differentiation ([23,22]).

To be able to apply this form of differentiation we have to ensure that the source system meets some requirements:

First, a set of *elementary operations* ($+, -, \cdot, /$ and elementary functions such as sine or exponential) is specified. It is assumed that the right hand side of the system $f$ consists only of operations from this set.

The second assumption is, that though the explicit expression for $f(x)$ remains unknown, its *algorithmic representation*, that is, a step-by-step specification of evaluation of this function for given $x$ in terms of the previously defined operations and functions, is given. An example of such a specification is a routine in some programming language. There are different ways to formalize this concept, for instance, a *computational graph*. It is here a directed acyclic graph, in which nodes and edges represent the elementary operations and their dependencies. This kind of formalization helps to develop data structures, which "record" the execution of the goal function and in this way make implementation of algorithmic differentiation possible.

At last, *elemental differentiability* is assumed, that is, that every elementary operation is continuously differentiable up to some order $p$, $0 \leq p \leq \infty$. Under this assumption, the *chain rule* can be applied to the mathematical formalization of the algorithmic representation [23].

The result of this application is a system of linear equations, which can be solved either by forward or by backward substitution. That is, by knowing only the derivatives for the elementary operations, it is possible to obtain the derivatives of an arbitrary function in which they consist of.

The two approaches mentioned above form the *forward* and *backward modes* of algorithmic differentiation respectively. The former was employed in MOBILE to get the Jacobian of the right side of a modeled initial value problem.

Also, Taylor coefficients up to the order $p$ of a given function can be obtained by knowing the rules of their calculation for elementary operations, of which this function consists ($p$ here is the order defined by the assumption of elemental differentiability) [21]. Combining forward mode and Taylor coefficients' algorithms, it is possible to find necessary Jacobians of the Taylor coefficients.

The next section specifies the software which implements the methods mentioned above and its place in IIVPS. Besides, different IIVPS are compared from the point of view of their later integration into MOBILE.

### 4.3 Available Software and Choice for Integration into MOBILE

It appears more difficult to build a validated solver than a standard one if we think in terms of additional software involved. In addition to implementation of the actual algorithm, a validated solver must incorporate an interval arithmetic library and a package for automatic differentiation to compute interval Taylor coefficients for the solution of an ODE and of the associated variational equation (see section 4.2).

As already mentioned, a package for automatic differentiation based on the algorithmic representation of functions is required in the present case. It was decided to use FADBAD [26] and TADIFF [27], both of which are built on top of the interval library PROFIL/BIAS. This library was also used to extend kinematics in MOBILE to the interval case (section 3), so that the choice of the algorithmic differentiation software was mostly predetermined by this fact as well as by the programming language (C++ as in MOBILE).

As to the IIVPS actual algorithm, one can choose between several packages: AWA [21] and its C++ version AWACOO [28], ADIODES [23], COSY INFINITY [29] and VNODE [24]. But as experience showed, none of them can be integrated into MOBILE as is. The choice between these packages has to be made primarily according to their programming language and availability of algorithmic differentiation option.

AWA and COSY INFINITY can be ruled out right away, because they are programmed in PASCAL-XSC and FORTRAN, respectively. All the others match MOBILE in language, but still they are not interchangeable. Though all of them solve systems given in their exact symbolic representation, which is not suitable for integration into MOBILE, VNODE and ADIODES do that in an "enhanced" way and use the packages FADBAD and TADIFF for differentiation. VNODE has some advantages in efficiency over ADIODES and is easier to use; besides, it introduces some new validated algorithms. For that reason it was decided to take this package as a basis of an interval solver in MOBILE.

Hence, the process of integration of an IIVPS into MOBILE can be carried out as follows: first, enhancing MOBILE with algorithmic differentiation; second, adjusting VNODE to MOBILE; third, assembling the parts to the verifying extension. This approach offers the facility of automatized calculation of derivatives in MOBILE directly, which helps to validate the parameters of a mechanical system and treat the uncertainty in input data.

### 4.4 An Interval Extension of MOBILE for Modeling of Dynamics

**Fusion of VNODE and MOBILE.** Before talking about implementation of an interval solver in MOBILE, it is necessary to consider programming of such a solver in general, which can give some insight into problems arising on VNODE and MOBILE's fusion. To implement the theory outlined in section 4.2, the Taylor coefficients $f^{[k]}$ of the solution to the given system as well as those of the solution to the associated variational equation have to be generated. To achieve this, one *transforms* the algorithm for $f$ to get an algorithm for derivatives. One of the popular techniques of such a transformation is *overloading*: The definitions of the quantities and elementary operations involved in the algorithmic representation of $f$ are supplemented with corresponding differentiation rules. Therefore, to solve an initial value problem with the help of algorithmic differentiation implemented through overloading, three data types are required:

1. One to get the actual values of the right hand side (for example, `INTERVAL` from PROFIL/BIAS)
2. One to get the Taylor coefficients of the solution (for example, `TINTERVAL` from TADIFF)
3. One to get the Taylor coefficients of the solution to the associated variational equation (`TFINTERVAL` from FADBAD/TADIFF)

As a result, we are confronted with a "designing problem": On the one hand, a function to compute the right hand side is given, on the other hand, two

further types of computational graphs are required, so two additional functions appear, which differ from the first one only in data types. It can inflate the code, especially in case of MOBILE, where the goal function is "assembled" from subroutines of many transmission elements.

VNODE solves this problem very elegantly with the help of templates. At the same time, this fact as well as some other usage features of the program (see [24]) prevents its "as is" integration into MOBILE. For example, one has either to know the exact expression for the right hand side or to modify all the sub functions involved in the evaluation of this right hand side into templates. As already mentioned, the former alternative is not acceptable for MOBILE, which does not provide any explicit symbolic representations. The latter one also seems impracticable, because the body of sub functions imposed by transmission elements is too large and intricate to be replaced by template analogs. Another such example is that a MOBILE user has to work with VNODE (and not MOBILE itself) to model the dynamics of his interval systems. These and other features of VNODE have to be adjusted to MOBILE. The next sections describe how this adjustment was put into practice.

**Enhancing MOBILE with Algorithmic Differentiation.** As already mentioned at the end of section 4.3, the process of integrating VNODE into MOBILE should start with making MOBILE capable of algorithmic differentiation to provide for the derivatives required. This enhancement is carried out with the help of the packages FADBAD/TADIFF, which use operator overloading for algorithm transformation. Therefore, a new data type is required to manage all the computations automatically, with minimum effort on the user's side. The use of algorithmic differentiation is legitimate in this case, because the set of elementary operations in MOBILE (for the transmission elements we are considering at present) consists of addition, subtraction, multiplication, division, sine and cosine functions, which satisfy the assumption of differentiability introduced in section 4.2. We would like to point out that it holds for an arbitrary $p$, because the branching (IF-statements) which is present in the transmission elements does not depend on the argument of the goal function and thus has no influence on the order of differentiability.

The basic AD data type of the extension is called `MoADInterval`. Its structure is shown in Fig. 7. With the help of this construction it is possible to obtain all computational graphs fast and through a single function call. On the other hand, the use of memory is inefficient, so this data type is in need of further optimization. For example, a hierarchy based on inheritance (`MoADInterval` from `MoTInterval` from `MoInterval`) would help to avoid unnecessary memory allocation for instances `TFINTERVAL` if only Taylor coefficients of the goal function (and not those of the variational equation) need to be computed.

To provide for algorithmic differentiation, all MOBILE structures have to be modified with the help of `MoADInterval`. This modification presupposes systematic replacement of all variables of ordinary MOBILE data types with ones of the corresponding algorithmic differentiation types. At first, all instances of `MoReal`

```
class MoADInterval {
// Data
     INTERVAL Enclosure;    // the function value
     TINTERVAL TEnclosure;  // the Taylor coefficients of the solution
     TFINTERVAL TFEnclosure;// those of the variational equation
//Methods
     . . .
}
```

**Fig. 7.** The basic AD data type

are replaced with `MoADInterval`, then those of `MoAngle` with `MoADIntervalAngle`, and so on. Also, some restructuring is required. It originates rather in the conversion from floating-point to interval computations than in algorithmic differentiation itself (e.g. usage of a verified linear systems solver instead of an ordinary one).

At present, the following transmission elements are extended to work with the new data type: `MoADIntervalMap`, `MoADIntervalElementaryJoint`, `MoADIntervalSphericalJoint`, `MoADIntervalMassElement`, `MoADIntervalRigidLink`, and the part of `MoADIntervalSpringDamper`, which does not contain square roots. The objects for pre-modeling of dynamics are transformed as well: `MoADIntervalMapChain` (provides the basis for the concatenation of elementary kinetostatic transmission elements into complex composite systems), `MoADIntervalEqmBuilder` (computes the equations of motion of a mechanical system), `MoADIntervalDynamicSystem` (provides a basic interface for obtaining of the model equations in state-space form), and `MoADIntervalMechanicalSystem` (computes the equations of motion of general mechanical systems in state-space form).

Most changes have to be made to the member function `SolveLinearSystem` of the class `MoADIntervalEqmBuilder`. This function is used to obtain accelerations from the known mass matrix and force vector. We cannot call the standard routine for linear systems' solving provided by PROFIL/BIAS. The function `Lss` from PROFIL/BIAS does not allow for its usage with data types other than `INTERVAL`. However, both computational graphs are necessary for accelerations, because they constitute a part of an ordinary differential equations system in state-space form. Therefore, the basic validated algorithm [30] has to be reimplemented for the data type `MoADInterval`. The simple duplication of this algorithm, which is already slower for intervals, proves to be too expensive in practice: apart from taking more time than floating-point algorithms, it enlarges computational graphs too much, which considerably slows down actual integration. Other solutions, which better suit algorithmic differentiation and accelerate further computations, require thorough understanding of MOBILE's inner algorithms and are being developed at present.

Some of the other methods employed in MOBILE's transmission elements can be optimized with respect to intervals as well (for example, to reduce overestimation). But at this stage of development the authors are more concerned

with the implementation of the validated integrator itself rather than with overestimation due to one-to-one replacements of floating-point values with interval ones. In case of a linear systems solver it is crucial to choose an appropriate method; in other cases this can be temporarily neglected.

With all mathematical, state objects, and transmission elements modified, one can start the actual implementation of an IIVPS for mechanical systems modeled by MOBILE.

**The Interval IVP Solver:** `MoADIntervalAWAIntegrator`**.** To comply with the usual MOBILE interface for solving initial value problems, two of its classes (`MoIntegrator`, `MoAdamsIntegrator`) were modified. If the changes required by the former were of rather formal character (mostly alterations in type names), the latter was to incorporate the modified VNODE and, consequently, had to be thoroughly transformed.

The base class for all integrator objects in MOBILE as well as in its verifying extension is `Mo(ADInterval)Integrator` (Fig. 8 shows the latter version of its implementation). It provides the basic variables and routines, which are to be inherent in all objects capable of solving modeled initial value problems for given mechanical systems.

```
class MoADIntervalIntegrator : public MoADIntervalMap{
protected:
  MoADIntervalDynamicSystem* System ;              // the system to be solved
  int neq ;                                 // number of equations in the system
  MoADInterval* Y ;                                              // the solution
  MoADInterval* Yd ;      // its time derivative (the right side of the system)
  MoReal Time ;                                  // the current integration point
  MoReal tStart ;  MoReal tEnd ;    //the starting and end point of integration
  MoReal tInterval ;                                              //the stepsize
  ...
public:
  MoADIntervalIntegrator () ;
  MoADIntervalIntegrator ( MoADIntervalDynamicSystem &sys_ ) ;
  ~MoADIntervalIntegrator () ;
  virtual int getOrder ( ) ;
  virtual void giveState ( MoADInterval* st ) ;
  ...
};
```

**Fig. 8.** The base class for integrator objects

This class is derived from the abstract class `MoADIntervalMap`. The "philosophy" of MOBILE presupposes that after setting the integration interval (`tStart` and `tEnd`), one can consider every integrator object as a transmission element (hence the derivation from `MoADIntervalMap`) that travels along the solution trajectory in small steps determined by `tInterval`. This way it is easier to communicate with the visualizing part of the software. In the interval exten-

sion of MOBILE, though, visualization is not as important as the problem of obtaining guaranteed enclosures. Besides, one of the special features of interval methods is the internal step size control, which either accelerates the process of system's integration or provides the time interval over which the solution is proved to exist. Therefore, the interval extension of MOBILE does not allow for user-administrated step size control at present. It implements its integrators as transmission elements which provide the solution at the end point, "documenting" their way to that state in a text file. On the other hand, a facility for interval visualization (whatever it may be) might become necessary in the future, so the variable `tInterval` is preserved in the base class. At present, it is equal to the difference between `tEnd` and `tStart`, but it is still possible to derive an object, in which this stepsize is controlled from the outside.

A class to handle the solution of initial value problems, which is derived from the class with above properties, is implemented as shown in Fig. 9.

```
class MoADIntervalAWAIntegrator : public MoADIntervalIntegrator {
  // work space
  MoADInterval* Tin; MoADInterval* Tout;
  ...
  // cahracteristics of the integrator
  int T_Ord;                                          // the order of the Taylor series
  IntegrationAlgorithm Algorithm;                  // the name of the algorithm of solution
  IntegrationFunction CompEncl;                     //execution of the respective algorithm
  ...
  // internal integrator functions
  void dfn ( ... ) ;                  //computational graphs of the right side of the system
  void tGenerateTerms(...); ...                       // Taylor coefficients of the solution
  void vGenerateTerms(...); ... // Taylor coefficients of the solution to the variational eq.
  bool PredictStep( ... ) ; ...                                          // stepsize control
  bool Validate(...) ;                                              // the proof of existence
  void CompEnclITSDirect(...) ;                          // the direct Taylor series method
  void CompEnclITSQr(...) ;                                 // the QR-factorization method
  ...
  // internal auxiliary functions
  ...
public:
  MoReal TOL;                                                                  //tolerances
  void doMotion( ... ) ;                                                    //integration itself
  ...
};
```

**Fig. 9.** The interval integrator class

Apart from data defined by `MoADIntervalIntegrator`, the object contains some arrays to store the necessary computational graphs (`Tin`, etc.), a relative tolerance variable, and some other auxiliary information. At present it incorporates two validated methods: the direct Taylor series method and the QR-factorization method. The constructor of `MoADIntervalAWAIntegrator` takes the name of the method (stored in `Algorithm`) and the order of Taylor series decomposition (`T_Ord`) as parameters and sets the function pointer `CompEncl` to either `CompEnclITSDirect` or `CompEnclITSQR` depending on what method has been chosen. Functions for the generation of the Taylor coefficients of the

solution to the system (`tGenerateTerms`, etc.) and to the respective variational equation (`vGenerateTerms`, etc.) are now member functions of the class `MoADIntervalAWAIntegrator`.

The member function `doMotion`, which is inherent in all transmission elements, starts the process of integration and calls the function `AWACOO`. This function is the adaptation of VNODE's `VODE_SOLVER::IntegrateTo`. It chooses the stepsize control (`PredictStep`) and existence proof (`Validate`) strategies as well as the methods for tight enclosures (`CompEncl`) according to the parameters predefined by the constructor and executes the actual algorithm.

A characteristic feature of VNODE's adjustment to MOBILE is the restructuring of the original object hierarchy of the former in subordination to that of the latter. It becomes necessary partly because of the decision to avoid templates, partly because of the complexity of that hierarchy, resulting from the intention of VNODE's developers to provide maximal flexibility in the choice of solution strategies in order to compare them, which is not the primary goal of an interval solver for MOBILE. The changes can be led down to the following:

- Merging `INTERVAL`, `TINTERVAL`, and `TFINTERVAL` into a single data type which helps to avoid templates
- Substitution of the branch in the VNODE's object hierarchy responsible for computational graphs by additional work space arrays and respective methods in `MoADIntervalAWAIntegrator`
- Absence of separate object branches for stepsize and order control strategies as well as for the existence proof algorithm
- Identification of the VNODE's solver classes with the MOBILE class `MoADIntervalAWAIntegrator`

One can still choose the order of Taylor series, the necessary integration method, etc. But unlike VNODE's assembling of the particular integrator object from many instances of other classes, this choice is performed through a single call of the corresponding `MoADIntervalAWAIntegrator`'s constructor.

`MoADIntervalAWAIntegrator` allows us to obtain validated enclosures of dynamic parameters for general mechanical systems modeled in MOBILE. The next section describes the usage and performance of this class.

**Basic Usage of `MoADIntervalAWAIntegrator` and its Performance.** With the help of the modified transmission elements, the feature of easy handling can be introduced for modeling of dynamics in the same manner as for kinematics in section 3. The main difference from the normal MOBILE program is the attachment of the identifier `ADInterval` to usual data type names. Some constants are also turned into interval ones. Then, the use of the validated initial problem solver `MoADIntervalAWAIntegrator` provides for the dynamics of the system. In the interval case two additional (in comparison with the usual mode) parameters are used by the constructor to determine what order of Taylor series and what validated method are to be applied to solve the initial value problem.

Let us consider the dynamics of a simple pendulum with a damper and compare our results with those obtained with the Adams-Moulton-Bashfort (AMB)

integration algorithm, which are represented in Fig. 10. The pendulum starts the movement at a twenty degree angle with zero velocity and is considered over the time interval $[0; 10]$. The tests were performed on a Pentium IV (CPU 2.26GHz, RAM 256 MB). The plot on the left shows the dependence of position and velocity of the pendulum on time, both for the validated and numerical integrator. In this scale one can discern neither the differences between the upper and lower bounds of the solution set, nor its difference from the trajectory obtained with the AMB algorithm. The plot on the right demonstrates that these differences exist. It represents the position in relation to midpoints of intervals obtained with the QR-factorization method. In the scale of $10^{-11}$, it is evident that the AMB curve lies outside of the validated boundaries in that section of the diagram, where the solution oscillates, but towards the end of integration interval, where the solution stabilizes, it lies within or near them. The explanation of this effect may be the error of the AMB algorithm, which is considerable for oscillating solutions. The computing time in this case is about ten seconds.

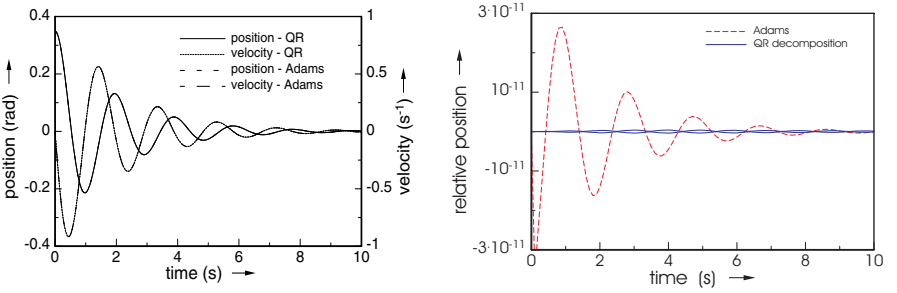



**Fig. 10.** Comparison of the validated and Adams-Moulton-Bashfort solution for a simple pendulum with a damper: trajectories (*left*) and their close-ups (*right*)

The example above is one of the simplest. Yet, in more complicated cases the extension does not work that fast. For instance, it takes eleven hours to model a triple pendulum. This system has three independent variables (angles of the three arms), so its state-space model has six unknowns, which is a few in terms of physical systems. However, this model is relatively complicated from the point of view of the transmission elements involved. The more transmission elements are used, the larger the computational graphs are and the longer it takes to traverse them.

The important loss of time is caused by the member function `SolveLinearSystem` of the class `MoADIntervalEqmBuilder`. It was not necessary to solve a system of linear equations in the first example. The mass and the force being scalar, we simply divided the former by the latter to obtain the sought out second equation of the state-space representation. In case of the triple pendulum it becomes necessary, because the dimension is higher and the quantities in question

are no more scalar. We have to use validated methods to solve the respective system of linear equations, which is very time consuming for the reasons mentioned at page 150.

The arms of the triple pendulum start to move from their initial angles $\beta_1 = 30°$, $\beta_2 = \beta_3 = 40°$ with zero velocities. The system is considered over the time interval $[0; 8]$. Again, the results shown in Fig. 11 demonstrate the overall similarity of the solution trajectories obtained with the QR-factorization interval Taylor series algorithm and the AMB method.

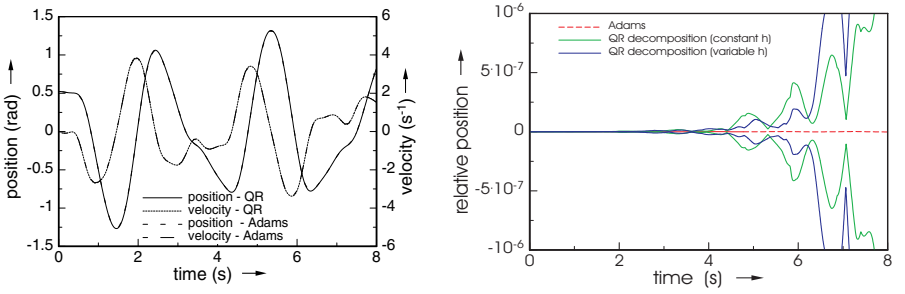



**Fig. 11.** Comparison of the validated and Adams-Moulton-Bashfort solution for a triple pendulum: trajectories (*left*) and their close-ups (*right*)

The diagram to the right compares the position obtained with the help of the QR-factorization algorithm using constant and variable stepsize control strategies with the AMB solution. Again, to see the differences clearer, the three trajectories are represented in relation to the midpoints of obtained intervals. This time the AMB trajectory lies within interval boundaries and its deviance from points of reference is indiscernible in the scale of $10^{-6}$. The intervals obtained with variable stepsize control strategy have in average bigger diameters than those of constant stepsize strategy, but the computing time increases considerably in the latter case. Judging by the scale of the diagram, the obtained intervals are not as tight as in the first example, which is caused partly by the use of the linear systems solver, partly by the validated method itself and the lack of interval optimization in transmission elements. Besides, the system in question is chaotic, that is, extremely sensitive to the initial conditions (so that initial nearby points can evolve quickly into very different states). It is interesting to point out that we obtained better enclosures for non-chaotic systems with point interval initial values. The separation of the influences of the wrapping effect and chaotic solution behavior on the results and the investigation on interconnections of chaotic systems and intervals may be a promising topic for further research.

### 4.5 Example: Point-Tracking Manipulator

As a close to life example of verified modeling in MOBILE, consider a two armed manipulator conceived to track a point $P$ with the help of a camera. $P$ moves along a straight line $\mathcal{L}$ (Fig. 12, left). The manipulator consists, as in the case of the example of section 3.3, of two revolute joints and two rigid links with masses, where all variable names are retained. The tracking of the motion of $P$ is accomplished by measuring the distances $g_z$, $g_y$ of $P$ relative to the coordinate planes orthogonal to the $z$ and $y$ axes of the end-effector, respectively, which corresponds to a simple camera model. Tracking error is compensated by applying a force

$$Q_g = -\left[P\, g + I \int_{\tau=0}^{t} g(\tau)\mathrm{d}\tau + D\, \dot{g}\right] \tag{9}$$

in the direction of the corresponding axis on the end effector, where $P, I, D$ are the constants of a corresponding PID controller. The overall differential equations involve two second order equations of the form

$$\mathbf{M}\,(\,\underline{q}\,)\,\ddot{\underline{q}} + \underline{b}\,(\,\underline{q}\,,\,\dot{\underline{q}}\,) = \underline{Q}\,(\,\underline{q}\,,\,\dot{\underline{q}}\,,\,\underline{x}\,;\,t\,)\quad, \tag{10}$$

where $\underline{q} = [\beta_1\,,\,\beta_2]^{\mathrm{T}}$, and $\underline{x}$ represent two additional variables stemming from the ordinary first order differential equations

$$\dot{\underline{x}} = [\dot{x}_1\,,\,\dot{x}_2]^{\mathrm{T}} = [g_y\,,\,g_z]^{\mathrm{T}} \tag{11}$$

by which the integral in the PID controllers are substituted by the new variables $x_i$. Thus, one obtains an overall system of six first order differential equations with the initial values for $\underline{q}$ corresponding to the initial configuration of the manipulator and $\underline{x}_0 = [0, 0]^{\mathrm{T}}$. Assuming that the point $P$ is moving by a known function of time, one obtains a non-autonomous system of differential equations.

To model the "PID-behavior" of this system, the new force transmission element `MoPIDForce` is required as well as its verified version. Two elements of this type are used to track the point and a `MoHarmonicVibration` to move it.

Fig. 12 (right) shows the position and velocity of the camera over the time interval $[0; 17]$, obtained with the QR factorization (the order of Taylor decomposition is 24, tolerances are set to $10^{-8}$) and AMB algorithms. Once again we observe the overall similarity of the respective trajectories as well as the tightness of the enclosures. Regrettably, the computing time amounts to ten hours, which is nonetheless faster than for the triple pendulum, because the linear equations' system to be solved is smaller. As already mentioned, we continue working on this problem.

It is interesting to point out that we had to model the dynamics for the non-autonomous system here, whereas the assumption for IIVPS (section 4.2) was autonomy. Having no expressions for the resulting differential equations, we were not able to transform the original system into the autonomous one. But using algorithmic differentiation, it was possible to solve the problem not only in this particular case, but also in general.

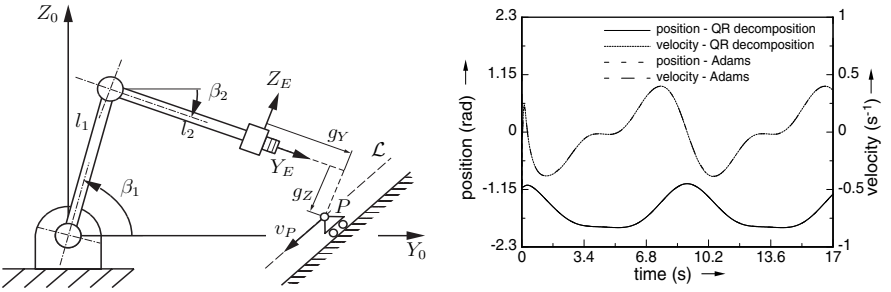



**Fig. 12.** The point-tracking manipulator (*left*) and the position and velocity of its camera obtained with validated and Adams-Moulton-Bashfort algorithms (*right*)

Another interesting question connected with the example would be modeling dynamics of the manipulator with `MyIntervalSlacknessJoint` instead of `MoElementaryJoint`, as shown in section 3.5 for its kinematics. The implementation in this case presupposes the enhancement of wrapping effect's reduction methods, described in section 3.5, with algorithmic differentiation and their integration into the IIVPS system of MOBILE. The current task for the authors is to allow for this modeling possibility, that is, to bring software from sections 3 and 4.4 together.

Modeling with `MyIntervalSlacknessJoint` would provide for computations with some uncertainty in the parameters, which would help to calculate, for example, their tolerances. That means, how much are the axes of the arms allowed to deviate from being concentric without influencing the overall system behavior too much, or how an inaccuracy in the length of the arms affects the tracking of the point. Until now, we were able to validate the numerical results and show their correctness.

## 5 Conclusion: Prospects and Achievements

We have shown how interval techniques and modeling software can be combined to the advantage of both: the latter acquires the opportunities of validated modeling and uncertainty treatment, the former its real life application. But we have also pointed out the areas, where "naive methods" of such an integration produce unsatisfactory results and new algorithms, based on better understanding of MOBILE's inner structures and principles have to be developed. Such is the improved treatment of interval transmission of velocity, acceleration, and force (similar to that presented in section 3.5) aiming at reducing the wrapping effect caused by rotations, which is at the final stage of implementation now. Such is also the attempt to change MOBILE's algorithm of building differential equations' systems in state-space form using additional information provided by knowledge of derivatives, which will reduce computing time for the simulation

of dynamics. For this simulation level, implementing further verification algorithms, optimizing the data structures as well as a thorough solver's testing are imminent.

Kinematics and dynamics of mechanical systems can now be modeled with interval methods. To achieve this, basic mathematical objects, kinetostatic state objects, and kinetostatic transmission elements as well as dynamics' premodeling objects from MOBILE were transformed to provide interval calculus along with automatic calculation of Taylor coefficients for the system itself and for the corresponding variational equation. As a result, the respective extensions of MOBILE were implemented.

The connection of MOBILE and interval arithmetic allows for easier integration of reliable algorithms based on intervals. For example, methods for verified distance calculation and an accurate fault tree algorithm for calculating a failure distribution of a mechanical system using the failure distributions of its key subsystems will be adapted to MOBILE in the future.

## References

1. A. Kecskeméthy. *Objektorientierte Modellierung der Dynamik von Mehrkörpersystemen mit Hilfe von Übertragungselementen*. Fortschrittberichte VDI, Reihe 20 Nr. 88. VDI-Verlag, Düsseldorf, 1993.
2. A. Kecskeméthy. *MOBILE Version 1.3. User's guide*, 1999.
3. O. Knüppel. PROFIL/BIAS—A Fast Interval Library. *Computing*, 53:277–287, 1994.
4. E. Dyllong, W. Luther, and H. Traczinski. Modelling Geometric Objects and Tolerances with Intervals: Data Exchange with ISO Standard STEP. Presented at Validated Computing 2002, submitted paper, May 2002.
5. Werner Schiehlen, editor. *Multibody Systems Handbook*. Springer-Verlag, Berlin, Heidelberg, New York, 1990.
6. N. Orlandea. ADAMS (Theory and applications). In A.D. DePater and H.B. Pacejka, editors, *Proc. 3rd Seminar on Advanced Vehicle Systems Dynamics (Amalfi, Mai 1986)*, pages 121–166, 1987.
7. D. Rosenthal. Order n formulation for equations of motion of multibody systems. In *SDIO/NASA Workshop on Multibody Simulations, 3 September*, pages 1122–1150, 1987.
8. H. Brandl, R. Johanni, and M. Otter. A very efficient algorithm for the simulation of robots and similar multibody systems without inversion of the mass matrix. In *IFAC/IFIP/IMACS Symposium on Robotics*, Wien, December 1986.
9. Roy Featherstone. *Robot Dynamics Algorithms*. Kluwer Academic Publishers, Boston, Dordrecht, Lancaster, 1987.
10. E. Kreuzer and W. Schiehlen. NEWEUL — Software for the generation of symbolical equations of motion. In Schiehlen [5], pages 181–202.
11. J. Wittenburg and U. Wolz. The program MESA VERDE for robot dynamics. In *Proc. 3rd International Symposium of Robotic Research*, Gonvieux (Chantilly), France, 1985.
12. Rebecca Wirfs-Brock and Brian Wilkerson. Object-oriented design: A responsibility-driven approach. In *OOPSLA '89 Proceedings*, pages 71–75, October 1989.

13. A. Kecskeméthy. Sparse-matrix generation of Jacobians for the object-oriented modelling of multibody dynamics. In M.S. Pereira and J.A.C. Ambrósio, editors, *Proceedings of the NATO-Advanced Study Institute on Computer Aided Analysis of Rigid and Flexible Mechanical Systems, Volume II (Contributed Papers)*, pages 71–90, Tróia, Portugal, 27 June – 9 July 1993.
14. American National Standards Institute / Institute of Electrical and Electronic Engineers, New York. *IEEE Standard for Binary Floating-Point Arithmetic*, 1985. ANSI/IEEE Std. 754-1985.
15. E. Loh and G. W. Walster. Rump's Example Revisited. *Reliable Computing*, 8(3):245–248, 2002.
16. W. Krämer. Die Berechnung von Standardfunktionen in Rechenanlagen. In S. D. Chatterji, B. Fuchssteiner, U. Kulisch, R. Liedl, and W. Purkert, editors, *Jahrbuch Überblicke Mathematik 1992*, pages 97–115. Vieweg, Braunschweig, 1992.
17. G. Alefeld and J. Herzberger. *Introduction to interval computations*. Academic Press, 1983.
18. W. Krämer. A Priori Worst Case Error Bounds for Floating-Point Computations. *IEEE transactions on computers*, 47(7):750–756, 1998.
19. W. Luther and H. Traczinski. Error propagation control in MOBILE : extended basic mathematical objects and kinetostatic transmission elements. In A. Kecskeméthy, M. Schneider, and C. Woernle, editors, *Advances in Multibody Systems and Mechatronics*, pages 267–276. Institut für Mechanik und Getriebelehre, Technische Universität Graz, Graz, 1999.
20. C. Hörsken and H. Traczinski. Modeling of Multibody Systems with Interval Arithmetic. In W. Krämer and J. Wolff von Gudenberg, editors, *Scientific Computing, Validated Numerics, Interval Methods*, pages 317–328, Dordrecht, 2001. Kluwer Academic Publishers.
21. R. Lohner. *Einschließung der Lösung gewönlicher Anfangs- und Randwertaufgaben und Anwendungen*. PhD thesis, Universität Karlsruhe, 1988.
22. A. Griewank. *Evaluating derivatives: principles and techniques of algorithmic differentiation*. SIAM, 2000.
23. O. Stauning. *Automatic validation of numerical solutions*. PhD thesis, Technical University of Denmark, Lyngby, 1997.
24. N.S. Nedialkov. *The design and implementation of an object-oriented validated ODE solver*. Kluwer Academic Publishers, 2002.
25. R. Lohner. On the ubiquity of the wrapping effect in the computation of the error bounds. In Ulrich Kulisch, Rudolf Lohner, and Axel Facius, editors, *Perspectives on Enclosure Methods*, pages 201–217. Springer Wien New York, 2001.
26. C. Bendsten and O. Stauning. FADBAD, a flexible C++ package for automatic differentiation using the forward and backward methods. Technical Report 1996-x5-94, Technical University of Denmark, Lyngby, 1996.
27. C. Bendsten and O. Stauning. TADIFF, a flexible C++ package for automatic differentiation using Taylor series. Technical Report 1997-x5-94, Technical University of Denmark, Lyngby, 1997.
28. E. Auer. Ein verifizierender Anfangswertproblemlöser in C++ zur Integration in MOBILE. Master's thesis, Gerhard Mercator Universität Duisburg, 2002.
29. M. Berz and K. Makino. Verified integration of ODEs and flows using differential algebraic methods on high-order Taylor models. *Reliable Computing* **4**, pages 361–369, 1998.
30. R. Hammer, M. Hocks, U. Kulish, and D. Ratz. *Basic Algorithms (Basic numerical problems). Numerical toolbox for verified computing*. Springer-Verlag, Berlin, Heidelberg, New York, 1995.

# Reliable Distance and Intersection Computation Using Finite Precision Geometry

Katja Bühler[1], Eva Dyllong[2], and Wolfram Luther[2]

[1] VRVis Zentrum für Virtual Reality und Visualisierung Forschungs-GmbH
Donau-City-Strasse 1
1220 Wien, Austria
buehler@vrvis.at
http://www.vrvis.at
[2] Universität Duisburg-Essen
Fakultät für Ingenieurwissenschaften
47048 Duisburg, Germany
{dyllong, luther}@informatik.uni-duisburg.de
http://www.informatik.uni-duisburg.de

**Abstract.** In this paper we discuss reliable methods in the field of finite precision geometry. We begin with a brief survey of geometric computing and approaches generally used in dealing with accuracy and robustness problems in finite precision geometry. Moreover, two reliable geometric algorithms based on these approaches are presented. The first one is a new distance algorithm for objects modeled in a common octree. The results are exact and include good bounds on all subdivision levels. Using smoother enclosures on the highest level, a link is provided to well-known algorithms for convex and non-convex objects.
We discuss the general concept and advantages of special bounding volumes with representations directly connected to the representation of the enclosed object: Implicit and parametric Linear Interval Estimations (I)LIEs are roughly speaking, just thick planes enclosing the object. They are constructed using Taylor models or affine arithmetic. The particular structure of (I)LIEs allows the construction of effective hierarchies of bounding volumes and the development of effective intersection tests for the enclosed object with rays, boxes and other LIEs. In addition, a fast reliable intersection test for two LIEs is presented in detail.

## 1 Introduction

Geometric algorithms are widely used in robotics, computer graphics, computer aided design or any simulations of a virtual environment. Common representations for objects are constructive solid geometry models (CSG-models), boundary representation models (B-Rep-models) or tessellations (e.g. octrees). Single surfaces or surface patches are mostly represented in parametric or implicit form, or as subdivision surfaces. The choice of the appropriate representation is dependent on the application.

R. Alt et al. (Eds.): Num. Software with Result Verification, LNCS 2991, pp. 160–190, 2004.

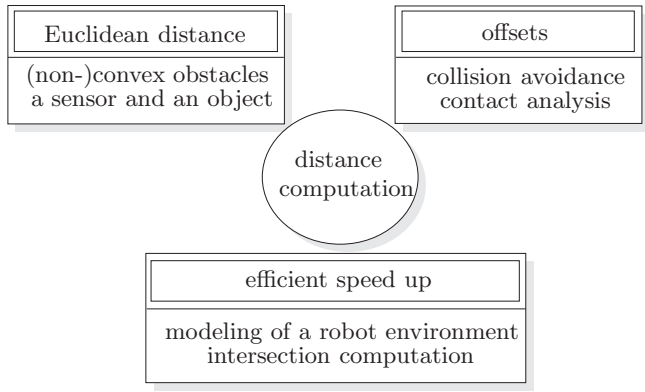



**Fig. 1.** Applications of distance computation.

Because exact modeling of an object is very time consuming and can be carried out only in certain special cases, polyhedral structures are recommended for path planning in robotics. Octrees are often used for scene reconstruction from sensor data. Parametric surfaces are an important tool for objects which are located near the robot. In the field of contact analysis and path planning, efficient distance and intersection algorithms play a decisive rule in most simulations.

Distance algorithms are most frequently used in robotics (see Figure 1) and also in computer games not only to determine the distance between two obstacles in the environment of a robot or between a sensor point and an object, but also to obtain the results of difficult geometric comparisons without actually doing them. If we know that two surfaces are too far apart to intersect, we do not need the more expensive intersection calculations. Here bounding volumes are a common technique, which relies on a hierarchical model representation of the two surfaces using axis-aligned bounding boxes (AABBs), oriented bounding boxes (OBBs), parallelepipeds, discrete-orientation polytopes (DOPs), spheres, or new concepts of parameterized bounding volumes such as Linear Interval Estimations (LIEs) [7] or Implicit Linear Interval Estimations (ILIEs) [8]. Hierarchies of bounding volumes provide a fast way to perform collision detection even between complex models. The determination of the offset to a surface is another example of a problem which can be formulated in terms of distance computation. Hierarchical algorithms are also applied in computer graphics to perform point- or box-surface incidence tests and ray-surface or surface-surface intersections. Here, it is of interest not only to test whether an intersection exists, but also to compute the (exact) intersection set. Some applications for such algorithms are, for instance, the rendering of implicit and parametric surfaces, the voxelization of implicit objects, the computation of surface-surface intersections, and visibility computations.

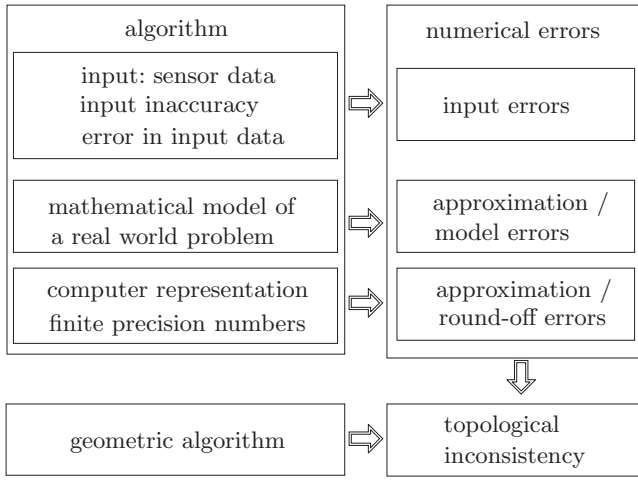



**Fig. 2.** Accuracy and robustness problems.

The methods mentioned here represent only a small selection of the geometric algorithms and structures commonly applied in the field of object modeling, contact analysis and path planning.

Usually, they are sophisticated algorithms designed and proven to be correct for objects defined over the domain of real numbers which can only be approximated on the computer. Due to rounding errors many implementations of geometric algorithms simply compute the wrong results for input values for which they are supposed to work. Numerical non-robustness in scientific computing is a well-known and widespread phenomenon. The implementation of an algorithm is in general considered robust if its output is always the correct response to some perturbation of the input, and stable if the perturbation is small.

Although non-robustness is already an issue in a purely numerical computation, it is more intractable in a geometric one. To appreciate why the robustness problem is especially hard for geometric computation, we need to understand what makes a computation geometric. Geometric computation involves not only numerical computations but also combinatorial structures as well as certain nontrivial consistency conditions between the numerical and combinatorial data. Consequently, in purely numerical computations a result becomes unusable when there is a severe loss of precision. In geometric computations errors become serious when the computed result leads to inconsistent states of the program or is qualitatively different from the true result, e.g. combinatorial structure is wrong. Accordingly, a loss of robustness related to geometric algorithms must always be understood in both its numerical and its topological meanings (see Figure 2).

Researchers trying to create robust geometric software use one of two approaches. The first is some form of exact computation in which every numerical quality is computed exactly (explicitly, if possible) and which relies on big num-

ber packages and use filters to make this approach viable. Alternatively, they can continue to use floating-point or some other finite precision arithmetic, and try to make their computation robust.

Although exact computation is a safe method of achieving robustness, it is somewhat inefficient for most robotic applications. Exact geometric computation requires that every evaluation is correct, which can be achieved either by computing every numeric value exactly (e.g. using exact integer or rational arithmetic) or by employing some implicit or symbolic representation that allows values to be computed exactly. But an exact computation is only possible whenever all numeric values are algebraic or if the result of the geometric algorithm depends only on the signs of some quantities to be known (such information's can be obtained with adaptive methods). Furthermore, the cost of an arithmetical operation is no longer constant, as in the case of floating-point arithmetic, but depends upon its context and increases due to geometric constructions in which a new geometric structure is produced from an old one. Because of this perceived performance cost, the exact geometric computation does not appear to be widely used in robotics. Besides, in most robotic applications the input data are arbitrary real numbers (e.g. sensor data) which have to be cleaned up into exact values (e.g. an inexact input point can be viewed as the center of a small ball) before being fed to the exact algorithm.
On the other hand, the common alternative to exact computation, finite precision geometry, is faster, readily available, and widely used in practice; however exactness and robustness are no longer guaranteed. Here, correct and verifiable geometric reasoning using finite precision arithmetic is demanded.

This paper aims to present new methods for the design of accurate and robust finite precision geometric algorithms which yield reliable results despite rounding errors caused by the limited precision of the computation. It begins with a short overview of the most common reliable techniques in the field of finite precision geometry: interval arithmetic or affine arithmetic, approaches which reduce the effect of overestimation caused by interval evaluations, Taylor models, and the exact scalar product.
Section 3 proposes a new algorithm for distance computation between octrees based on the use of the exact scalar product. Another center of interest in this section is the development of efficient and accurate algorithms for distance calculation between a sensor point fixed on a robot and a target or obstacle (or obstacles) in a complex environment. An accurate distance algorithm for convex and non-convex polyhedra with a priori error bounds of the computed values is provided. Robust solutions to these geometric problems are used in collision-free path planning if a given end-effector is moving amid a collection of (un)known obstacles from an initial to a desired final position as well as in dealing with the resulting contact problems. The advantages of the special structure of (implicit) linear interval estimations computed using Taylor models and affine arithmetic are demonstrated in Section 4, followed by a detailed discussion of robust intersection and enumeration algorithms for implicit and parametric surfaces based on spatial subdivision. Finally, Section 5 summarizes the results.

## 2 Handling of Robustness Problems

Because there is no general theory on how to deal with them, the handling of robustness problems in finite precision geometry takes a number of different approaches. In order to avoid inconsistent decisions these fall into two categories. The first places higher priority on topological and combinatorial data, while the second emphasizes numerical data.

The topology-oriented approach leads to robust algorithms which never crash and compute output with essential combinatorial properties, but the computed numerical values do not necessarily correspond to the real solution of the geometric problem being addressed. Typically a topology-oriented algorithm does not treat sign computations producing sign zero. In those cases where the numerical value of a sign computation is zero, it will be replaced by a positive or negative value, whichever is consistent with the current topology. For this reason the topology-oriented approach is not suitable for certain computations, such as determining the real distance points between two objects.

In such cases numerical approaches are more appropriate. Their typical strategies are based on an association of tolerances to geometric objects in order to represent uncertainties. The representation of a value by an approximation and an error bound or an interval is a numerical analogue of these strategies. In this context the term *interval geometry* can also be found [33].

### 2.1 Interval Arithmetic

Approximation and error bounds define an interval that contains an exact value. In interval arithmetic the real numbers are stored as intervals with floating-point endpoints. Computations on the numbers are performed as sets of computations on the interval bounds, e.g. $[a, b] + [c, d] = [a + c, b + d]$. Interval arithmetic is the most common technique providing reliable solutions for many numerical problems. Unfortunately, overestimation resulting from standard interval evaluations is an often criticized drawback of interval arithmetic. See Alefeld and Herzberger [1] for further reading.

### 2.2 Epsilon Geometry

Another method closely related to interval arithmetic is epsilon geometry, which was defined by Guibas, Salesin and Stolfi [21] and uses an epsilon predicate instead of a Boolean value to obtain information on how much the input satisfies the predicate. An epsilon predicate returns an interval that identifies a region over which the predicate is definitely true, definitely false or simply uncertain. So far, epsilon geometry has been applied only to a few basic geometric predicates. Moreover, it is not clear how to handle the regions of uncertainty.

### 2.3 Affine Arithmetic

Affine arithmetic, first proposed by Comba and Stolfi [11], is an extension to interval arithmetic which reduces the effect of overestimation by taking into

account the dependencies of the uncertainty factors of input data, approximation and rounding errors. In this way, error expansion can often be avoided and tighter bounds on the computed quantities achieved.
When using this approach, each numerical number is stored as an affine form

$$\hat{x} = x_0 + x_1\varepsilon_1 + x_2\varepsilon_2 + \ldots + x_n\varepsilon_n, \tag{1}$$

where $\epsilon_i \in [-1, 1]$ denotes a *noise symbol* representing one source of error or uncertainty. $x_0$ is the *central value* of the affine form and the $x_i$ are *partial deviations*. For each new source of error a new noise symbol $\epsilon_i$ is introduced and added to the affine form.

Each interval can be expressed as an affine form, but an affine form can only be approximated by an interval as it carries much more information. An interval describes only the general uncertainty of the data, whereas affine arithmetic splits this uncertainty into specific parts. Thus, a conversion from affine forms to intervals in most cases implies a loss of information.

*Let* $\hat{x} := x_0 + x_1\epsilon_1 + x_2\epsilon_2 + .... + x_n\epsilon_n$ *be the affine form of the fuzzy quantity* $x$. $x$ *lies in the interval*

$$[\hat{x}] := [x_0 - \xi, x_0 + \xi]; \qquad \xi := \sum_{i=1}^{n} |x_i|$$

$[\hat{x}]$ *is the smallest interval enclosing all possible values of* $x$.

Let $X = [a, b]$ be an interval representing the value $x$. Then $x$ can be represented as the affine form

$$\hat{x} = x_0 + x_k\epsilon_k$$

with $x_0 := (b + a)/2;\ x_k := (b - a)/2$.

Affine arithmetic is slower than standard interval arithmetic, but in cases where there might be error correlation from one computation step to the next, this approach is beneficial.

### 2.4 Arithmetical Approaches

Certain approaches might be described as being based primarily on arithmetical - as opposed to geometric - considerations. A highly precise evaluation of arithmetical expressions provides a solid tool for the solution of various geometric problems. The idea of arithmetical approaches is to isolate the basic operations (primitives) which have to be handled in a numerically correct way, where the manner in which the respective operands are represented is crucial. The primitives have to be implemented in such a way that they yield a result which is as close as it can be to the best possible machine representation. The computational depth of geometric algorithms has to be kept low to control the propagation of round-off errors.

Since scalar products occur frequently and are important basic operations in many geometric computations, it is advantageous to perform the scalar product calculation with the same precision as the basic arithmetical operations. Using

the exact scalar product delays the onset of qualitative errors and improves the robustness of the implementation. Other arithmetical approaches, like the permutation of operations combined with random rounding (up and down), can also be used [33].

### 2.5 Taylor Models

The idea of this approach is the representation of a (multivariate) function as a Taylor polynomial plus an interval that encloses the range of the remainder: the *Taylor model* of the function.

**Definition 1.** *Let be* $f \in C^{n+1}(D)$; $D \subset I\!R^m$ *and* $\mathbb{B} \in I\!I\!R^m$ *an interval box with* $\mathbb{B} \subset D$. *Let* $T$ *be the Taylor polynomial of order* $n$ *of* $f$ *around the point* $\boldsymbol{x}_0 \in \mathbb{B}$.

- *An interval* $I$ *with* $\forall \boldsymbol{x} \in \mathbb{B} : \quad f(\boldsymbol{x}) - T(\boldsymbol{x}) \in I$ *is called an* $n$-th order Remainder Bound of $f$ on $\mathbb{B}$.
- *A pair* $(T, I)$ *is called an* $n$-th order Taylor model of $f$.
- *A set of all remainder bounds is called the* Remainder Family*, the optimal enclosure of the remainder is called the* Optimal Remainder Bound.

Thus, a Taylor model is a polynomial of $n$-th order enclosing the approximated function on the interval box $\mathbb{B}$.

Berz and Hofstätter [5] define an arithmetic for Taylor models based on uni- and bivariate arithmetical operators and basic functions. It turns out that these methods are similar to interval arithmetic for the case $n = 0$.

Taylor models have a remarkable feature with respect to the quality of the approximation and its convergence: If $\mathbb{B}$ decreases, $I$ will decrease in size as the $(n+1)$-st power of the size of the box $\mathbb{B}$.

## 3 Accurate Distance Algorithms

Obstacles are often modeled or reconstructed from sonar and visual data leading to uncertain information. Descriptions based on polyhedral or hierarchical octree structures lead to a considerable reduction of data, which makes effective storing and processing possible. First, we will deal with objects represented by an octree in three dimensions and then with a more general $n$-tree in higher dimensions.

Octrees are very suitable for building environments where obstacles must be taken into account when considering collision-free path planning as they enable the location of free and occupied regions based on accurate distance calculations.

In Figure 3 a non-convex object is represented by an axis-aligned, level-three octree. The round nodes are gray because they have white and black leaves. Since the octree is constructed through the subsequent division of boxes, all constructed nodes are boxes whose boundary representations can be computed using an appropriate fixed-point arithmetic.

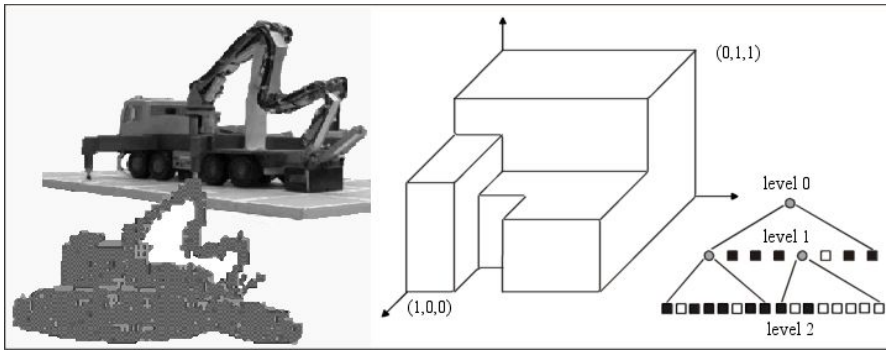



**Fig. 3.** Octree representing a non-convex object.

### 3.1 An Accurate Distance Algorithm for Octrees

The distance calculation between two objects represented by a common octree which has depth $N$ and extra color information in gray nodes is based on a simple computation of the distance between two boxes.

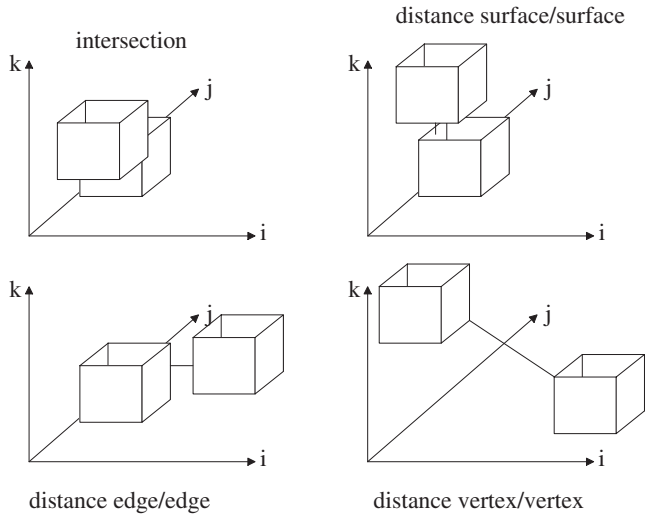



**Fig. 4.** Various examples of positioning two boxes.

First, we establish a procedure $dist^2(Q_1, Q_2)$ for the rectilinear axis-aligned boxes $Q_1, Q_2$ described by a vertex point with the smallest coordinates and the length of three edges:

$$Q_1 : [X_1, X_2, X_3, h_1, h_2, h_3] = I_1 \times I_2 \times I_3$$
$$Q_2 : [Y_1, Y_2, Y_3, k_1, k_2, k_3] = J_1 \times J_2 \times J_3$$

We introduce a case-selector determining by where the first box lies with respect to the other (outside below or above, cutting):

$$c_n := \begin{cases} (Y_n - X_n - h_n)^2, Y_n > X_n + h_n \\ (X_n - Y_n - k_n)^2, X_n > Y_n + k_n \\ 0, \text{ otherwise} \end{cases}, \quad n = 1, 2, 3.$$

The following cases appear (including also the other cases surface to vertex etc.):

- Intersection:

$$I_1 \cap J_1 \neq \emptyset \wedge I_2 \cap J_2 \neq \emptyset \wedge I_3 \cap J_3 \neq \emptyset \Longrightarrow dist^2 = 0$$

- Surface to surface (the distance vector may move on opposite facets; $l, m, n$ pairwise disjoint):

$$I_l \cap J_l \neq \emptyset \wedge I_m \cap J_m \neq \emptyset \wedge I_n \cap J_n = \emptyset \Longrightarrow dist^2 = c_n$$

- Edge to edge (the distance vector may move on opposite edges):

$$I_l \cap J_l \neq \emptyset \wedge I_m \cap J_m = \emptyset \wedge I_n \cap J_n = \emptyset \Longrightarrow dist^2 = c_n + c_m$$

- Vertex to vertex:

$$I_1 \cap J_1 = \emptyset \wedge I_2 \cap J_2 = \emptyset \wedge I_3 \cap J_3 = \emptyset \Longrightarrow dist^2 = c_1 + c_2 + c_3$$

If the entries $X_1$, $X_2$, $X_3$, $X_1 + h_1$, $X_2 + h_2$, $X_3 + h_3$, $Y_1$, $Y_2$, $Y_3$, $Y_1 + k_1$, $Y_2 + k_2$, $Y_3 + k_3$ are machine numbers, the square of the distance can be calculated up to 1 *ulp* with the aid of the exact scalar product. If a fixed point arithmetic is used, the results are exact.
We will now assume that the octree represents two objects, a white (w) and a black (b) one, and that the leaves are integrally white or black depending on the represented object or red (r) for the free space. We further assume that the octree has no bw-boxes which would yield $dist^2 = 0$.

The second part of our algorithm computes the distance between the two objects using the distance formulae between two cubes from part one:

- Initialize the lists $LB$, $LW$, $LG$, the distance $D = 3$, and boxes $W = [0, 0, 0, 0, 0, 0]$ and $B = [1, 1, 1, 0, 0, 0]$.
  /*The lists $LB$ and $LW$ are void, $LG$ contains the unit cube. $LB$ contains actual black boxes, $LW$ contains actual white boxes, $LG$ contains gray boxes of the $i$-th level */

- For all levels $i = 0, 1, ..., N$ /* $N$ depth of the octree */ do

  /* Step 1: Fill lists $LW, LB$ */

For all children $Q$ of all boxes of size $2^{-i}$ on level $i$

/* Update $LG$ */
If $Q = white$ then
{ $Q \to LW$; For all $T \in LB$ do
if $(dist^2(Q,T) < D)$
then { $D := dist^2(Q,T); W := Q; B := T$ }
}
else if $Q = black$ then
{ $Q \to LB$; For all $T \in LW$ do
if $(dist^2(Q,T) < D)$
then { $D := dist^2(Q,T); W := T; B := Q$ }
}
/* Two or more different kinds of subboxes*/
else if $Q = gray$ then $Q \to LG$;

/* Step 2: Drop all irrelevant boxes; define $\min(\emptyset) = 0$*/

For all $T \in LB, T \neq B$ do

For all $Q \in LG$ with attribute wr or bwr calculate $dist^2(Q,T)$;
$dist^2_{wr} := \min\{dist^2(Q,T) | Q \text{ has attribute wr}\}$;
$dist^2_{bwr} := \min\{dist^2(Q,T) | Q \text{ has attribute bwr}\}$;
if $dist^2_{wr} > D$ and $dist^2_{bwr} > 3 \cdot 2^{-2i-2}$ then drop $T$ in $LB$;

For all $T \in LW, T \neq W$ do

For all $Q \in LG$ with attribute br or bwr calculate $dist^2(Q,T)$;
$dist^2_{br} := \min\{dist^2(Q,T) | Q \text{ has attribute br}\}$;
$dist^2_{bwr} := \min\{dist^2(Q,T) | Q \text{ has attribute bwr}\}$;
if $dist^2_{br} > D$ and $dist^2_{bwr} > 3 \cdot 2^{-2i-2}$ then drop $T$ in $LW$;
return $D$.

### 3.2 Remarks

This algorithm can be modified to return a list of all solutions. To this end, it is necessary to establish a list of pairs of boxes with the same temporary distance. The algorithm provides good upper and lower bounds: the temporary distance $D$ is an upper bound, but we may use $D = 3 \cdot 2^{-2i}$ if there is a bwr-box on level $i$. It is also possible to compute lower bounds. To this behind determine the greatest level $i$ with bwr-boxes. Replace on an arbitrary level $j \geq i$ all br-boxes with black boxes and all wr-boxes with white boxes. Then apply the algorithm to return $D$ as a lower bound.
The algorithm works in any higher dimension when the definition of the case-selector is generalized to arbitrary dimensions.
On level $i$ we find $2^{6(i+1)}/3$ as an upper bound for the number of box comparisons and distance calculations. Thus, in the worst case, overall complexity is

$O\left(2^{6(N+1)}\right)$. If we do not drop irrelevant black and white boxes, the complexity is bounded by the product of the number of black and white boxes.
On the highest level tighter (convex) enclosures of the objects inside the boxes can be used to obtain better bounds for $D$. Then the simple distance computations in the first step are replaced by an algorithm for convex objects.
For an implementation it is not necessary to create the lists $LB$, $LW$, $LG$. All work can be done on the underlying data structure by traversing the octree in a certain manner and using appropriate flags in the nodes.

### 3.3 Examples

The first example concerns a level-three quadtree. In executing the algorithm the white box on the right-hand side is dropped. The result is found in the second and third quadrant. By applying a convex hull algorithm on the set of extreme vertices we find simple convex enclosing sets.

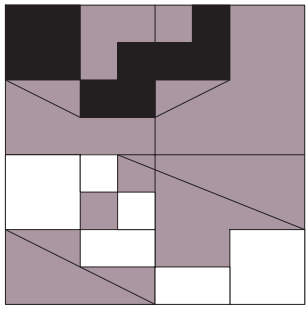


**Fig. 5.** Quadtrees and convex hull of two objects.

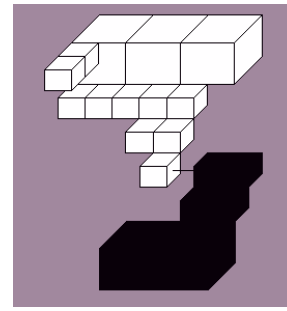


**Fig. 6.** Octree with two objects on level three.

The convex hull of the extreme vertices is shown in Figure 5. The distance remains unchanged. In the next example (see Figure 6) the algorithm eliminates the boxes near the boundary $z = 1$ with respect to the coordinate system shown in Figure 3.

### 3.4 Convex Hulls

Now let us turn our attention to the objects obtained by representing three-dimensional convex sets $S$ by octrees to apply distance theorems for this kind of sets. If the sets are non-convex, they can be split into convex parts. Building the octree corresponds to a certain kind of rasterization. So the question arises whether the objects are digital convex. If we replace each box on the highest level with its center point $x$ we obtain sets of grid points $S_\Delta$. This approach allows us to apply results from digital convexity (d.c.):

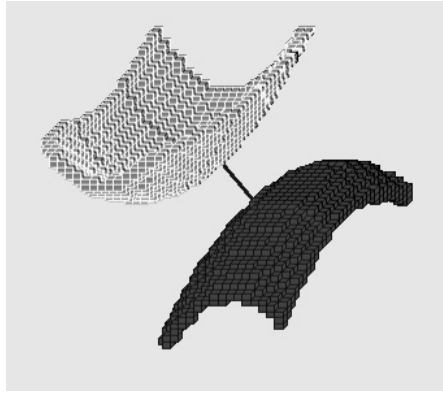


**Fig. 7.** Parabolic objects - level 5.

**Theorem 1 (see [16]).** *A digital set $S_\Delta \subseteq Z^d$, the set of all d-dimensional vectors whose components have integer values, is digital convex if and only if for each point of $x \in Z^d \setminus S_\Delta$ there is a hyperplane with normal vector $x'$ and distance $\alpha$ to the origin such that $x \cdot x' = \alpha$ and $y \cdot x' > \alpha$ for all $y \in S_\Delta$. If for each boundary point $x$ of $S_\Delta$ there is a hyperplane such that $x \cdot x' = \alpha$ and $y \cdot x' \geq \alpha$ for all $y \in S_\Delta$ then $S_\Delta$ is digital convex.*

This theorem finds its analogous result due to Tietze in the context of continuous convexity.

Unfortunately, Tietze's theorem, which says that the condition $x \cdot x' = \alpha$ and $y \cdot x' \geq \alpha$ should be verified only locally when deriving continuous convexity, does not hold in the digital world. For this reason, a test for digital convexity cannot be done in a time proportional to the number of neighbors and boundary points, as was shown by a counter example given in Eckhardt [16]. However, if the set $S_\Delta$ is simply connected and all the boundary points fulfill the interior point condition, i.e., each point $x \in \partial S_\Delta$ has at least two 8-neighbor points belonging to $S_\Delta$ and these points are all connected in the 4-neighborhood topology, then the result of Tietze's theorem holds true.

A simpler way to proceed is to use the concept of extreme vertices of boxes on the boundary. A vertex is said to be an extreme vertex if none of the adjacent boxes belongs to the object. In the case of a quadtree there are three neighboring boxes; for octrees there are seven boxes. The convex hull of all extreme vertices is constructed to obtain an enclosure of the object. Obviously, the convex hull also contains the original set $S$.

Then we can apply our distance algorithms for convex sets and obtain lower bounds for the distances. This approach also opens the way to dynamic algorithms for moving objects. It is well known that rotational motions of octrees lead to an unwanted wrapping-effect, which can be avoided by using the convex hulls of the objects [25].

### 3.5 Accurate Distance Algorithms for Convex and Non-convex Polyhedra

Generally, distance algorithms focus on objects represented by convex polyhedra, which are defined as the convex hull of points in three-dimensional space. Although these approaches can be applied for convex polytopes (bounded polyhedra) in three-dimensional space, a wider class of objects is permitted since it is also possible to treat conveniently non-convex shapes as a union of convex polytopes.

There are two main classes of distance algorithms for convex polyhedral models. In the first class algorithms are based on Voronoi regions, like the Lin-Canny (LC) algorithm [24] and its software implementations, such as I-Collide [10], V-Clip [27], or SWIFT++ [17]. Another class is the simplex-based Gilbert-Johnson-Keerthi (GJK) algorithm [19] and its various extensions, including non-convex objects [29] and proximity queries with collision detection [4].

One drawback of the original LC algorithm is that it does not readily handle penetrating polyhedra; a second is its lack of robustness when applied to models in degenerate configurations. The GJK-like algorithms are more robust than LC; they can also handle penetration cases. Nonetheless, with GJK-like algorithms, computations generally require more floating-point operations. The collision detection library Q-Collide [9] was spawned from I-Collide, which replaces LC with the GJK algorithm for low-level collision detection. A numerical comparison of some derivations of GJK and LC algorithms was done in [20].

Although the GJK algorithm is widely used in robotics, there has been no verification of the computed results. For this reason, we have implemented an interval version of the GJK distance algorithm for tracking the distance between convex polyhedra which is adapted to sensor-based input data [15].

We are also interested in simple accurate algorithms to calculate the distance between two objects, such as points, collections of axis-aligned boxes, (non-)convex polyhedra or NURBS-surfaces with interval vertices. Accurate finite precision algorithms have been developed based on suitable projections and using controlled rounding and the exact scalar product whereby a verified enclosure of the solution is ensured [12].

If the end-effector or the sensor is taken to be a single moving point, an efficient distance algorithm, which does not rely on convex properties and thus is applicable to non-convex polyhedral surfaces has been developed [13]. Under the same assumption the problem has been solved for the more difficult case of NURBS-defined solids based on subdivision techniques and using an algorithm for the solution of nonlinear polynomial systems proposed by Sherbrooke and Patrikalakis [30]. The extension of this algorithm introduces interval arithmetic, the interval version of the convex hull algorithm, and a modified Simplex algorithm. The new solver allows a verification of obtained results [14] using new criteria to guarantee the existence of zeros within the calculated inclusions [18].

Our algorithm to compute the distance between a point and a non-convex polyhedron does not require decomposing the polyhedron into convex parts or iteration and yields the result with high accuracy [13]. It is possible to derive

the explicit absolute or relative errors in a real distance point and the distance value to the (non-)convex polyhedron as well as the computed approximations of these values.

### 3.6 An Accurate Distance Algorithm between a Point and a (Non-)Convex Polyhedron

Given a point $\boldsymbol{y}$ outside a non-degenerated polyhedron $P$ bounded by $\partial P := \{S_i, i = 1, \ldots, m;\ \boldsymbol{v}_j, j = 1, \ldots, n\}$ with $m$ facets and $n$ vertices.
In the following, the vertices belonging to the facet $S_i$ are denoted by $\boldsymbol{s}_{ik}$, $k = 1, \ldots, t_i$, $t_i > 2$, given in counter-clockwise order, and by $[\boldsymbol{s}_{ik}, \boldsymbol{s}_{i(k+1)}]$, the edges of the facet $S_i$; $k = 1, \ldots, t_i$, $\boldsymbol{s}_{i(t_i+1)} := \boldsymbol{s}_{i1}$.

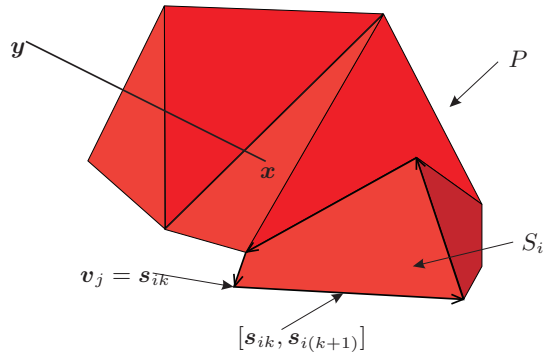



**Fig. 8.** A point $\boldsymbol{y}$ and a non-convex polyhedron $P$.

We are searching for the shortest straight line segment $[\boldsymbol{y}, \boldsymbol{x}]$ between point $\boldsymbol{y}$, which is any point outside of polyhedron $P$, and this polyhedron with $\boldsymbol{x} \in \partial P$. At the beginning, before starting the distance algorithm, we calculate the correctly rounded cross product

$$\boldsymbol{n}_{i2} = (\boldsymbol{s}_{i2} - \boldsymbol{s}_{i1}) \times (\boldsymbol{s}_{i3} - \boldsymbol{s}_{i2}) = \boldsymbol{s}_{i2} \times \boldsymbol{s}_{i3} + \boldsymbol{s}_{i1} \times \boldsymbol{s}_{i2} - \boldsymbol{s}_{i1} \times \boldsymbol{s}_{i3}$$

with $\boldsymbol{x} \times \boldsymbol{y} := (x_2 \cdot y_3 - x_3 \cdot y_2, x_3 \cdot y_1 - x_1 \cdot y_3, x_1 \cdot y_2 - x_2 \cdot y_1)$ for $\boldsymbol{x} = (x_1, x_2, x_3)$, $\boldsymbol{y} = (y_1, y_2, y_3)$, and a normal vector $\boldsymbol{n}_i = \boldsymbol{n}_{i2} / \sqrt{\boldsymbol{n}_{i2} \cdot \boldsymbol{n}_{i2}}$ for all $i = 1, \ldots, m$. Then $E_i$ denotes the plane described by

$$E_i : \ \boldsymbol{x} \cdot \boldsymbol{n}_i - \boldsymbol{s}_{i1} \cdot \boldsymbol{n}_i = 0.$$

For all scalar product computations the algorithm uses the exact scalar product followed by rounding (to nearest):

A: We calculate the distances between point $\boldsymbol{y}$ and each plane $E_i$

$$l_i := \boldsymbol{y} \cdot \boldsymbol{n}_i - \boldsymbol{s}_{i1} \cdot \boldsymbol{n}_i.$$

We store the sign of $l_i$, $i = 1, ..., m$ for future use. There is at least one $l_i > 0$ therefore the set $I := \{i \mid l_i > 0\}$ is not empty, and we can form the set $J$ of all $j \in \{1, \ldots, n\}$ with $\exists_{i \in I} \boldsymbol{v}_j \in S_i$ and the set $K$ of all pairs $(s, r)$ with

$$\exists_{i \in I} \exists_k [\boldsymbol{v}_s, \boldsymbol{v}_r] = [\boldsymbol{v}_{ik}, \boldsymbol{v}_{i(k+1)}],\ s, r \in \{1, \ldots, n\}.$$

Then, for all $i \in I$, the projections onto $E_i$ can be accurately calculated:

$$\boldsymbol{x}_i := \boldsymbol{y} - l_i \cdot \boldsymbol{n}_i.$$

Next, we have to decide whether $\boldsymbol{x}_i$ is in $S_i$. For that purpose we calculate the number of intersections of the ray $\boldsymbol{x}_i + t'(\boldsymbol{m} - \boldsymbol{x}_i)$, $t' \geq 0$, suitable $\boldsymbol{m} \in E_i$, with edges $[\boldsymbol{s}_{ik}, \boldsymbol{s}_{i(k+1)}]$, $k = 1, \ldots, t_i$, avoiding vertices, by solving a system of two equations with two variables. These equations result from setting the first derivatives of the function

$$f(t', t'') = \|\boldsymbol{s}_{ik} + t''(\boldsymbol{s}_{i(k+1)} - \boldsymbol{s}_{ik}) - \boldsymbol{x}_i - t'(\boldsymbol{m} - \boldsymbol{x}_i)\|^2$$

in the variables $t''$ and $t'$ to zero. If $\boldsymbol{x}_i$ belongs to the polygonal surface $S_i$, i.e. if the number of intersections is odd, we remove all edges from $K$ belonging to $S_i$ and calculate for the remaining $(s, r) \in K$ the scalar products

$$w_{si} := (\boldsymbol{y} - \boldsymbol{x}_i) \cdot (\boldsymbol{v}_s - \boldsymbol{x}_i) \quad \text{and} \quad w_{ri} := (\boldsymbol{y} - \boldsymbol{x}_i) \cdot (\boldsymbol{v}_r - \boldsymbol{x}_i)$$

and, if $w_{si} \leq 0$ and $w_{ri} \leq 0$, we redefine $K := K \backslash \{(s, r)\}$. Then we set a distance-point $\boldsymbol{x} := \boldsymbol{x}_i$ and the distance $d := l_i$ or update them (if there are points with the same distance, the result of the algorithm will be a list of them), and stop the algorithm if $K = \emptyset$.

B: If $K \neq \emptyset$ after step A, then we have to decide for all edges with $(s, r) \in K$, whether the projection of $\boldsymbol{y}$ to the line

$$\boldsymbol{u}(t) := \boldsymbol{v}_s + t\,(\boldsymbol{v}_r - \boldsymbol{v}_s)$$

meets a point with parameter $0 \leq t \leq 1$. To do so, we form the accurately calculated scalar products

$$\kappa := (\boldsymbol{y} - \boldsymbol{v}_s) \cdot (\boldsymbol{v}_r - \boldsymbol{v}_s) \quad \text{and} \quad \mu := (\boldsymbol{v}_r - \boldsymbol{v}_s) \cdot (\boldsymbol{v}_r - \boldsymbol{v}_s).$$

If $\kappa < 0$ or $\kappa > \mu$, then the projection ray does not meet the section between $\boldsymbol{v}_s$ and $\boldsymbol{v}_r$. Otherwise, the projection point on $[\boldsymbol{v}_s, \boldsymbol{v}_r]$ is given by

$$\boldsymbol{x}_{sr} := \boldsymbol{v}_s + \frac{\kappa}{\mu}\,(\boldsymbol{v}_r - \boldsymbol{v}_s)$$

and the square of the distance by

$$d_{sr}^2 := \frac{((\boldsymbol{v}_r - \boldsymbol{v}_s) \times (\boldsymbol{y} - \boldsymbol{v}_r)) \cdot ((\boldsymbol{v}_r - \boldsymbol{v}_s) \times (\boldsymbol{y} - \boldsymbol{v}_r))}{\boldsymbol{v}_r \cdot \boldsymbol{v}_r - \boldsymbol{v}_r \cdot \boldsymbol{v}_s - \boldsymbol{v}_s \cdot \boldsymbol{v}_r + \boldsymbol{v}_s \cdot \boldsymbol{v}_s}.$$

We replace $J$ by $J\backslash\{s,r\}$. Using the projection point $\boldsymbol{x}_{sr}$ we calculate for all $j \in J$ the scalar products

$$w_{jsr} := (\boldsymbol{y} - \boldsymbol{x}_{sr}) \cdot (\boldsymbol{v}_j - \boldsymbol{x}_{sr})$$

and if $w_{jsr} \leq 0$ we set $J := J\backslash\{j\}$.
If the projection point is the nearest distance point, we update $\boldsymbol{x}$ with $\boldsymbol{x}_{sr}$ and $d$ with $\sqrt{d_{sr}^2}$. We stop the algorithm if $J = \emptyset$.

C: If $J \neq \emptyset$ after step B, we compare the distance values of the paths joining point $\boldsymbol{y}$ and each vertex-point $\boldsymbol{x}_j$, $j \in J$, with the distance found so far and update $d$ and $\boldsymbol{x}$ if necessary.

The accurate distance algorithm works in linear time $O(Cn)$ with an order-constant $C$ depending on the number of successful projections onto facets and edges. Furthermore, it can be used to determine the local distance between a point and any polyhedral surface described by its vertices and oriented facets.

### 3.7 Error Discussion

Let $\varepsilon_l \leq 2^{-52}$ be the rounding error in the floating-point number space $S := (\mathcal{B}, l, em, eM)$ characterized by its base $\mathcal{B}$, mantissa length $l$ and $[em, eM]$ the smallest and largest allowable exponents. Then, for the error estimation of a calculated distance point $\boldsymbol{x} = (x_1, x_2, x_3)$ and the distance value $d = \|\boldsymbol{y} - \boldsymbol{x}\|$ in the cases discussed in steps A, B and C it can be shown [12] that the results in Table 1 are valid.

**Table 1.** Absolute or relative errors in the distance point and value

| Step: | Error estimations |
|---|---|
| A: | $x_v = X_v + \delta_{1,v}(11.032\|\boldsymbol{y}\| + 10.032\sigma_i)\varepsilon_l$<br>$d = D + 4.27\delta_1'(\|\boldsymbol{y}\| + \sigma_i)\varepsilon_l$ (a point to a surface) |
| B: | $x_v = X_v + \delta_{2,v}(2.505\|\boldsymbol{y}\| + 14.515\sigma_i)\varepsilon_l$<br>$d = D(1 + 3.003\delta_2'\varepsilon_l)$ (a point to an edge) |
| C: | $x_v = X_v$, $d = D(1 + 1.76\delta_3'\varepsilon_l)$ (a point to a point) |

$v = 1, 2, 3,\ \sigma_i := \max_k \|\boldsymbol{s}_{ik}\|,\ |\delta_{j,v}| \leq 1,\ |\delta_j'| \leq 1,\ j = 1, 2, 3,$
$D = \|\boldsymbol{y} - \boldsymbol{X}\|,\ \boldsymbol{X} \in \partial P$ the real distance point

### 3.8 Example

The algorithm was implemented in C++ using the library Profil/BIAS [23]. Figure 9 shows the ASCII input file of a non-convex polyhedron. The input file

consists of two parts: the fourteen vertex points of the polyhedron in a Cartesian coordinate system as geometric information and their positions on its nine faces as topological information. The corresponding program layout for the point $\boldsymbol{y}$ lying outside of the polyhedron in the origin of the Cartesian coordinate system is shown on the opposite side of the figure.

```
polyhedron
 -1.0     6.0     0.0
 -4.0     0.0     0.0
  0.0    -4.0     0.0
  4.0     0.0     0.0
  1.0     6.0     0.0
  1.0    -1.0     0.0
 -1.0    -1.0     0.0
 -1.0     6.0    -2.0
 -4.0     0.0    -2.0
  0.0    -4.0    -2.0
  4.0     0.0    -2.0
  1.0     6.0    -2.0
  1.0    -1.0    -2.0
 -1.0    -1.0    -2.0
E
1  2  3  4  5  6  7
14  13  12  11  10  9  8
9  2  1  8
10  3  2  9
11  4  3  10
12  5  4  11
13  6  5  12
14  7  6  13
8  1  7  14
```

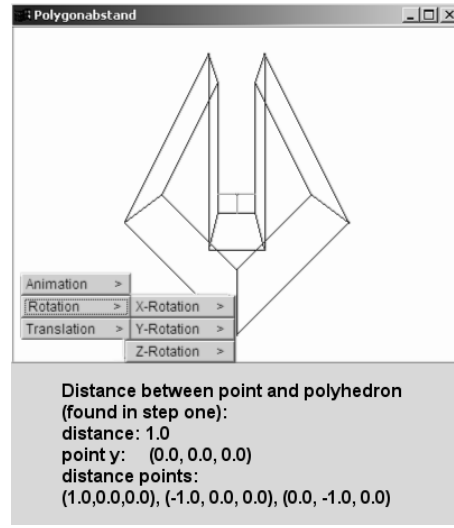



**Fig. 9.** Distance computation: the input file and program layout.

## 4 Reliable Intersection Algorithms

In the previous section accurate distance algorithms widely used for path planning in robotics were described. In computer graphics, it is important to know not only whether two objects intersect, but also where they intersect. Direct ray-tracing of parametric surfaces, rendering and voxelization of implicit curves, surfaces and volumes, as well as the computation of intersection curves are common tasks.

### 4.1 On Bounding Volumes and Subdivision

If a direct solution to the problem is not possible (which is generally the case), the application of a divide-and-conquer strategy is a widespread approach. A common technique for reducing the computational complexity of intersection problems is to subdivide the complex object into simpler objects and to simplify the shape using bounding volumes. Divide-and-conquer approaches to solve object-object intersection problems find by definition all possible intersections,

but due to the piecewise enclosure of the solution information on the overall topology of the intersection gets lost. Postprocessing steps like connectivity determination and sorting are necessary to restore this information. Solutions for this problem can be found in classical literature on computational geometry and e.g. in [2]. Classical bounding volumes are simple solids, such as axis-aligned or -oriented bounding boxes, parallelepipeds, polyhedra or spheres. In general they are computed using range analysis methods based on sampling, exploiting convex hull properties of control points, evaluation of derivatives, or applying affine or interval arithmetic. Bounding volumes should be a reliable enclosure of the object, which is not the case if sampling techniques are used to construct the bounding volume. The direct application of interval or affine arithmetic to compute a bounding volume produces reliable bounds, but these bounds overestimate the object because functional dependencies are not taken into account, or are lost during conversion from affine forms to intervals. Axis-aligned bounding boxes are easy to compute and intersect easily with other axis-aligned bounding boxes or rays; thus, they are well-suited for rapidly providing an insight into the structure of an environment with obstacles and targets. However, in most cases they significantly overestimate curves and surface patches. Therefore, in subdivision-based algorithms many more steps are necessary to reach precision than when using the much better fitting parallelepipeds. On the other hand, an intersection test for two parallelepipeds, for instance, is very complex and time-consuming. Furthermore, all classical bounding volumes are solids, i.e. they provide information only on the location of the whole object. Yet, especially for intersection algorithms for parametric objects, in order to accelerate the computation it would be interesting to be able to derive information on the location of the intersection of the enclosed objects in parameter space from the intersection of two bounding volumes. To summarize, the ideal bounding volume provides a tight and reliable enclosure of the object, is easily calculated, and intersects easily with other, similar bounding volumes.

### 4.2 Linear Interval Estimations

To overcome problems connected with classical bounding volumes, another form of enclosing objects satisfying the requirements for the ideal bounding volume listed above has been introduced for parametric and implicit objects: *Linear Interval Estimations* [7, 8] are defined as the linear approximation of the representation of the enclosed object combined with an interval estimation of the approximation error. An LIE is just a thick (hyper)plane, that can be understood as a continuous linear set of axis parallel bounding boxes. Furthermore, the representation of an LIE corresponds to the representation of the object. This means in the parametric case that the LIE can be parameterized in such a way that its parameterization corresponds to the parameterization of the enclosed object. Each point of the object is enclosed by an "interval point" (an interval box) of the LIE with the same parameters. In the case of the intersection of two LIEs this construction allows direct conclusions on the location of intersections

of the two enclosed objects in object *and* parameter space. This characteristic of LIEs is the most significant difference to other common bounding volumes.

But LIEs are also easy to compute and usually provide much tighter enclosures than common solid bounding volumes. If reliable methods are used to compute the LIE, it also provides a reliable enclosure of the patch. Furthermore, the diameter of the interval part of the LIE contains information about the flatness of the patch and its extension has been proven to be a good termination criterion for subdivision-based algorithms. The linear structure of the LIE reduces the intersection problem of parametric or implicit objects to the solution of (constrained) linear equation systems, which can, in general, be solved much more easily than the original problem.

### 4.3 Parametric LIEs

Parametric objects are widely used in computer graphics and computer aided geometric design. Bézier, B-Spline, and NURBS curves, surfaces and volumes are standard representations used for effective and exact modeling and representation of smooth objects. A general parametric object $S$ over a rectangular parameter domain can be defined as follows:

$$S : \left\{ \begin{array}{cc} = \prod_{i=1}^{m} I_i \in \mathbb{IR}^m & \rightarrow \mathbb{R}^n \\ \boldsymbol{x} & \mapsto \boldsymbol{f}(\boldsymbol{x}) := (f_1(\boldsymbol{x}), ...., f_n(\boldsymbol{x}))^T \end{array} \right\}$$

The corresponding linear interval estimation enclosing the object described above must fulfill the following requirements:

**Definition 2.** A linear map $\mathbb{L} : {}^{*} \in \mathbb{IR}^m \rightarrow \mathbb{R}^n$,

$$\mathbb{L}(\boldsymbol{x}^*) := \quad + \sum_{i=1}^{m} x_i^* \, \boldsymbol{v}_i, \quad \boldsymbol{x}^* = (x_1^*, ..., x_m^*) \in \prod_{i=1}^{m} I_i^* = {}^{*} \in \mathbb{IR}^m \tag{2}$$

with $\in \mathbb{IR}^n$, $\boldsymbol{v}_i \in \mathbb{R}^n, i = 1, ..., n$ is called a *linear interval estimation (LIE)* of the parametric object $\boldsymbol{f}(\boldsymbol{x}) \in \mathbb{R}^n$; $\boldsymbol{x} \in \quad \in \mathbb{IR}^m$ iff there exists a valid reparameterization

$$\phi : \left\{ \begin{array}{c} \rightarrow {}^{*} \\ \boldsymbol{x} \mapsto \phi(\boldsymbol{x}) := \boldsymbol{x}^* \end{array} \right.$$

of $\mathbb{L}$ so that for all $\boldsymbol{x} \in \quad$ holds $\boldsymbol{f}(\boldsymbol{x}) \in \mathbb{L}(\phi(\boldsymbol{x})) = \mathbb{L}(\boldsymbol{x}^*)$.

**Computation.** The general recipe for constructing parametric LIEs is quite simple (see also Figure 10 and 11):

1. Compute a linear approximation of the object.
2. Estimate and enclose the approximation error with an interval vector.
3. Reparameterize the linear approximation so that it corresponds to the parameterization of the object.

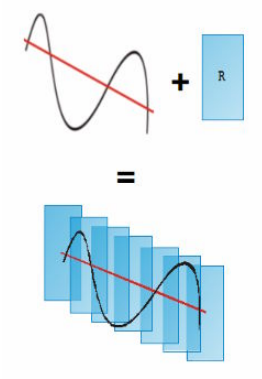



**Fig. 10.** Sketch of the construction of LIEs.

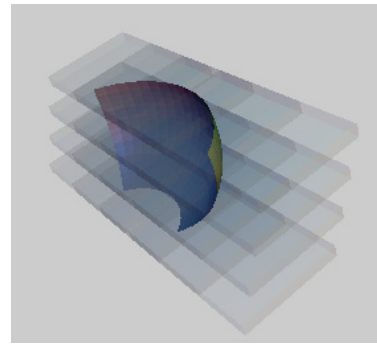


**Fig. 11.** Discrete representation of an LIE based on affine arithmetic.

Two different methods of computing LIEs have been proposed [7]. One is based on first order Taylor models and is straight forward:

**Theorem 2.** *Let* $(T, \quad)$ *be a first order Taylor model of the function* $\boldsymbol{f}(\boldsymbol{x}), \boldsymbol{x} \in \quad$. *Then*

$$\mathbb{L}(\boldsymbol{x}) = T(\boldsymbol{x}) +$$

*is an LIE of* $\boldsymbol{f}$.

$\mathbb{L}(\boldsymbol{x})$ can be written in the form $\quad + \sum_{i=1}^{m} x_i \frac{\partial}{\partial x_i} \boldsymbol{f}(\boldsymbol{x}_0)$ with $\quad := \boldsymbol{f}(\boldsymbol{x}_0) - \sum_{i=1}^{m} x_i^0 \frac{\partial}{\partial x_i} \boldsymbol{f}(\boldsymbol{x}_0) + \quad$, which already corresponds to the parameterization of the object. Thus, in this case, a reparameterization of the LIE is not necessary.

The second method exploits the intrinsic structure of affine arithmetic:

**Theorem 3.** *Let* $\boldsymbol{f} : \quad \in I\!R^m \rightarrow I\!R^n$ *be* $C^0$ *over* $\quad$ *and* $\quad := \prod_{k=1}^{m} I_k$ *with* $rad(I_k) > 0, k = 1, ..., m$. $\tilde{x}_k := x_0^k + x_1^k \epsilon_k; \quad \epsilon_k \in [-1, 1]$ *denote the affine forms corresponding to* $I_k, k = 1, ..., m$ *and* $\tilde{\boldsymbol{x}} = (\tilde{x}_1, ...., \tilde{x}_m)^T$.

$$\boldsymbol{f}(\tilde{\boldsymbol{x}}) = \tilde{\boldsymbol{f}}(\epsilon_1, ...., \epsilon_m, \gamma_1, ..., \gamma_l) = \boldsymbol{f}^0 + \sum_{k=1}^{m} \boldsymbol{f}^k \epsilon_k + \sum_{i=1}^{l} \boldsymbol{r}^i \gamma_i$$

*with* $\epsilon_k, \gamma_i \in [-1, 1]$ *and* $\boldsymbol{f}^k, \boldsymbol{r}^i \in I\!R^n$ *for all* $k = 1, ..., m, i = 1, ..., l$, *denotes the evaluation of* $\boldsymbol{f}$ *with* $\tilde{\boldsymbol{x}}$. *Furthermore let be* $\quad := \boldsymbol{f}^0 + \mathbb{q}$ *with* $\mathbb{q} := \left[ -\sum_{i=1}^{l} |\boldsymbol{r}^i|, \sum_{i=1}^{l} |\boldsymbol{r}^i| \right]$ *and* $|\boldsymbol{r}^i| := (|r_1^i|, ..., |r_n^i|)^T, \ i = 1, ..., l$. *Then*

$$\mathbb{L}_\epsilon(\epsilon_1, ..., \epsilon_m) := \quad + \sum_{k=1}^{m} \boldsymbol{f}^k \epsilon_k; \quad \epsilon_k \in [-1, 1], k = 1, ..., m \tag{3}$$

*is an LIE of* $\boldsymbol{f}$.

The evaluation of the function describing our object with respect to the affine forms representing the parameter domain, is equivalent to the computation of a point symmetric polytope enclosing our object. The term

$$\boldsymbol{f}^0 + \sum_{k=1}^{m} \boldsymbol{f}^k \epsilon_k \tag{4}$$

describes a subset of the polytope that is a linear approximation of the input object with respect to the input error symbols and therefore also with respect to the original parameters. The sum $\sum_{i=1}^{l} \boldsymbol{r}^i \gamma_i$ describes for each point of (4) an enclosure of approximation and rounding errors introduced during the evaluation, that is estimated in the theorem by the interval vector $\mathbb{q}$.

Formula (3) describes the combination of (4) with the error estimation $\mathbb{q}$ which is after Definition 2 a LIE of $\boldsymbol{f}$ with respect to the input error symbols $\epsilon_k, k = 1, ..., m$ and the parameter domain $[-1, 1]^m$. To reestablish the direct connection between the parameterization of the original object and the parametrization of the corresponding LIE a reparameterization of $\mathbb{L}_\epsilon$ is necessary. The map $\phi$ describes the correspondence between the two parameterizations:

$$\phi : \left\{ \begin{array}{ccc} := \prod_{k=1}^{m} [a_k, b_k] & \rightarrow & [-1, 1]^m \\ \boldsymbol{x} & \mapsto & \phi(\boldsymbol{x}) := (\alpha_1 x_1 - \beta_1, ..., \alpha_m x_m - \beta_m) = (\epsilon_1, ..., \epsilon_m) \end{array} \right.$$

with $\alpha_k := \frac{2}{b_k - a_k}$ and $\beta_k := \frac{b_k + a_k}{b_k - a_k}, k = 1, ..., m$. Finally $\mathbb{L}(\boldsymbol{x}) := \mathbb{L}_\epsilon(\phi^{-1}(\epsilon_1, ..., \epsilon_m))$ describes the parametrization of the LIE with respect to the same parameters and parameter domain of the enclosed object.

**Intersection.** All intersection problems for LIEs with boxes, rays or other LIEs can be described as a system of constrained linear interval equations of the form $A\boldsymbol{x} = \quad$, where $A$ is a thin matrix, $\quad$ is an interval vector and $\boldsymbol{x}$ is the vector of unknown parameters constrained by their predefined domains:

Let be $\boldsymbol{f}(\boldsymbol{x}) \in \mathbb{R}^n; \boldsymbol{x} \in \quad \in \mathbb{IR}^m$ and $\boldsymbol{g}(\boldsymbol{y}) \in \mathbb{R}^n; \boldsymbol{y} \in \quad \in \mathbb{IR}^k$ two parametric objects in $n$-space with their respective LIEs $\mathbb{F}(\boldsymbol{x}) = \ _0 + \sum_{i=1}^{m} x_i \boldsymbol{f}_i$ and $\mathbb{G}(\boldsymbol{y}) = \ _0 + \sum_{j=1}^{k} y_j \boldsymbol{g}_j$. The intersection of $\mathbb{L}$ and $\mathbb{G}$ can be described as the solution of the system of linear equations

$$\sum_{i=1}^{m} x_i \boldsymbol{f}_i - \sum_{j=1}^{k} y_j \boldsymbol{g}_j = \ _0 - \ _0$$

with the constraints $\boldsymbol{x} \in \quad$ and $\boldsymbol{y} \in \quad$. A detailed discussion about how an enclosure of a linear system of interval equations can be computed is found in a number of books and articles (see, for example, [3, 28]). In addition to an enclosure of the solution in object space, an enclosure of the solution in parameter space is also needed for effective parameter domain pruning during a subdivision procedure. If interval arithmetic is applied, these enclosures can be generated partly as a byproduct of the intersection algorithms; additional steps might be necessary to compute tight solutions for all parameters.

We are proposing an effective algorithm for computing the intersection of two LIEs of surface patches in 3-space. Its goal is to compute an interval line that encloses the intersection of two LIEs as narrowly as possible as well as to locate the parameter domains $\tilde{} \subseteq$ and $\tilde{\tilde{}} \subseteq$ defined in Theorem 4, which enclose the parameter values of the interval intersection line as narrowly as possible. The derivation of the algorithm follows a geometric approach similar to the intersection of two parallelograms in space:

1. For each parallelogram compute the intersection points of the four border lines with the carrying plane of the other parallelogram.

2. Intersect all four line segments formed by the intersection points of parallel lines (see Figure 12).

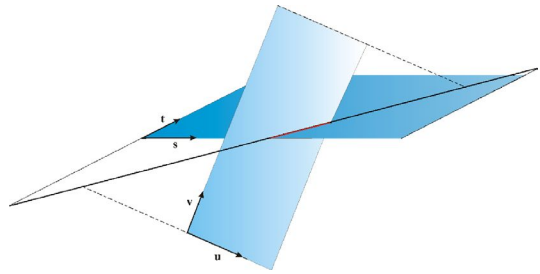



**Fig. 12.** Intersection of two parallelograms in space

Let be

$$\mathbb{L}_1(u,v) = \quad + u\boldsymbol{y}_1 + v\boldsymbol{y}_2; \qquad (u,v) \in I_u \times I_v = \tag{5}$$

$$\mathbb{L}_2(s,t) = \mathfrak{q} + s\boldsymbol{w}_1 + t\boldsymbol{w}_2; \qquad (s,t) \in J_s \times J_t = \tag{6}$$

two LIEs in $\mathbb{R}^3$. Equating (5) and (6) yields

$$(\boldsymbol{y}_1 \quad \boldsymbol{y}_2 \quad -\boldsymbol{w}_1 \quad -\boldsymbol{w}_2) \begin{pmatrix} u \\ v \\ s \\ t \end{pmatrix} = \boldsymbol{r}; \quad \boldsymbol{r} \in \mathbb{r} := \mathfrak{q} - \quad . \tag{7}$$

Under the assumption that each triple of vectors of $\boldsymbol{y}_1, \boldsymbol{y}_2, \boldsymbol{w}_1, \boldsymbol{w}_2$ is linearly independent, the solution set of this underdetermined system is either the empty set, an interval point, or an interval line enclosing the intersection of $\mathbb{L}_1$ and $\mathbb{L}_2$.

Following the geometric approach, the four line segments containing the intersection can be computed in the following way: For each line segment assume that one parameter of $(s,t,u,v)$ is fixed and apply Cramer's rule to the corresponding $3 \times 3$ sub-matrix to solve the system for the two parameters that belong to the other LIE. For example, if we consider $t$ the fixed parameter, (7)

changes to

$$(\boldsymbol{y}_1 \quad \boldsymbol{y}_2 \quad -\boldsymbol{w}_1)\begin{pmatrix} u \\ v \\ s \end{pmatrix} = \boldsymbol{r} + t\boldsymbol{w}_2; \quad \boldsymbol{r} \in \mathbb{r} := \mathbb{q} - \quad . \tag{8}$$

Applied to all parameters this yields the following equations:

$$\begin{array}{ll} S_1(u) = \frac{1}{\alpha}(K + \beta\, u) & U_1(s) = \frac{1}{\beta}(-K + \alpha\, s) \\ S_2(v) = \frac{1}{\gamma}(L - \beta\, v) & U_2(t) = \frac{1}{\delta}(M - \alpha\, t) \\ T_1(u) = \frac{1}{\alpha}(M - \delta\, u) & V_1(s) = \frac{1}{\beta}(L - \gamma\, s) \\ T_2(v) = \frac{1}{\gamma}(N + \delta\, v) & V_2(t) = \frac{1}{\delta}(-N + \gamma\, t) \end{array}$$

where

$$\begin{array}{ll} \alpha := |\boldsymbol{y}_2 \quad -\boldsymbol{w}_1 \quad -\boldsymbol{w}_2| & K := |\boldsymbol{y}_2 \quad \mathbb{r} \quad -\boldsymbol{w}_2| \\ \beta := |\boldsymbol{y}_1 \quad \boldsymbol{y}_2 \quad -\boldsymbol{w}_2| & L := |\boldsymbol{y}_1 \quad \mathbb{r} \quad -\boldsymbol{w}_2| \\ \gamma := |\boldsymbol{y}_1 \quad -\boldsymbol{w}_1 \quad -\boldsymbol{w}_2| & M := |\boldsymbol{y}_2 \quad -\boldsymbol{w}_1 \quad \mathbb{r}| \\ \delta := |\boldsymbol{y}_1 \quad \boldsymbol{y}_2 \quad -\boldsymbol{w}_1| & N := |\boldsymbol{y}_1 \quad -\boldsymbol{w}_1 \quad \mathbb{r}| \end{array}$$

The equations above can be combined to the four intersection lines $\ _i, i = 1, 2$ and $\ _j, j = 1, 2$ parameterized using the same parameters as the LIEs.

$$\begin{array}{ll} {}_1(u) := \begin{pmatrix} S_1(u) \\ T_1(u) \end{pmatrix};\ u \in I_u & {}_2(v) := \begin{pmatrix} S_2(v) \\ T_2(v) \end{pmatrix};\ v \in I_v \\ {}_1(s) := \begin{pmatrix} U_1(s) \\ V_1(s) \end{pmatrix};\ s \in J_s & {}_2(t) := \begin{pmatrix} U_2(t) \\ V_2(t) \end{pmatrix};\ t \in J_t \end{array} \tag{9}$$

Notice that computing $K, L, M$ and $N$ during the first expansion of the determinant according to the elements of $\mathbb{r} = (R_1, R_2, R_3)^T$ avoids overestimation. In this case, each interval $R_i, i = 1, ..., 3$ appears only once in the expression, and the result is an exact enclosure. Furthermore, all occurring matrices are thin and assumed to be regular, which implies that the enclosure of the solution and the solution are identical [28]. Thus, $\ _i, i = 1, 2$ and $\ _j, j = 1, 2$ are optimal enclosures of the non-constrained problem.

The following theorem clarifies how an enclosure of the intersecting line segment and interval enclosures of the intersection in the parameter domains can be computed:

**Theorem 4.** *Let $\mathbb{L}_1$ and $\mathbb{L}_2$ be the LIEs defined by equations (5) and (6), and $_1$, $_2$, $_1$, $_2$ the interval lines defined by equations (9). If each triple of the vectors $\boldsymbol{y}_1, \boldsymbol{y}_2, \boldsymbol{w}_1, \boldsymbol{w}_2$ is linearly independent, an enclosure of the intersection of $\mathbb{L}_1$ and $\mathbb{L}_2$ is provided by each of the interval line segments $_1(u)$, $u \in \tilde{I}_u$, $_2(v)$, $v \in \tilde{I}_v$, $_1(s)$ $s \in \tilde{\tilde{J}}_s$ and $_2(t)$ $t \in \tilde{\tilde{J}}_t$, where*

$$\tilde{} := \ _1(I_u) \cap \ _2(I_v) \cap$$

$$\tilde{} := \ _1(\tilde{J}_s) \cap \ _2(\tilde{J}_t) \cap$$

$$\tilde{\tilde{}} := \ _1(\tilde{I}_u) \cap \ _2(\tilde{I}_v) \cap \tilde{}$$

*The LIEs do not intersect if at least one of the parameter domains* $\tilde{}$, $\tilde{}$ *or* $\tilde{\tilde{}}$ *is empty.*

Notice that the computed intervals are very good enclosures of the solution but might still be slightly overestimated due to the computation of intermediate axis-aligned enclosures. Figures 13 – 15 illustrate the three pruning steps.

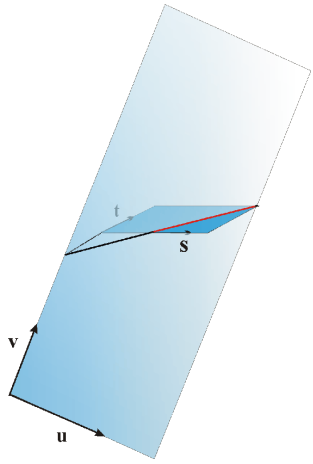



**Fig. 13.** The parallelograms of Figure 12 after the first reduction step,...

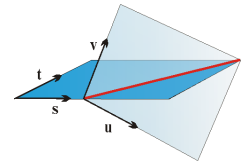



**Fig. 14.** ... after the second reduction,...

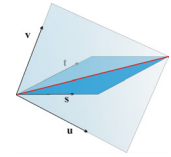



**Fig. 15.** ... and after the third reduction.

**Special cases** occur if two or more of the determinants $\alpha, \beta, \gamma$, and $\delta$ disappear (zeros appear always in pairs), which is equivalent to the LIEs having parallel edges. In the proposed algorithm special cases are handled with two different approaches depending on their type:

1. if two determinants are zero, for $\alpha = 0$ set $\boldsymbol{g}_1(u) :=$ , for $\gamma = 0$ set $\boldsymbol{g}_2(v) :=$ , for $\beta = 0$ set $\boldsymbol{h}_1(s) :=$ , and for $\delta = 0$ set $\boldsymbol{h}_2(t) :=$ ,
2. if more than two determinants are zero, compute an axis-aligned bounding box.

The complete algorithm is described in Figure 16.

**Application of the algorithm.** The algorithm has been implemented in C++ using the Profil/BIAS package [23] and a modification of the affine arithmetic package by van Iwaarden. The algorithm is part of the new subdivision algorithm for surface patches described in [7]. (The reader is referred to the publication for details.) The results can be summarized as follows: The use of LIEs allows

**Algorithm: IntersectionTest**
**Input:** A pair of surfaces $AB$ with corresponding LIEs LIE($A$) and LIE($B$).
**Output:** Explicit: TRUE, if A and B intersect; FALSE, if not. Implicit: If the test is positive, the algorithm also returns the pruned parameter spaces of $A$ and $B$. Value of the intersection flag AB.inters

```
Boolean IntersectionTest(Surface pair AB, LIE(A), LIE(B)){
    Compute α, β, γ, δ, K, L, M, N
    If (α = β = γ = δ = 0)                    // special case 2: LIEs parallel or equal
       If axis parallel bounding boxes intersect
          AB.inters = true; Return true;            // Surfaces might intersect.
       else
          AB.inters = false; Return false;          // Surfaces do not intersect.
    Compute ~;                                  // see theorem 4 and special cases 1
    If (~ = ∅)
       AB.inters = false; Return false;         // Surfaces do not intersect.
    Compute ~;                                  // see theorem 4 and special cases 1
    If (~ = ∅)
       AB.inters = false; Return false;         // Surfaces do not intersect.
    Compute ≈;                                  // see theorem 4 and special cases 1
    If (≈ = ∅)
       AB.inters = false; Return false;         // Surfaces do not intersect.
    A.domain = ~                  // ~ defines the reduced parameter space of patch A
    B.domain = ≈                  // ≈ defines the reduced parameter space of patch B
    Return true;                                //Surfaces might intersect.
}
```

**Fig. 16.** Intersection algorithm for two LIEs enclosing surface patches in 3-space

the subdivision algorithm to be optimized in almost all steps (effective bounding volumes, easy and fast intersection, parameter domain pruning, adaptive subdivision, termination criterion, and so forth). The number of subdivisions and intersection tests, as well as computation time have been reduced dramatically compared to subdivisions that use axis-aligned bounding volumes. For two simple quadric patches in almost rectangular position to one another, both defined on the parameter domain $[-2, 2]^2$, computing the intersection with precision of 0.01 needed 0.01 seconds applying the LIE algorithm, but 2.61 seconds to reach the same precision and reliability with a pure subdivision. The ILIE algorithm described in Section 4.4 needed just 58 subdivisions to enclose the results with interval line segments of a diameter less than 0.01, whereas a reliable enclosure with boxes required almost 100,000 subdivisions. This example demonstrates another important side-effect of the reduction of subdivisions: the amount of data needed to represent the result is much smaller. Final results are enclosed by the interval lines in parameter space and by the corresponding interval surface curves in object space. Tests also show that, applied in a subdivision algorithm

for surface-surface intersection, the proposed algorithm is on average about 25% faster and needs 15% fewer subdivisions than the ILSS algorithm included in the Profil/BIAS package. Particularly remarkable was the observation, that, especially for the two simple surface patches described above, the ILSS algorithm was about 50% slower while requiring the same number of subdivisions. An example is given in Figure 17.

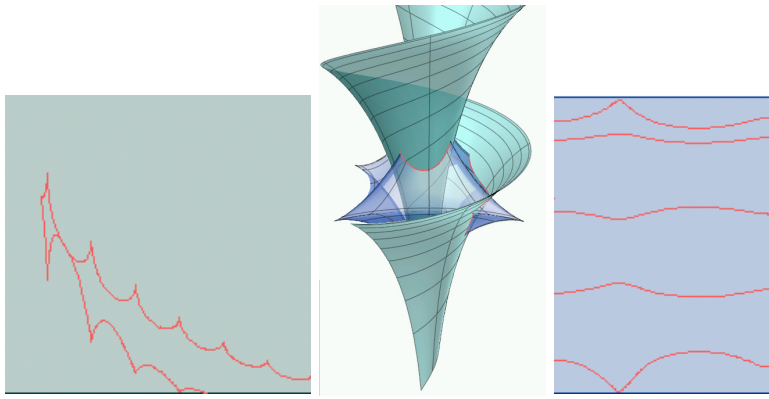


**Fig. 17.** Intersection of a Dini surface and an astroidal. The boxes left and right show the corresponding parameter domains with the intersection in parameter space.

### 4.4 Implicit LIEs

Implicit equations are a powerful tool for the representation of curves, surfaces and volumes in computer graphics. Besides the description of mathematical, physical, geological, and other scientific phenomena, implicit surfaces and volumes are mainly used in CSG-Systems to design complex objects by adding, subtracting, and inverting smooth surfaces.

An implicitly defined object in $\mathbb{R}^n$ is produced by an equation of the form $f(\boldsymbol{x}) = 0, \quad \boldsymbol{x} \in \mathbb{R}^n$, where $f$ can be either a polynomial or any other real valued function. The implicit representation has the advantage that it allows rapid determination of whether a point lies inside ($f(\boldsymbol{x}) < 0$), outside ($f(\boldsymbol{x}) > 0$) or on ($f(\boldsymbol{x}) = 0$) the object. Despite the many other positive features of implicit descriptions of objects, there is one main drawback: The points building the object are defined as zero-set of $f$ - an equation which is in general not explicitly resolvable. The computation of an approximation of this zero-set for visualization and collision detection has been the topic of many publications over the last two decades; finding solutions that are fast and guaranteed reliable is one of the subjects of recent research [26, 31, 32]. Existing solutions for the 3D case compute a polygonization or voxelization, or determine single points on the surface. Algorithms are based on simple space subdivision, marching cubes,

particle systems, ray tracing and stochastic differential equations. Implicit Linear Interval Estimations (ILIEs) can be used to accelerate those algorithms that are based on subdivision and/or incidence tests.

**Definition 3.** *Let* $\mathcal{F} : f(\boldsymbol{x}) = 0,\ \boldsymbol{x} = (x_1, ..., x_n)^T \in \mathbb{R}^n$ *be the implicit definition of an object in* $\mathbb{R}^n$ *and*

$$L(\boldsymbol{x}) := \sum_{i=1}^{n} a_i x_i + J \tag{10}$$

*with* $J \in I\mathbb{R}$ *and* $a_i \in \mathbb{R}, i = 1, ..., n$.
*The interval hyperplane segment inside the axis-aligned box* $\in I\mathbb{R}^n$

$$\mathcal{L} := \{\boldsymbol{x} \in \ \mid 0 \in L(\boldsymbol{x})\}$$

*is called the* Implicit Linear Interval Estimation (ILIE) of $\mathcal{F}$ on *, iff for all* $\boldsymbol{x} \in (\mathcal{F} \cap\ )$ *holds* $0 \in L(\boldsymbol{x})$.

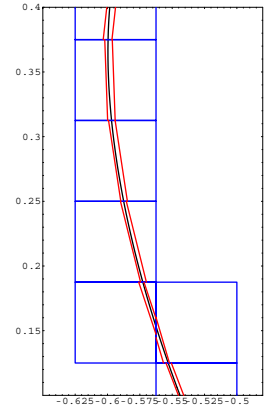



**Fig. 18.** ILIEs enclosing a curve.

**Computation.** The computation of ILIEs follows roughly the same strategy as in the parametric case. The general recipe for generating ILIEs can be described as follows:

1. Compute any linear approximation $l_f(\boldsymbol{x})$ of $f(\boldsymbol{x})$ on a cell .
2. Estimate the approximation error with an interval $J$.
3. Combine both to obtain an ILIE $L_F :\ 0 \in l_f(\boldsymbol{x}) + J$ of $F$.

Again the linearization can be done using, for example, affine arithmetic or Taylor models. The characteristics of ILIEs are also similar to those of parametric LIEs: They are a kind of thick linearization of the object and the diameter of the interval part can be used as criterion for flatness. Furthermore, if affine arithmetic is used for the computation, low additional computational costs are required compared to a cell/object evaluation, singularities are not a problem and the ILIEs provide a tight enclosure due to implied Tchebycheff approximation.

**Application of ILIEs to enumeration algorithms.** A classic enumeration algorithm for implicit objects works in the following way:

- Define an initial cell (an axis-aligned box) where the object has to be detected.
- Test whether this box interferes with the object.
- If it does, subdivide and test sub-cells until the termination criterion is fulfilled.
- If it does not, the object does not intersect the cell; stop.

In this algorithm, the cell-object interference test is the most expensive and important part. ILIEs can help to optimize the whole process in the following ways (for a detailed description see [8]):

- The cell-object incidence test and the computation of the corresponding ILIE can be done in (almost) one step.
- The diameter of the interval part of the ILIE can be used as termination criterion.
- ILIEs allow *cell pruning*. An ILIE encloses the object in many cases much more tightly than the corresponding cell. An axis-aligned cell can be easily reduced to those parts containing the ILIE and the enclosed object using interval/affine arithmetic. (Iterated) cell pruning applied on each computed sub-cell reduces the number of necessary subdivisions significantly.
- Unnecessary cell-object tests can be reduced doing a pre-test with the ILIE of the mother cell.
- Cell pruning also allows highly effective adaptive subdivision strategies.

**Implementation and experiments.** Up to now ILIEs have only been implemented using affine arithmetic based a modification of the affine arithmetic package of van Iwaarden. To prove the usability of ILIEs they have been applied for reliable plotting of implicit curves [6] and the enumeration of implicit surfaces [8] (see also Figure 19). The results can be summarized as follows:

The introduction of ILIEs allowed a complete redefinition of classic enumeration algorithms. In many cases ILIEs provide much better enclosures than axis-aligned cells. The results are much better adapted to the topology of the object. The number of necessary subdivisions decreased substantially. The number of necessary ILIEs to represent a result with certain precision is radically less using ILIEs than using axis-aligned cells. Thus, if the subdivision is used as a basis for polygonization, many fewer polygons are necessary; if it is used as basis for collision detection, many fewer interference tests are necessary; and, if it is used as a basis for ray tracing, it enables the performance of rapid ray/plane tests with unique results. Figure 19 shows two examples where ILIEs have been applied to curve plotting and surface enumeration. For a detailed discussion and more examples the reader is referred to the two papers mentioned above.

## 5 Conclusion

In this paper we have discussed the impact of the exact scalar product, common interval arithmetic and its refinements, for example, affine arithmetic and Taylor models. These techniques provide a complete framework for modeling geometric structures with different levels of detail and accuracy. Octrees are adequate data types when only accurate rough bounds for distances are needed. The exact scalar product is crucial to derive tight a priori error bounds for distance computation between convex polyhedra. Certain classic distance algorithms can be enhanced by introducing standard interval arithmetic. Linear Interval Estimations are a natural and intuitive generalization directly connected to the

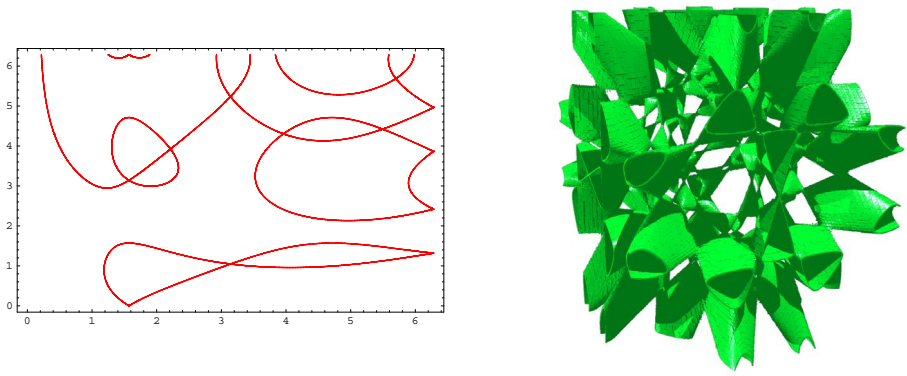



**Fig. 19.** The figure on the left shows the plot of a trigonometric implicit curve created using ILIEs. The figure on the right shows an enclosure of a Barth Decic built of ILIEs. The surface is algebraic degree 10. Notice, that in both examples singularities are properly enclosed by bounding volumes.

representation of the enclosed object. LIEs provide tight and reliable bounds for parametric and implicit objects and are easy to compute and to intersect. The proposed methods allow the development of adaptive algorithms and the reduction of the number of necessary subdivision steps in bounding volumes, intersection and enumeration algorithms in robotics, geometric modeling and computer graphics.

## References

1. Alefeld, G., Herzberger, J.: Introduction to Interval Computations. Academic Press Inc, New York (1983)
2. Barnhill, R. E., Kersey, S.N.: A marching method for parametric surface/surface intersection. CAGD 7(1-4) (1990) 257–280
3. Beaumont, O.: Solving interval linear systems with linear programming techniques. Linear Algebra Appl. 281(1-3) (1998) 293–309
4. van den Bergen, G.: A Fast and Robust GJK Implementation for Collision Detection of Convex Objects. Journal of Graphics Tools, Vol. 4, No. 2, (1999) 7–25
5. Berz, B., Hofstätter, G.: Computation and application of Taylor polynomials with interval remainder bounds. Reliable Computing **4** (1998) 83–97
6. Bühler, K.: Fast and reliable plotting of implicit curves. In Proc. of Workshop on Uncertainty in Geom. Comp., Sheffield University, UK, July 2001. Kluwer (2002)
7. Bühler, K.: Linear interval estimations for parametric objects. (Proc. of Eurographics 2001) Computer Graphics Forum 20(3) (2001)
8. Bühler, K.: Implicit linear interval estimations. In Proc. Spring Conference of Computer Graphics, SCCG, Budmerice (SK). April 2002. ACM Siggraph (2002)
9. Chung, Tat Leung.: An Efficient Collision Detection Algorithm for Polytopes in Virtual Environments. M. Phil. Thesis, University of Hong Kong (1996)
10. Cohen, J., Lin, M. C., Manocha, D., Ponamgi, K.: COLLIDE: An interactive and exact collision detection system for large-scale environments. Proc. Symp. of Interactive 3D Graphics (1995) 189–196

11. Comba, J. L. D., Stolfi, J.: Affine Arithmetic and Its Applications to Computer Graphics. Proceedings of the VI Sibgrapi. Recife, Brazil, October (1993)
12. Dyllong, E., Luther, W., Otten, W.: An Accurate Distance-Computation Algorithm for Convex Polyhedra. Reliable Computing, Vol. 5 (1999) 241–253
13. Dyllong, E., Luther, W.: An accurate computation of the distance between a point and a polyhedron. M. Berveiller, A. K. Louis and C. Fressengeas (eds.), ZAMM Zeitschrift für angewandte Mathematik und Mechanik, Vol. 80 of GAMM 99 Annual Meeting, Metz, France, April 12-16, WILEY-VCH, Berlin (2000) S771–S772
14. Dyllong, E., Luther, W.: Distance calculation between a point and a NURBS surface. Curve and Surface Design: Saint-Malo 1999, P.-J. Laurent, P. Sablonnière, L. L. Schumaker (eds.), Vanderbilt University Press, Nashville, TN (2000) 55–62
15. Dyllong, E., Luther, W.: The GJK Distance Algorithm: An Interval Version for Incremental Motions. Submitted to Scan 2002 Proceedings.
16. Eckardt, U.: Digital Lines and Digital Convexity. Hamburger Beiträge zur Angewandten Mathematik 164 (2001)
17. Ehmann, St. A., Lin, Ming C.: Accurate and Fast Proximity Queries between Polyhedra Using Surface Decomposition. (Proc. of Eurographics 2001) Computer Graphics Forum 20(3) (2001)
18. Fausten, D., Luther, W.: Verified solutions of systems of nonlinear polynomial equations. In Walter Krämer, Jürgen Wolff v. Gudenberg (eds.), Scientific Computing, Validated Numerics, Interval Methods. Kluwer (2001) 141–152
19. Gilbert, E. G., Johnson, D. W., Keerthi, S. S.: A fast procedure for computing the distance between complex objects in three-dimensional space. IEEE Journal of Robotics and Automation, Vol. 4 (1988) 193–203
20. Gilbert E. G., Ong, Chong Jin: Fast Versions of the Gilbert-Johnson-Keethi Distance Algorithm: Additional Results and Comparisons. IEEE Trans. Robotics and Automation, Vol. 17, No. 4 (2001) 531–539
21. Guibas, L. J., Salesin, D., Stolfi, J.: Epsilon Geometry: Building Robust Algorithms from Imprecise Computations. Symposium on Computational Geometry (1989) 208–217
22. van Iwaarden, R., Stolfi, J.: Affine arithmetic software (1997)
23. Knüppel, O.: PROFIL/BIAS – A fast interval library. Computing, Vol. 53 (1994) 277–288
24. Lin, M. C., Canny, J. F.: A fast algorithm for incremental distance calculation. Proc. IEEE Int. Conf. on Robotics and Automation, (1991) 1008–1014
25. Lohner, R.: On the Ubiquity of the Wrapping Effect in the Computation of the Error Bounds, in U. Kulisch and R. Lohner and A. Facius (eds.), Perspectives on Enclosure Methods, Springer Wien New York, (2001) 201–217.
26. Martin, R., Shou, H., Voiculescu, I., Bowyer, A., Wang, G.: Comparison of Interval Methods for Plotting Algebraic Curves. CAGD 19(7) (2002) 553–587
27. Mirtich, B.: V-Clip: Fast and robust polyhedral collision detection. ACM Trans. Graphics, Vol. 17 (1998) 177–208
28. Neumaier, A.: Interval Methods for Systems of Equations, Vol. 37 of Encyclopedia of Mathematics and its Applications. Cambridge University Press (1990)
29. Sato, Y., Hirita, M., Maruyama, T., Arita, Y.: Efficient collision detection for convex and nonconvex objects. Proc. IEEE Int. Conf. Robotics and Automation (Minneapolis, MN) (1996) 771–777
30. Sherbrooke E. C., Patrikalakis, N. M.: Computation of the solution of nonlinear polynomial systems. Computer Aided Geometric Design **10** (1993) 379–405

31. Stolte, N., Kaufman, A.: Parallel spatial enumeration of implicit surfaces using interval arithmetic for octree generation and its direct visualization. In Implicit Surfaces'98 (1998) 81–88
32. Voiculescu, I., Berchtold, J., Bowyer, A, Martin, R. and Zhang, Q.: Interval and affine arithmetic for surface location of power- and Bernstein-Form polynomials. In: Cipolla, R., Martin, R. (eds.): The Mathematics of Surfaces, Vol. IX. Springer-Verlag (2000) 410–423
33. Yap, C. K.: Robust geometric computation. In: Goodman, J. E., O'Rourke, J. (eds.): CRC Handbook in Computational Geometry. CRC Press (1997) 653–668

# On Singular Interval Systems

Götz Alefeld[1] and Günter Mayer[2]

[1] Universität Karlsruhe, 76128 Karlsruhe, Germany
goetz.alefeld@math.uni-karlsruhe.de
[2] Universität Rostock, 18051 Rostock, Germany
guenter.mayer@mathematik.uni-rostock.de

**Abstract.** We consider the interval iteration $[x]^{k+1} = [A][x]^k + [b]$ with $\rho(|[A]|) \leq 1$ where $|[A]|$ denotes the absolute value of the given interval matrix $[A]$. If $|[A]|$ is irreducible we derive a necessary and sufficient criterion for the existence of the limit $[x]^* = [x]^*([x]^0)$ of each sequence $([x]^k)$ of interval iterates. In this way we generalize a well–known theorem of O. Mayer [6] on the above–mentioned iteration, and we are able to enclose solutions of certain singular systems $(I - A)x = b$ with $A \in [A]$ and degenerate interval vectors $[b] \equiv b$. Moreover, we give a connection between the convergence of $([x]^k)$ and the convergence of the powers of $[A]$.

## 1 Introduction

Consider Poisson's equation

$$\frac{\partial^2 u}{\partial s^2} + \frac{\partial^2 u}{\partial t^2} = -f(s,t) \tag{1}$$

on the unit square $Q = [0,1] \times [0,1]$ with a continuous function $f$ defined on $Q$. If one looks for a solution $u(s,t)$ of (1) subject to the periodic boundary conditions

$$\left.\begin{array}{l} u(0,t) = u(1,t),\ 0 \leq t \leq 1 \\ u(s,0) = u(s,1),\ 0 \leq s \leq 1 \end{array}\right\} \tag{2}$$

and if one discretizes (1) using an equidistant grid of mesh size $h = \frac{1}{n},\ n \in \mathbb{N}\backslash\{1,2\}$, a row–wise ordering and the well–known five point central difference approximation one ends up with a system

$$Cx = b \tag{3}$$

of linear equations in which $C \in \mathbb{R}^{n^2 \times n^2}$ is defined by

$$C = \frac{1}{4}\begin{pmatrix} D & -I & O & \dots & O & -I \\ -I & D & -I & O & \dots & O \\ O & -I & D & -I & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & O \\ O & \dots & O & -I & D & -I \\ -I & O & \dots & O & -I & D \end{pmatrix},\ D = \begin{pmatrix} 4 & -1 & 0 & \dots & 0 & -1 \\ -1 & 4 & -1 & 0 & \dots & 0 \\ 0 & -1 & 4 & -1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & -1 & 4 & -1 \\ -1 & 0 & \dots & 0 & -1 & 4 \end{pmatrix},$$

R. Alt et al. (Eds.): Num. Software with Result Verification, LNCS 2991, pp. 191–197, 2004.

$D \in \mathbb{R}^{n\times n}$, $I$ = identity matrix. The components $b_i$ of $b \in \mathbb{R}^{n^2}$ are given by

$$b_i = \frac{h^2}{4} f(s_l, t_m), \quad i = 1, \dots, n^2,$$

with $s_l = t_l = lh$, $i = (m-1)\cdot n + l$, $l, m = 1, \dots, n$. When discretizing one assumes a periodic continuation of $u$ across the boundary of $Q$. The unknowns $x_i$ refer to the inner grid points and to the grid points of the right and upper boundary of $Q$. It is known (cf. [2], p. 196 ff) that $C$ is a singular matrix of rank $n^2 - 1$. This follows from the fact that it is a singular irreducible $M$ matrix with property c (cf. [2], Definition 6.4.10, Theorem 6.4.16 and p. 201). Richardson splitting applied to $C$ yields to the iterative process

$$x^{k+1} = Ax^k + b, \quad k = 0, 1, \dots , \tag{4}$$

where $A = I - C$. Since every diagonal element of $C$ is 1 the iteration (4) coincides here with the Jacobi method for (3). If $n$ is odd the matrix $A$ has the spectral radius $\rho(A) = 1$, and all eigenvalues $\lambda$ of $A$ with $|\lambda| = 1$ are one and have only linear elementary divisors, i.e., the corresponding Jordan blocks are $1 \times 1$. Such matrices – together with those of spectral radius less than one – are called semi–convergent ([2], p. 152). They represent just the matrices for which the limit $A^\infty = \lim_{k\to\infty} A^k$ exists.

We remark that the matrix $A$ arising from the discretization of (1), (2) is symmetric. Therefore, all eigenvalues of $A$ have only linear elementary divisors. In addition, $A$ is non–negative and irreducible. Hence the Theorem of Perron and Frobenius guarantees that the eigenvalue $\lambda = 1$ is even algebraically simple which is not required in the definition of semi–convergence and which is not necessary for our subsequent considerations. Moreover, $A$ is primitive if $n$ is odd, and cyclic of index 2 if $n$ is even. This can be seen by inspecting the lengths of the circuits in the directed graph associated with $A$ ([2], § 2.2). Therefore, the theory of Perron and Frobenius on non–negative irreducible matrices shows that $\lambda = 1$ is the only eigenvalue of $A$ with $|\lambda| = \rho(A) = 1$ in the case of $n$ being odd while $\lambda = -1$ is another eigenvalue with this property in the case of even $n$.

In this short note we will consider the case where $A$ is allowed to vary within a given interval matrix $[A]$ such that the absolute value $|[A]|$ of $[A]$ is irreducible and semi–convergent. We present – in a condensed form – results on the corresponding interval iteration

$$[x]^{k+1} = [A][x]^k + [b], \quad k = 0, 1, \dots \tag{5}$$

generalizing in this way a well–known theorem of O. Mayer [6]; cf. also [1], pp. 143 ff. By lack of space we must omit the very lengthy and by no means straightforward proofs. They will be published elsewhere.

We finally remark that singular linear systems also occur in other situations – cf. [2], § 7.6, in this respect.

## 2 Results

In order to recall some results for the iterative process (4) with (general) semi–convergent matrices $A$ we first define the Drazin inverse $A^D$ of an arbitrary $n \times n$ matrix $A = S \begin{pmatrix} \hat{J}_0 & O \\ O & \hat{J}_r \end{pmatrix} S^{-1}$ by $A^D = S \begin{pmatrix} O & O \\ O & (\hat{J}_r)^{-1} \end{pmatrix} S^{-1}$. Here, $J = \begin{pmatrix} \hat{J}_0 & O \\ O & \hat{J}_r \end{pmatrix}$ is the Jordan canonical form of $A$ with square blocks $\hat{J}_0,\ \hat{J}_r$, whose diagonal blocks are just the singular Jordan blocks of $J$, and the non–singular ones, respectively; cf. for instance [2], § 5.4.

The following theorem which is contained in Lemma 7.6.13 in [2] answers completely the question on the convergence of (4).

**Theorem 1.** *Let (3) (with a matrix $C$ not necessary equal to the one obtained by discretizing (1) and (2)) be solvable. Then each sequence $(x^k)$ of iterates defined by (4) is convergent if and only if $A$ is semi-convergent. The limit is independent of $x^0$ if and only if $\rho(A) < 1$. In any case this limit $x^*$ is a solution of (3) and a fixed point of (4). By means of Drazin inverses it can be expressed as*

$$x^* = (I - A)^D b + \{I - (I - A)(I - A)^D\} x^0.$$

If $\rho(A) < 1$ then $(I-A)^{-1}$ exists. Hence (3) is uniquely solvable and by virtue of $(I-A)^{-1} = (I-A)^D$ Theorem 1 reduces to a basic result of numerical analysis in this case. Therefore, it is essentially the case $\rho(A) = 1$ which is of interest in our paper.

For the interval iteration (5) we will replace the assumption of solvability in Theorem 1 by the existence of a fixed point of (5). For interval matrices $[A]$ with $\rho(|[A]|) < 1$ the above–mentioned theorem of O. Mayer [6] guarantees that (5) has a unique fixed point. If $|[A]|$ is irreducible and satisfies $\rho(|[A]|) = 1$ we could prove in [5] an exhaustive result on the existence and the shape of such fixed points. In order to formulate our main result we need the following definition.

**Definition 1.** *([3], [4]) Let $[A]$ be an $n \times n$ interval matrix. Let*

$$[A]^0 = I, \qquad [A]^{k+1} = [A]^k \cdot [A], \quad k = 0, 1, \dots \ .$$

*If $[A]^\infty = \lim_{k\to\infty} [A]^k$ exists then we call $[A]$ semi–convergent.*

**Theorem 2.** *Let $[A]$ be a non–degenerate $n \times n$ interval matrix with irreducible absolute value $|[A]|$. Let the iteration (5) have a fixed point $[z]^*$ (which implies $[b] \equiv b \in \mathbb{R}^n$ in the case $\rho(|[A]|) = 1$ according to Theorem 8 in [5]). Then the following three statements are equivalent.*

*a) Each sequence $([x]^k)$ of (5) is convergent.*
*b) The interval matrix $[A]$ is semi–convergent.*
*c) The absolute value $|[A]|$ is semi-convergent. Moreover, if $\rho(|[A]|) = 1$ and if $[A]$ contains only one matrix $\dot{A}$ with $|\dot{A}| = |[A]|$ then $\dot{A} \neq -D|[A]|D$ for all matrices $D$ with $|D| = I$.*

*In case of convergence of (5) the limit* $[x]^* = [x]^*([x]^0)$ *of* $([x]^k)$ *is a fixed point of the iteration (5). It contains the set* $S([x]^0)$ *of all solutions of (3) which one obtains as limits of sequences* $(x^k)$ *of iterates defined by (4) with* $x^0 \in [x]^0$*, i.e.,*

$$S([x]^0) = \Big\{ x^* \mid x^* = (I-A)^D b + \{I - (I-A)(I-A)^D\}x^0, \\ A \in [A],\ x^0 \in [x]^0,\ b \in [b] \Big\} \subseteq [x]^*([x]^0) .$$

*In case of convergence of (5) the limit* $[x]^*$ *of* $([x]^k)$ *does not depend on the starting vector* $[x]^0$ *if and only if one of the following equivalent properties holds:*

*(i)* $\rho(|[A]|) < 1$.
*(ii)* $\lim_{k\to\infty} |[A]|^k = O$.
*(iii)* $\lim_{k\to\infty} [A]^k = O$.

Note that the equivalence 'a) $\Leftrightarrow$ c)' remains true even if $[A]$ is degenerate (and $|[A]|$ is irreducible) while 'b) $\Rightarrow$ c)' becomes false as the example

$$[A] \equiv A = \begin{pmatrix} 2/3 & 2/3 \\ 2/3 & -2/3 \end{pmatrix} \tag{6}$$

shows. (Cf. [3], [4] for further details.) Since the statements b), c) do not depend on $[b]$ and since (5) has always the fixed point $[z]^* \equiv 0$ for $[b] \equiv 0$ the existence of $[z]^*$ does not need to be assumed in Theorem 2 for the equivalence of b) and c). If $|[A]|$ is reducible the equivalence of (ii) and (iii) becomes false as can be seen, e.g., by the $2 \times 2$ block diagonal matrix $[A] = \mathrm{diag}([0, 1/2], B)$ where $B$ is the matrix denoted by $A$ in (6). We refer to [3] or [7] in this case.

We conclude our contribution with a numerical example which illustrates the theory.

*Example 1.* Define the $n \times n$ interval matrix $[D]$ by

$$[D] = [D]_{[\alpha],[\beta]} = \begin{pmatrix} [\alpha] & [\beta] & 0 & \dots & 0 & [\beta] \\ [\beta] & [\alpha] & [\beta] & 0 & \dots & 0 \\ 0 & [\beta] & [\alpha] & [\beta] & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & [\beta] & [\alpha] & [\beta] \\ [\beta] & 0 & \dots & 0 & [\beta] & [\alpha] \end{pmatrix},$$

and the $n^2 \times n^2$ interval matrix $[A] = [\underline{A}, \overline{A}]$ in block form by

$$[A] = \begin{pmatrix} [D] & [\gamma]I & O & \dots & O & [\gamma]I \\ [\gamma]I & [D] & [\gamma]I & O & \dots & O \\ O & [\gamma]I & [D] & [\gamma]I & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & O \\ O & \dots & O & [\gamma]I & [D] & [\gamma]I \\ [\gamma]I & O & \dots & O & [\gamma]I & [D] \end{pmatrix},$$

where $[\alpha]$, $[\beta]$, $[\gamma]$ are intervals which are still to be chosen. By means of the Kronecker product $\otimes$ the matrix $[A]$ can be written as

$$[A] = I \otimes [D]_{[\alpha],[\beta]} + [D]_{0,[\gamma]} \otimes I, \qquad I \in \mathbb{R}^{n\times n}.$$

In this way it can easily be constructed in software packages like INTLAB [8] whose version 4.1.1 we used for our interval computations. We choose $[\alpha]$, $[\beta] \neq 0$, $[\gamma] \neq 0$ such that $|[\alpha]| + 2|[\beta]| + 2|[\gamma]| = 1$ holds. Then $|[A]|$ is irreducible and $\rho(|[A]|) = 1$ is guaranteed. Moreover, $[b] \equiv b = (b_i) \in \mathbb{R}^{n^2}$ is necessary for the existence of a fixed point of (5) which is required as assumption in Theorem 2.

We first use $n = 5$, $[\alpha] = 0$, $[\beta] = [\gamma] = 1/4$. This leads to the particular situation of Section 1 in which we showed that $[A] \equiv A = |[A]| \in \mathbb{R}^{n^2\times n^2}$ is semi–convergent with $\rho(|[A]|) = 1$. If $b = (I - A)\check{z}$ for some $\check{z} = (\check{z}_i) \in \mathbb{R}^{n^2}$ then Theorem 8 in [5] guarantees that (5) has the fixed points

$$[z]^* = \check{z} + se + t[-1, 1]e, \tag{7}$$

where $s, t$ are any real numbers with $t \geq 0$ and where $e = (1, 1, \dots, 1)^T \in \mathbb{R}^{n^2}$ is an eigenvector of $A \geq O$ associated with the eigenvalue $\lambda = \rho(A) = 1$. Therefore, by virtue of Theorem 2 a), c), extended by the first remark following this theorem, the limits $[x]^* = [x]^*([x]^0)$ exist for any starting vectors $[x]^0$ and are precisely the vectors $[z]^*$ in (7). We choose $b_i = b_{n^2+1-i} = 0.5$ for $i \in \{1, 3, 4, 7\}$, $b_2 = b_{n^2-1} = -2$, $b_i = 0$ otherwise. Then $\check{z} = (1, -1, 1, 1, \dots, 1, 1, -1, 1)^T \in \mathbb{R}^{n^2}$ satisfies $b = (I - A)\check{z}$ as required above. We iterated according to (5) with different starting vectors $[x]^0$. We stopped the iteration either when the criterion

$$[\tilde{x}]^k = [\tilde{x}]^{k-1} \tag{8}$$

was fulfilled for some $k = k_0$ or when $k$ reached a given upper bound $k_{\max}$, where here and in the sequel the tilde denotes computed, i.e., rounded quantities. By the outward rounding of the machine interval arithmetic (cf., e.g., [1]) we always have $[x]^k \subseteq [\tilde{x}]^k = ([\tilde{x}]_i^k)$, $k = 0, 1, \dots$ . Moreover, in the case (8) we can guarantee $[x]^k \subseteq [\tilde{x}]^{k_0}$, $k = k_0, k_0 + 1, \dots$ , whence $[x]^* \subseteq [\tilde{x}]^{k_0}$.

If (8) cannot be obtained, i.e., in the case where $k$ reaches $k_{\max}$ one can compute the midpoints $\tilde{m}_i = \operatorname{mid}\left([\tilde{x}]_i^{k_{\max}} - \check{z}_i\right)$, $i = 1, \dots, n^2$, and the radii $\tilde{r}_i = \operatorname{rad}\left([\tilde{x}]_i^{k_{\max}} - \check{z}_i\right)$, $i = 1, \dots, n^2$. Here, we assume that the *computed* values $\tilde{m}_i, \tilde{r}_i$ satisfy $[\tilde{x}]_i^{k_{\max}} - \check{z}_i \subseteq \tilde{m}_i + \tilde{r}_i[-1, 1]$. Define

$$\tilde{s} = \left(\max_i \tilde{m}_i + \min_i \tilde{m}_i\right)/2 \in \mathbb{R} \tag{9}$$

and

$$\tilde{t} = \max_i \left(\tilde{r}_i + |\tilde{s} - \tilde{m}_i|\right) \in \mathbb{R} \tag{10}$$

using upward rounding in the latter case. According to (7) the vector

$$[\hat{z}]^* = \check{z} + \tilde{s}e + \tilde{t}[-1, 1]e$$

(not to be confused with $[z]^*$ in Theorem 2) is a fixed point of (5) provided that $[\hat{z}]^*$ is computed with *exact* arithmetic. By construction, $[\hat{z}]^*$ contains $[\tilde{x}]^{k_{\max}}$. From $[x]^{k_{\max}} \subseteq [\tilde{x}]^{k_{\max}} \subseteq [\hat{z}]^*$ we get

$$[x]^k \subseteq [\hat{z}]^*, \quad k = k_{\max}, k_{\max}+1, \dots, \quad \text{whence } [x]^* \subseteq [\hat{z}]^*.$$

This holds also if $k_{\max}$ is replaced by $k_0$ in the case (8). In our tables we list $[\hat{z}]^*$ in both cases.

**Table 1.** Starting vector vs. enclosure $[\hat{z}]^* = \check{z} + \tilde{s}e + \tilde{t}[-1,1]e$

| $[x]^0$ | $\tilde{s}$ | $\tilde{t}$ | $k_0$ | $k_{\max}$ |
|---|---|---|---|---|
| $0$ | $-0.84$ | $1.021405182655144 \cdot 10^{-14}$ | 192 | |
| $e$ | $0.16 + 10^{-14}$ | $3.996802888650564 \cdot 10^{-14}$ | – | 200 |
| $[-1,1]e$ | $-0.84 + 2 \cdot 10^{-14}$ | $1.00000000000006$ | – | 200 |
| $[-2,1]e$ | $-1.34 + 10^{-14}$ | $1.50000000000006$ | 172 | |
| $\Big( (-1)^i[-1,2] \Big)_{i=1}^{n^2}$ | $-0.86 + 10^{-14}$ | $1.50000000000006$ | – | 200 |

Without further knowledge on a relation between $[x]^*$ and $[x]^0$ we cannot, of course, assess the quality which the enclosure $[\tilde{x}]^{k_0}$ or $[\hat{z}]^*$ of the true limit $[x]^*$ has with respect to $[x]^*$. For degenerate starting vectors $[x]^0 \equiv x^0$, however, the radius of $[\tilde{x}]^{k_0}$, and $[\hat{z}]^*$, respectively, may indicate this quality. In theory this radius is zero for such starting vectors, in practice it is not by virtue of rounding errors during the iteration. Table 1 contains the parameters $\tilde{s}$, $\tilde{t}$ from (9), (10) for different starting vectors.

**Table 2.** Starting vector vs. enclosure $[\hat{z}]^* = \check{z} + \tilde{s}e + \tilde{t}[-1,1]e$

| $[x]^0$ | $\tilde{s}$ | $\tilde{t}$ | $k_0$ | $k_{\max}$ |
|---|---|---|---|---|
| $0$ | $0$ | $1$ | 814 | |
| $e$ | $0$ | $1$ | 796 | |
| $[-1,1]e$ | $-0.42 - 10^{-14}$ | $1.42 + 10^{-14}$ | 796 | |
| $[-2,1]e$ | $-0.92 - 2 \cdot 10^{-14}$ | $1.92 + 2 \cdot 10^{-14}$ | 796 | |
| $\Big( (-1)^i[-1,2] \Big)_{i=1}^{n^2}$ | $-0.68 - 2 \cdot 10^{-14}$ | $1.68 + 2 \cdot 10^{-14}$ | 777 | |

We next choose $n = 5$, $[\alpha] = [0, 1/4]$, $[\beta] = [0, 1/8]$, $[\gamma] = [1/8, 1/4]$ and $b = (I - \overline{A})\check{z}$ with $\check{z}$ as above. Then nearly all earlier remarks hold analogously, and we obtain the results of Table 2. By virtue of Theorem 8 in [5] we get the restriction $t \geq |\pm 1 + s| = 1 + |s|$ for $s, t$ from (7). A short glance at Table 2 reveals that this inequality also holds for our computed values $\tilde{s}, \tilde{t}$ instead of $s, t$.

**Acknowledgement.** The authors are grateful to two anonymous referees for a series of comments and remarks which improved the paper.

## References

1. Alefeld, G., Herzberger, J.: Introduction to Interval Computations. Academic Press, New York, 1983
2. Berman, A., Plemmons, R.J.: Nonnegative Matrices in the Mathematical Sciences. Academic Press, New York, 1979
3. Mayer, G.: On the convergence of powers of interval matrices. Linear Algebra Appl. **58** (1984) 201 – 216
4. Mayer, G.: On the convergence of powers of interval matrices (2). Numer. Math. **46** (1985) 69 – 83
5. Mayer, G., Warnke, I.: On the fixed points of the interval function $[f]([x]) = [A][x] + [b]$. Linear Algebra Appl. **363** (2003) 201 – 216
6. Mayer, O.: Über die in der Intervallrechnung auftretenden Räume und einige Anwendungen, Ph.D. Thesis, Universität Karlsruhe, Karlsruhe, 1968
7. Pang, C.-T., Lur, Y.-Y., Guu, S.-M.: A new proof of Mayer's theorem, Linear Algebra Appl. **350** (2002) 273 – 278
8. Rump, S.M.: INTLAB – INTerval LABoratory. In: Csendes T. (ed.), Developements in Reliable Computing, Kluwer, Dordrecht, 1999, 77 – 104

# Result-Verifying Solution of Nonlinear Systems in the Analysis of Chemical Processes*

Thomas Beelitz[1], Christian Bischof[2], Bruno Lang[1], and Klaus Schulte Althoff[2]

[1] Universität Wuppertal
Fachbereich Mathematik und Naturwissenschaften
42097 Wuppertal, Germany
[2] RWTH Aachen
Institut für Wissenschaftliches Rechnen
52064 Aachen, Germany

**Abstract.** A framework for the verified solution of nonlinear systems arising in the analysis and design of chemical processes is described. The framework combines a symbolic preprocessing step with an interval–based branch–and–bound solver whose efficiency is increased with several acceleration techniques. One of these methods is based on order–2 Taylor expansion; it is also discussed in this note.

**Keywords:** chemical process analysis and design, singularities, verified solution of nonlinear systems, order–2 Taylor expansion

## 1 Introduction

Nonlinear systems arise in a variety of applications, one example being the search for singularities in a chemical process [1,2]. Let $\mathbf{p} \in \mathbb{R}^k$ denote the adjustable *parameters* controlling the process (such as heating, inflow concentrations, etc.), and let $\mathbf{x} \in \mathbb{R}^\ell$ describe its internal *state* (current temperature, reaction rates, etc.). Then the steady states of the process can be described by a set of equations, $\mathbf{f}(\mathbf{p}, \mathbf{x}) = \mathbf{0}$, and singularities mark those parameter values $\mathbf{p}^*$ where transitions from unique steady states $\mathbf{x} = \mathbf{x}(\mathbf{p})$ to multiple steady states occur. As multiple steady states can lead to fluctuations in the quality of the resulting product or can even cause severe damage to the facility, being able to *guarantee* the absence of singularities in the parameter range $[\mathbf{p}] = [\underline{p_1}, \overline{p_1}] \times \cdots \times [\underline{p_k}, \overline{p_k}]$ intended for operating the process is an important goal during process analysis and design.

One approach to achieve this goal first augments the system $\mathbf{f}(\mathbf{p}, \mathbf{x}) = \mathbf{0}$ with equations characterizing a specific type of singularity [3], and then applies a result-verifying nonlinear solver to the augmented system, $\mathbf{F}(\mathbf{z}) = \mathbf{0}$. Here, $\mathbf{z} \in \mathbb{R}^n$ comprises the variables $\mathbf{p}$ and $\mathbf{x}$, as well as auxiliary variables introduced during the augmentation, and $\mathbf{F}$ consists of the functions $\mathbf{f}$ and additional functions involving derivatives such as $\partial\mathbf{f}/\partial\mathbf{x}$; see [2] for more details.

* This work was supported by the VolkswagenStiftung within the project "Konstruktive Methoden der Nichtlinearen Dynamik zum Entwurf verfahrenstechnischer Prozesse", Geschäftszeichen I/79 288.

R. Alt et al. (Eds.): Num. Software with Result Verification, LNCS 2991, pp. 198–205, 2004.

In this note we describe our framework for the verified solution of such problems. Section 2 gives an overview of the overall structure and the interplay of its three main modules, while Section 3 explains one specific component to some detail. In Section 4 we report on numerical experiments demonstrating the effectiveness of our approach.

## 2 The Structure of the Solver

As shown in Figure 1, our framework consists of three modules. The central **solver** module implements an interval–based branch–and–bound nonlinear solver with several acceleration tools [2,4,5,6]. The system to be solved is set up in a symbolic **preprocessing** step, and the function and derivative values needed in the **solver** are provided in a third **evaluation** module.

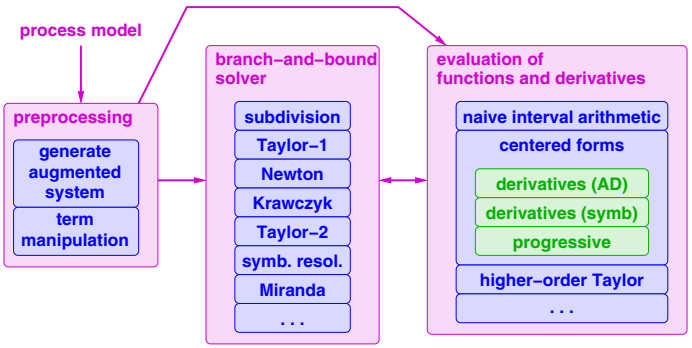



**Fig. 1.** The structure of the framework.

### 2.1 Symbolic Preprocessing

The **preprocessing** module prepares the system that is subsequently passed to the branch–and–bound solver. The preprocessing involves

- reading the problem specification (names of the parameter and state variables, ranges for these variables, (in)equalities describing the chemical process, type of the singularities under consideration, etc.),
- generating terms for the derivatives needed in the augmentation step and setting up the augmented system,
- generating terms for the augmented system's derivatives or slopes,
- "optimizing" the terms such that *narrow* enclosures for the ranges of the corresponding functions can be computed at *low complexity*, and
- generating an adequate representation that allows the **evaluation** module to efficiently evaluate the functions and derivatives. Currently, the function terms can be emitted in a tree representation or as C code.

With the "C code" option, the resulting file must be compiled and linked to the framework before the solver is started.

The term optimizer is a rule–based term rewriting system, complemented with algorithmic simplification. The term rewriting system [7] applies general rules to matching subterms. Taking the subdistributive law as an example, we know that replacing $t_1t_2+t_1t_3$ with $t_1(t_2+t_3)$ always reduces the evaluation time and never increases the overestimation. Here, the $t_i$ denote arbitrary subterms in the representation of a function. In the algorithmic simplification, optimizations are performed that are difficult to handle with a rule system, e.g., replacing constant subterms with their numerical value: $2 + [-1, 1] \longrightarrow [1, 3]$.

### 2.2 Enclosing the Ranges of Functions and Derivatives

The **evaluation** module provides several methods to compute enclosures for the ranges of the functions $\mathbf{F}$ and its derivatives:

- plain interval arithmetic, leading to an order–1 approximation of the respective range, or
- slope–based centered forms yielding order–2 approximations, where the slopes can be computed
  - "on the fly" with a slope arithmetic [8] implemented with operator overloading, similarly to Automatic Differentiation [9],
  - using explicit expressions for the slopes, or
  - relying on expressions for the partial derivatives.

  In the latter two cases, the expressions for the slopes and derivatives have been generated during the preprocessing phase.

If the slopes are based on expressions for the derivatives then in most cases they are evaluated "progressively" [10]: Let $f$ denote a scalar function whose range over some box $[\mathbf{z}]$ is sought, let $\mathbf{c} \in [\mathbf{z}]$ be a fixed center, and $\tilde{\mathbf{z}} \in [\mathbf{z}]$ an arbitrary point. Then repeated application of the mean value theorem yields

$$\begin{aligned} f(\tilde{\mathbf{z}}) &= f(\mathbf{c}) + \frac{\partial f}{\partial z_1}(\zeta_1, c_2, \ldots, c_n) + \frac{\partial f}{\partial z_2}(\tilde{z}_1, \zeta_2, c_3, \ldots, c_n) \\ &\quad + \ldots + \frac{\partial f}{\partial z_n}(\tilde{z}_1, \ldots, \tilde{z}_{n-1}, \zeta_n) \\ &\in f(\mathbf{c}) + \frac{\partial f}{\partial z_1}([z_1], c_2, \ldots, c_n) + \frac{\partial f}{\partial z_2}([z_1], [z_2], c_3, \ldots, c_n) \\ &\quad + \ldots + \frac{\partial f}{\partial z_n}([z_1], \ldots, [z_{n-1}], [z_n]) \ . \end{aligned}$$

This means that a slope vector $[\mathbf{s}]$ can be obtained by evaluating expressions for the partial derivatives in a particular way: $\partial f/\partial z_j$ is evaluated on the subbox $[\mathbf{z}^{(\mathbf{j})}] = ([z_1], \ldots, [z_j], c_{j+1}, \ldots, c_n)$ of $[\mathbf{z}]$ containing nondegenerate intervals only in its first $j$ places. In our experience [11] the enclosures determined this way are comparable to the ones obtained with slope arithmetic and take less time to compute.

Other methods aimed at obtaining sharper enclosures have also been investigated, e.g., the Taylor models available through the COSY INFINITY software [12,13]. Taylor models are currently not supported in our environment due to their significantly higher cost [14].

### 2.3 The Branch-and-Bound Nonlinear Solver

The nonlinear solver implements a branch–and–bound strategy for locating the solutions of the system $\mathbf{F}(\mathbf{z}) = \mathbf{0}$: If (an enclosure of) the range of some component $F_i$ over the box $[\mathbf{z}]$ does not contain zero, then the system cannot have a solution in $[\mathbf{z}]$, and $[\mathbf{z}]$ is discarded. Otherwise, $[\mathbf{z}]$ is split into two or more subboxes, and the above criterion is applied recursively to the subboxes until they are smaller than some prescribed threshold.

In order to make this simple scheme work in practice, acceleration techniques must be used that allow to restrict the search for zeros to smaller boxes and thus reduce the number of subdivisions. Several accelerators are currently supported in our framework:

- Tightening operators, such as Newton–Gauß–Seidel and Krawczyk [4,5]. Besides allowing to replace the current box $[\mathbf{z}]$ with a smaller box $[\mathbf{z}'] \subseteq [\mathbf{z}]$ that contains all zeros of $\mathbf{F}$ in $[\mathbf{z}]$, these operators can often also be used to *prove* the existence of a zero in the current box.
- Numerical resolution of equations for selected variables based on Taylor expansion up to order 1 (cf. [2]) and 2. The order–2 Taylor refinement technique is discussed in Sect. 3.

In addition, several schemes are available for choosing the directions for box subdivisions. The well-known "MaxDiam" subdivision scheme bisects the box along its longest edge. The "MaxSumMagnitude" scheme bisects the current box $[\mathbf{z}]$ along the axis $j$ that maximizes the quantity

$$\sum_{i=1}^{m} |[J_{ij}]| \cdot \mathrm{diam}([z_j])^{\alpha} \ ,$$

where $[J] = ([J_{ij}])$ denotes an interval enclosure of the Jacobian (or an appropriate slope matrix) of $\mathbf{F}$ over $[\mathbf{z}]$, $|[a]| = \max\{|\underline{a}|, |\overline{a}|\}$ denotes the magnitude (maximum absolute value of an entry) of an interval $[a]$, and $\alpha > 0$ is some user-defined exponent (default: $\alpha = 1$). Thus, "MaxSumMagnitude" tries to select the direction that may lead to the largest overall changes in the function values. In the "MaxSumMagnitude" strategy, the magnitude $|[J_{ij}]|$ is replaced with the mignitude, $\langle [J_{ij}] \rangle$, which is the minimum absolute value of an element of $[J_{ij}]$.

To those boxes that are small enough to meet the size threshold, but could not yet be proven to contain a zero, further verification tests of varying complexity can be applied.

### 2.4 Design and Implementation Issues

Our framework is implemented in C++. Instead of relying on a specific implementation of interval arithmetic, we use generic macros for accessing and manipulating interval quantities. The C preprocessor is then used to map these macros to the constructs available in C-XSC [15] and Sun ONE Studio 8 [16]. In this way we can combine the performance of a compiler-integrated interval implementation, as available on Sun machines, with the wide range of platforms supported by C-XSC.

Modularity has been an important goal in the design of the framework. Almost all features can be switched on and off independently from each other via a control file that is read at the beginning. The modular design also facilitates including new methods for evaluating the functions and slopes, accelerating the branch–and–bound solver, and verifying the existence of solutions.

The solver also has a breakpoint feature, that is, it may be interrupted at specified points and restarted, possibly with changed settings for the switches.

## 3 Second-Order Taylor Refinement

In this section we discuss one of the techniques for "refining" a box $[\mathbf{z}]$, i.e., replacing it with a smaller box $[\mathbf{z}']$ such that no zero $\mathbf{z}^* \in [\mathbf{z}]$ of $\mathbf{F}$ is lost.

Assume that $\mathbf{F}$ is twice continuously differentiable on $[\mathbf{z}]$. Then, by Taylor's theorem, for a fixed center $\mathbf{c} \in [\mathbf{z}]$ and any solution $\mathbf{z}^* \in [\mathbf{z}]$ we have

$$0 = F_i(\mathbf{z}^*) = F_i(\mathbf{c}) + \nabla F_i(\mathbf{c}) \cdot (\mathbf{z}^* - \mathbf{c}) + \frac{1}{2} \cdot (\mathbf{z}^* - \mathbf{c})^T \cdot F_i''(\zeta) \cdot (\mathbf{z}^* - \mathbf{c})$$

for some $\zeta$ between $\mathbf{c}$ and $\mathbf{z}^*$. Letting

$$u_k = \frac{\partial F_i}{\partial z_k}(\mathbf{c}) \ , \quad H_{k\ell} = \frac{\partial^2 F_i}{\partial z_k \partial z_\ell}(\zeta) \ ,$$

sorting by powers of a specific $z_j^* - c_j$, and making use of the symmetry of the Hesse matrix $H = (H_{k\ell})$ yields

$$\begin{aligned} 0 = \frac{1}{2} H_{jj} \cdot (z_j^* - c_j)^2 + \left( u_j + \sum_{k \neq j} H_{jk} \cdot (z_k^* - c_k) \right) \cdot (z_j^* - c_j) \\ + F_i(\mathbf{c}) + \sum_{k \neq j} u_k \cdot (z_k^* - c_k) + \frac{1}{2} \cdot \sum_{k,\ell \neq j} H_{k\ell} \cdot (z_k^* - c_k) \cdot (z_\ell^* - c_\ell) \ . \end{aligned}$$

Substituting $y = z_j^* - c_j$, this implies

$$0 \in [a] \cdot y^2 + [b] \cdot y + [c]$$

with suitable intervals $[a]$, $[b]$, $[c]$ *not depending on* $y$. Analogously to the well-known formulae for the point case, the solutions of this *quadratic interval equation* are given by

$$y_1 \in [y_1^{(a)}] := \frac{-[b] + \sqrt{[D]}}{2[a]} \ , \quad y_2 \in [y_2^{(a)}] := \frac{-[b] - \sqrt{[D]}}{2[a]} \ ,$$

where $[D] = [b]^2 - 4[a][c]$. In the point case, these formulae are known to be highly susceptible to rounding errors if $b \approx \pm\sqrt{D}$; see [17] for a thorough discussion. For the interval equation this means that one of the intervals $[y_1^{(a)}]$ and $[y_2^{(a)}]$ will significantly overestimate the true solution set. Then the alternative formulas

$$y_1 \in [y_1^{(b)}] := \frac{2[c]}{-[b] - \sqrt{[D]}} \quad \text{resp.} \quad y_2 \in [y_2^{(b)}] := \frac{2[c]}{-[b] + \sqrt{[D]}}$$

yield sharper bounds. As a consequence, we compute the solutions of the interval quadratic equation as

$$y_1 \in [y_1^{(a)}] \cap [y_1^{(b)}] \cap ([z_j] - c_j) \ , \ \ y_2 \in [y_2^{(a)}] \cap [y_2^{(b)}] \cap ([z_j] - c_j) \ .$$

Since only real roots are sought, we may also replace $[D]$ with $[D] \cap [0, \infty)$.

Note that solving the quadratic equation may split $[z_j]$ into *up to four subintervals* $[z_j'] = [y] + c_j$: Each of the two branches of the formula again results in one or two intervals, the latter being the case if a zero in the denominator enforces the use of extended interval arithmetic.

The following algorithm summarizes the order–2 Taylor refinement for the case when *no splitting* is necessary. The general case, together with additional implementation issues, is discussed in [6]. In the algorithm, $m$ and $n$ denote the number of equations and variables, respectively.

**repeat**
  **for** $i = 1 : m$
    compute $[u_k] = (\partial F_i / \partial z_k)([\mathbf{z}])$ **for** $k = 1 : n$
    compute $[H_{k\ell}] = (\partial^2 F_i / \partial z_k \partial z_\ell)([\mathbf{z}])$ **for** $k, \ell = 1 : n$
    **for** $j = 1 : n$ { solve $i$th equation for $z_j$ }
      compute $[z_j']$ and replace $[z_j]$ with $[z_j']$
**until** this did not reduce $[\mathbf{z}]$ sufficiently

Similarly to the order–1 Taylor refinement described in [2], the order–2 Taylor refinement can be applied in addition to the Newton–Gauß–Seidel and comparable tightening operators in order to enhance their effectiveness; cf. Section 4 for performance data supporting this approach. Both techniques also are applicable in situations where the tightening operators do not work, e.g., when some of the functions or derivatives cannot be evaluated over the box $[\mathbf{z}]$, or when $m \neq n$.

A similar technique can be used to refine the box $[\mathbf{z}]$ through inequalities. In this case, the solution sets of quadratic interval inequalities must be determined.

## 4 Numerical Experiments

Table 1 summarizes the performance indicators of our solver for two nonlinear systems. The "Reactor" system consists of 29 equations in 29 unknowns resulting from augmenting a process model with 14 equations in 15 unknowns by additional constraints for saddle–node singularities [2]. This problem was solved on

**Table 1.** Number of boxes considered and time (hours:minutes:seconds) for the solution of two nonlinear systems; "—" indicates that the solver did not complete within 4 hours.

| | Robotics | | Reactor | |
|---|---|---|---|---|
| Settings | Boxes | Time | Boxes | Time |
| Standard | 215 | 0.44 | 81 473 | 10:44 |
| Non-progressive derivative evaluation | 275 | 0.44 | 239 315 | 28:54 |
| No term simplification | — | > 4:00:00 | — | > 4:00:00 |
| No order–1 Taylor refinement | 541 | 0.45 | — | > 4:00:00 |
| No Newton operator | 1 025 | 1.82 | 274 663 | 15:17 |
| Krawczyk operator instead | 297 | 0.56 | 190 277 | 26:11 |
| Order–2 Taylor refinement enabled | 111 | 0.34 | 73 319 | 12:51 |
| Bisection strategy "MaxDiam" | 219 | 0.41 | — | > 4:00:00 |
| Bisection strategy "MaxSumMignitude" | 257 | 0.44 | — | > 4:00:00 |

a 900 MHz UltraSparc III using the interval support of the Sun ONE Studio 8 C++ compiler. The smaller "Robotics" system [18] arises in determining the angles that the joints of a robot's arm must take in order to place the grip at a specified position (*inverse kinematic problem*). It consists of 8 equations in 8 unknowns and is solved in "PlainSystem" mode, i.e., the system is solved as such without prior augmentation. This problem was solved on a 1.8 GHz Pentium 4 with the C-XSC interval arithmetic. As a rule, the Sun version achieved a speedup of 2 to 2.5 over the Pentium.

The default settings enable order–1 Taylor refinement, followed by a Newton–Gauß–Seidel iteration, but disable the less powerful Krawczyk operator and the computationally expensive order–2 Taylor refinement. Slopes are obtained by progressive evaluation of expressions for the derivatives, as explained in Subsection 2.2. Each run was made with these default settings, except for the switch explicitly mentioned in the "settings" column.

The data show that the order–2 Taylor refinement indeed reduces the number of boxes considered during the solution process. But due to the high computational cost this reduction is not necessarily reflected in the overall solution time. By contrast, order–1 Taylor refinement, which may be considered as a modified Newton–Gauß–Seidel iteration without preconditioning, significantly enhances the efficiency of the following (preconditioned) Newton–Gauß–Seidel step. Disabling this technique, the term simplification, or the progressive evaluation of the derivatives can have a serious negative impact on the overall performance. In the examples considered here, the "MaxDiam" subdivision scheme is not competitive with the default scheme "MaxSumMagnitude" discussed in Subsection 2.3.

## 5 Conclusions and Future Directions

We have described a framework for setting up and solving nonlinear systems arising in the analysis and design of chemical processes. The branch–and–bound type nonlinear solver is similar to GlobSol [4], significant differences lying in

the symbolic preprocessing and in the acceleration techniques. In particular, our solver incorporates order–1 and order–2 Taylor refinement steps to reduce the size of the box under consideration before subdividing it. Numerical experiments show that currently small to medium sized systems can be solved.

For the future we plan to further improve on the symbolic preprocessing and to incorporate additional acceleration techniques and verification tests into the solver. A parallel version of the solver will be implemented to allow the solution of larger problems.

## References


1. Uppal, A., Ray, W.H., Poore, A.B.: On the dynamic behavior of continuous stirred tank reactors. Chem. Engng Sci. **29** (1974) 967–985
2. Bischof, C.H., Lang, B., Marquardt, W., Mönnigmann, M.: Verified determination of singularities in chemical processes. In Krämer, W., Wolff von Gudenberg, J., eds.: Scientific Computing, Validated Numerics, Interval Methods, New York, Kluwer Academic/Plenum Publishers (2001) 305–316
3. Golubitsky, M., Schaeffer, D.G.: Singularities and Groups in Bifurcation Theory, Volume I. Springer-Verlag, New York (1985)
4. Kearfott, R.B.: Rigorous Global Search: Continuous Problems. Kluwer Academic Publishers, Dordrecht, The Netherlands (1996)
5. Neumaier, A.: Interval Methods for Systems of Equations. Cambridge University Press, Cambridge, UK (1990)
6. Schulte Althoff, K.: Algorithmen zum verifizierten Lösen nichtlinearer Gleichungssysteme. Diploma thesis, Aachen University, Germany (2002)
7. Charniak, E., Riesback, C., McDermott, D.: Artificial Intelligence Programming. 2nd edn. Lawrence Erlbaum Associates, Hillsdale NJ (1987)
8. Ratz, D.: Automatic Slope Computation and its Application in Nonsmooth Global Optimization. Shaker Verlag, Aachen, Germany (1998)
9. Griewank, A.: Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation. SIAM, Philadelphia (2000)
10. Oliveira, J.B.: New slope methods for sharper interval functions and a note on Fischer's acceleration method. Reliable Computing **2** (1996) 299–320
11. Beelitz, T.: Methoden zum Einschluss von Funktions- und Ableitungswerten. Diploma thesis, Aachen University, Germany (2002)
12. Berz, M., Hofstätter, G.: Computation and application of Taylor polynomials with interval remainder bounds. Reliable Computing **4** (1998) 83–97
13. Berz, M.: COSY INFINITY version 8.1 programming manual. Technical Report MSUHEP-20703, East Lansing, MI (2002)
14. Kienitz, A.: Untersuchungen zum Einsatz von Taylormodellen bei der verifizierten Lösung von Gleichungssystemen. Diploma thesis, Aachen University, Germany (2003)
15. Klatte, R., Kulisch, U., Wiethoff, A., Lawo, C., Rauch, M.: C-XSC — A C++ Class Library for Extended Scientific Computing. Springer-Verlag, Heidelberg (1993)
16. Sun Microsystems: C++ Interval Arithmetic Programming Reference. (2002)
17. Goldberg, D.: What every computer scientist should know about floating-point arithmetic. ACM Computing Surveys **23** (1991) 5–48
18. Morgan, A., Shapiro, V.: Box-bisection for solving second-degree systems and the problem of clustering. ACM Trans. Math. Software **13** (1987) 152–167

# Verified Numerical Analysis of the Performance of Switching Systems in Telecommunication

Daniela Fausten[1] and Gerhard Haßlinger[2]

[1] Universität Duisburg-Essen
Fakultät für Ingenieurwissenschaften
47048 Duisburg, Germany
fausten@informatik.uni-duisburg.de
[2] Deutsche Telekom, T-Systems
Am Kavalleriesand 3
64307 Darmstadt, Germany
gerhard.hasslinger@telekom.de

**Abstract.** The computation of workload distributions of service systems in telecommunication networks is essential for determining quality of service (QoS) parameters for various types of data transfer traffic. We discuss the relation of QoS demands from the end-to-end perspective of users to possible QoS degradation by network elements e.g. when overload situations are encountered. Modeling and analysis approaches of performance measures in multi service networks are also summarized with regard to the representation of usual traffic pattern being observed in the Internet. Depending on the model, there are different ways to determine workload distributions. We investigate Wiener-Hopf factorization as an efficient approach for discrete time semi-Markovian server systems. A numerical solution of the steady state workload distribution is computed in extension of an algorithm by Grassmann and Jain. After a verification step the guaranteed workload distribution can be computed using interval arithmetic. A C++ tool for this modeling approach using C-XSC is presented as well as examples of the evaluation.

## 1 Introduction: Traffic Modeling in Telecommunication Networks

The performance characteristics of services delivered through data transmission over telecommunication networks depends on many interacting elements and control functions in a multi-layered environment. Users, application developers, service providers, equipment manufacturers, standardization bodies for communication protocols and other contributors are involved in a complex process in order to provide acceptable communication services, where each of those parties is acting from a different viewpoint.

There is a trend to integrate multiple services over IP networks as a common infrastructure, which diversifies traffic types entering the network as well as the quality of service (QoS) demands. The transmission of huge data volumes with

R. Alt et al. (Eds.): Num. Software with Result Verification, LNCS 2991, pp. 206–225, 2004.

strict error and data loss requirements is an extreme case which usually has less strict time constraints and forms a class of elastic traffic. On the other hand, real time applications are subject to bounds on the delay and delay jitter, which are as low as 20 ms per switching element for speech, although most real time services are more tolerant regarding transmission errors.

Traffic engineering is a central task in the communication scenario, which is linked to the planning, operation and resource management of networks as well as to the quality of service requirements of the applications.

The nature of traffic in telecommunication networks is unpredictable from different viewpoints. Service providers do not exactly know the transmission volume and the time when new demands are created by the users and applications whereas the end systems cannot predict the utilization of network resources at the time when they require some communication service. Randomly changing system parameters are relevant in normal operation. In addition, failure events like transmission errors or breakdowns of transmission links and switching systems are unpredictable as well.

Therefore the performance and the quality of service properties of communication systems depend on random processes, which form a basis for the representation of traffic as it is developing over time. Usual stochastic traffic models in telecommunication consider random variables

- for the interarrival times of events corresponding e.g. to arrivals of packets, flows, connections or other units relevant at network elements as the classical approach in queueing and service systems [15],
- for the counting function of the number of arrivals in predefined intervals e.g. in time slotted multiplexer models [2,17],
- for the traffic rate e.g. in fluid flow models [3].

Two basic characteristics for the stochastic behavior of traffic are

- the distribution function of considered random variables and
- the autocorrelation of the process.

Distribution functions capture the variability of a process in stationary conditions which may be observed as the limiting behavior over arbitrary long time. The autocorrelation is a measure for the dependency of values observed in a process between two different points in time as a function of their time distance.

The distribution of the amount of arriving data for a transmission line is essential to estimate overload situations leading to data loss. From knowledge of the distribution function of the delay on a transmission path we can determine a delay threshold which is exceeded with 1% or the probability that packets miss their play out time at the receiver by violating a predefined delay bound. In this way, QoS guarantees are often formulated as service level agreements for telecommunication services on a probabilistic basis.

In addition, the autocorrelation function can distinguish whether the delay of a packet in a transfer flow is independent of the delay of previous packets

or if consecutive packets have a similar delay. This indicates whether too large delays are typically observed for single packet without impact on the next ones or otherwise for coherent bursts of many packets. Both cases have different impact on the QoS since e.g. the loss of single voice packet may be bridged by appropriate coding without being noticed, which is not possible for longer gaps.

Autocorrelation often spreads over multiple time scales where a different context is responsible for dependencies in each of them. In a transmission system distinct levels at different time scales include

- a packet level characterized by interarrival times between successive packets,
- a packet burst or packet train level, which comprises a number of packets generated as a non-interrupted transmission sequence e.g. for data downloaded from a server or for a speech phase of a voice coder,
- a connection or call level for the complete data transmitted during a TCP connection or a telephone call etc.

Further dependencies at a certain time scale are relevant at a session level for the complete time when an online user is connected to the Internet. Video coding schemes with periodically changing transmission rates for groups of pictures introduce a context in the time scale of about a second and for complete scenes on a longer time scale. A periodical profile is also observed at a daily time scale for the complete traffic on links of service provider networks with heavy and low traffic hours.

Semi-Markov Processes (SMP) can be used to model the distribution function of arriving data as well as the autocorrelation function. The analysis of such models and the modeling itself should be done as exactly as possible, since the results have influence on service level agreements. The proposed Wiener-Hopf factorization method to analyze a SMP has favorable computational complexity, but the convergence and numerical stability are not ensured.

Consequently, the application of interval arithmetic is included to guarantee the results of the analysis and to validate the method. Hence, we can achieve assured information about data delay and loss probabilities. Prior results for GI/GI/1 systems, which can be viewed as simple special cases of semi-Markov processes, show that results given in literature are not always exact [4]. Nevertheless, a validation of the model itself is not provided.

We proceed with a closer look at the context of QoS demands and traffic characteristics in section 2, followed by a summary of analysis methods to determine the QoS performance of switching elements in section 3. Sections 4-5 focus on Wiener-Hopf factorization as an efficient alternative with low computational complexity to analyze especially the workload and waiting time distribution of a buffered switching system. Section 6 presents a computation algorithm with result verification, which accounts for the non-assured numerical stability of the Wiener-Hopf method. The paper is rounded up by numerical examples in section 7 and the conclusions.

## 2 Quality of Service Aspects

Users, applications and end-systems are requesting telecommunication services together with specific quality of service demands. The QoS performance may be estimated based on the users sensitiveness of impairments, e.g. by the MOS (mean opinion score) scale for voice recognition ranging from small to severe disturbances which make a conversation more or less impossible. Similarly, transmission errors in a video stream may range from an unnoticeable to unbearable impact. In addition, the level of impact depends on the sensitivity of users and may be estimated differently by another person. While QoS measures are difficult to define on a user perception level, we can identify the following parameters in order to determine QoS properties of transmission channels

- the required bandwidth,
- the transmission delay from sending to receiving data,
- the delay jitter, i.e. differences in the delay between successively transmitted data packets,
- the reliability with regard to failures of network resources and bit and packet errors.

In principle, communication services may demand for an instantaneous and non-modified delivery of transmitted data, but in practice limited bandwidth, variable delays and transmission errors impose restrictions on the QoS.

Table 1 shows typical parameter values of QoS requirements for voice, video and data transmission as the basic service types in nowadays multi-service networks. It depends on the applications and the communication services, whether the limitations and uncertainties in transmission are acceptable. The necessary bandwidth ranges from a few kbit/s for compressed voice traffic to several Gbit/s for a single video transmission in HDTV quality without compression and is even more diversified for data transmission.

The delay sensitivity mainly distinguishes real time applications like voice, video conferencing etc. where interaction is essentially disrupted by delays beyond 0.25-0.4 s from delay tolerant elastic transmission. If the receiver is unable to store data in a play-out buffer then even the variation of delay (delay jitter) for successive transmission units has to be small as supported by synchronous link layer architectures (SDH, SONET). In packet networks, play-out buffers have to compensate for delay jitter since no guarantees on jitter are possible at the IP network boundaries.

There is a general relation between the QoS objectives for delay and data loss in real time services with a fixed delay bound as pointed out in figure 1. The distribution of the delay composed of fixed and variable portions determines the probability that a bound for the maximum tolerable delay is exceeded, which corresponds to the data loss probability. Variable delay mainly depends on buffering and contention of packets in switching systems. The analysis of a single switching system or multiplexer is addressed in sections 4-7, especially with regard to the distribution of the delay and the data loss rate as main QoS parameters.

**Table 1.** QoS characteristics of telecommunication service types

| Quality of service demand parameters for service types | bandwidth [Mbit/s] | tolerable delay [s] | bit error rate | burst factor: $\frac{\text{peak rate}}{\text{mean rate}}$ |
|---|---|---|---|---|
| VOICE | | | | |
| no compression (ISDN) | 0.064 | $< 0.25$ | $\approx 10^{-3}$ | 1 |
| with compression | 0.002–0.016 | $< 0.25$ | $\approx 10^{-4}$ | 2–3 |
| (VoIP, GSM, UMTS) | | | | |
| VIDEO | | | | |
| no compression | 1–5 000 | | $\approx 10^{-3}$ | 1 |
| with compression | 0.1–500 | | $< 10^{-5}$ | $\approx$ 2–30 |
| on demand | | $< 10$ | | $\approx$ 3–30 |
| conferencing | | $< 0.25$ | | $\approx$ 2–10 |
| DATA TRANSFER | | | | |
| peer to peer, file transfer | in a wide | $\approx$ 10–10000 | 0 | depends on |
| E-mail | range from | $\approx$ 1–1000 | 0 | application; |
| WWW access | smallband | $< 5$ | 0 | often |
| realtime data | to broadband | $< 1$ | 0 | very large |

Data compression is applicable to most service types and has a major impact on traffic variability and QoS parameters. Compression schemes are a main driver of increased traffic variability, often converting constant bit rate source traffic into variable bit rate. The burstiness or burst factor of traffic represents an indicator of the variability, which is captured in table 1 as the ratio of the peak rate to the mean rate. For voice over IP with silence suppression the burst factor is about 2-3; video and data transmission often have much higher burstiness. On the other hand, efficient coding may introduce a non-negligible delay for real time applications.

Finally, considering interfaces between service provider networks as well as between service providers and customer networks, the QoS properties of services are negotiated and defined as service level agreements (SLA), which have to be monitored in operational state by adequate measurement of thresholds indicating possible degradation of service and SLA violation. A challenging problem addressed e.g. in RFC 2990 (www.rfc-editor.org) is then to establish a chain of control items to enforce the required service quality on all involved subnets and network elements and to give direct feedback to the end systems about the network status. Regarding a negotiated SLA and an appropriate monitoring of the QoS from the users perspective, service providers will then deliver the offered services over a platform in a cost efficient way by the help of traffic engineering methods and tools.

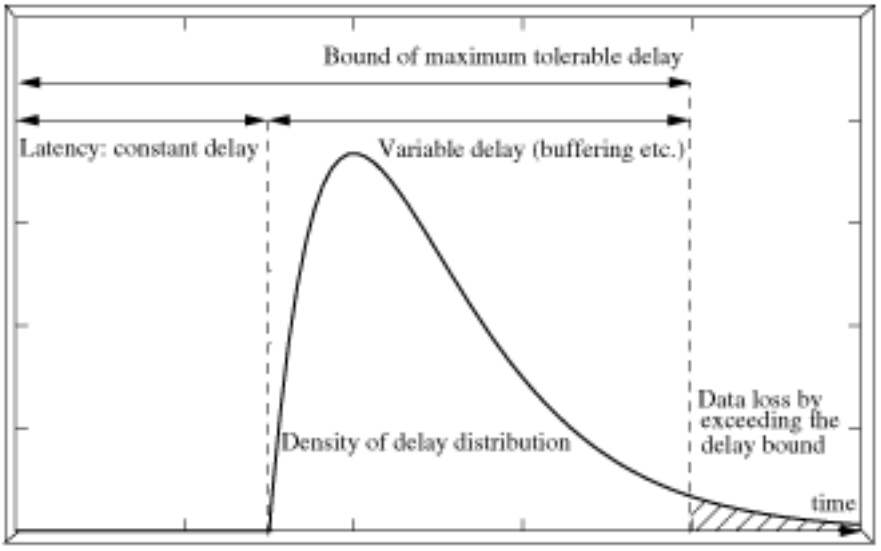



**Fig. 1.** Relation of tolerable delay and data loss for real time services

## 3 Traffic Characterization and Performance Analysis

Traffic modeling is important for the management and planning of telecommunications networks. In order to make an efficient use of resources, network operators are required to perform frequent traffic measurement, from which descriptors and parameters have to be extracted appropriately for traffic characterization. When selecting a stochastic traffic source model, there is the need to consider fitting procedures appropriate for parameter estimation. The design of the fitting procedure is a trade-off between computational complexity and accuracy and requires careful consideration of the model parameters that are most relevant to the performance metrics of interest.

The unique traffic representation is a basis for network-wide traffic modeling and evaluation. Open queueing networks provide a usual approach for the analysis of (tele-)communication systems being decomposed into nodes representing switches and routers and edges as transmission links. The main items of this approach are

- traffic flows on the edges of a network topology being characterized by stochastic processes,
- server systems for switching and forwarding of traffic with the help of buffers to store data in phases of temporary overload and
- sources and destinations, which generate and terminate traffic flows in the network.

The main analysis steps in a decomposition approach determine

- the aggregation of a number of traffic flows entering a switching system,
- the evaluation of the switch performance with regard to waiting times and data loss together with the transformation of input traffic flows into output traffic and
- the branching of traffic into subflows with different destinations departing from a switch, which appear as the input at other network nodes.

In this environment the traffic on each network link is represented in an appropriate form by aggregation and branching starting from the source traffic. As the main result, bottlenecks can be found for a predefined topology and traffic demands and a network dimensioning by sufficient link bandwidth, forwarding and buffering capacities can be performed.

### 3.1 Traffic Characterization by Semi-Markov Processes

Semi-Markov processes (SMP) extend the well known Markov models with finite state space and memoryless states with exponential or geometrical distribution, such that each state is associated with an arbitrary state specific distribution [13, 22]. As usual, a matrix of state transition probabilities determines the underlying Markov chain, see section 4 for a definition of SMP. Efficient analysis methods are available for multiplexing and switching systems with semi-Markov input in discrete time domain [7,9]. This approach is appropriate to capture the distribution of traffic rates on network links, which are limited by the capacity. The modeling approach is similar to other popular models by fluid flow or time slotted systems [2,3,17].

Some basic properties of the SMP autocorrelation function are summarized in [10]. Although including arbitrary distributions in each state, the SMP autocorrelation depends only on the mean sojourn time of each state and the underlying Markov chain. Therefore it does not become more complex than for models with memoryless states. For a SMP($M$) with $M$ states in the underlying chain the autocorrelation has the form of a superposition of $M-1$ geometrical terms including complex coefficients. In general, this implies a geometrical decay of the SMP autocorrelation, which is different from the slowly decaying or heavy tail behavior for long term correlation. Nevertheless, the fitting procedures based on MMPP and MAP [6] for autocorrelation over multiple time scales show that Markovian and SMP models are useful to represent long term correlation as well.

In recent years it has been clearly shown through experimental evidence, that network traffic may exhibit properties of self-similarity and long-range dependence (LRD) [11,20]. These characteristics have significant impact on network performance especially with regard to buffer efficiency. However, as pointed out in [6], matching the LRD is only required within limited time scales of interest to the system under study. For example, in order to analyze queuing behavior, the selected traffic model needs only to capture the correlation structure of the source up to the so-called critical time scale or correlation horizon, which is

directly related to the maximum buffer size. One of the consequences of this result is that (semi-)Markov processes can still be used to model traffic exhibiting LRD, although a large state space may be required.

On the other hand, modeling of short range dependency is also necessary for some traffic types. The aggregation of a sufficient large number of on-off voice sources (VoIP) comes close to an autoregressive process [10] which represents a basic, although simplifying model for video traffic.

A simple and general result determines the autocorrelation of aggregated traffic with rate $S = R_1 + \cdots + R_N$ composed by independent flows $R_1, \cdots, R_N$. Backbone links in the Internet are usually loaded with a large number of independent flows. Then the autocorrelation function $A_S(n)$ is determined by the autocorrelation $A_{R_k}(n)$ of the included flow components, each of them weighted by the variance $\sigma_k^2$ of the traffic rate:

$$A_S(n) = \sum_{k=1}^{N} \sigma_k^2 \, A_{R_k}(n) \Big/ \sum_{k=1}^{N} \sigma_k^2.$$

For homogeneous traffic flows, which are all of the same type with a specific autocorrelation, the same autocorrelation is still valid for the aggregated traffic. Thus superposed traffic of a number of voice sources with identical coding and transmission scheme preserves the autocorrelation structure of the source traffic. As a consequence, the traffic aggregated from flows belonging to the same application still shows autocorrelation in those time scales which are characteristic for the application, as observed e.g. by [10] for voice and video. On the other hand, self-similarity and correlation over arbitrary time scales arises from the superposition of heterogeneous traffic composed of many different applications. This is again predicted by the previous formula, where the autocorrelation accumulates in any time scale which has relevant contributions from some of the aggregated flows.

### 3.2 Analysis of Multiplexers with Semi-Markov Input

From the modeling aspect, usual approaches by fluid flow analysis or time slotted systems are based on Lindley's equation [18] for the steady state workload, which makes the analysis more or less equivalent to the analysis of semi-Markovian servers.

Analysis methods for multiplexing, routing and switching systems integrate the input and service process into a corresponding Markovian state model. Implications on QoS parameters like delay, data loss and buffer occupancy then become apparent from the steady state solution of the system. While the QoS analysis is simple to handle without buffers, it becomes complex and requires elaborate queueing theoretic approaches with regard to computational expense and stability when buffers are included. Among many different approaches in the huge amount of published work in this area, there are two main principles being addressed by numerous researchers:

- Matrix-analytic approaches have been established in a book by M.F. Neuts [19] as a standard method and have been improved in many ways in the last decades, where recent results can be followed on the conference series especially on matrix analytic methods, the "Series of International Conferences on Matrix Analytic Methods in Stochastic Models (MAM1 - MAM4)".
- A second widely used class of methods may be denoted as factorization approaches, but no unique notation is used in literature. Similar algorithmic approaches in this area may be found under key words like spectrum expansion, spectral or polynomial (Wiener-Hopf) factorization, method of generating functions, ($z$-, Laplace-)transform inversion, root finding algorithm, eigenvalue representation, fluid flow analysis, etc.

A detailed description of the basic approach can be found in a book by L. Kleinrock [15] (chapter 8 in vol. 1 and half of vol. 2) for classical queueing systems, in works by D. Mitra et al. [3] for fluid flow analysis and by H. Bruneel [2] and S.-Q. Li [17] et al. for discrete time as well as in [1] for phase type analysis of multiplexers.

The major trade off between both principles is that factorization approaches often have a lower computational complexity, i.e. a quadratic increase in computation time and a linear increase in storage occupancy with regard to the number of stages in classical GI/GI/1 queueing systems with general independent interarrival and service time distributions, whereas a cubic computation time and a quadratic storage consumption is required by comparable matrix analytic approaches [1,16], which on the other hand provide assured properties for numerical stability in computation.

The time consuming part of polynomial factorization is the determination of $h$ roots of the characteristic polynomial of degree $g + h$, where $g$ and $h$ are the number of stages (phases) of the arrival and the service time distribution. The boundary conditions of the system form a system of linear equations, which reveals a Vandermonde structure and can be solved by an explicit expression. Thus matrix computation can be completely avoided for the GI/GI/1 case yielding a quadratic run time complexity $\mathcal{O}((g + h)\, h)$ and a linear space complexity $\mathcal{O}(g + h)$.

Wiener-Hopf factorization leads to an equivalent form of the steady state solution represented on the basis of generating functions. By the help of an attractive algorithm proposed by Grassmann and Jain [7] the computational evaluation is even faster than the determination of the roots and eigenvalues in polynomial factorization.

In the GI/GI/1 case the complexity of both algorithms depends on $g$ and $h$, but is fairly independent of other system parameters e.g. the utilization. For SMP/GI/1 systems our implementation of the Wiener-Hopf factorization spends increasing computation time when the SMP autocorrelation becomes larger [10].

Both factorization approaches have some drawbacks concerning numerical stability in a straightforward implementation: The computation of the characteristic polynomial's roots normally accumulates errors, so that for larger degree of the polynomial the computed roots become more and more inaccurate.

Even the decision, which of the computed roots are located inside the unit disk and therefore are relevant for the solution may become uncertain. On the other hand, the algorithm of Grassmann and Jain is not proven to be convergent in the SMP/GI/1 case.

Therefore we have implemented the factorization approaches for the GI/GI/1 case with regard to achieve verified results by using interval arithmetic [4,5]. The implementation of Wiener-Hopf factorization for systems with semi-Markovian arrival with result verification is described in the following sections.

## 4 Workload of a Discrete Time Semi-Markovian Server

We inspect a single server system at a sequence of embedded points in time $t_n, \forall\, n \in \mathbb{N}_0 : t_n < t_{n+1}$. $W_n$ denotes the workload in the system at time $t_n$. We consider a discrete time system whose workload $W_n$ is developing according to Lindley's equation [18]

$$\forall\, n \in \mathbb{N}_0 : \;\; W_{n+1} = \max(W_n + U_n, 0), \;\; W_n \in \mathbb{N}_0, \;\; U_n \in \mathbb{Z}, \;\; -g \le U_n \le h, \quad (1)$$

where $U_n$ determines the differences observed in the workload between embedded time points. We assume distributions of finite support ($-g \le U_n \le h$), which enable to apply efficient computation schemes in the analysis on account of restricting the process class. Nevertheless, the discrete time approach provides simple adaptation schemes for many continuous processes, e.g. the analysis of a multiplexer with a first order autoregressive input [10]. An extension of the analysis for $h \to \infty$ is possible as shown in [11] for GI/G/1 servers. The probabilities of a level $k$ to be reached by the workload are expressed by a convolution for known distribution of $U_n$ starting e.g. from $W_0 = 0$:

$$\Pr\,(W_{n+1} = k) = \sum_{i=-h}^{g} \Pr\,(W_n = k + i) \Pr\,(U_n = -i) \;\text{ for }\; k > 0. \quad (2)$$

A discrete semi-Markov server is characterized by state specific distributions for $U_n$ depending on a finite Markov chain $\sigma_n \in \{1, \ldots, m\}$ with a state transition at each embedded time $t_n$. A series $\{U_n, \sigma_n\}$, $n \in \mathbb{N}_0$, forms a discrete semi-Markov process, if

$$\Pr\,(U_{n+1} = k, \sigma_{n+1} = j \,|\, U_0, \ldots, U_n,\, \sigma_0, \ldots, \sigma_n) = \Pr\,(U_{n+1} = k, \sigma_{n+1} = j \,|\, \sigma_n),$$

where the considered probabilities are independent of $n$. The state specific distributions $u_{ij}(k)$ completely determine a SMP including the transition probabilities $p_{ij}$ and the steady state probabilities $p_i$ of the underlying chain.

$$u_{ij}(k) \stackrel{\text{def}}{=} \Pr\,(U_{n+1} = k,\, \sigma_{n+1} = j \,|\, \sigma_n = i); \qquad i, j \in \{1, \ldots, m\};$$
$$\boldsymbol{u}(k) \stackrel{\text{def}}{=} \big(u_{ij}(k)\big); \qquad -g \le k \le h.$$

As the limit for stationary distributions we use $p_i \stackrel{\text{def}}{=} \lim_{N\to\infty} \sum_{n=1}^{N} \Pr(\sigma_n = i)/N$ regardless of periodicity in the underlying chain.

$$p_{ij} \stackrel{\text{def}}{=} \Pr(\sigma_{n+1} = j \,|\, \sigma_n = i) = \sum_k u_{ij}(k); \qquad \boldsymbol{P} \stackrel{\text{def}}{=} (p_{ij}); \tag{3}$$

$$p_i = \sum_{j=1}^{m} p_j\, p_{ji} \quad \text{and} \quad \sum_{i=1}^{m} p_i = 1; \qquad \boldsymbol{p} \stackrel{\text{def}}{=} (p_1, \ldots, p_m)^T.$$

The transition matrix $\boldsymbol{P}$ is assumed to be irreducible thus providing unique stationary probabilities $p_1, \ldots, p_m$ of the underlying Markov chain. We are interested in a solution for the distribution $w(k)$ for the workload in steady state

$$w(k) = \lim_{N\to\infty} \sum_{n=1}^{N} \Pr(W_n = k) \;\text{ for }\; k \in \mathbb{N}_0.$$

The state specific characteristics of the semi-Markov server is also relevant for the workload. Therefore we denote

$$w_i(k) \stackrel{\text{def}}{=} \lim_{N\to\infty} \sum_{n=1}^{N} \Pr(W_n = k,\, \sigma_n = i) \;\text{ for }\; k \in \mathbb{N}_0;\, i = 1, \ldots, m.$$

As a necessary precondition of the existence of a steady state distribution we assume the mean $E(U)$ of the workload differences $U_n$ to be negative

$$E(U) \stackrel{\text{def}}{=} \sum_k \sum_{i=1}^{m} \sum_{j=1}^{m} k\, p_i\, u_{ij}(k) < 0.$$

In addition, the generating functions $\mathcal{U}_{ij}(z)$ are defined and summarized in a matrix representation $\boldsymbol{\mathcal{U}}(z)$

$$\mathcal{U}_{ij}(z) = \sum_{k=-g}^{h} u_{ij}(k) z^k; \qquad \boldsymbol{\mathcal{U}}(z) = (\mathcal{U}_{ij}(z))_{i,j=1,\ldots,m}.$$

Finally, we refer to several special cases of the considered semi-Markov server system.

- A classical SMP/GI/1 service system with semi-Markov arrival process is included with time points $t_n$ being embedded at arrival instances. Then the interarrival times $A_n$ and the general independent service times $S_n$ have distributions of finite support $0 < A_n \le g$ and $0 \le S_n \le h$ and $U_n = S_n - A_n$.
- A GI/SMP/1 system with semi-Markov service process is included with time points $t_n$ being embedded at instances when a service is completed. Again, the interarrival times $A_n$ and service times $S_n$ have distributions of finite support $0 \le A_n \le g$ and $0 < S_n \le h$ and $U_n = S_n - A_n$.

- A time slotted multiplexer model [17] with equidistant time points $t_n = n \cdot \Delta$ and semi-Markov arrival and service process is included. When the number of arrivals $A_n$ and the service capacity $S_n$ available in the $n$-th slot are limited $0 \leq A_n \leq h$ and $0 \leq S_n \leq g$ and when a common underlying Markov chain $\sigma$ is combined from the underlying chains of both SMP for the arrival and service process, then we again have a semi-Markov server for $U_n = A_n - S_n$. Time-slotted modeling is often applied in communication systems since the time scale ($\Delta$) can be chosen arbitrarily and independent from arrival or service events as in classical queueing systems.
- Fluid flow systems [3] are a closely related approach, since they also apply Lindley's equation to a state specific process determining differences in the workload, when instances of state transitions are considered for the embedded sequence $t_n$. Then $U_n$ is always exponentially distributed with a state specific mean [3].

## 5 Wiener-Hopf Factorization Approach

In the following, we consider a single server system with semi-Markovian arrival and general independent service time distributions, denoted as SMP/GI/1.

In order to solve Lindley's equation (1), we use the Wiener-Hopf factorization. This approach is well known and explained in detail in [7] and [9]. For the sake of completeness we summarize the main steps.

The main idea of Wiener-Hopf factorization is the division of a server system's busy period into phases. A phase starts at the arrival of a service demand, each time after the workload has reached a lowest level in the current busy period. The level of a phase is the workload at its start, not including the arriving service demand. A phase is finished as soon as the workload drops below the level of the phase. From this view, a busy period consists of an initial interarrival time and a number of phases and interphase intervals, so called idle periods.

We consider the states of the underlying Markov chain immediately prior to a busy period as well as prior an inherent phase, which are referred to as initial states of a busy period or a phase respectively. Final states of a busy period or phase are initial states of the following one, since there are no transitions in between. Let $\sigma_A$ and $\sigma_E$ be the initial and final state of a given busy period. The state dependent distribution of the subsequent idle period $I$ is denoted by

$$l_{ij}(k) \stackrel{\text{def}}{=} \Pr\left(I = k,\, \sigma_E = j \,|\, \sigma_A = i\right); \tag{4}$$

$$l_{ij} \stackrel{\text{def}}{=} \sum_k l_{ij}(k) = \Pr\left(\sigma_E = j \,|\, \sigma_A = i\right). \tag{5}$$

$l_{ij}(k)$ is the probability for an idle period of duration $k$, $l_{ij}$ are the transition probabilities in the underlying chain from an idle period to the next one. $v_{ij}(k)$ denotes the probability that a phase with level $k$ and initial state $j$ is observed within a busy period having initial state $\sigma_A = i$. Let $\boldsymbol{l}(k)$ and $\boldsymbol{v}(k)$ denote

corresponding matrices

$$\boldsymbol{l}(k) = (l_{ij}(k)); \;\; \boldsymbol{v}(k) = (v_{ij}(k)); \;\; i,j \in \{1,\ldots,m\}.$$

The division of a busy period implies the following relationships among the probabilities $v_{ij}(k)$ and $l_{ij}(k)$

$$\boldsymbol{v}(n) = \quad \boldsymbol{u}(n) + \sum_{m=1}^{\min(h-n,g)} \boldsymbol{v}(n+m)\,\boldsymbol{l}(m) \;\text{ for }\; n = h,\ldots,0; \tag{6}$$

$$\boldsymbol{l}(n) = \boldsymbol{u}(-n) + \sum_{m=0}^{\min(g-n,h)} \boldsymbol{v}(m)\,\boldsymbol{l}(m+n) \;\text{ for }\; n = g,\ldots,1. \tag{7}$$

Here it should be noted that a busy period with $\sigma_A = i$ can be viewed as a phase of level 0 starting from state $i$, and that a phase of level $n+m$ starting from state $j$ is followed with probability $l_{jk}(m)$ by a phase of level $n$ starting from state $k$.

The first equation (6) gives a recursive scheme to determine $\boldsymbol{v}(h), \ldots, \boldsymbol{v}(0)$ when $\boldsymbol{l}(1), \ldots, \boldsymbol{l}(g)$ are known, and the second equation (7) yields a recursive scheme for $\boldsymbol{l}(g), \ldots, \boldsymbol{l}(1)$ from known $\boldsymbol{v}(0), \ldots, \boldsymbol{v}(h)$.

Grassmann and Jain [7] proposed an iterative solution for equations (6) and (7) in the GI/GI/1 case, which we have transferred to the SMP/GI/1 model. Section 6 describes the method in detail.

From the solutions of (6) and (7) we obtain the stationary probabilities $l_j$ of the underlying Markov chain in idle periods and the distribution $l(k)$ of their length

$$l_j \stackrel{\text{def}}{=} \Pr(\sigma_A = j) = \Pr(\sigma_E = j) = \sum_{i=1}^{m} l_i l_{ij} \;\text{ for }\; j = 1,\ldots,m; \;\; \sum_{j=1}^{m} l_j = 1; \tag{8}$$

$$l(k) \stackrel{\text{def}}{=} \sum_{i,j} l_i l_{ij}(k) \;\text{ for }\; k \in \mathbb{N}_0.$$

In order to compute the waiting time distribution, we distinguish the states of the underlying chain prior a busy period. As defined in section 4, $w_i(k)$ denotes the probability that the waiting time of a service demand lasts $k$ time units, with generating function $\mathcal{W}_i(z)$. $N_i$ denotes the number of demands served per busy period with $\sigma_A = i$. Then the following relationships hold

$$E(N_i) = 1 + \sum_{j=1}^{m} \sum_{n=0}^{h} v_{ij}(n) E(N_j); \tag{9}$$

$$\mathcal{W}_i(z) E(N_i) = 1 + \sum_{j=1}^{m} \sum_{n=0}^{h} v_{ij}(n) z^n \mathcal{W}_j(z) E(N_j). \tag{10}$$

An interpretation of these formulas is given in [9]. Let

$$\mathcal{V}_{ij}(z) \stackrel{\text{def}}{=} \sum_{n=0}^{h} v_{ij}(n) z^n, \;\; \boldsymbol{\mathcal{V}}(z) \stackrel{\text{def}}{=} (\mathcal{V}_{ij}(z)),$$

$\mathbf{1} \stackrel{\text{def}}{=} (1,\ldots,1)^T$ and let $\boldsymbol{I}$ denote the $(m \times m)$ identity matrix. $\boldsymbol{X}^T$ indicates transposition of a vector $\boldsymbol{X}$. Then we obtain from equations (9) and (10)

$$(\boldsymbol{I} - \boldsymbol{\mathcal{V}}(1)) \cdot (E(N_1),\ldots,E(N_m))^T = \mathbf{1}; \tag{11}$$

$$(\boldsymbol{I} - \boldsymbol{\mathcal{V}}(z)) \cdot (E(N_1)\mathcal{W}_1(z),\ldots,E(N_m)\mathcal{W}_m(z))^T = \mathbf{1}; \tag{12}$$

and

$$\mathcal{W}(z) = \sum_{i=1}^{m} l_i E(N_i)\mathcal{W}_i(z) / \sum_{i=1}^{m} l_i E(N_i) \tag{13}$$

for the generating function $\mathcal{W}(z)$ of the waiting time of an arbitrary service demand. The $n$-th derivatives at $z = 0$ of (12) determine the probabilities $w(n)$ of the waiting time:

$$(\boldsymbol{I} - \boldsymbol{\mathcal{V}}(0)) \cdot (E(N_1)w_1(0),\ldots,E(N_m)w_m(0))^T = \mathbf{1} \tag{14}$$

and

$$E(N_i)\mathcal{W}_i^{(n)}(z) = \sum_{j=1}^{m}\sum_{k=0}^{n} \binom{n}{k} \mathcal{V}_{ij}^{(k)}(z) E(N_j)\mathcal{W}_j^{(n-k)}(z) \text{ for } n \in \mathbb{N}$$

$$\Rightarrow (\boldsymbol{I} - \boldsymbol{\mathcal{V}}(0)) \cdot (E(N_1)w_1(n),\ldots,E(N_m)w_m(n))^T$$

$$= \sum_{k=1}^{\min(h,n)} \boldsymbol{v}(k) \cdot (E(N_1)w_1(n-k),\ldots,E(N_m)w_m(n-k))^T. \tag{15}$$

Finally, we obtain from equation (13)

$$w(n) = \frac{\sum_{i=1}^{m} l_i E(N_i) w_i(n)}{\sum_{i=1}^{m} l_i E(N_i)}. \tag{16}$$

## 6 Verified Computation Using Wiener-Hopf Factorization

In the following, we will present our method to achieve verified enclosures of the system's workload, using the described techniques of the Wiener-Hopf factorization approach.

We have implemented the Wiener-Hopf factorization in C++ using C-XSC and the associated toolbox [8,12,14,23]. C-XSC is a C++ class library providing interval arithmetic. The toolbox uses C-XSC and consists of tools to find verified solutions for basic numerical problems. We use 2.0 $\beta$2 versions of C-XSC and the toolbox, which can be downloaded from [23].

Our program consist of two parts: First, it computes the Wiener-Hopf factorization numerically using the method developed by Grassmann and Jain [7], adapted to the SMP/GI/1 case. After this, there is a verification step. A schematic overview of our tool is given in figure 2. In detail, our method proceeds as follows: First we enclose the input file data in intervals. Then we compute the difference distribution (interval-)matrices $\boldsymbol{u}(k)$, $(-g \le k \le h)$.

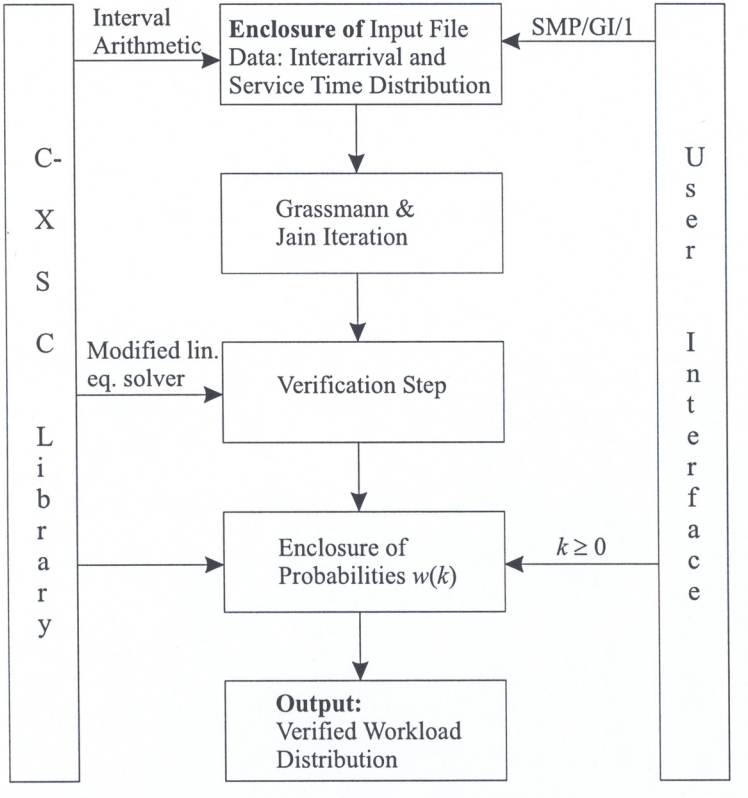



**Fig. 2.** Schematic overview on verified Wiener-Hopf factorization

In order to solve equations (6) and (7), the iteration process introduced by Grassmann and Jain takes initial approximations for $\boldsymbol{l}(n)$, for example $l_{ij}^{(0)}(n) = \mathrm{mid}(u_{ij}(-n))/\sum_{k=1}^{g} \mathrm{mid}(u_{ij}(-k))$, $n = 1, \ldots, g$, and then uses these values in equation (6) to compute approximations $\boldsymbol{v}^{(0)}(n)$. Here $\mathrm{mid}(x)$ denotes the midpoint of the corresponding interval $x$.

Applying formulas (6) and (7) alternately we obtain a sequence of distributions $\boldsymbol{l}^{(k)}(n)$, $\boldsymbol{v}^{(k)}(n)$ for $k = 0, 1, \ldots$ Note, that equation (7) forms an implicit system for $\boldsymbol{l}(n)$, so we use the following equation instead

$$\boldsymbol{l}(n) = (\boldsymbol{I} - \boldsymbol{v}(0))^{-1} \left( \mathrm{mid}(\boldsymbol{u}(-n)) + \sum_{m=1}^{\min(g-n,h)} \boldsymbol{v}(m)\, \boldsymbol{l}(m+n) \right)$$

for $n = g, \ldots, 1$. The computation of the inverse matrix is done using the function `mathInv` provided by the toolbox [8]. The iteration stops if the differences between the new and the old values are sufficiently small. Now we have suitable approximations for the values of $\boldsymbol{v}(n)$ and $\boldsymbol{l}(n)$.

Assume the iteration has stopped for a $k \in \mathbb{N}$. We then enclose the computed values for $\boldsymbol{v}^{(k)}(n)$ by intervals with small diameters, e.g. $\Delta = 10^{-14}$. Thus we obtain interval matrices $[\boldsymbol{v}^{(k)}(n)]$ for $n = 0, \ldots, h$. Then we compute one step of the Grassmann and Jain iteration using interval operations. The formulas (6) and (7) are therefore transformed into interval functions

$$[\boldsymbol{v}(n)] = \quad \boldsymbol{u}(n) + \sum_{m=1}^{\min(h-n,g)} [\boldsymbol{v}(n+m)]\,[\boldsymbol{l}(m)] \;\text{ for }\; n = h, \ldots, 0; \tag{17}$$

$$[\boldsymbol{l}(n)] = (\boldsymbol{I} - [\boldsymbol{v}(0)])^{-1} \left( \boldsymbol{u}(-n) + \sum_{m=1}^{\min(g-n,h)} [\boldsymbol{v}(m)]\,[\boldsymbol{l}(m+n)] \right) \tag{18}$$

for $n = g, \ldots, 1$. To compute the inverse interval matrix, we use the toolbox' algorithm for verified solutions of linear systems of equations, modified as outlined in [21] to cope with interval matrices as input.

If the condition $[\boldsymbol{v}^{(k+1)}(n)] \subset [\boldsymbol{v}^{(k)}(n)]$ then holds for all $n = 0, \ldots, h$, Brouwer's fixed point theorem guarantees that the correct solution $\boldsymbol{v}(n)$ is contained in $[\boldsymbol{v}^{(k+1)}(n)]$ for all $n = 0, \ldots, h$. With this verified result for $\boldsymbol{v}(n)$ we compute a verified enclosure of $\boldsymbol{l}(n)$ using formula (18).

Now, we solve the linear system of equations for the stationary probabilities $l_j$ of the underlying Markov chain in idle periods, given by (8), again using the modified algorithm of the toolbox. After this, we solve (11) for the mean values $E(N_i)$ of the number of demands served per busy period with initial state $i$.

Finally, we recursively compute the values of $E(N_i)w_i(k)$, starting with $k = 0$ and equation (14). Equation (15) has to be solved for each $k \leq n$ for a user-defined maximum value $n$. Here we also need the modified solver for systems of linear equations. With these values we can finally determine enclosures of the probabilities $w(k)$, $k = 0, 1, \ldots, n$ using (16).

## 7 Examples

We will illustrate the proposed procedure by two examples.

### 7.1 Example 1: A System with Deterministic Service

We consider as first example a SMP/D/1 system with

$$A_{ij}(z) = p_{ij} A_i(z), \;\; 1 \leq i, j \leq 2.$$

The arrival process is modeled by a two step SMP, and we have

$$A_1(z) = 0.05 + \sum_{i=1}^{9} 0.1 z^i + 0.05 z^{10},$$

$$A_2(z) = 0.05 + 0.1z + 0.1z^6 + 0.2z^7 + 0.2z^8 + 0.2z^9 + 0.15z^{10},$$

and

$$\boldsymbol{P} = \begin{pmatrix} 0.2 & 0.8 \\ 0.6 & 0.4 \end{pmatrix}.$$

The deterministic service time distribution is given by $S(z) = z^5$.

Since the coefficients of $A_i$, $i = 1, 2$ and the entries of $\boldsymbol{P}$ are not presentable as machine numbers we enclose them in intervals.

The results for the distributions $\boldsymbol{v}(n)$ and $\boldsymbol{l}(n)$ are given in table 2 and 3, respectively. Table 4 shows some results for the waiting time distribution.

**Table 2.** Distributions $\boldsymbol{v}(n)$ of Example 1

| $n$ | $\boldsymbol{v}(n)$ |
|---|---|
| 0 | $\begin{pmatrix} 9.160202536\mathbf{27}^{5454}_{2213} \cdot 10^{-2} & 1.4734682389\mathbf{6}^{6214}_{5879} \cdot 10^{-1} \\ 2.7378910326^{11265}_{09633} \cdot 10^{-2} & 3.0765052083\mathbf{7}^{6273}_{4533} \cdot 10^{-2} \end{pmatrix}$ |
| 1 | $\begin{pmatrix} 7.136347088\mathbf{0}^{51376}_{49188} \cdot 10^{-2} & 1.27668266829\mathbf{9}^{252}_{027} \cdot 10^{-1} \\ 2.4830582206\mathbf{1}^{1462}_{0134} \cdot 10^{-2} & 2.8505523355\mathbf{0}^{7495}_{6077} \cdot 10^{-2} \end{pmatrix}$ |
| 2 | $\begin{pmatrix} 5.272175213\mathbf{3}^{60093}_{58749} \cdot 10^{-2} & 1.10097944032\mathbf{4}^{998}_{860} \cdot 10^{-1} \\ 2.1680433205\mathbf{2}^{2446}_{1391} \cdot 10^{-2} & 2.4930642689\mathbf{8}^{6836}_{5710} \cdot 10^{-2} \end{pmatrix}$ |
| 3 | $\begin{pmatrix} 3.6616448427\mathbf{8}^{7013}_{6322} \cdot 10^{-2} & 9.546942130\mathbf{7}^{10274}_{09563} \cdot 10^{-2} \\ 1.6949917794\mathbf{1}^{7420}_{6607} \cdot 10^{-2} & 2.0659404505\mathbf{0}^{6816}_{5942} \cdot 10^{-2} \end{pmatrix}$ |
| 4 | $\begin{pmatrix} 2.45732954721\mathbf{3}^{804}_{599} \cdot 10^{-2} & 8.44212632298\mathbf{4}^{798}_{581} \cdot 10^{-2} \\ 6.5030436188\mathbf{94}^{266}_{016} \cdot 10^{-2} & 4.6367612311\mathbf{2}^{8145}_{7877} \cdot 10^{-2} \end{pmatrix}$ |
| 5 | $\begin{pmatrix} 1.000000000000001 \\ 9.999999999999996 \end{pmatrix} \cdot 10^{-2} \quad \begin{pmatrix} 4.000000000000001 \\ 3.999999999999998 \end{pmatrix} \cdot 10^{-2}$ <br> $\begin{pmatrix} 3.000000000000001 \\ 2.999999999999999 \end{pmatrix} \cdot 10^{-2} \quad \begin{pmatrix} 2.000000000000001 \\ 1.999999999999999 \end{pmatrix} \cdot 10^{-2}$ |

### 7.2 Example 2

Now we consider a SMP/GI/1 system given by

$$\begin{aligned} &\Pr(A_1 = 0) = 0.01, \quad \Pr(A_1 = k) = 0.02 \quad \text{for } k = 1, \ldots, 49, \\ &\Pr(A_1 = 50) = 0.01, \end{aligned}$$

$$\begin{aligned} &\Pr(A_2 = 0) = 0.02, \quad \Pr(A_2 = k) = 0.04 \quad \text{for } k = 1, \ldots, 24, \\ &\Pr(A_2 = 25) = 0.02, \ \Pr(A_2 = k) = 0 \quad\ \text{for } k = 26, \ldots, 50, \end{aligned}$$

$$\begin{aligned} &\Pr(A_3 = 0) = 0.02, \quad \Pr(A_3 = k) = 0 \quad\ \text{for } k = 1, \ldots, 25, \\ &\Pr(A_3 = 50) = 0.02, \ \Pr(A_3 = k) = 0.04 \quad \text{for } k = 26, \ldots, 49, \end{aligned}$$

and

$$\boldsymbol{P} = \begin{pmatrix} 0.2 & 0.6 & 0.2 \\ 0.6 & 0.3 & 0.1 \\ 0.4 & 0.3 & 0.3 \end{pmatrix}.$$

For the service time distribution we have

$$S(z) = 0.05z + 0.3z^2 + 0.6z^3 + 0.05z^8.$$

**Table 3.** Distributions $\boldsymbol{l}(n)$ of Example 1

| $n$ | $\boldsymbol{l}(n)$ |
|---|---|
| 1 | $\begin{pmatrix} 1.097515381114^{9275}_{8908} \cdot 10^{-1} & 1.662792278542^{795}_{389} \cdot 10^{-1} \\ 8.689450227^{70382}_{68167} \cdot 10^{-2} & 6.896177378^{61257}_{59077} \cdot 10^{-2} \end{pmatrix}$ |
| 2 | $\begin{pmatrix} 9.9670516118^{4278}_{1173} \cdot 10^{-2} & 1.54712986842^{470}_{129} \cdot 10^{-1} \\ 1.396760660^{7167}_{6932} \cdot 10^{-1} & 1.00360569610^{726}_{515} \cdot 10^{-1} \end{pmatrix}$ |
| 3 | $\begin{pmatrix} 7.9347891258^{8805}_{6555} \cdot 10^{-2} & 1.35893758180^{439}_{184} \cdot 10^{-1} \\ 1.34323271420^{515}_{322} \cdot 10^{-1} & 9.4808425998^{9419}_{7757} \cdot 10^{-2} \end{pmatrix}$ |
| 4 | $\begin{pmatrix} 5.8132700802^{1451}_{0070} \cdot 10^{-2} & 1.15697445571^{714}_{548} \cdot 10^{-1} \\ 1.288747873^{451}_{299} \cdot 10^{-1} & 8.90650115292^{5448}_{4220} \cdot 10^{-2} \end{pmatrix}$ |
| 5 | $\begin{pmatrix} 2.619023488^{5392}_{4873} \cdot 10^{-2} & 5.43237003762^{910}_{279} \cdot 10^{-2} \\ 9.35965632349^{980}_{070} \cdot 10^{-2} & 6.34390287446^{974}_{298} \cdot 10^{-2} \end{pmatrix}$ |
| 6 | $\begin{pmatrix} 0.000000000000000 & 0.000000000000000 \\ 0.000000000000000 & 0.000000000000000 \end{pmatrix}$ |
| 7 | $\begin{pmatrix} 0.000000000000000 & 0.000000000000000 \\ 0.000000000000000 & 0.000000000000000 \end{pmatrix}$ |
| 8 | $\begin{pmatrix} 0.000000000000000 & 0.000000000000000 \\ 0.000000000000000 & 0.000000000000000 \end{pmatrix}$ |
| 9 | $\begin{pmatrix} 0.000000000000000 & 0.000000000000000 \\ 0.000000000000000 & 0.000000000000000 \end{pmatrix}$ |
| 10 | $\begin{pmatrix} 0.000000000000000 & 0.000000000000000 \\ 0.000000000000000 & 0.000000000000000 \end{pmatrix}$ |

**Table 4.** Some results of Example 1

| $k$ | $w(k)$ | Diameter |
|---|---|---|
| 0 | $4.82868606766^{8663}_{2947} \cdot 10^{-1}$ | $5.714873019257993 \cdot 10^{-13}$ |
| 1 | $6.75016397776^{89639}_{79007} \cdot 10^{-2}$ | $1.063038546078587 \cdot 10^{-13}$ |
| 5 | $5.81363798841^{24647}_{15303} \cdot 10^{-2}$ | $9.342526752220692 \cdot 10^{-14}$ |
| 10 | $1.56100699948^{20398}_{17133} \cdot 10^{-2}$ | $3.263882220050363 \cdot 10^{-14}$ |
| 25 | $5.16768434513^{32693}_{15748} \cdot 10^{-4}$ | $1.694391155160346 \cdot 10^{-15}$ |
| 50 | $1.71716324^{505335}_{496267} \cdot 10^{-6}$ | $9.067275942120472 \cdot 10^{-18}$ |
| 100 | $1.895745195^{408086}_{390485} \cdot 10^{-11}$ | $1.760052935405311 \cdot 10^{-22}$ |

Some results are listed in Table 5. We have chosen some $10^{-n}$-quantiles for the value $k$, i.e. $k = q^{-n} := \min_x(\Pr(W > x) < 10^{-n})$. Since the values of $w(k)$ decrease very fast, we can not make any statements about higher order quantiles.

**Table 5.** Some results of Example 2

| $k$ | $w(k)$ | Quantile |
|---|---|---|
| 0 | $9.342000807697^{8476}_{7654} \cdot 10^{-1}$ | |
| 1 | $2.449675933498^{798}_{612} \cdot 10^{-2}$ | $\Pr(W > 1) < 0.1$ |
| 3 | $1.166858324238^{741}_{634} \cdot 10^{-2}$ | $\Pr(W > 3) < 0.01$ |
| 10 | $7.883270671474^{403}_{3722} \cdot 10^{-5}$ | $\Pr(W > 10) < 10^{-4}$ |
| 16 | $1.405803830270^{678}_{548} \cdot 10^{-6}$ | $\Pr(W > 16) < 10^{-6}$ |
| 26 | $6.106570806838^{9264}_{8691} \cdot 10^{-10}$ | $\Pr(W > 26) < 10^{-9}$ |
| 50 | $1.146819437078^{368}_{246} \cdot 10^{-17}$ | |

## 8 Conclusions

Performance analysis for ensuring the achievable quality of service of components and end-to-end delivery in telecommunication networks has been considered from the general framework of service level agreements to the detailed analysis of switching systems using verified numerical methods.

It is shown that Wiener-Hopf factorization as an approach with favorable computation time properties often yields verified results enclosed within narrow intervals. Presently we study alternative approaches like factorization by root-finding methods in ongoing work.

**Acknowledgements.** This research and software development for algorithms with result verification has been carried out by the authors in a recent project funded by the German Research Council (DFG).

## References

1. Boudec, J.-Y. Le: Steady-State Probabilities of the PH/PH/1 Queue. Queueing Systems **3** (1988) 73–87
2. Bruneel, H. and B. Kim: Discrete-Time Models for Communication Systems Including ATM. Kluwer Acad. Publ. (1993)
3. Elwalid, A.I. and D. Mitra: Effective bandwidth of general Markovian traffic sources and admission control of high speed networks. IEEE/ACM Transactions on Networking **1** (1993) 329–343
4. Fausten, D., W. Luther, and G. Haßlinger: Verified Computing of the Stationary Workload Distribution of a GI/GI/1 Server. In: Mastorakis, N. and Antoniou, G., (eds.): Recent Advances in Circuits, Systems and Signal Processing. WSEAS Press (2002) 169–174
5. Fausten, D., W. Luther, and G. Haßlinger: Accurate computation of traffic workload distributions. Accepted paper of Proceedings of SCAN 2002, Special issue of Numerical Algorithms (2002)
6. Feldmann, A. and W. Whitt: Fitting mixtures of exponentials to long-tail distributions. Performance Evaluation **31** (1997) 245–279

7. Grassmann, W.K. and J.L. Jain: Numerical solutions of the waiting time distribution and idle time distribution of the arithmetic GI/G/1 queue. Operations Research **37** (1989) 141–150
8. Hammer, R., M. Hocks, U. Kulisch, and D. Ratz: C++ Toolbox for Verified Computing. Springer (1995)
9. Haßlinger, G.: Waiting times, busy periods and output models of a server analyzed via Wiener-Hopf factorization. Perf. Eval. **40** (2000) 3–26
10. Haßlinger, G.: Quality-of-Service analysis for statistical multiplexing with Gaussian distributed & autoregressive input modelling. Telecom. Systems **16** (2001) 315–334
11. Haßlinger, G. and D. Fausten: Analysis of the Workload in Communication Systems Including Data Transfers over Arbitrary Time Scales. I. J. of Simulation Vol. **3**, No. **3-4** (2002) 25–37
12. Hofschuster, W., Krämer, W., Wedner, S., and Wiethoff, A.: C-XSC 2.0 - A C++ Class Library for Extended Scientific Computing. Preprint 2001/1, Universität Wuppertal (2001)
13. Jansen, J. and N. Limnios, (Eds.): Proceedings of the 2nd Internat. Symposium on Semi-Markov Models. Compiegne, France (1998)
14. Klatte, R., U. Kulisch, A. Wiethoff, C. Lawo, and M. Rauch: C-XSC. Springer (1993)
15. Kleinrock, L.: Queueing systems. Vol. 1/2. Wiley (1975/6)
16. Latouche, G. and V. Ramaswami: The PH/PH/1 queue at epochs of queue size change. Queueing Systems **25** (1997) 97–114
17. Li, S.-Q.: A general solution technique for discrete queueing analysis of multimedia traffic on ATM. IEEE Trans. on Commun. **COM-39** (1991) 1115–1132
18. Lindley, D.V.: The theory of queues with a single server. Proc. Cambridge Philos. Soc. **48** (1952) 277–289
19. Neuts, M.F.: Structured Stochastic Matrices of M/G/1 Type and their Applications. M. Dekker (1989)
20. Paxson, V. and S. Floyd: Wide area traffic: The failure of Poisson modelling. IEEE/ACM Transactions on Networking **3** (1995) 226–244.
21. Rump, S.M.: Solving Algebraic Problems with High Accuracy. In: Kulisch, U. and W.L. Miranker (eds.): A New Approach to Scientific Computation. Academic Press (1983) 53–120
22. Sengupta, B.: The semi-Markovian queue: Theory and applications. Commun. Statist.-Stochastic Models **6** (1990) 383–413
23. http://www.math.uni-wuppertal.de/wrswt/xsc-sprachen.html

# Result Verification for Computational Problems in Geodesy

Stefan Borovac and Gerhard Heindl

Universität Wuppertal
Fachbereich Mathematik und Naturwissenschaften
Institut für Angewandte Informatik
42097 Wuppertal, Germany,
{Stefan.Borovac, Gerhard.Heindl}@math.uni-wuppertal.de

**Abstract.** The subject of the paper is to present verified methods to solve three important geodetic problems: The Direct and Inverse Problem of geodetic surveying and the three-dimensional resection problem, often also designated as the main problem of Photogrammetry.
The Direct Problem reads as follows: Given a point $p$ on a rotational ellipsoid $E$, a direction tangential to $E$ and a distance, determine the point $q$ which is reached after "walking" the given distance along a geodesic on $E$, starting at $p$ in the given direction, and compute the direction of the geodesic in $q$.
In the Inverse Problem there are given two points $p, q$ on $E$, and one has to compute the length of a geodesic on $E$ connecting $p$ and $q$ as well as its directions in $p$ and $q$.
It is shown that the Direct Problem can be treated as an ordinary initial value problem with verified solvers like AWA.
The inverse problem is much more complicated because no characteristic quantity of the geodesic is known. In fact it is an ordinary open boundary value problem and the question whether it is uniquely sovable must be treated. It is shown how to compute an enclosure of the direction of the geodesic in one point. By the aid of this enclosure the problem can be solved by a verified shooting technique using again solvers like AWA.
The three-dimensional resection problem can be reduced to the problem to find the set $S$ of all real positive $s_1, s_2, s_3$ satisfying the equations

$$\begin{aligned} s_2^2 + s_3^2 - 2s_2 s_3 c_1 &= a_1^2 \\ s_3^2 + s_1^2 - 2s_3 s_1 c_2 &= a_2^2 \\ s_1^2 + s_2^2 - 2s_1 s_2 c_3 &= a_3^2 \end{aligned}$$

where $c_i, a_i,\ i = 1, 2, 3$, are real parameters restricted by the inequalities
$-1 < c_i < 1,\ i = 1, 2, 3,\ 1 - c_1^2 - c_2^2 - c_3^2 + 2c_1c_2c_3 \geq 0,$
$a_i > 0,\ i = 1, 2, 3,\ a_1 + a_2 > a_3, a_2 + a_3 > a_1, a_3 + a_1 > a_2.$
There are discussed three approaches for constructing algorithms for computing narrow inclusions of $S$. In the first one the fact is used that an initial box containing $S$ can be easily derived. Consequently standard modules like *GlobSol* or `nlss` can be applied to achieve the goal.

R. Alt et al. (Eds.): Num. Software with Result Verification, LNCS 2991, pp. 226–242, 2004.

The idea of the second approach is to construct verifying versions of algorithms for computing $S$ wich are used in practice. However, it turns out that there are instabilities in the resulting methods which are not present in the methods resulting from the first approach.
The aim of the third approach is to derive sufficiently narrow inclusions of $S$ from sufficiently narrow inclusions of all zeros of at most four simple functions of one variable. At the moment experiences with several practical and artificial examples indicate that the third approach is best.

**Keywords:** Direct and Inverse Problem of geodetic surveying, three-dimensional resection problem, automatic result verification

## 1 Introduction

Geodesists naturally must be highly interested in the reliability of their computations. But although it can be assumed that most of their results are precise enough, their traditional techniques cannot give any certainty. Measuring errors mostly are treated statistically only. Errors introduced by linearisation or other approximations and rounding errors are considered, if at all, only qualitatively.

Therefore, making use of the tools for verified computing, can improve the quality of geodetical computations considerably.

In the following we will demonstrate this for some frequently appearing problems: The two main problems of Geodesy concerned with geodesics on a rotational ellipsoid (section 2), and the three-dimensional resection problem, also designated as the main problem of Photogrammetry (section 3).

## 2 Verified Solution of the Direct and the Inverse Problem on the Rotational Ellipsoid (by Stefan Borovac)

### 2.1 Overview

In section 2.2 the objects of interest will be defined. The problems are described in section 2.3 and a solution for the Direct Problem will be derived. In section 2.4 the solver AWA will be introduced and in section 2.5 two examples for the Direct Problem are given. Section 2.6 describes a way to the solution of the Inverse Problem. In the last section two examples for the Inverse Problem are presented.

Note that this paper is based on [6], where it was tried to give the objects and methods a strictly mathematical shape.

### 2.2 Preliminaries

All objects that will be described below can be found in [1] or [2].

Let $\mathbb{R}^3$ denote the 3-dimensional euclidian space with the standard norm $\|\cdot\|_2$ and inner product $\langle\cdot,\cdot\rangle$. For a function $f(x,y)$ we denote by $f_x(x,y)$ the partial derivative of $f$ with respect to $x$. The analogue is valid for $f_y(x,y)$.

**The Rotational Ellipsoid $E$**
The rotational ellipsoid $E \subset \mathbb{R}^3$ is given here as

$$E := \left\{ (x,y,z)^T \in \mathbb{R}^3 : \frac{x^2}{a^2} + \frac{y^2}{a^2} + \frac{z^2}{b^2} = 1 \, , a,b \in \mathbb{R} \, , a > b > 0 \right\}.$$

Note that our ellipsoid is flattened at the z-axis. A well known parametrisation [12], which is suitable for geodetic purposes, is given by

$$\Psi : \mathbb{R} \times \mathbb{R} \to \mathbb{R}^3$$

$$\Psi(\phi,\lambda) := \frac{a}{\sqrt{1 - e^2 \sin^2(\phi)}} \big(\cos(\phi)\cos(\lambda), \cos(\phi)\sin(\lambda), (1-e^2)\sin(\phi)\big)^T \quad (1)$$

with

$$e^2 := \frac{a^2 - b^2}{a^2}. \quad (2)$$

$\phi$ is called the *(geographic) latitude* and $\lambda$ the *(geographic) longitude*. We restrict $\Psi$ to $D := \left]-\frac{\pi}{2}, \frac{\pi}{2}\right[ \times [0, 2\pi[$, and $\Psi$ maps $D$ bijectively onto $E \setminus \{(0,0,-b)^T, (0,0,b)^T\}$. $b$ can obviously be described in terms of $e$ and $a$. Because $a$ is in all quantities of interest only a multiplicative constant, we assume $a = 1$. The number $e$ is also known as the *first numerical excentricity*.

**Tangent and Normal Vectors**
Let $p = \Psi(\phi,\lambda) \in E$. By $T_p(E) \subset \mathbb{R}^2$ we denote the set of all tangential vectors of $E$ in $p$. $T_p(E)$ is spanned by $\Psi_\phi$ and $\Psi_\lambda$ which are orthogonal vectors. A normal field for $E$ is given by

$$N(p) = -\left(\cos\phi\cos\lambda, \cos\phi\sin\lambda, \sin\phi\right)^T. \quad (3)$$

$E$ is therefore an *orientable surface*.

**Curves in $E$**
A curve in $E$ is given by a function $\alpha : [c,d] \to E$. We assume $[c,d] \subset \mathbb{R}$, $\alpha$ sufficiently often differentiable and $\alpha'(t) = (\alpha_1'(t), \alpha_2'(t), \alpha_3'(t))^T \neq (0,0,0)\ \forall t$. That means, that $\alpha$ is a regular curve. Note that $\alpha'(t) \in T_{\alpha(t)}(E)$.

An alternative form of $\alpha$ in terms of the parametrisation $\Psi$ of $E$ is given by

$$\alpha(t) = \Psi(\phi(t), \lambda(t)) \quad (4)$$

with differentiable parameters $(\phi(t), \lambda(t)) = \Psi^{-1} \circ \alpha(t)$. $N(\alpha(t))$ is a normal to $E$ in $\alpha(t)$.

**Geodesics**

Now our main subject will be defined.

**Definition 1.** *Let $S$ be a regular orientable surface and $\alpha$ be a regular curve in $S$ parametrised by its arc-length (i.e. $\|\alpha'(t)\|_2 = 1\ \forall t$). We say that $\alpha$ is a geodesic iff*

$$\alpha''(t) = (\alpha_1''(t), \alpha_2''(t), \alpha_3''(t))^T = \gamma(t) \cdot N(\alpha(t)),\ \forall t \text{ with } \gamma(t) \in \mathbb{R} \setminus \{0\}.$$

The above definition says, that a regular curve is a geodesic if and only if the normal vector of the curve, represented by $\alpha''(t)$, is a scalar multiple of the normal vector of the surface in $\alpha(t)$.

It is a well known fact that if a regular curve $\alpha$ is the shortest connection between two points of a regular surface, then $\alpha$ is a geodesic. The latter does not mean that there is necessarily only one geodesic connecting these points. In the case of the rotational ellipsoid we have two more properties [2]. In fact

- $E$ is *geodetical complete*. That means that every two points on $E$ can be connected by a geodesic.
- $\alpha$ is a shortest geodesic connecting two points if

$$s < \pi\sqrt{1 - e^2}. \tag{5}$$

Here $s$ stands for the arc length of $\alpha$ and $e$ is given by (2).

**Coordinate Curves and the Azimuth**

A special class of curves in $E$ are the coordinate curves. A coordinate curve given by $\Psi(\phi, \lambda_0)$ with $\lambda_0$ fixed is called a *meridian*. $\Psi(\phi_0, \lambda)$ is called a *parallel circle* or *parallel*.

Every curve $\alpha$ in $E$ intersects in each point $p = \Psi(\phi, \lambda) = \alpha(t)$ a meridian and a parallel. The tangent vector of a meridian is given by $\Psi_\phi(\phi, \lambda)$ and the tangent vector of a parallel by $\Psi_\lambda(\phi, \lambda)$. As already mentioned, we know that $B = \{\Psi_\phi(\phi, \lambda), \Psi_\lambda(\phi, \lambda)\}$ is an orthogonal basis for $T_p(E)$. So we can find coordinates for the normalised tangent vector of $\alpha$ in $p$ with respect to $B$.

This situation makes it possible to define a direction of a curve $\alpha$ in $p$ by the unique number $A \in ]-\pi, \pi]$ satisfying

$$\cos(A) = \frac{\langle \Psi_\phi(\phi, \lambda), \alpha'(t) \rangle}{\|\Psi_\phi(\phi, \lambda)\|_2 \|\alpha(t)\|_2}, \qquad \sin(A) = \frac{\langle \Psi_\lambda(\phi, \lambda), \alpha'(t) \rangle}{\|\Psi_\lambda(\phi, \lambda)\|_2 \|\alpha(t)\|_2}. \tag{6}$$

The angle $A$ is also known as the *Azimuth* ([13], [12]). By $A(t)$ we denote the set of all azimuths along a curve $\alpha = \alpha(t)$. Note that $A(t)$ depends differentiably on $t$.

**Differential Equations for a Geodesic in $E$**

We would like to describe a geodesic $\alpha$ in $E$ by a system of ordinary differential equations. By (4), (6) and the fact that $\langle \Psi_\phi, \Psi_\lambda \rangle = 0$ holds, we get

$$\cos(A) = \frac{\langle \Psi_\phi, \alpha' \rangle}{\|\Psi_\phi\|_2} = \frac{\langle \Psi_\phi, \phi' \Psi_\phi + \lambda' \Psi_\lambda \rangle}{\|\Psi_\phi\|_2} = \frac{\phi' \|\Psi_\phi\|_2^2}{\|\Psi_\phi\|_2} = \phi' \|\Psi_\phi\|_2.$$

An analogous argument leads to

$$\sin(A) = \lambda' \|\Psi_\lambda\|_2.$$

Due to the dependencies of $\phi'$ and $\lambda'$ on $A$, we need an equation for $A'$ in terms of $\phi$ and $\lambda$. This equation is not so easy to derive and we only state the result:

$$A' = -\frac{(\|\Psi_\lambda\|_2)_\phi \sin(A)}{\|\Psi_\phi\|_2 \|\Psi_\lambda\|_2}.$$

Note that the last equation is only valid for geodesics, while the others are valid for arbitrary curves in $E$.

Now calculating explicit expressions for $\|\Psi_\phi\|_2$ and $\|\Psi_\lambda\|_2$ and using

$$M(\phi) := \frac{(1-e^2)}{\left(\sqrt{1-e^2\sin^2(\phi)}\right)^3}, \tag{7}$$

$$p(\phi) := \frac{\cos(\phi)}{\sqrt{1-e^2\sin^2(\phi)}}, \tag{8}$$

leads to the following system depending on the curve parameter $t$:

$$\begin{aligned} \frac{d\phi(t)}{dt} &= \frac{\cos(A(t))}{M(\phi(t))} \\ \frac{d\lambda(t)}{dt} &= \frac{\sin(A(t))}{p(\phi(t))} \\ \frac{dA(t)}{dt} &= \frac{\sin(A(t))\sin(\phi(t))}{p(\phi(t))}. \end{aligned} \tag{9}$$

Note that $t$ represents the length of the curve.

By the aid of (9) and the so called Relation of Clairaut (see (11) in section 2.6) it is easy to derive the behavior of a geodesic in $E$. If e.g. the azimuth is in one point different from 0 and $\pi$, so it is in every point and the longitudes strictly increase (decrease). Hence $t$ can be written as a function of $\lambda$ and we get the following representation of a geodesic by a system depending on the longitudes:

$$\begin{aligned} \frac{dt(\lambda_0)}{d\lambda} &= \frac{p(\phi(\lambda_0))}{\sin(A(\lambda_0))} \\ \frac{d\phi(\lambda_0)}{d\lambda} &= \frac{\cos(A(\lambda_0))p(\phi(\lambda_0))}{\sin(A(\lambda_0))M(\phi(\lambda_0))} \\ \frac{dA(\lambda_0)}{d\lambda} &= \sin(\phi(\lambda_0)). \end{aligned} \tag{10}$$

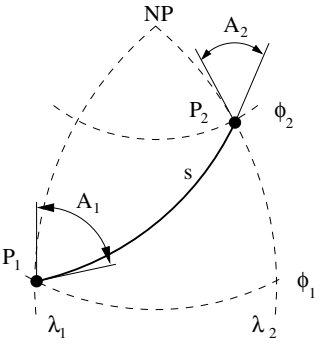
### 2.3 The Problems

Now we are able to state the problems (note that the picture below describes both situations).

**Direct Problem**
*Given:* A point $P_1 = \Psi(\phi_1, \lambda_1)$ in $E$, an azimuth $A_1$ and a distance $s$.

*Sought:* The point $P_2 = \Psi(\phi_2, \lambda_2)$ to be reached after "walking" the distance along a geodesic starting from $P_1$ in the given direction $A_1$, and the direction of the geodesic in $P_2$.

It is obvious that the Direct Problem is an initial value problem if we use (9) since all initial values are given. So we have to look for a tool which can calculate guaranteed enclosures of the solution of such a problem. The tool of choice will be introduced in the next section. Note that we assume $A_1 \notin \{0, \pi\}$, since these cases are trivial. Now we state the other problem.




**Inverse Problem**
*Given:* Two points $P_1 = \Psi(\phi_1, \lambda_1)$ and $P_2 = \Psi(\phi_2, \lambda_2)$ in $E$.

*Sought:* The length $s$ of a geodesic connecting these points as well as the directions $A_1$ and $A_2$ the geodesic will have in $P_1$ and $P_2$.

A closer look reveals that using (10), the Inverse Problem is an ordinary open boundary value problem. Hence it can be solved iteratively by a shooting method [5]. But it is not obvious that a shooting method can be used to compute verified solutions. The appropriate assumptions will be made in section 2.6. Note also that we assume $\lambda_1 \neq \lambda_2$ since the other case is trivial again.

### 2.4 The Solver AWA

To solve the both problems we use the program AWA due to Rudolf Lohner [7]. It is a program to compute enclosures of solutions of systems of ordinary differential equations (ODEs). The method used by AWA is based on Taylor-Series-Expansions of the right hand side, and makes excessive use of automatic differentiation (see [7] and the references therein).

It should be mentioned here, that one can explicitly fix the order of the Taylor-Series-Expansion and choose between five different possibilities to enclose the global error, so AWA is a good tool for experiments. One can also determine the bounds of the absolute and relative error which can be violated due to overestimations. However, these bounds play an important role in determining an appropriate stepsize.

The program is written in PASCAL-XSC [4]. A C-version is unfortunately not available. One might find references to other (newer) solvers in [8,9,10,11], but these solvers were not tested with the problems above. However, the examples in this paper show that AWA still is a good and stable solver.

### 2.5 Numerical Examples for the Direct Problem

As already mentioned one can treat the Direct Problem as an initial value problem. Now two examples taken from [14] will be presented. The verified results were originally presented in [6]

To gain a sufficient precision the diameter $d$ of the enclosure of each component of the solution must satisfy $d \leq 10^{-8}$. In terms of the latitude or longitude this bound represents the distance of nearly 1 mm (or 0.0001 arc seconds) on the earth surface. But AWA did not compute the best possible error bounds and we will loose some digits due to overestimations and by transformations. Therefore we set the error bounds to $10^{-16}$, hence keep the stepsize small and hopefully get the desired precision. Furthermore the examples are calculated with an order of 20. This is simple because one gets the best results with that order. The ellipsoid for which the results are calculated is the Hayford ellipsoid defined by $a = 6378.388$ km and $e^2 = 0,00672267$.

Note that $s_E$ stands for the distance on the normalized ellipsoid (i.e. $a = 1$) on which all calculation have been done. Furthermore one can assume $\lambda_1 = 0$, because all the problems are invariant under rotations around the z-axis.

**Example 1**
*Given:* $\phi_1 = 40^0$, $A_1 = 10^0$, $s = 1000$ km ($s_E = 0.1567794245191732$)

| *Results from [14]:* | *Results by AWA:* |
|---|---|
| $\phi_2 = 48^0\ 50'\ 25''.1215864$ | $\phi_2 = 48^0\ 50'\ 25''.12158616^{249}_{130}$ |
| $\lambda_2 = 2^0\ 21'\ 23''.3180857$ | $\lambda_2 = 2^0\ 21'\ 23''.318085^{7045}_{6935}$ |
| $A_2 = 11^0\ 39'\ 15''.7789103$ | $A_2 = 11^0\ 39'\ 15''.778910^{305}_{267}$ |

**Example 2**
*Given:* $\phi_1 = 50^0$, $A_1 = 140^0$, $s = 15000$ km ($s_E = 2.351691367787597$)

| *Results from [14]:* | *Results by AWA:* |
|---|---|
| $\phi_2 = -62^0\ 57'\ 03''.2038671$ | $\phi_2 = -62^0\ 57'\ 03''.203864^{363}_{509}$ |
| $\lambda_2 = 95^0\ 5'\ 38''.2996643$ | $\lambda_2 = 95^0\ 5'\ 38''.2996566^{997}_{745}$ |
| $A_2 = 114^0\ 46'\ 41''.48439034$ | $A_2 = 114^0\ 46'\ 41''.4839127^{37}_{01}$ |

The examples show that the results from [14] have the desired precision. But in the second example the last digits are almost all wrong. This probably gives the illusion of having a better precision than it is. Furthermore the examples show that it is possible to compute enclosures within reliable bounds.

### 2.6 Solution of the Inverse Problem

The key to the solution of the Inverse Problem is to construct a *corresponding* geodesic on the unit sphere $S$ (see [12], [15]). Once one has constructed this geodesic, some quantities can be calculated by spherical trigonometry. The latter is due to the fact that a geodesic on a sphere is a part of a great circle.

A parametrisation $\Sigma$ of $S$ can be obtained from (1) by setting $e^2 = 0$.

One tool which is essential in this situation is the relation of Clairaut [1]. If we assume $\alpha$ to be a geodesic on $E$ which has in arbitrary points $\Psi(\phi_1, \lambda_1), \Psi(\phi_2, \lambda_2)$ the azimuths $A_1$ and $A_2$ the relation reads as follows:

$$p(\phi_1) \cdot \sin(A_1) = p(\phi_2) \cdot \sin(A_2) = const \tag{11}$$

Here $p(\phi)$ is given by (8) and is the radius of the parallel through points with the latitude $\phi$.

Now suppose $p_1 = \Psi(\phi_1, \lambda_1)$ is the starting point of a geodesic $\alpha$ in $E$. One can assign a point $s_1 = \Sigma(\beta_1, \lambda_1) \in S$ in a manner that the radius of the parallel through $s_1$ is the same as in $p_1$ by choosing $\beta \in ]-\frac{\pi}{2}, \frac{\pi}{2}[$ such that

$$\tan(\beta_1) = \frac{b}{a} \tan(\phi_1).$$

$\beta_1$ is called the *reduced latitude* of $\phi_1$ and is unique. But as the reduced latitude can be assigned to an arbitrary point of $\alpha$ one gets from (11)

$$\cos(\beta_1) \sin(A_1) = \cos(\beta_2) \sin(A_2).$$

This is the Relation of Clairaut for a geodesic in $S$, hence a great circle. So we can find a geodesic in $S$ starting at the point $\Sigma(\beta_1, \lambda_1)$ which has in points $\Sigma(\beta, \lambda_S)$ the same azimut $A$ as a point $\Psi(\phi, \lambda)$ of $\alpha$, if $\beta$ is the reduced latitude of $\phi$.

The only drawback is that the longitudes are not the same. But one can find a relation between the longitudes [12]. In the situation of the Inverse Problem one has given two points $P_1 = \Psi(\phi_1, 0)$ and $P_2 = \Psi(\phi_2, \lambda_2)$, and gets for the longitude $\lambda_{S_2}$ of a point with reduced latitude $\beta_2$ the following relation

$$\lambda_2 \leq \lambda_{S_2} \leq \frac{\lambda_2}{\sqrt{1-e^2}}. \tag{12}$$

The enclosure (12) makes it possible to compute an enclosure of the azimuth $A_1$ at $P_1$ by the aid of spherical trigonometry. In fact with $I = [\lambda_2, \frac{\lambda_2}{\sqrt{1-e^2}}]$ one gets

$$A_1 \in I_A := \operatorname{arccot}\left(\frac{\tan(\beta_1)\cos(\beta_2) - \sin(\beta_2)\cos(I)}{\sin(I)}\right). \tag{13}$$

The above construction makes it also possible to decide whether there is only one geodesic connecting $P_1$ and $P_2$ or not. If $s$ is the length of the geodesic in $E$ and $\sigma$ the length of the corresponding geodesic in $S$ there is the following inclusion [12].

$$s \in [\sigma\sqrt{1-e^2}, \sigma]$$

Therefore one gets with (5), that the geodesic is the shortest if $\sigma < \pi\sqrt{1-e^2}$. But this is also computable by spherical trigonometry.

It is now obvious how to solve the Inverse Problem. If one takes the system (10) and the initial enclosure (13), verified enclosures of the latitude at $\lambda_2$ can

be computed at the boundaries of (13) and the midpoint by using AWA. If the computed geodesics do not intersect, we can decide by the true value of $\phi_2$, which half of (13) to skip. With the other half we proceed further on until the desired precision is reached.

The final assumption (see [6]) that two geodesics starting in $P_1 = \Psi(\phi_1, 0)$ cannot intersect before reaching the meridian in $\lambda_2$ is

$$\lambda_2 < \pi\sqrt{1 - e^2}.$$

### 2.7 Numerical Examples for the Inverse Problem

As in section 2.5 two examples are given for the Inverse Problem. The first one is the inverse of example 2 in section 2.5. The second is again from [14] and also calculated on the Hayford ellipsoid. The verified results are again from [6].

The desired precision, the error bounds and the order are the same as in the foregoing examples. Note that $A_1.\sup$ denotes the computed upper bound and $A_1.\inf$ the computed lower bound of the enclosure. The same is valid for $s.\sup$ and $s.\inf$. Again we assume $\lambda_1 = 0$.

**Example 3**
*Given:* $\phi_1 = 49^0$, $\phi_2 = -62^0 57' 03''.203864^{509}_{363}$, $\lambda_2 = 95^0 5' 38''.2996566^{997}_{745}$.

*Initial enclosure for* $A_1$*:* $[139^0\ 56'\ 58''.14779207, 140^0\ 3'\ 55''.95605677]$

| *Values given by example 2:* | *Results by AWA:* |
|---|---|
| $A_1 = 140^0$ | $A_1.\sup = 140^0\ \ 0'\ 00''.000000036$ |
| | $A_1.\inf = 139^0\ 59'\ 59''.999999999$ |
| $A_2 = 114^0\ 46'\ 41''.4839127^{37}_{01}$ | $A_2 = 114^0\ 46'\ 41''.4839128^{8}_{6}$ |
| $s = 15000$ km | $s.\sup = 15000.00000001$ km |
| | $s.\inf = 14999.99999999$ km |

**Example 4**
*Given:* $\phi_1 = 65^0$, $\phi_2 = 70^0 1' 22''.722249$, $\lambda_2 = 169^0 38' 51''.251406$

*Initial enclosure for* $A_1$*:* $[4^0\ 43'\ 39''.16197007, 5^0\ 0'\ 9''.68708845]$

| *Results from [14]:* | *Results by AWA:* |
|---|---|
| $A_1 = \ \ 5^0$ | $A_1 = \ \ 5^0\ \ 0'\ 00''.0000000^{3}_{1}$ |
| $A_2 = 173^0\ 48'\ 43''.3289378$ | $A_2 = 173^0\ 48'\ 43''.3289377^{78}_{44}$ |
| $s = 4999.99999999$ km | $s = 4999.9999999^{47}_{41}$ km |

Example 3 confirms example 2 and shows together with example 4 that the applied iterative method delivers reliable results. The iterations always take 34 to 37 steps because the diameter of the initial enclosure is always nearly the same and is only halved in each iteration.

# 3 On Methods for Computing Narrow Inclusions of All Solutions of the Three-Dimensional Resection Problem (by Gerhard Heindl)

## 3.1 The Problem and Its Reduced Version

The three-dimensional resection problem, often designated as the Main Problem of Photogrammetry, can be stated as follows:

(**P**): Given the coordinate vectors $x_1, x_2, x_3$ of three affinely independent points $P_1, P_2, P_3$ in 3-space with respect to a Cartesian coordinate system, and for a point $P \notin \{P_1, P_2, P_3\}$ the measured angles

$$\begin{aligned}\alpha_1 &= \angle(P_3, P, P_2) \in ]0, \pi[,\\ \alpha_2 &= \angle(P_1, P, P_3) \in ]0, \pi[,\\ \alpha_3 &= \angle(P_2, P, P_1) \in ]0, \pi[,\end{aligned}$$

what can be said about the coordinate vector $x$ of $P$?

We will show first that this question can be answered satisfactorily if we can solve the following reduced problem:

(**P**$_{\mathbf{red}}$): Given $c_i \in ]-1, 1[$, $a_i \in ]0, \infty[$, $i = 1, 2, 3$, such that

$$w := 1 - c_1^2 - c_2^2 - c_3^2 + 2c_1c_2c_3 \geq 0, \tag{14}$$

$$a_1 + a_2 > a_3,\ a_2 + a_3 > a_1,\ a_3 + a_1 > a_2, \tag{15}$$

and let

$$f : \mathbb{R}^3 \ni s = (s_1, s_2, s_3)^T \mapsto (f_1(s), f_2(s), f_3(s))^T \in \mathbb{R}^3$$

be the quadratic mapping defined by

$$\begin{aligned}f_1(s) &:= s_2^2 + s_3^2 - 2s_2s_3c_1 - a_1^2,\\ f_2(s) &:= s_3^2 + s_1^2 - 2s_3s_1c_2 - a_2^2,\\ f_3(s) &:= s_1^2 + s_2^2 - 2s_1s_2c_3 - a_3^2.\end{aligned}$$

The problem is to find the set $S$ of all positive zeros of $f$ (i.e. all $s = (s_1, s_2, s_3)^T \in \mathbb{R}^3$ such that $f_i(s) = 0$ and $s_i > 0$, $i = 1, 2, 3$).

Concerning the relations between (**P**) and (**P**$_{\mathbf{red}}$) we observe first that the quantities

$$c_i := \cos(\alpha_i),\ i = 1, 2, 3,$$

$$a_1 := |x_2 - x_3|,\ a_2 := |x_3 - x_1|,\ a_3 := |x_1 - x_2|,$$

where $|\cdot|$ denotes the Euclidean norm, and $\alpha_i, x_i$, $i = 1, 2, 3$, are the data given in (**P**), satisfy the inequalities (14) and (15), and that for these quantities $s := (|x_1 - x|, |x_2 - x|, |x_3 - x|)^T$ is in $S$.

In fact it is well-known that for the considered $c_i$ and $s_i$ we have

$$w = \frac{36V^2}{s_1^2 s_2^2 s_3^2},$$

where $V$ denotes the volume of the tetrahedron $[P_1, P_2, P_3, P]$ which is possibly degenerate. Hence $w \geq 0$. (15) is a consequence of the assumed affine independence of the points $P_1, P_2, P_3$.

Finally $s \in S$ is verified by applying the cosine theorem to each of the (possibly degenerate) triangles $[P_3, P, P_2]$, $[P_1, P, P_3]$, $[P_2, P, P_1]$.

On the other hand, if we know $S$ for the considered $c_i$ and $a_i$ then it can be verified using e.g. Lemma 1 in [19]:

**Lemma 1.** *The set of all possible candidates for the coordinate vector $x$ of $P$ is given by the set of all*

$$x_+ = x_1 + \lambda_2(x_2 - x_1) + \lambda_3(x_3 - x_1) + \tau n$$

*and*

$$x_- = x_1 + \lambda_2(x_2 - x_1) + \lambda_3(x_3 - x_1) - \tau n$$

*where*

$$\begin{aligned}
\lambda_2 &:= s_3 s_1 \left(s_3 s_1 (1 - c_2^2) + s_2 s_3 (c_1 c_2 - c_3) + s_1 s_2 (c_3 c_2 - c_1)\right) / (4F^2), \\
\lambda_3 &:= s_1 s_2 \left(s_1 s_2 (1 - c_3^2) + s_3 s_1 (c_2 c_3 - c_1) + s_2 s_3 (c_1 c_3 - c_2)\right) / (4F^2), \\
\tau &:= s_1 s_2 s_3 \sqrt{w} / (2F), \\
s &= (s_1, s_2, s_3)^T \in S,
\end{aligned}$$

*$F$ denotes the area of the triangle $[P_1, P_2, P_3]$ and $n$ is one of the two possible unit vectors orthogonal to the plane spanned by $P_1, P_2, P_3$.*

### 3.2 Approaches for Including $S$

Even if the data given in problem (**P**) are exact and machine representable, the quantities $c_i, a_i$ are usually not machine representable. Therefore we are exclusively interested in algorithms suitable for computing (sufficiently narrow) inclusions of $S$ from (suitably thin) machine intervals containing the $c_i$ and $a_i$.

**First Approach**

In a first approach ($\mathbf{P_{red}}$) was converted into the problem to find all minimizers of $f_1^2 + f_2^2 + f_3^2$ with positive components in order to solve the converted problem by applying one of the standard packages for verified global optimization. These packages can be applied since it is possible to compute an initial box containing $S$.

**Lemma 2.** *$S$ is contained in the box*

$$B := \begin{pmatrix} [0, \min\{b_2, b_3\}] \\ [0, \min\{b_3, b_1\}] \\ [0, \min\{b_1, b_2\}] \end{pmatrix},$$

*where*

$$b_i := \begin{cases} a_i / \sqrt{1 - c_i^2} & \text{if } c_i > 0 \\ a_i & \text{if } c_i \leq 0 \end{cases}, \quad i = 1, 2, 3.$$

*Proof.* Let $s$ be in $S$. Then from $f_2(s) = 0$ we conclude

$$s_3 = s_1c_2 \pm \sqrt{a_2^2 - (1 - c_2^2)s_1^2},$$
$$s_1 = s_3c_2 \pm \sqrt{a_2^2 - (1 - c_2^2)s_3^2},$$

with $a_2^2 - (1 - c_2^2)s_1^2 \geq 0$ since $s_3$ is real. Hence $s_1 \leq a_2/\sqrt{1 - c_2^2}$.
If $c_2 \leq 0$ then $s_1 = s_3c_2 + \sqrt{a_2^2 - (1 - c_2^2)s_3^2}$, since $s_1 > 0$.
Hence $s_1 \leq \sqrt{a_2^2 - (1 - c_2^2)s_3^2} < a_2$, and $s_1 \leq b_2$ is shown.
Similarly $s_1 \leq b_3$ can be concluded from $f_3(s) = 0$. Index shifts complete the proof.

Applying now one of the mentioned packages results in a list of usually small boxes the union of which covers $S$. Computational results obtained by A. Stephan with the module `gop` listed in [18] and with his improved version `mgop` of `gop` presented in [22] indicate that in most practical cases the list consists of at most four pairwise disjoint boxes for which it can be verified that each of them contains exactly one $s \in S$ and that this $s$ is not singular. However in exceptional cases $S$ can contain singular elements, i.e. elements $s$ for which the Jacobian

$$Jf(s) := 2 \begin{pmatrix} 0 & s_2 - s_3c_1 & s_3 - s_2c_1 \\ s_1 - s_3c_2 & 0 & s_3 - s_1c_2 \\ s_1 - s_2c_3 & s_2 - s_1c_3 & 0 \end{pmatrix}$$

of $f$ in $s$ is singular.

Referring to the situation considered in problem (**P**) it can be shown, using some well-known trigonometric identities:

**Lemma 3.** *Let $r$ denote the radius, and $M$ the center of the circum circle of the triangle $[P_1, P_2, P_3]$, $R$ the distance between $M$ and the orthogonal projection of $P$ to the plane through $P_1, P_2, P_3$. Then for $s := (|x_1 - x|, |x_2 - x|, |x_3 - x|)^T$ we have*

$$\det Jf(s) = 128r^4(r^2 - R^2) \left(\sin\tilde{\alpha}_1 \sin\tilde{\alpha}_2 \sin\tilde{\alpha}_3\right)^2 / (s_1s_2s_3),$$

*where*

$$\tilde{\alpha}_1 := \angle(P_3, P_1, P_2), \tilde{\alpha}_2 := \angle(P_1, P_2, P_3), \tilde{\alpha}_3 := \angle(P_2, P_3, P_1).$$

Hence $Jf(s)$ is singular for $s := (|x_1 - x|, |x_2 - x|, |x_3 - x|)^T$ iff $P$ is on the circular cylinder orthogonal to the plane through $P_1, P_2, P_3$ and intersecting this plane in the circum circle of the triangle $[P_1, P_2, P_3]$. In practice $P$ is always chosen sufficiently far from this *critical cylinder*.

**Second Approach**

Although A. Stephan's improved version `mgop` of the module `gop` is usually much faster than `gop`, e.g. it solved the problem of type ($\mathbf{P_{red}}$) defined by the realistic

data $c_1 = 0.846735205, c_2 = 0.928981803, c_3 = 0.912299033, a_1 = 1871.1, a_2 = 1592.4, a_3 = 1471.9$ about 38 times faster than the original `gop`, it needed still 71 seconds applying Pascal-XSC-Version T3.01 on a PC with a 133MHz processor.

Therefore, mainly in order to search for a faster method, the aim of a second approach was to develop verifying versions of algorithms relying more on the special structure of $(\mathbf{P_{red}})$. Sorting out the methods used in practice which can be considered for a verifying extension, one observes that there are essentially two different ones.

The first one is presented e.g. by J.A. Grunert in [17]. It is based on a result of J.L. Lagrange [20] who converted $(\mathbf{P_{red}})$ to the problem to compute all positive zeros of a polynomial $p$ of degree $\leq 4$.

The second one is presented in the more recent paper [16] by E.W. Grafarend, P. Lohse and B. Schaffrin. They worked out the details of a method based on a result of M.G. Lamé [21] which allows to convert $(\mathbf{P_{red}})$ essentially to the problem to compute all zeros of a polynomial $q$ of degree $\leq 3$.

They also programmed their method and presented (in Teil IV of [16]) numerical solutions for four real life examples.

Unfortunately verifying versions of these methods can fail even if $P$ is far away from the critical cylinder. This is because one has to include zeros of polynomials with interval coefficients and it may happen in very harmless examples that the interval for the leading coefficient or for the constant or for a discriminant contains zero. In such a case inclusions of the zeros of the polynomial, useful for further computations, are usually not possible. Problems with the polynomial $p$ can appear e.g. if $\alpha_i \approx \tilde{\alpha}_i$ for some $i$, indicating that there is an $s = (s_1, s_2, s_3)^T$ in $S$ with $s_i \approx 0$. But usually $P$ is not close to one of the points $P_i$.

A simple example in which the described problem appears for the polynomial $q$ is given by the data $c_1 = c_2 = c_3 = 0.76604443$, $a_1 = a_2 = a_3 = 5.0$, for which A. Stephan has computed with `mgop` very tight inclusions of all $s \in S$ without any difficulties ($S$ consists of four nonsingular elements in this example).

**Remark.** Although the considered algebraic methods seem not to be suited for verified extensions, they give more insight into the structure of $S$. They show e.g. that either $S$ contains not more than four elements or infinitely many. The latter happens if $P$ is on the circum circle of the triangle $[P_1, P_2, P_3]$.

**Third Approach**

A very promising approach is based on the following simple idea:

Consider the conditions

$$s_1, s_2, s_3 > 0$$
$$s_2^2 + s_3^2 - 2s_2s_3c_1 - a_1^2 = 0 \tag{16}$$
$$s_3^2 + s_1^2 - 2s_3s_1c_2 - a_2^2 = 0 \tag{17}$$
$$s_1^2 + s_2^2 - 2s_1s_2c_3 - a_3^2 = 0 \tag{18}$$

characterizing $s = (s_1, s_2, s_3)^T \in S$.

Derive $s_3$ from the second equation as a function of $s_1$. There might be two possibilities: $s_3 = s_3^{(1)}(s_1)$ or $s_3 = s_3^{(-1)}(s_1)$, where
$s_3^{(\sigma)}(s_1) := s_1 c_2 + \sigma(a_2^2 - (1-c_2^2)s_1^2)^{1/2}$, $\sigma \in \{-1, 1\}$.

Derive $s_2$ from the third equation as a function of $s_1$. There might be also two possibilities: $s_2 = s_2^{(1)}(s_1)$ or $s_2 = s_2^{(-1)}(s_1)$, where
$s_2^{(\tau)}(s_1) := s_1 c_3 + \tau(a_3^2 - (1-c_3^2)s_1^2)^{1/2}$, $\tau \in \{-1, 1\}$.

Inserting now $s_3^{(\sigma)}(s_1)$ and $s_2^{(\tau)}(s_1)$ into the first equation leads to the condition

$$h^{(\sigma,\tau)}(s_1) = 0,$$

where

$$h^{(\sigma,\tau)}(s_1) = s_2^{(\tau)}(s_1)^2 + s_3^{(\sigma)}(s_1)^2 - 2s_2^{(\tau)}(s_1)s_3^{(\sigma)}(s_1)c_1 - a_1^2.$$

Thus it can be expected that we can derive inclusions of $S$ from inclusion of all positive zeros of the four functions $h^{(1,1)}, h^{(1,-1)}, h^{(-1,1)}, h^{(-1,-1)}$ of one variable.

In fact, a corresponding characterization of $S$ can be given after fixing the proper domains of the functions $s_3^{(\sigma)}, s_2^{(\tau)}$ and $h^{(\sigma,\tau)}$. The proper domain of the function $s_3^{(\sigma)}(s_2^{(\tau)})$ is the largest subset of $]0, \infty[$ on which $s_3^{(\sigma)}(s_2^{(\tau)})$ is real and positive.

Introducing the abbreviations

$$a_i' := a_i/(1-c_i)^{1/2}\ (> a_i),\ i = 1, 2, 3,$$

we have

$$\text{dom } s_3^{(1)} = \begin{cases} ]0, a_2'] & \text{if } c_2 > 0 \\ ]0, a_2[ & \text{if } c_2 \le 0, \end{cases}$$

$$\text{dom } s_3^{(-1)} = \begin{cases} ]a_2, a_2'] & \text{if } c_2 > 0 \\ \emptyset & \text{if } c_2 \le 0, \end{cases}$$

$$\text{dom } s_2^{(1)} = \begin{cases} ]0, a_3'] & \text{if } c_3 > 0 \\ ]0, a_3[ & \text{if } c_3 \le 0, \end{cases}$$

$$\text{dom } s_2^{(-1)} = \begin{cases} ]a_3, a_3'] & \text{if } c_3 > 0 \\ \emptyset & \text{if } c_3 \le 0. \end{cases}$$

The proper domains of the $h^{(\sigma,\tau)}$ result from the proper domains of the $s_3^{(\sigma)}, s_2^{(\tau)}$ as follows:

$$\text{dom } h^{(1,1)} = \begin{cases} ]0, \min\{a_2', a_3'\}] & \text{if } c_2 > 0 \text{ and } c_3 > 0 \\ ]0, a_2'] \cap ]0, a_3[ & \text{if } c_2 > 0 \text{ and } c_3 \le 0 \\ ]0, a_2[ \cap ]0, a_3'] & \text{if } c_2 \le 0 \text{ and } c_3 > 0 \\ ]0, \min\{a_3, a_2\}[ & \text{if } c_2 \le 0 \text{ and } c_3 \le 0, \end{cases}$$

$$\text{dom } h^{(1,-1)} = \begin{cases} ]0, a_2'] \cap ]a_3, a_3'] & \text{if } c_2 > 0 \text{ and } c_3 > 0 \\ ]0, a_2[ \cap ]a_3, a_3'] & \text{if } c_2 \le 0 \text{ and } c_3 > 0 \\ \emptyset & \text{if } c_3 \le 0, \end{cases}$$

$$\text{dom } h^{(-1,1)} = \begin{cases} ]a_2, a_2'] \cap ]0, a_3'] & \text{if } c_2 > 0 \text{ and } c_3 > 0 \\ ]a_2, a_2'] \cap ]0, a_3[ & \text{if } c_2 > 0 \text{ and } c_3 \leq 0 \\ \emptyset & \text{if } c_2 \leq 0, \end{cases}$$

$$\text{dom } h^{(-1,-1)} = \begin{cases} ]a_2, a_2'] \cap ]a_3, a_3'] & \text{if } c_2 > 0 \text{ and } c_3 > 0 \\ \emptyset & \text{otherwise }, \end{cases}$$

where some of the considered intersections might be empty.

Now a straightforward verification shows

**Lemma 4.**

$$S = S^{(1,1)} \cup S^{(1,-1)} \cup S^{(-1,1)} \cup S^{(-1,-1)}$$

*where*

$$S^{(\sigma,\tau)} := \left\{ (s_1, s_2, s_3)^T \in \mathbb{R}^3 : h^{(\sigma,\tau)}(s_1) = 0, s_2 = s_2^{(\tau)}(s_1), s_3 = s_3^{(\sigma)}(s_1) \right\}$$

$\forall \sigma, \tau \in \{-1, 1\}$.

### 3.3 Examples

In a preliminary Pascal-XSC program based on Lemma 4 three main steps are carried out:

- The first step is designed for computing relatively crude but nonoverlapping inclusions of the zeros of the $h^{(\sigma,\tau)}$. This step is carried out with a simplified version of the module `nlfzero` from the Toolbox [18].
- After a successful run of the first step, in the second step a set of usually not very narrow boxes covering $S$ is computed.
- The third step is a *refine and verification step* as it is used also in the first approach.

The program was tested successfully with several realistic and artificial examples. Let us consider the results obtained for two of them:

**Example 1**

$$\begin{aligned} c_1 &\in -0.02800932059_6^2, \\ c_2 &\in -0.73599476269_9^5, \\ c_3 &\in 0.69732782693_3^5, \\ a_1 &\in 20883.7483382_0^2, \\ a_2 &\in 22652.5934423_0^2, \\ a_3 &\in 11606.583393_{09}^{11}. \end{aligned}$$

These data are computed from given coordinate vectors and measured horizontal and vertical angles in the Westharz-Example presented by E.W. Grafarend, P. Lohse and B. Schaffrin (in Teil IV of [16]).

Result produced by the mentioned Pascal-XSC program:
$S = \{(s_1, s_2, s_3)^T\}$ where

$$s_1 \in 11562.4436_0^2,$$
$$s_2 \in 16188.7991_6^8,$$
$$s_3 \in 12747.2949_6^8,$$

and $(s_1, s_2, s_3)^T$ is nonsingular.

This result shows that the numerical values $11562.454, 16188.809, 12747.290$ presented for $s_1, s_2, s_3$ by the authors of [16] do not have the obviously desired computational precision.

**Example 2**
For the artificial data $c_1 = c_2 = c_3 = 0$, $a_1 = 2.0, a_2 = 5.4, a_3 = 4.0$, the mentioned Pascal-XSC program produced the right answer $S = \emptyset$. In fact it can be easily shown that in case $c_1 = c_2 = c_3 = 0$ $S$ is empty iff $a_1, a_2, a_3$ constitute a triangle with a nonacute angle. Otherwise the set $S = \{(s_1, s_2, s_3)^T\}$ where

$$\begin{pmatrix} s_1 \\ s_2 \\ s_3 \end{pmatrix} = \begin{pmatrix} \left(\frac{1}{2}(-a_1^2 + a_2^2 + a_3^2)\right)^{1/2} \\ \left(\frac{1}{2}(a_1^2 - a_2^2 + a_3^2)\right)^{1/2} \\ \left(\frac{1}{2}(a_1^2 + a_2^2 - a_3^2)\right)^{1/2} \end{pmatrix}.$$

### 3.4 Concluding Remarks

1. Since B. Kearfott has demonstrated that his package GLOBSOL can compute sufficiently narrow inclusions of $S$ very fast, it would be desirable to check the third approach against the first one on a common basis, e.g. by including the zeros of the functions $h^{(\sigma,\tau)}$ also with GLOBSOL.
2. The question how to single out possible singular or almost singular elements of $S$ fast enough is not yet answered satisfactorily.

## References

1. DoCarmo, M.P.: Differential Geometry of Curves and Surfaces, Prentice-Hall, New Jersey 1976.
2. Blaschke, W., Leichtweiß K.: Elementare Differentialgeometrie, Springer, Berlin, Heidelberg, 1973.
3. Alefeld, G., Herzberger H.: Introduction to interval computations, Academic Press, New York, London, Paris, San Diego, San Francisco, São Paulo, Sydney, Tokyo, Toronto, 1983
4. Klatte, R., Kulisch, U., Neaga, M., Ratz, D., Ullrich, Ch.: PASCAL - XSC - Sprachbeschreibung mit Beispielen, Springer-Verlag, Heidelberg, 1991
5. Walter, W.: Gewöhnliche Differentialgleichungen, Springer, Berlin, Heidelberg, New York, 1976.

6. Borovac, S.: Zur Theorie und verifizierten Lösung der ersten und zweiten geodätischen Hauptaufgabe auf dem Rotationsellipsoid, Diplomarbeit, Universität Wuppertal, 1998
7. Lohner, R.: Einschließung der Lösung gewöhnlicher Anfangs- und Randwertaufgaben und Anwendungen, Dissertation, Universität Karlsruhe, 1988
8. Nedialkov, N.S., Jackson, K.R.: The design and implementation of an object-oriented validated ODE solver, Technical Report, Department of Computer Science, University of Toronto, 2002
9. Auer, E.: Ein verifizierender Anfangswertproblemlöser in C++ zur Integration in MOBILE, Master's thesis, Universität Duisburg, 2002
10. Berz, M., Makino, K.: Verfified integration of ODEs and flows using differential algebraic methods on high-order Taylor models, Reliable Computing, 4 (1998), 361-369
11. Stauning, O.: Automatic validation of numerical solutions, PhD thesis, Technical University of Denmark, Lyngby, 1997
12. Großmann, W.: Geodätische Rechnungen und Abbildungen in der Landvermessung, Konrad Wittner Verlag, Stuttgart, 1976
13. Torge, W.: Geodesy, de Gruyter, Berlin, New York, 1980
14. Klotz, J.: Eine analytische Lösung kanonischer Gleichungen der geodätischen Linie zur Transformation ellipsoidischer Fächenkoordinaten, Deutsche Geodätische Kommission, Reihe C, Nr. 385, München, 1991
15. Bodemüller, H.: Die geodätischen Linien des Rotationsellipsoides und die Lösung der geodätischen Hauptaufgaben für große Strecken unter besonderer Berücksichtigung der Bessel-Helmertschen Lösungsmethode, Deutsche Geodätische Kommission, Reihe B, Nr. 13, München, 1954
16. Grafarend E.W., Lohse P., Schaffrin B.: Dreidimensionaler Rückwärtsschnitt
    Teil I: Die projektiven Gleichungen, Zeitschrift für Vermessungswesen (ZfV) 2, 61-67, 1989
    Teil II: Dreistufige Lösung der algebraischen Gleichungen – Strecken –, ZfV 3, 127-137, 1989
    Teil III: Dreistufige Lösung der algebraischen Gleichungen – Orientierungsparameter, Koordinaten –, ZfV 4, 172-175, 1989
    Teil IV: Numerik, Beispiele, ZfV 5, 225-234, 1989
    Teil V: Alternative numerische Lösungen, ZfV 6, 278-287, 1989
17. Grunert J.A.: Das Pothenotsche Problem, in erweiterter Gestalt; nebst Bemerkungen über seine Anwendungen in der Geodäsie, Grunerts Archiv für Mathematik und Physik I, 238-248, 1841
18. Hammer R., Hocks M., Kulisch U., Ratz D.: Numerical Toolbox for Verified Computing I, Springer, Berlin, 1993
19. Heindl G.: Best possible componentwise parameter inclusions computable from a priori estimates, measurements and bounds for the measurement errors, Journal of Computational and Applied Mathematics 152, 175-185, 2003
20. Lagrange J.L.: Leçons élémentaires sur les mathématiques données à l'École Normale en 1795; in Serret M.J.A. (Ed.), Oeuvres de Lagrange, Tome 7, Section IV, Paris, 183-288, 1877
21. Lamé M.G.: Examen des différentes méthodes employées pour résoudre les problèmes de géometrie, 70-72, Paris 1818
22. Stephan A.: Über Strategien zur Verbesserung des Pascal-XSC-Moduls GOp zur verifizierten globalen Optimierung, Diplomarbeit, Universität Wuppertal, 1998

# Global Optimization in the COCONUT Project

Hermann Schichl*

Universität Wien
Institut für Mathematik
1090 Wien, Austria
Hermann.Schichl@esi.ac.at

**Abstract.** In this article, a solver platform for global optimization is presented, as it is developed in the COCONUT project. After a short introduction, a short description is given of the basic algorithmic concept and of all relevant components, the strategy engine, inference engines, and the remaining modules. A compact description of the search graph and its nodes and of the internal model representation using directed acyclic graphs (DAGs) completes the presentation.

## 1 Introduction

The COCONUT project [4] is aimed at the integration of the existing approaches to continuous global optimization and constraint satisfaction. It is a project funded by the Future and Emerging Technologies (FET) arm of the IST programme FET-Open scheme of the European Community (IST-2000-26063). Six academic and one industrial partner are involved: ILOG Inc., the industrial partner and project coordinator, *France*, TU Darmstadt, *Germany*, IRIN Nantes, *France*, EPFL Lausanne, *Switzerland*, University of Vienna, *Austria*, University of Louvain-la-Neuve, *Belgium*, University of Coimbra, *Portugal*.

The COCONUT consortium is planning to provide at the end of the project (February 2004) a modular solver environment for nonlinear global optimization problems with an open-source kernel, which can be expanded by commercial and open-source solver components (inference engines, see Section 5).

The application programmer's interface (API) is designed to make the development of the various module types independent of each other and independent of the internal model representation. It will be a collection of open-source `C++` classes protected by the LGPL license model, so that it could be used as part of commercial software. It uses the `FILIB++` [8] library for interval computations and the matrix template library (`MTL`) [12] for the internal representation of various matrix classes. Support for dynamic linking will relieve the user from recompilation when modules are added or removed. In addition, it is designed for distributed computing, and will probably be developed further (in the years after the end of the COCONUT project) to support parallel computing as well.

* supported by the EU project COCONUT (IST-2000-26063)

R. Alt et al. (Eds.): Num. Software with Result Verification, LNCS 2991, pp. 243–249, 2004.

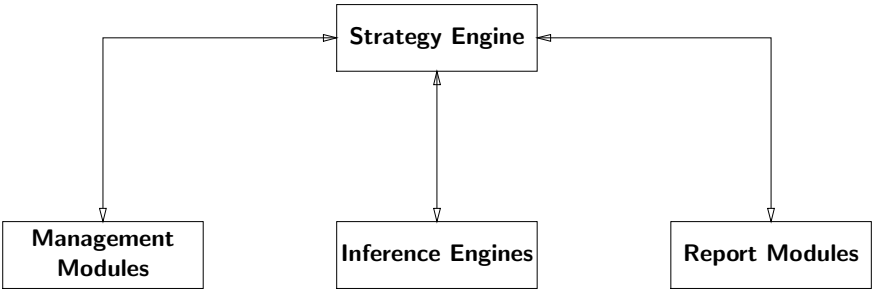



**Fig. 1.** Basic Scheme of the Algorithmic Design

The API kernel implementation consists of more than 50.000 lines of `C++` code and a few `perl` scripts, organized into about 150 files, occupying 1.5 MB of disk space.

The algorithmic design follows the scheme depicted in Figure 1; its various parts are described in more detail in the following sections.

## 2 Models and the Search Graph

The solution algorithm is an advanced branch-and-bound scheme which proceeds by working on the **search graph**, a *directed acyclic graph (DAG)* of search nodes, each representing an optimization problem, a **model**. The **search nodes** come in two flavors: **full nodes** which record the complete description of a model, and **delta nodes** which only contain the difference between the model represented by the node and its (then only) parent. All search nodes "know" in addition their relation to their ancestors. They can be splits, reductions, relaxations, or glueings. The latter turn the graph into a DAG instead of a tree, as usual in branch-and-bound algorithms. The search graph is implemented using the Vienna Graph Template Library (`VGTL`), a library following the generic programming spirit of the `C++ STL` (Standard Template Library).

A **reduction** is a problem, with additional or stronger constraints (**cuts** or **tightenings**), whose solution set can be shown to be equal to the solution set of its parent. A **relaxation** is a problem with fewer or weakened constraints, or a "weaker" objective function, whose solution set contains the solution set of its parent. Usually, relaxed problems have a simpler structure than its original. Typically *linear* or *convex* relaxations are used.

A problem is a **split** of its parent if it is one of at least two descendants and the union of the solution sets of all splits equals the solution set of their parent. Finally, a model is a **glueing** of several problems, if its solution set equals the solution sets of all the glued problems.

During the solution process some, and hopefully most, of the generated nodes will be solved, and hence become **terminal nodes**. These can be removed from

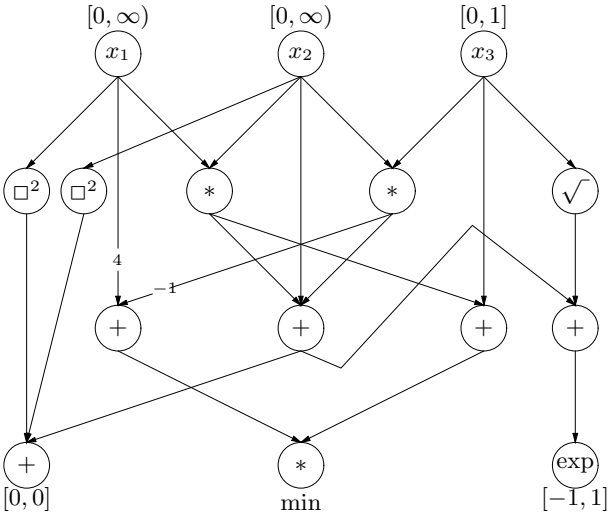



**Fig. 2.** DAG representation of problem (1)

the graph after their consequences (e.g., optimal solutions, ... ) have been stored in the **search database**. This has the consequence that the ancestor relation of a node can change in the course of the algorithm. If, e.g., all the splits but one have become terminal nodes, this split turns into a reduction. If all children of a node become terminal, the node itself becomes terminal, and so on.

The search graph has a **focus** pointing to the model which is worked upon. This model is copied into an enhanced structure - the **work node**. A reference to this work node is passed to each inference engine activated by the strategy engine. The graph itself can be analyzed by the strategy engine using so-called **search inspectors**.

## 3 Mathematical Representation of Problems, Directed Acyclic Graphs

The optimization problems stored in the work nodes, which are passed to the various inference engines, are kept as directed acyclic graphs (DAG), as well. This representation has big advantages; see [11] for a detailed analysis.

A complete optimization problem is always represented by a *single* DAG. The vertices of the graph represent operators similar to computational trees. Constants and variables are sources, objective and constraints are sinks of the DAG.

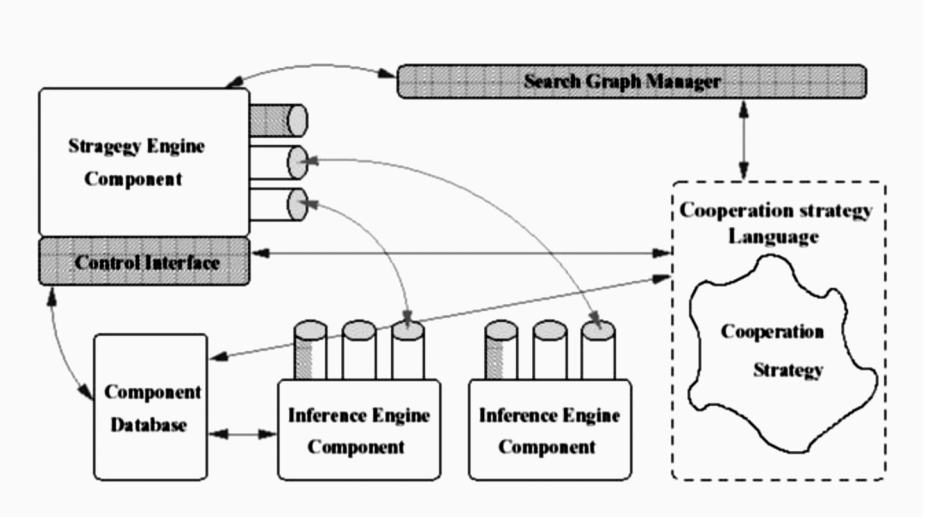



**Fig. 3.** The Strategy Engine Component Framework

Consider for example the optimization problem

$$\begin{array}{ll} \min & (4x_1 - x_2x_3)(x_1x_2 + x_3) \\ \text{s.t.} & x_1^2 + x_2^2 + x_1x_2 + x_2x_3 + x_2 = 0 \\ & e^{x_1x_2+x_2x_3+x_2+\sqrt{x_3}} \in [-1, 1] \\ & x_1 \geq 0, x_2 \geq 0, x_3 \in [0, 1]. \end{array} \tag{1}$$

This defines the DAG depicted in Figure 2.

This DAG is optimally small in the sense that it contains every subexpression of objective function and constraints only once.

Every vertex represents a function $F : \mathbb{R}^N \to \mathbb{R}$ for some $N$. Predefined functions include `sum`, `product`, `max`, `min`, elementary real functions (`exp`, `log`, `pow`, `sqrt`, ... ), and also some discrete operators like `all_diff` and `count`.

For expression graphs (DAG or tree), special forward and backward evaluators are provided. Currently implemented are *real function values*, *function ranges*, *gradients* (real, interval), and *slopes*. In the near future evaluators for *Hessians* (real, interval) and *second order slopes* (see, e.g., [10]) will be provided, as well.

## 4 The Strategy Engine

The **strategy engine** is the main part of the algorithm. It makes decisions, directs the search, and invokes the various modules.

The strategy engine consists of the logic core ("**search**") which is essentially the main solution loop, special **decision makers** (very specialized inference engines, see Section 5) for determining the next action at every point in the algorithm. It calls the management modules, the report modules, and the inference engines in a sequence defined by programmable search strategies.

The engine can be programmed using a simple **strategy language**, an interpreted language based on Python. Since it is interpreted, (semi-)interactive and automatic solution processes are possible, and even debugging and single-stepping of strategies is supported. The language is object oriented, garbage collecting, and provides dynamically typed objects. These features make the system easily extendable.

Furthermore, the strategy engine manages the search graph via the **search graph manager**, and the search database via the **database manager**.

The strategy engine uses a component framework (see Figure 3) to communicate with the inference engines. This makes it possible to launch inference engines dynamically (on need, also remote) to avoid memory overload. Since the strategy engine is itself a component, even multilevel strategies are possible.

## 5 Inference Engines

For the solution strategy, the most important class of modules are the **inference engines**. They provide the computational base for the algorithm, namely methods for problem structure analysis, local optimization, constraint propagation, interval analysis, linear relaxation, convex optimization, bisection, . . . .

Corresponding to every type of problem change, a class of inference engines is designed: **model analysis** (e.g. find convex part), **model reduction** (e.g. pruning, fathoming), **model relaxation** (e.g. linear relaxation), **model splitting** (e.g. bisection), **model glueing** (e.g. undo excessive splitting), **computing of local information** (e.g. probing, local optimization).

Inference engines calculate changes to a model that do not change the solution set. But they **never** change the model; the decision to apply the changes if they are considered useful is left to the strategy engine. Therefore, the result of an inference engine is a list of changes to the model together with a weight (the higher the weight the more important the change). Whether an advertised change is actually performed is decided by the strategy engine, and the actual change is executed by an appropriate management module. The inference engines are implemented as subclass of a single `C++` base class. In addition, there is a fixed documentation structure defined.

Several state of the art techniques are already provided:

- DONLP2-INTV, a general purpose nonlinear local optimizer for continuous variables [14],
- STOP, a heuristic starting point generator,
- Karush-John-Condition generator using symbolic differentiation,
- Point Verifier for verifying solution points,
- Exclusion Box generator, calculating an exclusion region around local optima [10],
- Interval constraint propagation [2,1],
- Linear Relaxation,
- CPLEX, a wrapper for the state of the art commercial linear programming solver by ILOG,

- Basic Splitter,
- BCS, a box covering solver [9,13],
- Convexity detection, for simple convexity analysis.

## 6 Management and Report Modules

**Management modules** are the interface between the strategy engine and the internal representation of data and modules, taking care of the management of models, resources, initialization, the search graph, the search database, . . . .

They are provided to make it possible to change the implementation of the search graph and the internal representation of problems without having to change all of the modules. Management modules just perform some of the changes which have been advertised by inference engines; they **never** calculate anything.

The final class of modules, called **report modules**, produce output. Human or machine readable progress indicators, solution reports, the interface to modeling languages [7] (currently only `AMPL` [6] is supported), and the biggest part of the checkpointing is realized via report modules.

## 7 Conclusion

The open design of the solver architecture, and its extensibility to include both open source modules and commercial programs, was chosen in the hope that the system will be a unique platform for global optimization in the future, serving the major part of the community, bringing their members closer together. The documentation of the interfaces and on how to write new inference engines can be found on the home page of the COCONUT environment [5].

We are happy that researchers and companies from outside the COCONUT project have already agreed to complement our efforts in integrating the known techniques. Thus there will be in the near future *Bernstein modules* by J. Garloff and A. Smith (U. Konstanz), *verified lower bounds for convex relaxations* by Ch. Keil and Ch. Jansson (TU Hamburg-Harburg), a *GAMS reader* by the GAMS consortium [3], *Taylor arithmetic* by G. Corliss (Marquette U.), *asymptotic arithmetic* by K. Petras (U. Braunschweig), and an interface to `XPRESS`, a commercial LP-solver by Dash Optimization.

**Acknowledgments.** I want to thank Arnold Neumaier (University of Vienna) for his support and his advice, and Eric Monfroy (IRIN, Nantes) for the picture of the strategy engine components.

## References

1. F. Benhamou, David McAllester, and Pascal Van Hentenryck. CLP intervals revisited. In Maurice Bruynooghe, editor, *Proceedings of ILPS'94, International Logic Programming Symposium*, pages 124–138, Ithaca, NY, USA, 1994. MIT Press.

2. F. Benhamou and W. J. Older. Applying interval arithmetic to real, integer, and boolean constraints. *Journal of Logic Programming*, 32(1):1–24, 1997.
3. Anthony Brooke, David Kendrick, and Alexander Meeraus. *GAMS - A User's Guide (Release 2.25)*. Boyd & Fraser Publishing Company, Danvers, Massachusetts, 1992.
4. The COCONUT project home page. http://www.mat.univie.ac.at/coconut.
5. The COCONUT Environment home page. http://www.mat.univie.ac.at/coconut-environment.
6. Robert Fourer, David M. Gay, and Brian W. Kernighan. *AMPL — A Mathematical Programming Language*. Thomson, second edition, 2003.
7. Josef Kallrath, editor. *Modeling Languages in Mathematical Optimization*. Kluwer Academic Publishers, Boston Dordrecht London, 2003.
8. M. Lerch, G. Tischler, and J. Wolff von Gudenberg. *filib++–Interval Library, Specification and Reference Manual*. Informatik, Universität Würzburg, techn. report 279 edition, August 2001.
9. D. Sam-Haroud and B. Faltings. Consistency techniques for continuous constraints. *Constraints*, 1:85–118, 1996.
10. Hermann Schichl and Arnold Neumaier. Exclusion regions for systems of equations. *SIAM J. Num. Analysis*, 2003. to appear.
11. Hermann Schichl and Arnold Neumaier. Interval analysis on directed acyclic graphs for global optimization, 2003. Preprint.
12. J. Siek, A. Lumsdaine, and L.-Q. Lee. Generic programming for high performance numerical linear algebra. In *Proceedings of the SIAM Workshop on Object Oriented Methods for Inter-operable Scientific and Engineering Computing (OO'98)*. SIAM Press, 1999.
13. M. Silaghi, D. Sam-Haroud, and B. Faltings. Search techniques for non-linear CSPS with inequalities. In *Proc. of the 14th Canadian Conf. on AI*, 2001.
14. Peter Spellucci. An SQP method for general nonlinear programs using only equality constrained subproblems. *Mathematical Programming*, 82:413–448, 1998.

# An Application of Wavelet Theory to Early Breast Cancer

Baya Oussena[1,2], Abderrezak Henni[2], and René Alt[1]

LIP6
Université de Paris VI,
75015 Paris, France
and
Institut National d'Informatique (INI)
Oued Smar
Algiers, Algeria

**Abstract.** Mammography is one of the principal kinds of the medical images which is considered as the most efficient for the detection of breast cancer at its first step. Microcalcifications and Clustered microcalcifications are known to be the first sign of a development of an eventual cancer. They appear as small and bright regions with irregular shape in the breast. Their diversity in their shape, their orientation, their size and localisation in a dense mammogram are the cause of the major difficulty for their classification. The aim of our scheme is the development of a method for the detection and classification of all type of microcalcifications. The wavelet transform and its multiresolution analysis is well known as a powerful tool for non-stationary signal analysis. In fact its discrete version is closely related to filter banks. In this paper a technique using two different discrete wavelet transforms is used to provide a method for enhancing and controlling the detection of all types of small scale objects, microcalcifications, and separating them from large background structures. Hence, the detection and the enhancement of the microcalcifications is assured for each different type by using appropriate wavelet coefficients. Some numerical experiments are given.

## 1 Introduction

Breast cancer is a major cause of premature death in women. Treatment is most successful if the tumour is detected early and mammography has been proved to be the most effective primary diagnostic procedure. Between 60 and 80% of nonpalpable breast carcinomas reveal microcalcifications on mammograms and therefore clustered microcalcifications on mammograms are an important indicator of the onset of breast carcinoma [1,3]. However an important of cancer cases are known to be missed by radiologists. To assist radiologists in their diagnosis, several techniques have been developed for the automated detection of microcalcifications. Some have reported the efficiency of using a specialised pre-processing step known as the difference-image technique [4,6]. Others have

R. Alt et al. (Eds.): Num. Software with Result Verification, LNCS 2991, pp. 250–258, 2004.

reported applying statistical methods [7,8]. Recently, there has been some efforts to use the wavelet transform for analysing mammography features.

Our study follows the method developed by Yoshida et al [9,10] using a wavelet transform to enhance the microcalcifications images. Our approach extends this previous investigations of the wavelet transform method in that we have studied the various classes of microcalcifications which differ in shape, alignment, size and location. Le Gal and al [11] have grouped the observed combinations of attributes of microcalcifications into five basic types which are specified in Table 1 and denoted Type I to Type V. The Type I microcalcifications are annular and rounded in shape, reaching a size of around 1mm across. The microcalcifications of Type II are roughly spherical and large enough to be felt by palpation. Type II image. Clustered microcalcifications of Type III are the most difficult to observe, and are sometimes confused with noise, since their size does not exceed 3 pixels. and finally, microcalcifications of Types IV and V are slightly bigger than Type III which makes them easier to spot in practice. The characteristics of these microcalcifications are summarized in Table 1.

For each of these types we have investigated the parametrisation of the wavelet transform to optimise the enhancement of the microcalcification image.

**Table 1.** Le Gal Microcalcifications Classification and Specificity.

| Types | Caracteristics | Le Gal (Probability of being a cancer) |
|---|---|---|
| Type I | Annular and rounded shape | 0 % |
| Type II | Roughly spherical and regular | 22 % |
| Type III | Confused with noise and too small to distinguish the shape | 36 % |
| Type IV | Irregular and could be confused with noise | 56 % |
| Type V | They are shaped as Y or W | 90 % |

## 2 Method

### 2.1 The Use of the Wavelet Transform

The principle of the wavelet transform can be explained in the following manner [12,14]. Mammograms contain structures with a wide range of sizes and contrasts. For example, a mammogram may contain large-scale structures such as masses, as well as small structures such as microcalcifications. These suspicious regions are surrounded by normal dense tissues and vessels that may make the radiologists' identification of carcinomas difficult. To find microcalcifications efficiently, it is useful to use a method that can focus on localised fine structures while removing coarse background structures. The wavelet transform is an ideal tool for analyzing images with such a mixture of coarse and fine patterns. The wavelet transform decomposes mammograms into different scale components. To

extract and examine the small-scale structures contained in the original mammogram, the wavelet transform uses a fine "probe" that is represented by a high-level wavelet well localized in the space domain. By performing a convolution operation between such a wavelet and the mammogram, one can substantially enhance small size structures. The same process can as well be applied to large-scale structures. In fact, this is done by using low level wavelets which have a large domain of definition in space. There exist many types of mother wavelets and associated wavelets. Depending on the properties of the mother wavelets, the wavelet transform can be divided into two categories: the redundant wavelet transform and the orthogonal wavelet transform. In this study, we use the orthogonal Haar and Daubechies wavelets [15,16] since they allow an input image to be decomposed into a set of independent coefficients corresponding to each basis vector and have proved experimentally to be efficient on the studied family of images probably because of their asymmetry.

### 2.2 Wavelet Transform Combination

The following technique has been used in the present study. First the histogram of the image is equalized and then two discrete wavelet transforms (DWT), called $P_1$ and $P_2$ each with distinct parameters which may be separately optimised, are applied to the same original image and the output results are combined with decision criteria determined by a threshold on the grey scale. The combination process starts with the original image which is the entry data to processes $P_1$ and $P_2$, producing outputs $O_1$ and $O_2$.

For example:

- $P_1$ decomposes the image up to some level for example 3 using Less Asymmetric Daubechies wavelets LAD8. With level 1 set to zero, the resulting image $O_1$ would be reconstructed from levels 2 and 3.

- $P_2$ decomposes the image up to another level for example 6 using a different family of wavelet such as a another Daubechies or a Haar filter. WSetting levels 4 and 5 set to zero, $O_2$ is then reconstructed from levels 1,2 and 3.

The final image is reconstructed from $O_1$ and $O_2$

## 3 Enhancement of the Microcalcification Images: Evaluation and Results

### 3.1 First Experiment

To consider quality of the image resulting from the treatment by the wavelets, we proceeded to a comparison of the histograms before and after the application of each wavelet transform. The various parameters characterising the model implemented are: the choice of the type of wavelets which have been chosen here as Daubechies, Haar and Coifman wavelets; The size of the filters which varies for Daubechies from 4 to 20 but remains fixed to 2 for Haar and to 6 for Coifman; the levels of decompositions which varies from 1 to 6 and the levels

of rebuilding which correspond to the coefficients of wavelets taking part in the image reconstruction process.

### 3.2 Second Experiment

To appreciate the quality of safeguard for the microcalcifications in the mammogram treated by wavelets, we undertook an analysis by operations of detection of contours [17].

The contours extracted for each one of them must be neither two thick (thickness higher than a pixel), nor unsignificant. Thus the results of three different operators have been compared and combined to provide a non ambiguous significant localisation. This has been tested on a set of 30 images.

### 3.3 Evaluation Using the Statistical Parameters

The reliability of the image are often measured according to two criteria : the Mean Squared-Error and the Peak Signal to Noise Ratio (PSNR). Unfortunately these criteria could not be taken in our tests because of the dephasing of the image resulting from an analysis by the Daubechies wavelets compared to the original image. The Mean Squared-Error resulting from this treatment appears large whereas the two images are identical but are only out of phase. We thus directed ourselves towards other statistical parameters of the images such as the mean, the median, the variance and as well as the maximum frequencies and this before and after the analysis.

### 3.4 Evaluation by the Method of Detection of Contours

In this evaluation we assessed our system of localisation of the microcalcifications by the method of contours and to highlight the influence of the parameters of analysis by wavelet. Intuitively, in a mammogram treated by wavelet, contours are situated between the pixels belonging to areas having different intensities which is the case of microcalcifications. In an image, the variations of intensities represent changes of physical or geometrical proximity of the scene or object observed; and in a number of cases, as it is the case in our study, the variations of intensities are significant information in the localisation of possible microcalcifications and their forms.

Let us call "MCI" the region containing localised microcalcifications. Let us also consider an unspecified pixel a(i), in the process of detection of contour characterizing the microcalcification. Then three possible cases are possible for the assignment of the pixel to area MCI :

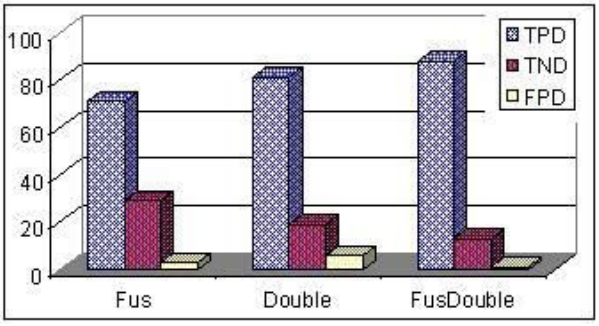
1 ) a(i) belongs indeed to MCI and is detected as being so. The whole of these pixels is called "True Positive Detection:TPD."

2 ) a(i) does not belong indeed to MCI and is detected as being so. The whole of these pixels is called "False Positive Detection FPD"

3 ) a(i) belongs to MCI and is not detected as being so. The whole of these pixels is called "True Negative Detection TND".

According to these three criteria, the cancer specialists compared the localisation of the microcalcifications for the various tests.Table 2 summarises the TPD, TND, FPD resulted altogether with the different modules m1, m2, m3.

**Table 2.** True Pos. Det., True Neg. Det., False Pos. Det.




### 3.5 Enhancement of the Microcalcifications

To evaluate the performance of the parameters adapted to each of the five microcalcifations type, we used a database consisting of about thirty true regions of interest containing actual microcalcifications and normal regions of interest that were randomly selected from the mammogram database we used. The results of the experiments and the parameters retained for each type are given in the following and summarised in Table 3.

**Table 3.** Parameters retained for each microcalcification type

| Microcalcifications Type | Applied Methode | Adapted Parameters |
|---|---|---|
| Type I | Double DWT | P1: LAD8;Up to 6,RESET:5 And P2: LAD12;Up to 6;RESET:5 |
| Type II | One DWT | LAD8;UP to 6; RESET:5 Or LAD12; Up to 6, RESET 4,5, Hist.Eq. |
| Type III | Double DWT | P1: LAD8; Up to 6; RESET: 4,5 And P2: LAD20; Up to 6; RESET: 4,5 |
| Type IV-V | One DWT | LAD8; Up to 6; RESET: 1,5 |

We experimented Type I microcalcifications first using a LAD4 mother wavelet. The original image was decomposed up to level 4 with levels 1 and 4 set

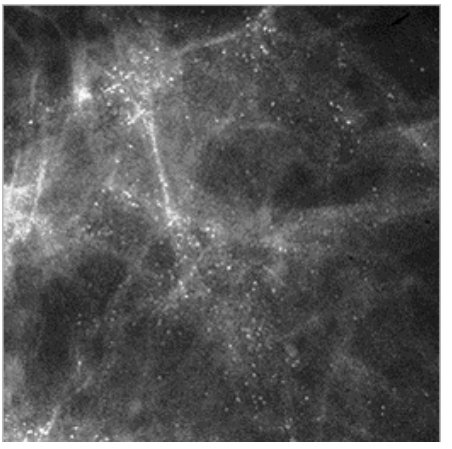
to zero. Setting level 4 to zero spoils the shape definition of the reconstructed image and setting level 1 to zero reduces the contrast of the fine microcalcifications. We experimented then with a 2D-DWT processing, with LAD8 decomposed up to level 6 and level 5 set to zero in the first operation, and LAD12 decomposed up to level 6 with level 5 set to zero for the second. Over and above the brightness effect that the use of LAD8 produces, the use of LAD12 gave better resolution of the smallest microcalcifications, due to the longer length of the filter. Setting level 5 to zero resulted in a considerable smoothing of the background distribution, thus highlighting the microcalcification structures.

The Type II microcalcifications have been experimented using a simple discrete wavelet transform with LAD12, image decomposition up to level 6 and setting to zero levels 4 and 5. Good background smoothing was achieved, but the microcalcifications contrast and edge definition were blurred to an unsatisfactory level. A histogram equalization performed on the resulting image has improved the microcalcifications definition. An alternative Type-II operation used LAD 8 and decomposition up to level 3 with level 1 set to zero. This process also showed a good enhancement of the microcalcifications.

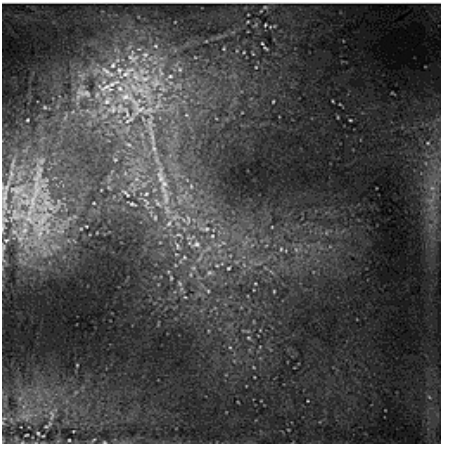
The Type III microcalcifications were experimented with a simple wavelet transform, using LAD 8 and decomposition up to level 6 with levels 1, 4 and 5 set to zero. Good background smoothing was achieved but the contrast on the microcalcifications image was inadequate. Improved results were obtained with a 2D-DWT using LAD8 and decomposition up to 6 with level 4 and 5 set to zero for P1 and LAD20 with decomposition up to level 6 and levels 4,5 set to zero for P2. LAD8, due to its length gave a bright microcalcifications image and good background smoothing was obtained after setting to zero levels 4 and 5. The use of LAD20 resulted in very good spatial resolution of the very fine microcalcifications and in the level 1-3 reconstruction, level 1 was the most important for preservation of the fine detail.

Finally, the Type IV and V microcalcifications showed similar response to wavelet transform processing so, we grouped these two types together in the processing. A processing of Type-V image, identical to the operation on a Type-III image described above enhanced the fine microcalcifications structures well but also created false microcalcifactions images. An alternative, simple discrete wavelet transform procedure using LAD8 and decomposition up to level 6 with levels 4 and 5 set to zero of both types IV and V respectively. The histograms resulted from the processed images of type IV and V showed a new and dense repartition of the grey scales; a good enhancement of grey level frequencies situated in the interval [0..130] corresponding to the background and a translation of both histograms toward decreased grey levels hence a good smoothing of the background. The correct parameters for each type of micro-calcification are reported in Table 3. Some examples of enhancement of microcalcifications in images are reported in Table 4.

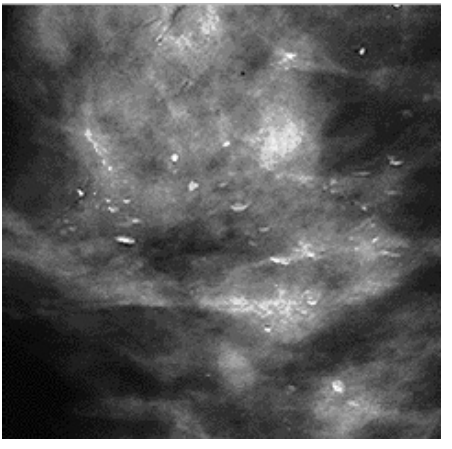
**Table 4.** Enhancement of type 3 and type 5 microcalcifications.



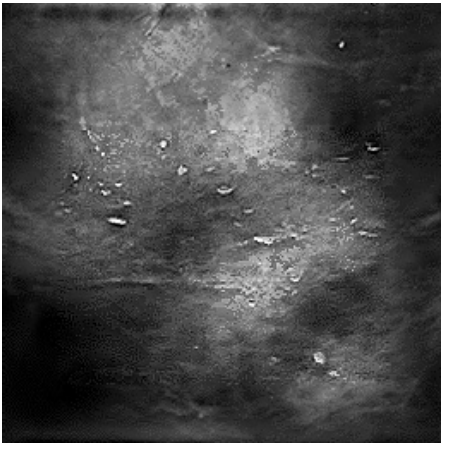
Original



Enhanced



Original



Enhanced

## 4 Conclusion

The detection of microcalcifications and the control of the correctness of the result have been achieved using combinations of two different wavelet transforms and of three different techniques of detection of contours. The comparison of the results obtained with two different wavelet transforms and the combination of them can be considered as an experimental way of checking the results. Optimisation of the procedure involved an appropriate choice of the wavelet transforms, the mother wavelet and the reconstruction levels. The quality of the reconstructed images has been checked at the Cancer Hospital of Algiers and a user-friendly interface to the image-reconstruction software has been devel-

oped using the C++ language. This has been installed on a PC computer at the hospital. A wavelet transform analysis of 30 mammograms has been made to determine optimum procedures for the positive identification of microcalcification structures of Types I-V. In order to obtain a realistic assessment of the numbers and types of prompts, the system could be expected to produce in a clinical setting. The resulting images were analysed by the computer system and a radiologists was asked to label the apparent cause of each of the prompts which were produced. This has allowed us to quantify some of the problems that will be faced by any group of researchers who aim to produce microcalcification detection software for mass screening clinics.

**Acknowledgement.** The authors wish to thank Pr. Bendib, Dr Aoudia, Dr Meziani from Algiers Hospital " Mustapha Bacha - Service CPMC " for their interest in the present framework and in particular for their suggestions in leading the various tests, which helped us choose and improve the wavelet parameters retained. We also remain grateful to Pr. Jean Vignes from LIP6 laboratory for his fruitful comments in various ways.

## References

1. Burbenne,L., Goldberg,F., Hislop,T., Worth,A.J., Rebbeck,P.M., Kan,L. : Interval breast cancers in the screenning mammography program of British Columbia: Analysis and classification. AJR, 162:1067-1071, 1994.
2. Bird,R.G., Wallace,T.W., Yankaskas,B.C.: Analysis of cancers missed at screening mammography Radiology, 184:613-617, 1992.
3. Feig,S.A.: Decreased cancer mortality through mammographic screening : Results of clinical trials Radiology 167, 659-665 , 1988.
4. Yoshida,H., Doi,K., Nishikawa ,R.M.,Giger,M.L., Schmidt ,R.A. : An Improved CAD Scheme Using Wavelet Transform for Detection of Clustered Microcalcifications in Digital Mammograms Academic Radiology, 3: 621-627, 1996.
5. Chan H.P., Doi K., Vyborny, C.J., Schmidt,R.A., Metz,C.E.,Lam K.L., Ogura T., Wu Y.,MacMahon H. : Improvement in radiologists' detection of clustered microcalcifications on mammograms : The potential of computed-aided diagnosis. Invest Radiol 1990; 25:1102-1110.
6. Nishikawa,R.M., Jiang,Y., Giger,M.L.,and al : Performance of Automated CAD schemes for the Detection and Classification of Clustered Microcalcifications. In: Gale AG, Ashley SM, Dance DR, Cairns AY eds. Digital Mammography. Amsterdam, Elsevier Science, 1994; pp. 13-20.
7. Karssemeijer, N. : A stochastic method for automated detection of microcalcifications in digital mammograms Information Processing in Medical Imaging, Springer-Verlag, New York, 1991; 76:227-238.
8. Yoshida,H., Doi,K.,and Nishikawa,R.M. : Automated detection of clustered microcalcifications in digital mammograms using wavelet transform techniques. Proc. SPIE, 2167:868-886, 1994.
9. Yoshida,H., Doi,K., Nishikawa,R.M., Muto,K., Tsuda,M. : Application of the wavelet transform to automated detection of clustered microcalcifications in digital mammograms. Acad. Reports of Tokyo Institute of Polytecnics 16: 24-37, 1994.

10. Clarke,L.P., Kallergi,M., Qian,W., Li,H.D., Clar,R.A., Silbiger,M.L. : Tree structured nonlinear filter and wavelet transform for microcalcification segmentation in digital mammography. Cancer Letters 1994; 77:173-181.
11. Gal,M.L., Chavanne,G., Pellier,D. : Valeur diagnostique des microcalcifications groupées découvertes par mammographies (á propos de 227 cas avec vérification histologique et sans tumeur palpable, Cancer (paris) Masson, 71(1):57Bull, 1994.
12. Mallat S. A theory for multiscale signal decomposition: The wavelet representation IEEE Trans. On Pattern and Machine Intelligence, 1989; 11:674-693.
13. Mallat S. Multiresolution approximations and wavelet orthonormal bases of L2(IR) Trans. AMS, Vol. 315, No. 1, pp. 69-87, 1989.
14. Cohen,A., Daubechies,I., Feauveau,J. : Bi-orthogonal bases of compactly supported wavelets. Comm. Pure Appl. Math., 45:485-560, 1992.
15. Daubechies,I. : Ten Lectures on Wavelets CBMS-NSF Regional Conference Series in Applied Mathematics, No 61, SIAM, Philadelphia, PA, 1992.
16. Daubechies,I. : The wavelet transform, time-frequency localization and signal analysis IEEE tans. Inf. Theory, Vol. 36, No 5, pp. 961-1005, 1990.
17. Oussena,B., Belhadj,F., Hebboub,W. : Outil de détection de Masses Cancéreuses du sein par une Méthode d'Analyse de Texture d'images mammographiques digitales. CARI2002, 391-398. Act du 5eme colloque sur la recherche en informatique Octobre 2000, Antananarivo, Madagascar.

# Using PVS to Validate the Inverse Trigonometric Functions of an Exact Arithmetic

David Lester

Department of Computer Science
Manchester University
Manchester M13 9PL, UK.

**Abstract.** This paper outlines the PVS development for the inverse trigonometric functions: $atan(x)$, $asin(x)$ and $acos(x)$. This is then used to validate exact arithmetic algorithms based on fast binary cauchy sequences [14,17] for these functions in a further PVS development.
The principle results of using PVS in this process is the detection of four errors in an implementation that had previously been believed to be correct. In addition, an error was detected in the handbook of formulæ used (Abramowitz and Stegun, Formula 4.4.34).

## 1 Introduction

This is not a paper about *Mathematics*. The mathematics presented here is essentially trivial, indeed it is probably of a pre-college level. Instead, the focus is on a branch of *Software Engineering*; in particular using *Formal Methods* to validate algorithms [2,3]. Instead of testing an algorithm on a test set, a formal proof is constructed so that we are assured that the algorithm is correct for all possible input data. The problems encountered are usually trivial: a loop is executed once too often, or once less than needed, a variable was not updated correctly *etc.* In other words: conducting a formal proof forces a programmer to think more carefully about their program, and usually these trivial errors are detected and corrected. Note that the programming process is extremely non-linear: a small error in the program will usually result in a massive failure of the program to perform correctly.

However, an unpleasant thought occurs: if a programmer is capable of making mistakes when programming, why should they make fewer mistakes when they attempt a mathematical proof? Why might there not be the same "out by one" errors in the proofs as there are in programs? The worst part about these essentially trivial errors is that they permit programs to be written incorporating these errors, *and in general*, detecting these mistakes in programs can be very hard. For example, in the area addressed by this paper of Exact Arithmetic, there is currently no reference implementation against which one might compare answers for a 10,000 digit evaluation of an arithmetic expression. The aim of the work presented here is to provide such a reference implementation for inverse trigonometric functions capable of working to any chosen accuracy, and guaranteeing the accuracy of the answer.

R. Alt et al. (Eds.): Num. Software with Result Verification, LNCS 2991, pp. 259–273, 2004.

Recently, increases in the power of computers and advances in mechanical theorem provers or proof assistants have reached the stage where it is possible to have real – rather than toy example – formal methods proofs checked by computer for validity. This paper is therefore an exploration of the feasibility of using one of these tools (PVS) to address the algorithms needed for an Exact Arithmetic. In addition the number of errors detected, and their nature is reported.

In detail this paper briefly outlines a PVS development for the inverse trigonometric functions: $atan(x)$, $asin(x)$ and $acos(x)$. This is then used to validate exact arithmetic algorithms based on fast binary cauchy sequences [14,17] for these functions in a further PVS development. The principle results of using PVS in this process is the detection of four errors in an implementation that had previously been believed to be correct. In addition, errors were detected in the NASA library of PVS theorems and an error was even detected in the handbook of formulæ used [1]. Although disconcerting, this proved to have no impact on the validation of the algorithms.

In this paper our interest lies in minimizing the precision required for each argument to an operation whilst still ensuring that the answer for the operation is accurate.

## 2 PVS: Prototype Verification System

"The Prototype Verification System (PVS) provides an integrated environment for the development and analysis of formal specifications, and supports a wide range of activities involved in creating, analyzing, modifying, managing and documenting theories and proofs." So says the PVS system guide[12]. The system is based on a typed higher-order logic, and provides a close match to what is required to prove properties of functional programs, such as those written in Haskell. For our purposes, we will be interested in the way recursive functions are defined and in how "proof" is conducted.

In PVS a non-recursive function $f$ from type $T_1$ to $T_2$ would be defined as follows:

$$f(x : T_1) : T_2 = E$$

It would be normal to expect the variable to occur in the expression $E$, but this is not required. In contrast a recursive function would be defined as:

$$f(x : T_1) : \text{RECURSIVE } T_2 = E \text{ MEASURE } g$$

where $g$ is a function from the type $T_1$ to the natural numbers, that 'measures' the depth of recursive function calls that will be required to evaluate the function $f$ applied to a particular argument.

Conducting a proof in PVS involves manipulating a logical term of the form $A_1 \wedge A_2 \wedge \ldots \wedge A_n \Rightarrow B$, using very simple steps. The most commonly used ones are now shown.

`(skolem! -`$i$`)` 'Skolemizes' the $i$-th antecedent formula. That is, replaces an existentially quantified variable $x$ with a constant $x'$.

$$(A_1 \wedge \ldots \wedge (\exists(x:T):P(x)) \wedge \ldots \wedge A_n \Rightarrow B) \equiv \\ (A_1 \wedge \ldots \wedge P(x') \wedge \ldots \wedge A_n \Rightarrow B)$$

`(skolem! 1)` 'Skolemizes' the consequent formula. That is, replaces a universally quantified variable $x$ with a constant $x'$.

$$(A_1 \wedge \ldots \wedge A_n \Rightarrow (\forall(x:T):P(x)) \equiv (A_1 \wedge \ldots \wedge A_n \Rightarrow P(x'))$$

`(inst -`$i$` "`$E$`")` Instantiates the $i$-th antecedent formula with expression $E$, *i.e.* replaces a universally quantified variable $x$ with the expression $E$.

$$(A_1 \wedge \ldots \wedge (\forall(x:T):P(x)) \wedge \ldots \wedge A_n \Rightarrow B) \equiv \\ (A_1 \wedge \ldots \wedge P(E) \wedge \ldots \wedge A_n \Rightarrow B)$$

It will generate an extra type-check condition (TCC) – that the expression $E$ has type $T$ – and this will need to be proved:

$$(A_1 \wedge \ldots \wedge (\forall(x:T):P(x)) \wedge \ldots \wedge A_n \Rightarrow (T\_\mathrm{pred}(E) \vee B))$$

It can sometimes be difficult to spot which of the consequent formulæ should be addressed; the temptation is to continue trying to prove $B$!

`(inst 1 "`$E$`")` Instantiates the consequent formula with expression $E$. That is, replaces an existentially quantified variable $x$ with the expression $E$.

$$(A_1 \wedge \ldots \wedge A_n \Rightarrow (\exists(x:T):P(x))) \equiv (A_1 \wedge \ldots \wedge A_n \Rightarrow P(E))$$

It will generate an extra type-check condition (TCC) – that the expression $E$ has type $T$ – and this will need to be proved:

$$(A_1 \wedge \ldots \wedge A_n \Rightarrow T\_\mathrm{pred}(E))$$

`(lemma "L")` Introduces a previously proved theorem `L` whose formula is $L$.

$$(A_1 \wedge \ldots \wedge A_n \Rightarrow B \equiv (L \wedge A_1 \wedge \ldots \wedge A_n \Rightarrow B$$

`(flatten -`$i$`)` "Flatten"s the $i$-th antecedent formula.

$$(A_1 \wedge \ldots \wedge (A_i^1 \wedge A_i^2) \wedge \ldots \wedge A_n \Rightarrow B) \equiv \\ (A_1 \wedge \ldots \wedge A_i^1 \wedge A_i^2 \wedge \ldots \wedge A_n \Rightarrow B)$$

`(split -`$i$`)` "Split"s the $i$-th antecedent formula.

$$(A_1 \wedge \ldots \wedge (A_i^1 \vee A_i^2) \wedge \ldots \wedge A_n \Rightarrow B)$$

now becomes two separate proofs: $(A_1 \wedge \ldots \wedge A_i^1 \wedge \ldots \wedge A_n \Rightarrow B)$ and $(A_1 \wedge \ldots \wedge A_i^2 \wedge \ldots \wedge A_n \Rightarrow B)$. This can also be used to simplify an antecedent formula of the form $A^1 \Rightarrow A^2$.

`(copy -`$i$`)` Copies the $i$-th antecedent formula.

$$(A_1 \wedge \ldots \wedge A_n \Rightarrow B) \equiv (A_i \wedge A_1 \wedge \ldots \wedge A_n \Rightarrow B)$$

This is useful when one wishes to instantiate an existentially quantified formula with different values.

`(hide -`$i$`)` Hides the $i$-th antecedent formula.

$$\begin{aligned}(A_1 \wedge \ldots \wedge A_{i-1} \wedge A_i \wedge A_{i+1} \ldots \wedge A_n \Rightarrow B) \equiv \\ (A_1 \wedge \ldots \wedge A_{i-1} \wedge A_{i+1} \wedge \ldots \wedge A_n \Rightarrow B)\end{aligned}$$

This should be used cautiously because although it's possible to find hidden formulæ again, the proofs are less robust if the PVS system is upgraded (and formula's numbers are changed). It is used when the list of antecedent clauses gets too large to be easily handled; it can become complicated keeping track of more than about 25–30 formulæ.

`(assert)` Simplifies all of the formulæ, substitutes constants, and checks whether the consequent has been proved.

`(expand "f")` Expands a function definition; if the function is recursive, it is expanded just once. We can control where this expansion takes place by specifying a formula (*i.e.* `-1`) and the occurence in the formula. For example: `(expand "f" -1 2)` would expand the following definition of `f(x:real):real = x+1`:

$$(f(3) + f(5) + f(7) = 18) \equiv (f(3) + (5 + 1) + f(7) = 18)$$

`(induct "n")` Performs an induction on the universally quantified consequent over the natural numbers. The term

$$(A_1 \wedge \ldots \wedge A_n \Rightarrow (\forall (n : \text{nat}) : P(n)))$$

requires that two proofs are undertaken: $(A_1 \wedge \ldots \wedge A_n \Rightarrow P(0))$ and $(\forall (j : \text{nat}) : j \leq n' \Rightarrow P(j)) \wedge A_1 \wedge \ldots \wedge A_n \Rightarrow P(n' + 1))$

`(case "`$A$`")` Introduces a "case" of the proof: $(A_1 \wedge \ldots \wedge A_n \Rightarrow B)$. It requires that two proofs are undertaken: $(A \wedge A_1 \wedge \ldots \wedge A_n \Rightarrow B)$ and $(A_1 \wedge \ldots \wedge A_n \Rightarrow (A \vee B))$ This can be useful when $A$ is universally quantified and multiply instantiated; it is vital if we intend to use induction to prove $A$.

There are a number of other commands in the system, and the ones shown are in fact slightly more general than I have indicated. For example `split` will split a proof into $n$ cases if the antecedent is of the form $C_1 \vee \ldots \vee C_n$.

## 3 The Representation

### 3.1 Mathematics

At the heart of our implementation is the representation of a computable real number as *Fast Binary Cauchy Sequence*. It is *fast* because we have an implicit

modulus of convergence function which is the identity function; it is *binary* because the denominator of the $p$-th element of the sequence is always $2^p$. This means that we do not need to store the denominator, and *critically for space efficiency* the size of the numerator grows linearly with the precision of the stored real number.

**Definition 1**

A computable real number $x$ is represented as a *Fast Binary Cauchy Sequence* if there is an infinite computable sequence of integers $\{n_0,\ n_1,\ \ldots,\ n_p,\ \ldots\}$, such that

$$|x - 2^{-p} n_p| < 2^{-p}$$

This definition has appeared in many papers [4,9,14]. [8] states it to be the most favourable definition for calculating the terms of a sequence to any accuracy desired.

As the main result of this section, we now show how our representation is related to the effective Cauchy representation [4,8,13,14,15,16,17] of the Computable Reals.

**Definition 2**

A computable real number $x$ is represented as an *Effective Cauchy Sequence* if there is an infinite computable sequence of rationals $\{\frac{n_0}{d_0},\ \frac{n_1}{d_1},\ \ldots,\ \frac{n_p}{d_p},\ \ldots\}$, with $d_i > 0$, and a *modulus of convergence function* $e : \mathbb{N} \to \mathbb{N}$ which is recursive, such that for all $p \in \mathbb{N}$:

$$k \geq e(p) \text{ implies } \left| x - \frac{n_k}{d_k} \right| < 2^{-p}$$

**Theorem 3**

*Any computable real $x$ represented as an Effective Cauchy Sequence ($\acute{x}$), with modulus of convergence function $e$, can be converted into the Fast Binary Cauchy Sequence ($\hat{x}$) and vice versa. This interconversion being effective in both directions.*

**Proof**

We will take a sequence to be a function from the naturals to the target type. Therefore, the effective cauchy representation of a computable real $x$ is a tuple, consisting of a function $f : \mathbb{N} \to \mathbb{Q}$ and a modulus of convergence $e : \mathbb{N} \to \mathbb{N}$, whereas the fast binary cauchy sequence for $x$ is a function $g : \mathbb{N} \to \mathbb{Z}$. Now, given the tuple $(f,\ e)$ we can construct $g$ as

$$g(n) = \lfloor f(e(n+2))/4 \rceil$$

and if given the function $g$, we can construct the tuple $(f,\ e)$ as:

$$\begin{aligned} f(n) &= g(n)/2^n \\ e(n) &= n \end{aligned}$$

These will satisfy the criteria laid out in Definitions 1 and 2. □

Although this proof is straightforward, there is one important point: to convert a computable real represented as an Effective Cauchy Sequence we need one more bit of precision than we might have naïvely expected so that the rounding to a denominator that is a power of two does not lose accuracy.

### 3.2 The PVS Representation

In our arithmetic package, we represent computable real numbers as fast binary cauchy sequences. In PVS these sequences are most easily represented as functions $c$, with the property that for all desired precisions $p$ of the answer the (computable) real number $x$ satisfies:

$$(c(p) - 1)2^{-p} < x < (c(p) + 1)2^{-p}.$$

The precisions are taken to be natural numbers, and the function $c$ returns integers. This is what is defined by the function cauchy_prop in Theory cauchy. There is however a constraint on the acceptable functions of this form: the rational approximations $c(p)/2^p$ must be converging to a particular real number $x$. The predicate cauchy_real? captures this property. Finally, the type cauchy_real can be defined as the set of functions satisfying the predicate cauchy_real?, which happens to be nonempty, because $(\lambda p : 0)$ represents the real number 0.

```
cauchy: THEORY
 BEGIN
  ...
  cauchy_prop(x:real, c:[nat → int]) : bool
      = ∀(p:nat):   c(p) − 1 < x × 2^p ∧ x × 2^p < c(p) + 1
  cauchy_real?(c:[nat → int]):   bool
      = ∃(x:real):   cauchy_prop(x, c)
  cauchy_real: NONEMPTY_TYPE
      = (cauchy_real?) CONTAINING (λp:   0)
  ...
 END cauchy
```

The elided part of the theory file defines subsets of the cauchy_reals to match the subsets of the reals in PVS – for example positive reals, nonnegative reals *etc.* – and involves a great deal of repetition.

## 4 Validating the Algorithms

### 4.1 mul2n: An Easy Example

To give a feel for the way in which proof is conducted in PVS, we give a simple example. The PVS file (in simplified form) is

```
shift: THEORY
 BEGIN
  mul2n(x : real, n : nat) : real = x × 2^n
```

cauchy_mul2n(cx : cauchy_real, $n$ : nat) : cauchy_real
$= (\lambda p : \mathrm{cx}(p+n))$
lemma_mul2n : LEMMA cauchy_prop$(x, \mathrm{cx}) \Rightarrow$
cauchy_prop(mul2n$(x, n)$, cauchy_mul2n(cx, $n$))
END shift

The function mul2n multiplies a real by $2^n$; cauchy_mul2n performs the equivalent operation on cauchy_reals. This equivalence is established by proving lemma lemma_mul2n, which we now do.

`lemma_mul2n:`

{1} $\forall$ (cx: cauchy_real, $n$: nat, $x$: real): cauchy_prop($x$, cx) $\Rightarrow$
cauchy_prop(mul2n($x$, $n$), cauchy_mul2n(cx, $n$))

Expanding the definitions of cauchy_prop, cauchy_mul2n and mul2n,

`lemma_mul2n:`

{1} $\forall$ (cx: cauchy_real, $n$: nat, $x$: real):
($\forall$ ($p$: nat): $\mathrm{cx}(p) - 1 < x \times 2^p \wedge x \times 2^p < 1 + \mathrm{cx}(p)$) $\Rightarrow$
($\forall$ ($p$: nat): $\mathrm{cx}(n+p) - 1 < x \times 2^n \times 2^p \wedge$
$x \times 2^n \times 2^p < 1 + \mathrm{cx}(n+p)$)

Repeatedly Skolemizing and flattening,

`lemma_mul2n:`

{-1} $\forall$ ($p$: nat): $\mathrm{cx}'(p) - 1 < x' \times 2^p \wedge x' \times 2^p < 1 + \mathrm{cx}'(p)$

{1} $\mathrm{cx}'(n'+p') - 1 < x' \times 2^{n'} \times 2^{p'} \wedge x' \times 2^{n'} \times 2^{p'} < 1 + \mathrm{cx}'(n'+p')$

Instantiating the top quantifier in {-1} with the terms: $n'+p'$,

`lemma_mul2n:`

{-1} $\mathrm{cx}'(n'+p') - 1 < x' \times 2^{n'+p'} \wedge x' \times 2^{n'+p'} < 1 + \mathrm{cx}'(n'+p')$

{1} $\mathrm{cx}'(n'+p') - 1 < x' \times 2^{n'} \times 2^{p'} \wedge x' \times 2^{n'} \times 2^{p'} < 1 + \mathrm{cx}'(n'+p')$

Applying expt_plus where n0x gets 2, i gets $n'$, j gets $p'$,

`lemma_mul2n:`

{-1} $2^{n'+p'} = 2^{n'} \times 2^{p'}$

{-2} $\mathrm{cx}'(n'+p') - 1 < x' \times 2^{n'+p'} \wedge x' \times 2^{n'+p'} < 1 + \mathrm{cx}'(n'+p')$

{1} $\mathrm{cx}'(n'+p') - 1 < x' \times 2^{n'} \times 2^{p'} \wedge x' \times 2^{n'} \times 2^{p'} < 1 + \mathrm{cx}'(n'+p')$

Trying repeated skolemization, instantiation, and if-lifting, This completes the proof of `lemma_mul2n`. Q.E.D.

Before validating the remaining algorithms, we will need definitions of the operations on the reals. We have two alternatives. Firstly, it would be possible

to provide an axiomatic definition; as we shall see this approach has dangers. The second approach is to prove from first principles the required properties of the operations.

### 4.2 atan: The Difficult Case

*Definition of atan.* So that we can show that atan is a bijection (and hence invertible) we will need to restrict it's range to the interval $(-\pi/2,\ \pi/2)$. Initially we define it's value as:

$$\text{atan\_value}(x : \text{real}) : \text{real} = \int_0^x \frac{1}{1+x^2} dx$$

From this, once we have established limits on the range of atan_value, we can define

$$\text{atan}(x : \text{real}) : \{y : \text{real} \mid |y| < 2\text{atan\_value}(1)\} = \text{atan\_value}(x)$$

*Subtraction: The pitfalls of using published formulæ.* One might imagine that one could safely incorporate theorems from handbooks of mathematical theorems, such as Abramovitz and Stegun [1]. Let this be a warning. Formula 4.4.34 gives:

$$\text{Arctan } z_1 \pm \text{Arctan } z_2 = \text{Arctan } \left(\frac{z_1 \pm z_2}{1 \mp z_1 z_2}\right)$$

The first obvious thing wrong with Formula 4.4.34 is that it is undefined when denominator $1 \mp z_1 z_2$ is 0. Certainly one could define $\text{Arctan}(\infty)$ as $\pi/2$, but in PVS one would be inclined to define Arctan as a function from the reals to the open interval $(-\pi/2,\ \pi/2)$.

On a very careful inspection of Abramovitz and Stegun this formula is defining the relationship between the *sets* of inverse values, not the relationship between the principle values. This is the almost undocumented distinction between arctan and Arctan. Of course it is still wrong, because it fails to specify that $1 + z_1 z_2 \neq 0$.

A great deal of time was wasted attempting to prove this theorem. As one of the reviewer's has pointed out I ought to have made a check that the theorem was correct before attempting a proof. All I can say is that, at the time, it would have seemed to be needless paranoia! Eventually a check was made (ironically using the calculator being proved correct) and the following, corrected, formula was substituted.

atan_minus: LEMMA $\forall(x, y : \text{real})$:

$$\begin{aligned}
&(-1 < xy \quad \Rightarrow\ \text{atan}(x) - \text{atan}(y) \quad = \text{atan}\left(\tfrac{x-y}{1+xy}\right)) \wedge \\
&(xy < -1 \wedge y > 0 \Rightarrow\ \text{atan}(x) - \text{atan}(y) + \pi = \text{atan}\left(\tfrac{x-y}{1+xy}\right)) \wedge \\
&(xy < -1 \wedge y < 0 \Rightarrow\ \text{atan}(x) - \text{atan}(y) - \pi = \text{atan}\left(\tfrac{x-y}{1+xy}\right))
\end{aligned}$$

*Recurrence relation between successive derivatives.* The longest proof in the PVS development involves showing that there is a recurrence relation between the $2n+1$-st and $2n+3$-rd derivative of atan. Note that

$$\frac{d^{2n+1}\text{atan}(x)}{dx^{2n+1}} = (-1)^n \frac{\sum_{i=0}^{n}(2n)!\ C_{2i}^{2n+1}(-x^2)^i}{(1+x^2)^{2n+1}},$$

where

$$C_m^n = \frac{n!}{m!(n-m)!}$$

The relation that needs to be proved is:

$$\frac{d^2}{dx^2}\left(\frac{\sum_{i=0}^{n}(2n)!\ C_{2i}^{2n+1}(-x^2)^i}{(1+x^2)^{2n+1}}\right) = -\frac{\sum_{i=0}^{n+1}(2n+2)!\ C_{2i}^{2n+3}(-x^2)^i}{(1+x^2)^{2n+3}}$$

Eventually – at somewhere near the internal limits of PVS – all of the terms can be persuaded to cancel out.

*Taylor's Theorem for atan.* It is now relatively straightforward to prove that the atan function and it's series satisfy Taylor's Theorem:

atan_taylors: LEMMA $\forall(x : \text{real}, n : \text{nat})$: $\exists(c : \text{between}(0, x))$:
$\text{atan}(x) = \sum_{i=0}^{n} \frac{(-1)^i}{2i+1}x^{2i+1} + \left(\frac{d^{2n+3}}{dx^{2n+3}}atan(x)\right)(c)\frac{x^{2n+3}}{(2n+3)!}$

As mathematicians we'd be inclined to argue that the error term is "obviously" less than $x^{2n+3}/(2n+3)$, since the series is alternating. However, this is *not* what the theorem says: it is couched in terms of the value of the $2n+3$-rd derivative of atan over the interval $(0,\ x)$ (or $(x,\ 0)$, if $x < 0$). Graphing this function shows it oscillates wildly over the interval $(0,\ 1)$.

*Harmonic polynomials.* Initially, it appears that proving the bound on the error term will be easy. All we have to do is show that

$$|\sum_{j=0}^{n} C_{2j}^{2n+1}(-x^2)^j| \le (1+x^2)^{2n+1}.$$

Observing that

$$(1+x^2)^{2n+1} = \sum_{j=0}^{2n+1} C_j^{2n+1}\ x^{2j},$$

one makes the claim that

$$C_{2j}^{2n+1}\ x^{2j} \le C_{2j}^{2n+1}\ x^{4j}.$$

And this is usually true, because $x \le x^2$ most of the time, *i.e.* whenever $x \le 0$ or $1 \le x$. The key observation turns out to be:

$$\sum_{j=0}^{n} C_{2j}^{2n+1}(-x^2)^j = \Re\left(\sum_{j=0}^{2n+1} C_j^{2n+1} i x^j\right) = \Re\left((1+ix)^{2n+1}\right)$$

It is now easy to see why this function oscillates for small values of $x$. Because the real part of a complex number must be less than or equal to it's modulus, we're able to show that:

$$\left|\sum_{i=0}^{n} C_{2i}^{2n+1}(-x^2)^i\right| \leq (1+x^2)^{n+1/2} \leq (1+x^2)^{2n+1},$$

as required. Note that this time the term being squared is greater than or equal to one, and so the final inequality is justified.

*Convergence of the series for atan.* Our final result is that:

atan_series: LEMMA $\forall(x : \text{real}, n : \text{nat})$:

$$\left|\text{atan}(x) - \textstyle\sum_{i=0}^{n} \frac{(-1)^i}{2i+1} x^{2i+1}\right| \leq \frac{|x^{2n+3}|}{2n+3}$$

*Validating the atan algorithm.* Because the sort of software errors I usually find remaining my programs are trivial, it should come as no surprise that the main result of the is paper is the algorithm is correct. This elides the fact that there were in fact *four* errors detected as a result of the PVS validation process. The main function `atan` can be applied to the full range of computable reals. A range reduction ensures that `atan_dr` is only applied to values of $x$ in the range $-1 < x < 1$, a further range reduction ensures that *atan*_drx is only applied to values of $x$ in the range $-1/\sqrt{2} < x < 1/\sqrt{2}$. Finally, the series expansion is performed by the function `atan_drxx`. The final form of the algorithm written in HASKELL is.

```
instance Floating CR where
  atan x  = if t <  -4 then atan_dr (negate (recip x)) - pi/2 else
            if t == -4 then -pi/4 - atan_dr (xp1/xm1)         else
            if t <   4 then atan_dr x                         else
            if t ==  4 then pi/4 + atan_dr (xm1/xp1)          else
            {- t >   4 -}   pi/2 - atan_dr (recip x)
            where (CR_ x') = x; t = x' 2
                  xp1 = x+fromInteger 1; xm1 = x-fromInteger 1
  asin x  = if t == -1 then -pi/2 - atan(s/x) else
            if t ==  0 then atan(x/s)         else
                            p/2 - atan(s/x)
            where (CR_ x') = x; t = x' 0; s = sqrt(1-x*x)
  acos x  = pi/2 - asin x

atan_dr :: CR -> CR
atan_dr x = if t < -4 then atan_drx n - pi/6 else
            if t <  5 then atan_drx x        else
                           atan_drx p + pi/6
             where (CR_ x') = x; t = x' 3
                   n = (x*sqrt 3+fromInteger 1)/(sqrt 3-x)
                   p = (x*sqrt 3-fromInteger 1)/(sqrt 3+x)
```

```
pi :: CR
pi = 16*atan_drx(fromRational(1%5))
                                  -4*atan_drx(fromRational(1%239))

atan_drx :: CR -> CR
atan_drx x = x * atan_drxx (x*x)

atan_drxx :: CR -> CR
atan_drxx = CR_ (\p -> round_uk(x' (p+2)%4))
            where (CR_ x') = power_series [(-1)^n%(2*n+1)|
                                                  n <- [0..]] (+2)
```

Originally, two lines were incorrect:

```
  atan x = if t < -5 then atan_dr (negate (recip x)) - piBy2 else
...
atan_drx = power_series ss (+1)
```

The first error (having $-5$ instead of $-4$) should have resulted in errors occuring when $x$ lies between $-1.5$ and $-1$ and has the wrong representation. This should have resulted in the power series calculation taking place outside of it's expected radius of convergence; no evidence of this was ever observed, possibly because for the large precisions at which the package was used, there was more than sufficient accuracy and terms to get the right answer.

For the same reason (conservative estimates of accuracy and number of terms required, and insufficient testing at low precisons), the need for one extra term in the series and two extra bits of accuracy also remained undetected.

Finally, as a result of rechecking the reviewers' comments, a further stage of range reduction was performed.

### 4.3 Asin and Acos

For the purposes of this paper little work needs to be done, as the chosen definitions of asin and acos closely match those used in the algorithms. We have defined asin as:

$$\text{asin}(x : \text{real\_abs\_le1}) : \text{real} =$$
$$\text{IF } |x| = 1 \text{ THEN } x \times \pi/2 \text{ ELSE } \text{atan}(x/\sqrt{1-x^2}) \text{ ENDIF}$$

Given the theorems developed about the properties of atan, proving lemma_asin is straightforward. Since we have defined acos in both definitions as:

$$\text{acos}(x) = \pi/2 - \text{asin}(x)$$

it's validation is even easier.

Lets just summarize what has been proved (and mechanically checked by PVS):

$$\text{lemma\_atan} : \text{LEMMA}\text{cauchy\_prop}(x, \text{cx}) \Rightarrow$$
$$\text{cauchy\_prop}(\text{atan}(x), \text{cauchy\_atan}(\text{cx}))$$

That is, if $x$ is any computable real number which is represented as a fast binary cauchy sequence cx, then we can calculate a fast binary cauchy sequence cauchy_atan(cx) which represents the computable real atan($x$). In particular, if $x = 1$ and cx = (LAMBDA($p$ : nat) : $2^p$) then we could in principle calculate a rational approximation to $\pi$, accurate to $10^{100}$ decimal places as:

$$(\text{cauchy\_atan}(\text{cx})(4 \times 10^{100}))/(2^{4(1+10^{100})}10^{100})$$

In practice, we will need to await machines powerful enough to perform this calculation in a reasonable period of time. The important point is that our concerns about the reliability of the result must lie in only two places: firstly, did PVS make any logical slips during the proof checking, and secondly, does the implementation of the programming language HASKELL faithfully match it's formal semantics? We need have no concerns about the particular algorithm (cauchy_atan in this case) under consideration.

Similar properties have been established for *asin* and *acos*.

## 5 Remarks

### 5.1 Bugs

Although the final result – that the representation and its associated operations are correct – comes as no surprise, nevertheless, completion of this proof led to a number of small changes to the implementation. All of these bugs had the ability to cause arbitrarily large errors in certain (contrived) examples. For the morbid, we present the changes in tabular form in Table 1.

**Table 1.** Bugs found

| Function | Originally | Found |
|---|---|---|
| cauchy_atan | IF$t < -5$THEN ... | PVS |
| cauchy_atan_drx | More Range Reduction | PVS |
| cauchy_atan_drxx | cauchy_powerseries(cx, ss, $p$) | PVS |
| cauchy_atan_drxx | cauchy_powerseries(cx, ss, $p$)($p$) | PVS |

The work most closely related to this paper is that of Valérie Ménissier-Morain [7], in which she proved slightly more general results (in base $b$ rather than just base 2). Our approach to implementing the transcendental functions also differs in that in her work, several functions are implemented separately: *e.g.* exp($x$), sin($x$), *etc.* In our work, we have instead define a power series algorithm and use this to construct our transcendental functions.

Müller's iRRAM [11,10] can also be used to evaluate expressions to any accuracy. However when performing calculations the iRRAM usually checks the error bounds of the result, then if they have grown beyond $2^{-p}$ the result is thrown out and the calculation repeated to a greater accuracy. A comparison of the approaches can be found in [5].

### 5.2 PVS

To give some sense of the scale of the task undertaken we tabulate the sizes of the files in the development in Table 2. Much of the work in reals and analysis was written by the NASA group. The theories developed in series and trig are a mixture of NASA work with my own additions. The cong directory is my own work and consists of validations relative to the NASA libraries. The number of theorems is – as expected the number of individual theories that were proved correct. Stating these theories often extends over a number of lines, and the final column shows how much effort is required to prove the theories correct. To a first approximation, each line in the proof files consists of one of the proof transformations described in Section 2.

**Table 2.** Size of the files used to validate the Exact Arithmetic

| Directory | Theorems | Lines in Theorems | Lines of Proof |
|---|---|---|---|
| reals | 333 | 2061 | 9044 |
| analysis | 396 | 3893 | 34946 |
| series | 130 | 929 | 18007 |
| trig | 436 | 1665 | 69117 |
| cong | 355 | 2217 | 38350 |
| **Totals** | 1650 | 10765 | 169464 |

There are any number of frustrations working with PVS! Perhaps the most ubiquitous is it's inability to directly deduce:

$$x > 0 \ \wedge \ y > 0 \Rightarrow x \times y > 0$$

At least half of the picky details could be solved with this simple strategy. It's also probably optimistic to expect PVS to deduce that:

$$x < k_1 \ \wedge \ y < k_2 \Rightarrow x \times y < k_1 \times k_2$$

since we'd need to know that $x \geq 0$ and $y \geq 0$, but this feature would be nice.

The full PVS development can be found by following the links from:

`http://www.cs.man.ac.uk/arch/dlester/exact.html`

### 5.3 Mathematics

The most interesting consequence of using the theorem prover is in showing just how often a manual proof goes wrong! Here are two particularly pernicious examples. In a hand proof I find it difficult to resist converting $z/y < x$ to $z < x \times y$, but of course this is only valid when $y > 0$. The other "factoid" that is easily assumed is that: $x < x^2$, this is of course true most of the time; but unfortunately not when $0 \leq x \leq 1$.

Perhaps more interesting is just how often the "obvious" proof of some fact involves some sort of circular argument. It is all too tempting to use a general theorem to prove some subsidiary lemma, and to have the general theorem turn out to require the lemma to be proved first.

For example, one way to show that $1 = \sin^2(x) + \cos^2(x)$ would be to show that

$$0 = \frac{d}{dx}(1) = \frac{d}{dx}\left(\sin^2(x) + \cos^2(x)\right) = \sin(x)\cos(x) - \sin(x)\cos(x) = 0,$$

and then calculate the constant of integration, which is zero. However, we almost certainly want to know that $1 = \sin^2(x)+\cos^2(x)$ before proving properties about the derivatives of $\sin(x)$ and $\cos(x)$. Many examples of this have occurred.

## References


1. M. Abramovitz, I.A. Stegun, Handbook of Mathematical Functions (Ninth Edition), Dover Publications, New York, 1972.
2. E.W. Dijkstra, A Discipline of Programming, Prentice-Hall, Englewood Cliffs, 1976.
3. C.A.R. Hoare, An Axiomatic Basis for Computer Programming. Communications of the ACM, 12(10):576–580, October 1969.
4. K.-I. Ko, On the definitions of some complexity classes of real numbers, Math. Systems Theory 16 (1983) 95–109.
5. V. A. Lee Jr., H.-J. Boehm, Optimizing programs over the constructive reals, in: Proceedings of the ACM SIGPLAN'90 conference on Programming Language design and implementation, 1990, pp. 102–111.
6. D. R. Lester, P. Gowland, Using PVS to validate the algorithms of an exact arithmetic, in: TCS 291, 2003, pp 203-218.
7. V. Ménissier-Morain, Arithmétique exacte, Ph.D. thesis, L'Université Paris VII (Dec. 1994).
8. A. Mostowski, On computable sequences, Fundamenta Mathematicae 44 (1957) 37–51.
9. N. T. Müller, Subpolynomial complexity classes of real functions and real numbers, in: L. Kott (Ed.), Proceedings of the 13th International Colloquium on Automata, Languages, and Programming, Vol. 226 of Lecture Notes in Computer Science, Springer, Berlin, 1986, pp. 284–293.
10. N. T. Müller, Towards a real Real RAM: a prototype using C++, in: K.-I. Ko, N. Müller, K. Weihrauch (Eds.), Computability and Complexity in Analysis, Universität Trier, 1996, pp. 59–66, second CCA Workshop, Trier, August 22–23, 1996.
11. N. T. Müller, Implementing limits in an interactive RealRAM, in: J.-M. Chesneaux, F. Jézéquel, J.-L. Lamotte, J. Vignes (Eds.), Third Real Numbers and Computers Conference, Université Pierre et Marie Curie, Paris, 1998, pp. 59–66, paris, France, April 27-29, 1998.
12. S. Owre, N. Shaker, J.M Rushby, D.W.J. Stringer-Calvert, PVS System Guide, Computer Science Laboratory, SRI International, Menlo Park, CA, September 1999. Available from `http://pvs.csl.sri.com`
13. M. B. Pour-El, J. I. Richards, Computability in Analysis and Physics, Perspectives in Mathematical Logic, Springer, Berlin, 1989.

14. H. Rice, Recursive real numbers, Proc. Amer. Math. Soc. 5 (1954) 784–791.
15. R. Robinson, Review of "Peter, R., Rekursive Funktionen", The Journal of Symbolic Logic 16 (1951) 280–282.
16. E. Specker, Nicht konstruktiv beweisbare Sätze der Analysis, The Journal of Symbolic Logic 14 (3) (1949) 145–158.
17. K. Weihrauch, Computability, Vol. 9 of EATCS Monographs on Theoretical Computer Science, Springer, Berlin, 1987.

# Novel Approaches to Numerical Software with Result Verification

Laurent Granvilliers[1], Vladik Kreinovich[2], and Norbert Müller[3]

[1] IRIN
Université de Nantes, France
granvilliers@irin.univ-nantes.fr
[2] Department of Computer Science
University of Texas at El Paso
El Paso, TX 79968, USA
vladik@cs.utep.edu
[3] Abteilung Informatik
Universität Trier
54286 Trier, Germany
mueller@uni-trier.de

**Abstract.** Traditional design of numerical software with result verification is based on the assumption that we know the algorithm $f(x_1, \ldots, x_n)$ that transforms inputs $x_1, \ldots, x_n$ into the output $y = f(x_1, \ldots, x_n)$, and we know the intervals of possible values of the inputs. Many real-life problems go beyond this paradigm. In some cases, we do not have an algorithm $f$, we only know some relation (constraints) between $x_i$ and $y$. In other cases, in addition to knowing the intervals $\mathbf{x}_i$, we may know some relations between $x_i$; we may have some information about the probabilities of different values of $x_i$, and we may know the exact values of some of the inputs (e.g., we may know that $x_1 = \pi/2$). In this paper, we describe the approaches for solving these real-life problems. In Section 2, we describe interval consistency techniques related to handling constraints; in Section 3, we describe techniques that take probabilistic information into consideration, and in Section 4, we overview techniques for processing exact real numbers.

## 1 Introduction

**Why data processing?** In many real-life situations, we are interested in the value of a physical quantity $y$ that is difficult or impossible to measure directly. Examples of such quantities are the distance to a star and the amount of oil in a given well. Since we cannot measure $y$ directly, a natural idea is to measure $y$ *indirectly*. Specifically, we find some easier-to-measure quantities $x_1, \ldots, x_n$ which are related to $y$ by a known relation $y = f(x_1, \ldots, x_n)$; this relation may be a simple functional transformation, or complex algorithm (e.g., for the amount of oil, numerical solution to an inverse problem). Then, to estimate $y$, we first measure the values of the quantities $x_1, \ldots, x_n$, and then we use the

R. Alt et al. (Eds.): Num. Software with Result Verification, LNCS 2991, pp. 274–305, 2004.

results $\widetilde{x}_1, \ldots, \widetilde{x}_n$ of these measurements to compute an estimate $\widetilde{y}$ for $y$ as $\widetilde{y} = f(\widetilde{x}_1, \ldots, \widetilde{x}_n)$.

For example, to find the resistance $R$, we measure current $I$ and voltage $V$, and then use the known relation $R = V/I$ to estimate resistance as $\widetilde{R} = \widetilde{V}/\widetilde{I}$.

Computing an estimate for $y$ based on the results of direct measurements is called *data processing*; data processing is the main reason why computers were invented in the first place, and data processing is still one of the main uses of computers as number crunching devices.

**Traditional approach to numerical software with result verification: from computing with numbers to probabilities to intervals.** Measurement are never 100% accurate, so in reality, the actual value $x_i$ of $i$-th measured quantity can differ from the measurement result $\widetilde{x}_i$. Because of these *measurement errors* $\Delta x_i \stackrel{\text{def}}{=} \widetilde{x}_i - x_i$, the result $\widetilde{y} = f(\widetilde{x}_1, \ldots, \widetilde{x}_n)$ of data processing is, in general, different from the actual value $y = f(x_1, \ldots, x_n)$ of the desired quantity $y$ [53].

It is desirable to describe the error $\Delta y \stackrel{\text{def}}{=} \widetilde{y} - y$ of the result of data processing. To do that, we must have some information about the errors of direct measurements.

What do we know about the errors $\Delta x_i$ of direct measurements? First, the manufacturer of the measuring instrument must supply us with an upper bound $\Delta_i$ on the measurement error. (If no such upper bound was supplied, this would mean that no accuracy is guaranteed, and the corresponding "measuring instrument" would be practically useless.) Since the upper bound $\Delta_i$ is supplied, once we performed a measurement and got a measurement result $\widetilde{x}_i$, we know that the actual (unknown) value $x_i$ of the measured quantity belongs to the interval $\mathbf{x}_i = [\underline{x}_i, \overline{x}_i]$, where $\underline{x}_i = \widetilde{x}_i - \Delta_i$ and $\overline{x}_i = \widetilde{x}_i + \Delta_i$.

In many practical situations, we not only know the interval $[-\Delta_i, \Delta_i]$ of possible values of the measurement error; we also know the probability of different values $\Delta x_i$ within this interval. This knowledge underlies the traditional engineering approach to estimating the error of indirect measurement, in which we assume that we know the probability distributions for measurement errors $\Delta x_i$.

In practice, we can determine the desired probabilities of different values of $\Delta x_i$ by comparing the results of measuring with this instrument with the results of measuring the same quantity by a standard (much more accurate) measuring instrument. Since the standard measuring instrument is much more accurate than the one used, the difference between these two measurement results is practically equal to the measurement error; thus, the empirical distribution of this difference is close to the desired probability distribution for measurement error. There are two cases, however, when this determination is not done:

- First is the case of cutting-edge measurements, e.g., measurements in fundamental science. When a Hubble telescope detects the light from a distant galaxy, there is no "standard" (much more accurate) telescope floating nearby that we can use to calibrate the Hubble: the Hubble telescope is the best we have.

– The second case is the case of measurements on the shop floor. In this case, in principle, every sensor can be thoroughly calibrated, but sensor calibration is so costly – usually costing ten times more than the sensor itself – that manufacturers rarely do it.

In both cases, we have no information about the probabilities of $\Delta x_i$; the only information we have is the upper bound on the measurement error.

In this case, after we performed a measurement and got a measurement result $\widetilde{x}_i$, the only information that we have about the actual value $x_i$ of the measured quantity is that it belongs to the interval $\mathbf{x}_i = [\widetilde{x}_i - \Delta_i, \widetilde{x}_i + \Delta_i]$. In such situations, the only information that we have about the (unknown) actual value of $y = f(x_1, \ldots, x_n)$ is that $y$ belongs to the range $\mathbf{y} = [\underline{y}, \overline{y}]$ of the function $f$ over the box $\mathbf{x}_1 \times \ldots \times \mathbf{x}_n$:

$$\mathbf{y} = [\underline{y}, \overline{y}] = \{f(x_1, \ldots, x_n) \,|\, x_1 \in \mathbf{x}_1, \ldots, x_n \in \mathbf{x}_n\}.$$

The process of computing this interval range based on the input intervals $\mathbf{x}_i$ is called *interval computations*; see, e.g., [30,31,32,44].

**Limitations of the traditional approach.** Traditional design of numerical software with result verification is based on the assumption that we know the algorithm $f(x_1, \ldots, x_n)$ that transforms input $x_1, \ldots, x_n$ into the output $y = f(x_1, \ldots, x_n)$, and we know the intervals $\mathbf{x}_1, \ldots, \mathbf{x}_n$ of possible values of the inputs. Many real-life problems go beyond this paradigm:

– In some cases, we do not have an algorithm $f$, we only know some relation (constraints) between $x_i$ and $y$.
– In other cases, in addition to knowing the intervals $\mathbf{x}_i$ of possible values of $x_i$, we may have some additional information:
  • we may have known some relation *between* different quantities $x_i$;
  • we may also have some additional information about each of these quantities:
    ∗ we may have some information about the *probabilities* of different values of $x_i$;
    ∗ in some cases, we may even know the *exact* values of some of the inputs (e.g., we may know that $x_1 = \pi/2$).

**What we need.** In view of the above limitations, in addition to traditional interval techniques, we must also have:

– techniques that translate known constraints between $x_i$ and $y$ into an algorithm that inputs the values $x_1, \ldots, x_n$ and computes the value(s) $y$ that satisfies all the given constraints (for given values of $x_i$);
– techniques that use the algorithm $f$, the ranges $\mathbf{x}_i$, *and* additional constraints between $x_i$ and $y$ to get a better estimate for the range of possible values of $y$;

- techniques that use the additional information about probabilities of different values of $x_i \in \mathbf{x}_i$ to come up with the information about the probabilities of possible values of $y = f(x_1, \ldots, x_n)$; and
- techniques that would enable us to deal with *exact* real numbers in addition to the numbers known with interval uncertainty.

**What we are planning to do.** In this paper, we describe the approaches for solving these real-life problems:

- In Section 2, written by L. Granvilliers, we describe interval consistency techniques related to handling constraints–both constraints that are known *instead* of the algorithm $f$ and *in addition* to the algorithm $f$.
- In Section 3, written by V. Kreinovich, we describe techniques that take probabilistic information into consideration.
- Finally, in Section 4, written by N. Müller, we overview techniques for processing exact real numbers.

**Why we decided to get together.** At first glance, this paper may seem quite inhomogeneous. As we have mentioned, this paper consists of three separate parts, treating three different topics. Each of these three parts concerns an important practical problem, but, as a reader may notice, as of now, there is little interaction between the three parts.

A reader may ask: why did we decide to make it a joint paper as opposed to three separate papers on three topics? The main reason for this is that we have a joint vision – which we describe in this Introduction. According to this vision, to get more practical numerical software with result verification, we must overcome the above described limitations and thus, we must move from the traditional interval techniques to the techniques that incorporate interval consistency and constraint techniques, interval probabilistic techniques, and techniques for handling exact real numbers.

Ideally, we would like all these techniques to be incorporated in a single tool. Yes, at present, there is very little interaction, little integration between these three techniques. By presenting different techniques within a single paper, we want to emphasize the need for integration and to encourage the readers to think not only about the further development of each of these techniques but also about their possible integration.

Let us work together towards this ambitious but noble goal!

## 2 Interval Consistency Techniques

Mathematical modeling is heavily used to simulate real-life phenomena from engineering, biology or economics. This is a mean for analysis of behavior, optimization or simulation of extreme situations. In many applications the problem is to solve (in)equality or differential equation systems, which may be parametric. Moreover observed data are often uncertain. Uncertainty can be efficiently

handled by interval methods [43,47,46], implementing set computations over real numbers to derive global information on systems. Recently constraint propagation using consistency techniques [5,60,20] have been shown to enhance pure interval methods.

Consistency techniques over real numbers originate from two concurrent works, the introduction of continuous domains in the constraint satisfaction framework [19] and the use of interval arithmetic in constraint logic programming [16] in order to define a logical meaning of arithmetic. In Cleary's work constraint systems are processed by hull-consistency, a consistency property exploiting the convex hull of (in)equalities over the reals. These pioneering ideas have been extended in several ways, *e.g.*, for heterogeneous constraint processing [6] or implementing strong consistencies [40]. `CLP(BNR)` [50] was the premier CLP system using interval consistency techniques.

The next revolution was the design of box-consistency, a local consistency property implemented in `Newton` [5]. For the first time constraint solving smoothly combines Newton-like iterative methods from interval analysis and constraint propagation. These techniques have been further developed and implemented in `Numerica` [60]. Furthermore box-consistency has been shown to drastically improve hull-consistency for a large set of problems in [59].

Recent works have been interested in advanced propagation techniques [41], relaxations for specific problems [64,38], solver cooperation [28], processing of differential equations [20], quantified formulas [54] or applications [58,18,24,27, 30,15]. The combination of box-consistency and hull-consistency has been shown to be efficient in [4]. In the following we review the main lines of interval consistency techniques.

### 2.1 Constraint Satisfaction Problems

A numeric constraint satisfaction problem (NCSP) $P$ is a triple $\langle \mathcal{C}, \mathcal{V}, \mathcal{D} \rangle$ where $\mathcal{C} = \{c_1, \ldots, c_m\}$ is a set of (in)equalities over a set $\mathcal{V} = \{x_1, \ldots, x_n\}$ of real-valued variables to which is associated a domain $\mathcal{D} = \boldsymbol{d}_1 \times \cdots \times \boldsymbol{d}_n$ called a box.

By inequalities, we mean, as usual, inequalities of the type $f = g$, $f \geq g$, or $f \leq g$, where $f$ and $g$ are made of real numbers, variables, arithmetic operations, and elementary functions.

Each $\boldsymbol{d}_i$ is an interval bounded by floating-point numbers that represents the set of possible values of $x_i$. A solution to $P$ is an assignment of variables $(a_1, \ldots, a_n) \in \mathcal{D}$ such that all the constraints are satisfied. Let $\mathsf{Sol}\,(P)$ denote the solution set of $P$.

The purpose of consistency techniques is to compute a NCSP $P' = \langle \mathcal{C}, \mathcal{V}, \mathcal{D}' \rangle$ from $P$ such that the following properties hold:

- completeness: $\mathsf{Sol}\,(P') = \mathsf{Sol}\,(P)$
- contractance: $\mathcal{D}' \subseteq \mathcal{D}$

To this end, given an integer $i \in \{1, \ldots, n\}$, define the $i$-th projection of a relation $\rho \subseteq \mathrm{R}^n$ as the set

$$\pi_i(\rho) = \{a_i \in \mathrm{R} \mid \exists (a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n) \in \mathrm{R}^{n-1} : \ (a_1, \ldots, a_n) \in \rho\}.$$

An inconsistent value is an element $a_i \in \boldsymbol{d}_i$ that does not belong to the $i$-th projection of $\mathsf{Sol}(P)$. It directly follows that $a_i$ can be removed from $\boldsymbol{d}_i$ while preserving $\mathsf{Sol}(P)$. In practice the main problem is to characterize the set of inconsistent values for each variable. Unfortunately, the whole set $\mathsf{Sol}(P)$ cannot be computed due to the limitations of machine arithmetic. As a consequence weaker techniques using constraint projections and interval enclosures have been developed.

### 2.2 Local Consistency Techniques

Each constraint $c_j$ from $P$ defines a relation $\rho_j$ composed of all the assignments of variables $(a_1, \ldots, a_n) \in \mathcal{D}$ satisfying $c_j$. A constraint projection is just a projection of the associated relation. Hull-consistency is a consistency property based on the convex hull of constraint projections. The convex hull of a set of real numbers is the smallest enclosing interval.

**Definition 1.** *Given a constraint $c_j$ and an integer $i \in \{1, \ldots, n\}$ $\boldsymbol{d}_i$ is hull-consistent wrt. $c_j$ and $\mathcal{D}$ iff*

$$\boldsymbol{d}_i = \mathsf{hull}\,(\pi_i(\rho_j)).$$

The meaning of the definition is that no value is declared to be inconsistent if the hull of the $i$-th projection of $c_j$ is equal to $\boldsymbol{d}_i$. In the contrary, if the consistency property is not verified, inconsistent values can be removed at bounds of $\boldsymbol{d}_i$. More precisely a new domain is computed as $\boldsymbol{d}_i \cap \mathsf{hull}\,(\pi_i(\rho_j))$. However the computation of the hull of constraint projections cannot be achieved in general due to rounding errors of numerical computations. This problem has led to the definition of box-consistency using interval functions, *i.e.*, computable objects.

An interval form of a function $f : D_f \to \mathrm{R}$ with $D_f \subseteq \mathrm{R}^n$ is an interval function $\boldsymbol{f} : \mathrm{I}^n \to \mathrm{I}$ such that for every box $\mathcal{D} = \boldsymbol{d}_1 \times \cdots \times \boldsymbol{d}_n$ the following property holds.

$$\boldsymbol{f}(\boldsymbol{d}_1, \cdots, \boldsymbol{d}_n) \supseteq \{f(a_1, \ldots, a_n) \mid (a_1 \ldots, a_n) \in \mathcal{D} \cap D_f\}$$

In other words the evaluation of $\boldsymbol{f}$ encloses the range of $f$. The natural form is the syntactic extension of a given expression of $f$ to the intervals. Interval forms can be used to implement a proof by refutation for constraint satisfaction. Given two expressions $f(x_1, \ldots, x_n)$ and $g(x_1, \ldots, x_n)$, an interval form $\boldsymbol{f}$ of $f$, an interval form $\boldsymbol{g}$ of $g$ and a box $\mathcal{D}$ such that $\boldsymbol{f}(\boldsymbol{d}_1, \cdots, \boldsymbol{d}_n) = [u, v]$ and $\boldsymbol{g}(\boldsymbol{d}_1, \cdots, \boldsymbol{d}_n) = [a, b]$ the interval reasonings are as follows.

$$\begin{array}{ll} (i) & [u, v] \cap [a, b] = \varnothing \implies f = g \text{ has no solution in } \mathcal{D} \\ (ii) & v < a \implies f \geqslant g \text{ has no solution in } \mathcal{D} \end{array}$$

The constraint is proved to be violated when the interval test succeeds (i.e., when either we have an equality constraint $f = g$ and the intersection of $[u, v]$ and $[a, b]$ is empty, or we have an inequality constraint $f \leq g$ and $v < a$). This method is based on the possibly interpretation of constraint relation symbols over the intervals. Given a constraint $c_j$, a box $\mathcal{D}$, an integer $i \in \{1, \ldots, n\}$ and a real number $a_i \in \boldsymbol{d}_i$ let $\boldsymbol{c}_j^i(a_i, \mathcal{D})$ denote a failure of the interval test over the box $\boldsymbol{d}_1 \times \cdots \times \boldsymbol{d}_{i-1} \times \mathsf{hull}\,(\{a_i\}) \times \boldsymbol{d}_{i+1} \times \cdots \times \boldsymbol{d}_n$.

**Definition 2.** *Given a constraint $c_j(x_1, \ldots, x_n)$, a box $\mathcal{D}$, an interval test for $c_j$ and an integer $i \in \{1, \ldots, n\}$ $\boldsymbol{d}_i$ is box-consistent wrt. $c_j$ and $\mathcal{D}$ iff*

$$\boldsymbol{d}_i = \mathsf{hull}\,(\{a_i \in \boldsymbol{d}_i \mid \boldsymbol{c}_j^i(a_i, \mathcal{D})\}).$$

Each real number in $\boldsymbol{d}_i$ is characterized using the interval test, *i.e.*, the bounds of $\boldsymbol{d}_i$ cannot be declared inconsistent using the interval test. We see that the projection appearing in the definition of hull-consistency is just replaced with interval tests.

There are two kinds of algorithms implementing box-consistency: constraint inversion and dichotomous search using interval tests. Constraint inversion[1] uses inverse real operations, as illustrated in Fig. 1. The initial box $[u, v] \times [a, b]$ is reduced to $[u, \lceil \log(b) \rceil] \times [\lfloor \exp(u) \rfloor, b]$ using rounded interval arithmetic (operations $\lceil \cdot \rceil$ and $\lfloor \cdot \rfloor$ are machine rounding operations). A numerical constraint inversion algorithm for processing complex constraints has been introduced in [4]. This method is efficient when variables occur once in constraints, because in this case, straightforward interval computations lead to the exact range (see, e.g., [29]); in more general situations, due to the dependency problem of interval arithmetic, this method may not always be so efficient.

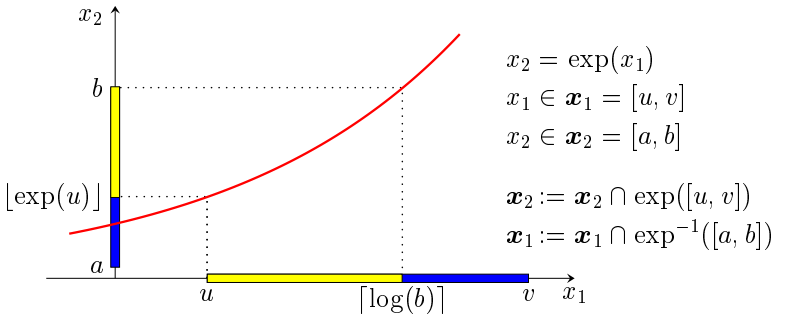



**Fig. 1.** Inversion of $x_2 = \exp(x_1)$ for computing domain reductions.

The dichotomous search procedure enforces the interval test over sub-domains from $\boldsymbol{d}_i$. The aim is to compute the leftmost real number $a \in \boldsymbol{d}_i$ and the

[1] Note that constraint inversion has often been presented as a computational technique for hull-consistency but this notion is not computable, as shown in [4].

rightmost real number $b \in \boldsymbol{d}_i$ such that the interval tests $\boldsymbol{c}_j^i(a, \mathcal{D})$ and $\boldsymbol{c}_j^i(b, \mathcal{D})$ fail (the constraint is not violated). Doing so the new domain is $[a, b]$. The method is in three steps: if the whole domain $\boldsymbol{d}_i$ is rejected using the interval test then stop and return the empty set; otherwise try to reduce the left bound of $\boldsymbol{d}_i$ and then the right bound. The search of outermost points is just implemented by a dichotomous search using the interval tests.

*Example 1.* Consider the constraint $(x_1-1)^2 = 0$, the interval $\boldsymbol{d}_1 = [-10, 10]$ and the interval test based on natural forms. Domain of $x_1$ is not box-consistent since the interval test succeeds for the right bound $((10-1)^2 \neq 0)$. As a consequence $\boldsymbol{d}_1$ can be reduced using *e.g.* constraint inversion, as follows.

$$\boldsymbol{d}_1 := \boldsymbol{d}_1 \cap \left(square^{-1}(0) + 1\right) = [1, 1]$$

Now suppose that the constraint is given in a developed form $x_1^2 - 2x_1 + 1 = 0$. The problem of constraint inversion is to choose one occurrence of $x_1$. Suppose this is the one in the quadratic term. Then try to reduce $\boldsymbol{d}_1$.

$$\boldsymbol{d}_1 := \boldsymbol{d}_1 \cap square^{-1}(2\boldsymbol{d}_1 - 1) \approx [-\sqrt{19}, \sqrt{19}]$$

The approximation is weak, which is due to the dependency problem of interval arithmetic (two occurrences of $x_1$ considered as different variables during interval computations). A solution is to implement a dichotomous search. In this case a tight interval enclosing 1 is computed since each small sub-domain can be rejected using the interval test. For instance the real number $2 \in \boldsymbol{d}_1$ can be rejected since $(2^2 - 2 \times 2 + 1 \neq 0)$. Using the search procedure the dependency problem over the considered variable vanishes. However, in this case, a second problem happens, a slow convergence, since there is a multiple root of function $(x_1 - 1)^2$. This phenomenon has to be controlled and the search stopped if necessary. The reader is referred to [26] for more details.

### 2.3 Constraint Propagation and Strong Consistency Techniques

Given a NCSP constraint projections have to be processed in sequence in order to reach the consistency of the whole problem. The fixed-point algorithm implementing such a sequence of computations is called constraint propagation. Given a constraint $c(x_1, \ldots, x_n)$ and an integer $i \in \{1, \ldots, n\}$ let $\mathsf{revise}_{ij}$ denote the function from boxes to boxes such that for each box $\mathcal{D}$

$$\mathsf{revise}_{ij}(\mathcal{D}) = \mathcal{D}' = \boldsymbol{d}_1 \times \cdots \times \boldsymbol{d}_{i-1} \times \boldsymbol{d}_i' \times \boldsymbol{d}_{i+1} \times \cdots \times \boldsymbol{d}_n$$

and $\boldsymbol{d}_i'$ is the largest interval included in $\boldsymbol{d}_i$ that is box-consistent wrt. $c_j$ and $\mathcal{D}'$. The revise function is implemented by constraint inversion or dichotomous search. A generic constraint propagation algorithm implementing box-consistency over NCSPs is presented in Table 1. Each step consists in an application of a revise function for reducing the current box. The loop invariant states that at the beginning of each step of the loop, each function $\theta$ not in $\mathcal{S}$ is such that

$\theta(\mathcal{D}) = \mathcal{D}$. Then every revise function associated to a constraint $c_q$ that contains a variable $x_i$ which domain has been modified is added in $\mathcal{S}$.

It is worth mentioning that the algorithm presented in Table 1 is actually intended for a more general situation that the situation that we have described. In our particular case, according to the definition of $\mathsf{revise}_{ij}$, the function $\mathsf{revise}_{ij}$ changes only the $i$-th component $\boldsymbol{d}_i$ to $\boldsymbol{d}'_i$. Therefore, there is only value $k$ which should be considered in the line foreach $k \in \{1,\ldots,n\}$ such that $\boldsymbol{d}'_k \neq \boldsymbol{d}_k$ do – the value $k = i$. In a more general situation, however, we do need a foreach-loop.

**Table 1.** Constraint propagation algorithm for box-consistency.

```
propagate (C = {c_1, ..., c_m}, V = {x_1, ..., x_n}, D = d_1 × ··· × d_n): box
begin
    S := {revise_ij | i ∈ {1, ..., n}, j ∈ {1, ..., m}}
    repeat
        revise_ij := pop (S)
        D' := revise_ij(D)
        foreach k ∈ {1, ..., n} such that d'_k ≠ d_k do
            S := push ({revise_pq | 1 ⩽ p ⩽ n, 1 ⩽ q ⩽ m, x_i occurs in c_q})
        endfor
        D := D'
    until S = ∅ or D = ∅
    return D
end
```

The propagate algorithm terminates in finite time since every step is contracting and the computational domain is finite. It is complete, *i.e.*, no solution is lost, since every revise function is complete. It converges and computes the greatest common fixed-point of the revise functions. As a consequence the order of applications of the revise functions is not relevant. The proofs of these properties come from domain theory [2].

*Example 2.* Consider the NCSP $\langle\{x_2 = x_1^2, x_1^2 + x_2^2 = 2\}, \{x_1, x_2\}, [-10, 10]^2\rangle$ which solutions are $(1, 1)$ and $(-1, 1)$. Applying $\mathsf{revise}_{21}$ reduces the domain of $x_2$ to $[0, 10]$. Applying $\mathsf{revise}_{12}$ reduces the domain of $x_1$ to $[-\sqrt{2}, \sqrt{2}]$, and so on. The result of propagate is the box $[-1.19, 1.19] \times [0.76, 1.42]$. Unfortunately this process does not compute the enclosure of the solution set, namely $[-1, 1] \times [1, 1]$. This weakness is due to the locality problem.

The locality problem originates from the way to reduce domains, since each constraint projection is used independently. However we may say that "the intersection of projections is weaker than the projection of the intersection". In Fig. 2 the NCSP represents an intersection of two curves $c_1$ and $c_2$. In this case the box cannot be reliably reduced using $c_1$ because this would loose solutions of $c_1$ (idem for $c_2$). This problem has led to the definition of stronger consistency

techniques, namely $k$B-consistencies [40]. The main idea is shown in Fig. 2: prove the inconsistency of a sub-box using the constraint propagation algorithm. In this case the leftmost sub-box is discarded and the rightmost sub-box is reduced. If this process is iterated then a tight enclosure of the solution can be computed. Unfortunately it has been shown in [52] that strong consistency techniques do not have a good practical complexity. A formal comparison of consistency techniques can be found in [17].

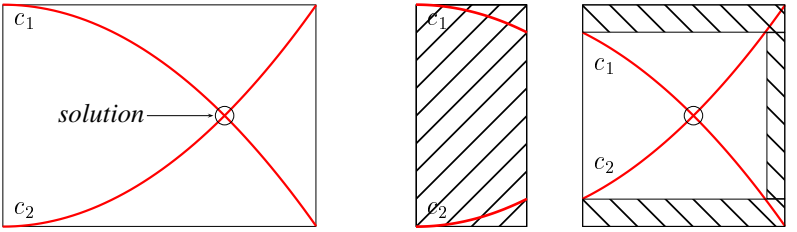



**Fig. 2.** Locality problem and strong consistency technique.

### 2.4 Use of Consistency Techniques

Mathematical models are often heterogeneous, involving integer or real numbers, differential equations, inequalities, objectives or quantified constraints. Actually consistency techniques can be implemented as components in more general solving processes, *e.g.*, constrained optimization, ODE solving or decomposition of quantified inequalities. They are used to reduce domains of possible values of the unknowns or to prove the inconsistency of a given problem.

As an example we describe the processing of universally quantified inequalities [3]. Consider the formula $\phi : \forall y \in \boldsymbol{y}\ f(y, x_1, \ldots, x_n) \leqslant 0$ over the box $\mathcal{D}$. A solution to $\phi$ is a tuple $(a_1, \ldots, a_n) \in \mathcal{D}$ such that for all $b \in \boldsymbol{y}$, $f(b, a_1, \ldots, a_n) \leqslant 0$ holds. Now suppose there exists a box $\boldsymbol{y} \times \mathcal{D}'$ such that every of its elements violates $f(y, x_1, \ldots, x_n) > 0$. As a consequence every element of $\mathcal{D}'$ is a solution to $\phi$, since $f > 0$ is the negation of $f \leqslant 0$. Consistency techniques are used to compute such a box: (1) starting from the box $\boldsymbol{y} \times \mathcal{D}$ apply a consistency technique over the negation of the constraint; (2) for each eliminated box $\boldsymbol{y}' \times \mathcal{D}'$ return $\mathcal{D}'$ if $\boldsymbol{y}' = \boldsymbol{y}$. In practice strict inequalities resulting from negations are relaxed. Note that a box that is inconsistent wrt. $f \geqslant 0$ is clearly inconsistent wrt. $f > 0$. In [54] a general framework for solving quantified constraints using consistency techniques has been proposed.

### 2.5 Perspectives

Consistency techniques have been developed for solving NCSPs, namely conjunctions of (in)equalities. Only recently they have been extended to tackle quantifiers, conditional constraints, mixed problems, differential equations or

optimization problems. We believe that the main perspective is to develop a suite of tools to be used in real-world applications and to integrate these tools in development frameworks. The main advantages that should be highlighted are that the use of intervals lead to robust decisions and that heterogeneous systems are handled without additional work.

A particularly interesting field of applications is the so-called robust design. Solutions in automatic control are proposed in [30]. For instance, state estimation problems are efficiently solved by constraint propagation alone since many redundant constraints, provided by redundant captors, are available. On the contrary problems from image synthesis [15] are often under-constrained. Consistency techniques have to be combined with local search and optimization methods. In conceptual design [51,24] the aim is to derive all possible concepts from specifications, and possibly to take a decision. Consistency techniques embedding decomposition techniques of NCSPs [8] may be directly applied. However further research has to be done in several ways such as uncertainty quantification given approximate models and sensitivity analysis.

## 3 Interval-Probability Techniques

**What we are planning to do in this section.** As we have mentioned in the Introduction, in many practical situations, in addition to the interval information, we also have some information about the probabilities.

In this section, first, we analyze a specific interval computations problem – when we use traditional statistical data processing algorithms $f(x_1, \ldots, x_n)$ to process the results of direct measurements.

Then, we extend our analysis to the case when for each input $x_i$, in addition to the interval $\mathbf{x}_i = [\underline{x}_i, \overline{x}_i]$ of possible values, we have partial information about the probabilities: specifically, we know its mean $E_i$ (or an interval $\mathbf{E}_i$ of possible values of the mean).

### 3.1 First Step Beyond Intervals: Error Estimation for Traditional Statistical Data Processing Algorithms under Interval Uncertainty

When we have $n$ results $x_1, \ldots, x_n$ of repeated measurement of the same quantity (at different points, or at different moments of time), traditional statistical approach usually starts with computing their sample average $E = (x_1 + \ldots + x_n)/n$ and their (sample) variance

$$V = \frac{(x_1 - E)^2 + \ldots + (x_n - E)^2}{n} \tag{1}$$

(or, equivalently, the sample standard deviation $\sigma = \sqrt{V}$); see, e.g., [53].

In this section, we consider situations when we do not know the exact values of the quantities $x_1, \ldots, x_n$, we only know the intervals $\mathbf{x}_1, \ldots, \mathbf{x}_n$ of possible values of $x_i$. In such situations, for different possible values $x_i \in \mathbf{x}_i$, we get

different values of $E$ and $V$. The question is: what are the intervals $\mathbf{E}$ and $\mathbf{V}$ of possible values of $E$ and $V$?

The practical importance of this question was emphasized, e.g., in [48,49] on the example of processing geophysical data.

For $E$, the straightforward interval computations [30,31,32,44] leads to the exact range:

$$\mathbf{E} = \frac{\mathbf{x}_1 + \ldots + \mathbf{x}_n}{n}, \text{ i.e., } \underline{E} = \frac{\underline{x}_1 + \ldots + \underline{x}_n}{n}, \text{ and } \overline{E} = \frac{\overline{x}_1 + \ldots + \overline{x}_n}{n}.$$

For $V$, straightforward interval computations lead to an excess width. For example, for $\mathbf{x}_1 = \mathbf{x}_2 = [0,1]$, the variance is $V = (x_1 - x_2)^2/4$ and hence, the actual range $\mathbf{V} = [0, 0.25]$. On the other hand, $\mathbf{E} = [0,1]$, hence

$$\frac{(\mathbf{x}_1 - \mathbf{E})^2 + (\mathbf{x}_2 - \mathbf{E})^2}{2} = [0,1] \supset [0, 0.25].$$

More sophisticated methods of interval computations also sometimes lead to an excess width. Reason: in the formula for the average $E$, each variable only occurs once, and it is known that for such formulas, straightforward interval computations lead to the exact range (see, e.g., [29]). In the expression for variance, each variable $x_i$ occurs several times: explicitly, in $(x_i - E)^2$, and explicitly, in the expression for $E$. In such cases, often, dependence between intermediate computation results leads to excess width of the results of straightforward interval computations. Not surprisingly, we do get excess width when applying straightforward interval computations to the formula (1).

For variance, we can actually prove that the corresponding optimization problem is difficult:

**Theorem 1.** [23] *Computing $\overline{V}$ is NP-hard.*

The very fact that computing the range of a quadratic function is NP-hard was first proven by Vavasis [61] (see also [35]). We have shown that this difficulty happens even for very simple quadratic functions frequently used in data processing.

A natural question is: maybe the difficulty comes from the requirement that the range be computed exactly? In practice, it is often sufficient to compute, in a reasonable amount of time, a usefully accurate estimate $\widetilde{\overline{V}}$ for $\overline{V}$, i.e., an estimate $\widetilde{\overline{V}}$ which is accurate with a given accuracy $\varepsilon > 0$: $\left|\widetilde{\overline{V}} - \overline{V}\right| \leq \varepsilon$. Alas, for any $\varepsilon$, such computations are also NP-hard:

**Theorem 2.** [23] *For every $\varepsilon > 0$, the problem of computing $\overline{V}$ with accuracy $\varepsilon$ is NP-hard.*

It is worth mentioning that $\overline{V}$ can be computed exactly in exponential time $O(2^n)$:

**Theorem 3.** [23] *There exists an algorithm that computes $\overline{V}$ in exponential time.*

This algorithm consists of testing all $2^n$ possible combinations of values $x_i = \underline{x}_i$ and $x_i = \overline{x}_i$. For small number of variables, this is quite doable, but when we have, say, $n = 10^3$ observations, we cannot do that anymore. In the following text, we will describe what we can do in such cases.

For computing $\underline{V}$, there exists feasible algorithms. In [23], we described a quadratic-time algorithm for computing $\underline{V}$. It turns out that we can actually compute $\underline{V}$ in
$O(n \cdot \log(n))$ computational steps (arithmetic operations or comparisons) for $n$ interval data points $\mathbf{x}_i = [\underline{x}_i, \overline{x}_i]$.

The algorithm $\underline{\mathcal{A}}$ is as follows:

- First, we sort all $2n$ values $\underline{x}_i$, $\overline{x}_i$ into a sequence $x_{(1)} \le x_{(2)} \le \ldots \le x_{(2n)}$.
- Second, we use bisection to find the value $k$ ($1 \le k \le 2n$) for which the following two inequalities hold:

$$\sum_{j:\overline{x}_j \le x_{(k)}} (x_{(k)} - \overline{x}_j) \le \sum_{i:\underline{x}_i \ge x_{(k+1)}} (\underline{x}_i - x_{(k)}); \tag{2}$$

$$\sum_{j:\overline{x}_j \le x_{(k)}} (x_{(k+1)} - \overline{x}_j) \ge \sum_{i:\underline{x}_i \ge x_{(k+1)}} (\underline{x}_i - x_{(k+1)}). \tag{3}$$

  At each iteration of this bisection, we have an interval $[k^-, k^+]$ that is guaranteed to contain $k$. In the beginning, $k^- = 1$ and $k^+ = 2n$. At each stage, we compute the midpoint $k_{\rm mid} = \lfloor (k^- + k^+)/2 \rfloor$, and check both inequalities (2) and (3) for $k = k_{\rm mid}$. Then:
  - If both inequalities (2) and (3) hold for his $k$, this means that we have found the desired $k$.
  - If (2) holds but (3) does not hold, this means that the desired value $k$ is larger than $k_{\rm mid}$, so we keep $k^+$ and replace $k^-$ with $k_{\rm mid} + 1$.
  - If (3) holds but (2) does not hold, this means that the desired value $k$ is smaller than $k_{\rm mid}$, so we keep $k^-$ and replace $k^+$ with $k_{\rm mid} - 1$.
- Once $k$ is found, we compute

$$S_k \stackrel{\rm def}{=} \sum_{i:\underline{x}_i \ge x_{(k+1)}} \underline{x}_i + \sum_{j:\overline{x}_j \le x_{(k)}} \overline{x}_j, \tag{4}$$

  then $r_k = S_k / N_k$, where $N_k$ is the total number of such $i$s and $j$s, and, finally,

$$\underline{V} = \frac{1}{n} \cdot \left( \sum_{j:\overline{x}_j \le x_{(k)}} (\overline{x}_j - r_k)^2 + \sum_{i:\underline{x}_i \ge x_{(k+1)}} (\underline{x}_i - r_k)^2 \right).$$

**Theorem 4.** (Kreinovich, Longpré) *The algorithm $\underline{\mathcal{A}}$ always computes $\underline{V}$ in time $O(n \cdot \log(n))$.*

**Proof.** Let us start with simple calculus. Let $f(x_1, \ldots, x_n)$ be a differentiable function on a box $B \stackrel{\text{def}}{=} \mathbf{x}_1 \times \ldots \times \mathbf{x}_n$, and let $x^- = (x_1^-, \ldots, x_n^-) \in B$ be a point at which $f$ attains its smallest value on this box.

Then, for every $i$, the function $f_i(x_i) \stackrel{\text{def}}{=} f(x_1^-, \ldots, x_{i-1}^-, x_i, x_{i+1}^-, \ldots, x_n^-)$ also attains its minimum on the interval $[\underline{x}_i, \overline{x}_i]$ at the point $x_i = x_i^-$.

According to the basic calculus, this minimum is either attained in the interior of the interval, in which case $df_i/dx_i = 0$ for $x_i = x_i^-$, or the minimum is attained at one of the endpoints of the interval $[\underline{x}_i, \overline{x}_i]$. If the minimum is attained at the left endpoint $\underline{x}_i$, then the function $f_i$ cannot be decreasing at this point, so $df_i/dx_i \geq 0$. Similarly, if the minimum is attained at the right endpoint $\overline{x}_i$, then $df_i/dx_i \leq 0$.

By definition of the function $f_i(x_i)$, the value of the derivative $df_i/dx_i$ for $x_i = x_i^-$ is equal to the value of the partial derivative $\partial f/\partial x_i$ at the point $x^-$. Thus, for each $i$, we have one of the following three cases:

- either $\underline{x}_i < x_i^- < \overline{x}_i$ and $\partial f/\partial x_i = 0$;
- or $x_i^- = \underline{x}_i$ and $\partial f/\partial x_i \geq 0$;
- or $x_i^- = \overline{x}_i$ and $\partial f/\partial x_i \leq 0$.

For $f = V$, as one can easily see, $\partial V/\partial x_i = (2/n) \cdot (x_i - E)$, so the sign of this derivative is the same as the sing of the difference $x_i - E$. Therefore, for the point $x^-$ at which the variance $V$ attains its minimum, we have one of the following three situations:

- either $\underline{x}_i < x_i^- < \overline{x}_i$ and $x_i^- = E$;
- or $x_i^- = \underline{x}_i$ and $x_i^- \geq E$;
- or $x_i^- = \overline{x}_i$ and $x_i^- \leq E$.

In the first case, $\underline{x}_i < E < \overline{x}_i$; in the second case, $E \leq \underline{x}_i$, and in the third case, $\overline{x}_i \leq E$.

Let us show that if we know where $E$ is in comparison to the endpoints of all the intervals, i.e., to which "small interval" $[x_{(k)}, x_{(k+1)}]$ the value $E$ belongs, we can uniquely determine the values $x_i^-$ for all $i$.

Indeed, when $x_{(k+1)} \leq \underline{x}_i$, this means that $E \leq x_{(k+1)} \leq \underline{x}_i$, so $E \leq \underline{x}_i$. Thus, we cannot have the first case (in which $E > \underline{x}_i$), so we must have either the second or the third cases, i.e., we must have $x_i = \underline{x}_i$ or $x_i = \overline{x}_i$. If the interval $\mathbf{x}_i$ is degenerate, then both cases lead to the same result. If the interval is non-degenerate, then we cannot have the third case – in which $\underline{x}_i < \overline{x}_i \leq E$ hence $\underline{x}_i < E$ – and thus, we must have the second case, i.e., $x_i^- = \overline{x}_i$. Thus, $x_{(k+1)} \leq \underline{x}_i$ implies that $x_i^- = \underline{x}_i$.

Similarly, $x_{(k)} \geq \overline{x}_i$ implies that $x_i^- = \overline{x}_i$, and in all other cases, we have $x_i^- = E$.

All that remains is to find the appropriate $k$. Once $k$ is fixed, we can find the values $x_i^-$ in linear time, and then compute the corresponding value $V$ is linear time. The only condition on $k$ is that the average of the corresponding values $x_i^-$ should be within the corresponding small interval $[x_{(k)}, x_{(k+1)}]$.

In [23], we proposed to find $k$ by exhaustive (linear) search. Since there are $2n$ possible small intervals, we must therefore repeat $O(n)$ computations $2n$ times, which takes $2n \cdot O(n) = O(n^2)$ time. Together with the original sorting – that takes $O(n \cdot \log(n))$ time – we thus get a quadratic time algorithm, since

$$O(n^2) + O(n \cdot \log(n)) = O(n^2).$$

Let us now show that we can find $k$ faster. We want to satisfy the conditions $x^{(k)} \le E$ and $E \le x^{(k+1)}$. The value $E$ is the average of all the values $x_i^-$, i.e., we have

$$n \cdot E = S_k + (n - N_k) \cdot E, \tag{5}$$

where $S_k$ is defined by the formula (4) and $N_k$ is defined in the description of the algorithm $\underline{\mathcal{A}}$. By moving all the terms proportional to $E$ to the left-hand side of (5), we conclude that $N_k \cdot E = S_k$, i.e., that $E = S_k/N_k$ $(= r_k)$. The first desired inequality $x^{(k)} \le E$ thus takes the form $S_k/N_k \le x^{(k)}$, i.e., equivalently, $N_k \cdot x^{(k)} \le S_k$, i.e.,

$$N_k \cdot x_{(k)} \le \sum_{i:\underline{x}_i \ge x_{(k+1)}} \underline{x}_i + \sum_{j:\overline{x}_j \le x_{(k)}} \overline{x}_j. \tag{6}$$

In the left-hand side of (6), we have as many terms $x_{(k)}$ as there are terms in the right-hand side. Thus, we can subtract $x_{(k)}$ from each of $N_k$ term in the right-hand side. When $\overline{x}_j \le x_{(k)}$, the difference $\overline{x}_j - x_{(k)}$ is negative, so we can move it to the left-hand side of the inequality, resulting in the inequality (2).

When $k$ increases, the left-hand side of the inequality (2) increases – because each term increases and new terms may appear. Similarly, the right-hand side of this inequality decreases with $k$. Thus, if this inequality holds for $k$, it should also hold for all smaller values, i.e., for $k-1$, $k-2$, etc.

Similarly, the second desired inequality $E \le x^{(k+1)}$ takes the equivalent form (3). When $k$ increases, the left-hand side of this inequality increases, while the right-hand side decreases. Thus, if this inequality is true for $k$, it is also true for $k+1$, $k+2$, ...

If both inequalities (2) and (3) are true for two different values $k < k'$, then they should both be true for all the values intermediate between $k$ and $k'$, i.e., for $k+1, k+2, \ldots, k'-1$. Let us show that both inequalities cannot be true for $k$ and for $k+1$. Indeed, if the inequality (2) is true for $k+1$, this means that

$$\sum_{j:\overline{x}_j \le x_{(k+1)}} (x_{(k+1)} - \overline{x}_j) \le \sum_{i:\underline{x}_i \ge x_{(k+2)}} (\underline{x}_i - x_{(k+1)}). \tag{7}$$

However, the left-hand side of this inequality is not smaller than the left-hand side of (3), while the right-hand side of this inequality is not larger than the right-hand side of (3). Thus, (7) is inconsistent with (3). This inconsistency proves that there is only one $k$ for which both inequalities are true, and this $k$ can be found by the bisection method as described in the above algorithm $\underline{\mathcal{A}}$.

How long does this algorithm take? In the beginning, we only know that $k$ belongs to the interval $[1, 2n]$ of width $O(n)$. At each stage of the bisection step,

we divide the interval (containing $k$) in half. After $I$ iterations, we decrease the width of this interval by a factor of $2^I$. Thus, to find the exact value of $k$, we must have $I$ for which $O(n)/2^I = 1$, i.e., we need $I = O(\log(n))$ iterations. On each iteration, we need $O(n)$ steps, so we need a total of $O(n \cdot \log(n))$ steps. With $O(n \cdot \log(n))$ steps for sorting, and $O(n)$ for computing the variance, we get a $O(n \cdot \log(n))$ algorithm. The theorem is proven.

NP-hardness of computing $\overline{V}$ means, crudely speaking, that there are no general ways for solving all particular cases of this problem (i.e., computing $\overline{V}$) in reasonable time.

However, there are algorithms for computing $\overline{V}$ for many reasonable situations. In [23], we proposed an efficient algorithm that computes $\overline{V}$ for the case when all the interval midpoints ("measured values") $\widetilde{x}_i = (\underline{x}_i + \overline{x}_i)/2$ are definitely different from each other, in the sense that the "narrowed" intervals $[\widetilde{x}_i - \Delta_i/n, \widetilde{x}_i + \Delta_i/n]$ – where $\Delta_i = (\underline{x}_i - \overline{x}_i)/2$ is the interval's half-width – do not intersect with each other.

This algorithm $\overline{\mathcal{A}}$ is as follows:

- First, we sort all $2n$ endpoints of the narrowed intervals $\widetilde{x}_i - \Delta_i/n$ and $\widetilde{x}_i + \Delta_i/n$ into a sequence $x_{(1)} \le x_{(2)} \le \ldots \le x_{(2n)}$. This enables us to divide the real line into $2n + 1$ segments ("small intervals") $[x_{(k)}, x_{(k+1)}]$, where we denoted $x_{(0)} \stackrel{\text{def}}{=} -\infty$ and $x_{(2n+1)} \stackrel{\text{def}}{=} +\infty$.
- Second, we compute $\underline{E}$ and $\overline{E}$ and pick all "small intervals" $[x_{(k)}, x_{(k+1)}]$ that intersect with $[\underline{E}, \overline{E}]$.
- For each of remaining small intervals $[x_{(k)}, x_{(k+1)}]$, for each $i$ from 1 to $n$, we pick the following value of $x_i$:
  - if $x_{(k+1)} < \widetilde{x}_i - \Delta_i/n$, then we pick $x_i = \overline{x}_i$;
  - if $x_{(k)} > \widetilde{x}_i + \Delta_i/n$, then we pick $x_i = \underline{x}_i$;
  - for all other $i$, we consider both possible values $x_i = \overline{x}_i$ and $x_i = \underline{x}_i$.

  As a result, we get one or several sequences of $x_i$. For each of these sequences, we check whether the average $E$ of the selected values $x_1, \ldots, x_n$ is indeed within this small interval, and if it is, compute the variance by using the formula (1).
- Finally, we return the largest of the computed variances as $\overline{V}$.

**Theorem 5.** *The algorithm $\overline{\mathcal{A}}$ computes $\overline{V}$ in quadratic time for all the cases in which the "narrowed" intervals do not intersect with each other.*

This algorithm also works when, for some fixed $k$, no more than $k$ "narrowed" intervals can have a common point:

**Theorem 6.** *For every positive integer $k$, the algorithm $\overline{\mathcal{A}}$ computes $\overline{V}$ in quadratic time for all the cases in which no more than $k$ "narrowed" intervals can have a common point.*

Can we compute $\overline{V}$ faster? In [23], we provided quadratic-time algorithms for computing $\underline{V}$ and $\overline{V}$. For $\underline{V}$, we have shown, in this paper, that we can replace the exhaustive (linear) search over $k$ with an appropriate binary search and thus

reduce the computation time from $O(n^2)$ to $O(n \cdot \log(n))$. It turns out [65] that a similar modification can be done for the algorithm $\overline{V}$. Thus (provided that no more than $k$ narrowed intervals have a common point), we can compute $\overline{V}$ in time $O(n \cdot \log(n))$ as well.

### 3.2 Second Step Beyond Intervals: Extension of Interval Arithmetic to Situations with Partial Information about Probabilities

**Practical problem.** In some practical situations, in addition to the lower and upper bounds on each random variable $x_i$, we know the bounds $\mathbf{E}_i = [\underline{E}_i, \overline{E}_i]$ on its mean $E_i$.

Indeed, in measurement practice (see, e.g., [53]), the overall measurement error $\Delta x$ is usually represented as a sum of two components:

- a *systematic* error component $\Delta_s x$ which is defined as the expected value $E[\Delta x]$, and
- a *random* error component $\Delta_r x$ which is defined as the difference between the overall measurement error and the systematic error component: $\Delta_r x \stackrel{\text{def}}{=} \Delta x - \Delta_s x$.

In addition to the bound $\Delta$ on the overall measurement error, the manufacturers of the measuring instrument often provide an upper bound $\Delta_s$ on the systematic error component: $|\Delta_s x| \le \Delta_s$.

This additional information is provided because, with this additional information, we not only get a bound on the accuracy of a single measurement, but we also get an idea of what accuracy we can attain if we use repeated measurements to increase the measurement accuracy. Indeed, the very idea that repeated measurements can improve the measurement accuracy is natural: we measure the same quantity by using the same measurement instrument several ($N$) times, and then take, e.g., an arithmetic average $\bar{x} = (\widetilde{x}^{(1)} + \ldots + \widetilde{x}^{(N)})/N$ of the corresponding measurement results $\widetilde{x}^{(1)} = x + \Delta x^{(1)}$, $\ldots$, $\widetilde{x}^{(N)} = x + \Delta x^{(N)}$.

- If systematic error is the only error component, then all the measurements lead to exactly the same value $\widetilde{x}^{(1)} = \ldots = \widetilde{x}^{(N)}$, and averaging does not change the value – hence does not improve the accuracy.
- On the other hand, if we know that the systematic error component is 0, i.e., $E[\Delta x] = 0$ and $E[\widetilde{x}] = x$, then, as $N \to \infty$, the arithmetic average tends to the actual value $x$. In this case, by repeating the measurements sufficiently many times, we can determine the actual value of $x$ with an arbitrary given accuracy.

In general, by repeating measurements sufficiently many times, we can arbitrarily decrease the random error component and thus attain accuracy as close to $\Delta_s$ as we want.

When this additional information is given, then, after we performed a measurement and got a measurement result $\widetilde{x}$, then not only we get the information that the actual value $x$ of the measured quantity belongs to the interval $\mathbf{x} = [\widetilde{x} - \Delta, \widetilde{x} + \Delta]$, but we can also conclude that the expected value of

$x = \widetilde{x} - \Delta x$ (which is equal to $E[x] = \widetilde{x} - E[\Delta x] = \widetilde{x} - \Delta_s x$) belongs to the interval $\mathbf{E} = [\widetilde{x} - \Delta_s, \widetilde{x} + \Delta_s]$.

If we have this information for every $x_i$, then, in addition to the interval $\mathbf{y}$ of possible value of $y$, we would also like to know the interval of possible values of $E[y]$. This additional interval will hopefully provide us with the information on how repeated measurements can improve the accuracy of this indirect measurement. Thus, we arrive at the following problem.

**Resulting optimization problem.** In more optimization terms, we want to solve the following problem: given an algorithm computing a function $f(x_1, \ldots, x_n)$ from $R^n$ to $R$; and values $\underline{x}_1$, $\overline{x}_1$, $\ldots$, $\underline{x}_n$, $\overline{x}_n$, $\underline{E}_1$, $\overline{E}_1$, $\ldots$, $\underline{E}_n$, $\overline{E}_n$, we want to find

$$\underline{E} \stackrel{\text{def}}{=} \min\{E[f(x_1, \ldots, x_n)] \mid \text{all distributions of } (x_1, \ldots, x_n) \text{ for which}$$

$$x_1 \in [\underline{x}_1, \overline{x}_1], \ldots, x_n \in [\underline{x}_n, \overline{x}_n], E[x_1] \in [\underline{E}_1, \overline{E}_1], \ldots E[x_n] \in [\underline{E}_n, \overline{E}_n]\};$$

and $\overline{E}$ which is the maximum of $E[f(x_1, \ldots, x_n)]$ for all such distributions.

In addition to considering all possible distributions, we can also consider the case when all the variables $x_i$ are independent.

**Analog of straightforward interval computations.** The main idea behind straightforward interval computations can be applied here as well. Namely, first, we find out how to solve this problem for the case when $n = 2$ and $f(x_1, x_2)$ is one of the standard arithmetic operations. Then, once we have an arbitrary algorithm $f(x_1, \ldots, x_n)$, we parse it and replace each elementary operation on real numbers with the corresponding operation on quadruples $(\underline{x}, \underline{E}, \overline{E}, \overline{x})$.

To implement this idea, we must therefore know how to solve the above problem for elementary operations.

For *addition*, the answer is simple. Since $E[x_1 + x_2] = E[x_1] + E[x_2]$, if $y = x_1 + x_2$, there is only one possible value for $E = E[y]$: the value $E = E_1 + E_2$. This value does not depend on whether we have correlation or nor, and whether we have any information about the correlation. Thus, $\mathbf{E} = \mathbf{E}_1 + \mathbf{E}_2$.

Similarly, the answer is simple for *subtraction:* if $y = x_1 - x_2$, there is only one possible value for $E = E[y]$: the value $E = E_1 - E_2$. Thus, $\mathbf{E} = \mathbf{E}_1 - \mathbf{E}_2$.

For *multiplication*, if the variables $x_1$ and $x_2$ are independent, then $E[x_1 \cdot x_2] = E[x_1] \cdot E[x_2]$. Hence, if $y = x_1 \cdot x_2$ and $x_1$ and $x_2$ are independent, there is only one possible value for $E = E[y]$: the value $E = E_1 \cdot E_2$; hence $\mathbf{E} = \mathbf{E}_1 \cdot \mathbf{E}_2$.

The first non-trivial case is the case of multiplication in the presence of possible correlation. When we know the exact values of $E_1$ and $E_2$, the solution to the above problem is as follows (see, e.g., [34]):

**Theorem 7.** *For multiplication $y = x_1 \cdot x_2$, when we have no information about the correlation,*

$$\underline{E} = \max(p_1 + p_2 - 1, 0) \cdot \overline{x}_1 \cdot \overline{x}_2 + \min(p_1, 1 - p_2) \cdot \overline{x}_1 \cdot \underline{x}_2 + \min(1 - p_1, p_2) \cdot \underline{x}_1 \cdot \overline{x}_2 +$$

$$\max(1 - p_1 - p_2, 0) \cdot \underline{x}_1 \cdot \underline{x}_2;$$

*and*

$$\overline{E} = \min(p_1, p_2) \cdot \overline{x}_1 \cdot \overline{x}_2 + \max(p_1 - p_2, 0) \cdot \overline{x}_1 \cdot \underline{x}_2 + \max(p_2 - p_1, 0) \cdot \underline{x}_1 \cdot \overline{x}_2 +$$

$$\min(1 - p_1, 1 - p_2) \cdot \underline{x}_1 \cdot \underline{x}_2,$$

*where* $p_i \stackrel{\text{def}}{=} (E_i - \underline{x}_i)/(\overline{x}_i - \underline{x}_i)$.

When we only know the intervals $\mathbf{E}_i$ of possible values of $E_i$, instead of the values $p_i$, we have the corresponding intervals $\mathbf{p}_i = (\mathbf{E}_i - \underline{x}_i)/(\overline{E}_i - \underline{x}_i)$. In terms of these intervals, we get the following results:

**Theorem 8.** *For multiplication under no information about dependence, to find* $\underline{E}$*, it is sufficient to consider the following combinations of* $p_1$ *and* $p_2$*:*

- $p_1 = \underline{p}_1$ *and* $p_2 = \underline{p}_2$*;* $p_1 = \underline{p}_1$ *and* $p_2 = \overline{p}_2$*;* $p_1 = \overline{p}_1$ *and* $p_2 = \underline{p}_2$*;* $p_1 = \overline{p}_1$ *and* $p_2 = \overline{p}_2$*;*
- $p_1 = \max(\underline{p}_1, 1 - \overline{p}_2)$ *and* $p_2 = 1 - p_1$ *(if* $1 \in \mathbf{p}_1 + \mathbf{p}_2$*); and*
- $p_1 = \min(\overline{p}_1, 1 - \underline{p}_2)$ *and* $p_2 = 1 - p_1$ *(if* $1 \in \mathbf{p}_1 + \mathbf{p}_2$*).*

*The smallest value of* $\underline{E}$ *for all these cases is the desired lower bound* $\underline{E}$*.*

**Theorem 9.** *For multiplication under no information about dependence, to find* $\overline{E}$*, it is sufficient to consider the following combinations of* $p_1$ *and* $p_2$*:*

- $p_1 = \underline{p}_1$ *and* $p_2 = \underline{p}_2$*;* $p_1 = \underline{p}_1$ *and* $p_2 = \overline{p}_2$*;* $p_1 = \overline{p}_1$ *and* $p_2 = \underline{p}_2$*;* $p_1 = \overline{p}_1$ *and* $p_2 = \overline{p}_2$*;*
- $p_1 = p_2 = \max(\underline{p}_1, \underline{p}_2)$ *(if* $\mathbf{p}_1 \cap \mathbf{p}_2 \neq \varnothing$*); and*
- $p_1 = p_2 = \min(\overline{p}_1, \overline{p}_2)$ *(if* $\mathbf{p}_1 \cap \mathbf{p}_2 \neq \varnothing$*).*

*The largest value of* $\overline{E}$ *for all these cases is the desired upper bound* $\overline{E}$*.*

- For the *inverse* $y = 1/x_1$, bounds for $E$ can be deduced from convexity [57]: $\mathbf{E} = [1/E_1, p_1/\overline{x}_1 + (1 - p_1)/\underline{x}_1]$.
- For min and independent $x_i$, we have $\overline{E} = \min(E_1, E_2)$ and

$$\underline{E} = p_1 \cdot p_2 \cdot \min(\overline{x}_1, \overline{x}_2) + p_1 \cdot (1 - p_2) \cdot \min(\overline{x}_1, \underline{x}_2) + (1 - p_1) \cdot p_2 \cdot \min(\underline{x}_1, \overline{x}_2) +$$

$$(1 - p_1) \cdot (1 - p_2) \cdot \min(\underline{x}_1, \underline{x}_2).$$

- For max and independent $x_i$, we have $\underline{E} = \max(E_1, E_2)$ and

$$\overline{E} = p_1 \cdot p_2 \cdot \max(\overline{x}_1, \overline{x}_2) + p_1 \cdot (1 - p_2) \cdot \max(\overline{x}_1, \underline{x}_2) + (1 - p_1) \cdot p_2 \cdot \max(\underline{x}_1, \overline{x}_2) +$$

$$(1 - p_1) \cdot (1 - p_2) \cdot \max(\underline{x}_1, \underline{x}_2).$$

- For min in the general case, $\overline{E} = \min(E_1, E_2)$,

$$\underline{E} = \max(p_1 + p_2 - 1, 0) \cdot \min(\overline{x}_1, \overline{x}_2) + \min(p_1, 1 - p_2) \cdot \min(\overline{x}_1, \underline{x}_2) +$$

$$\min(1 - p_1, p_2) \cdot \min(\underline{x}_1, \overline{x}_2) + \max(1 - p_1 - p_2, 0) \cdot \min(\underline{x}_1, \underline{x}_2).$$

– For max in the general case, $\underline{E} = \max(E_1, E_2)$ and

$$\overline{E} = \min(p_1, p_2) \cdot \max(\overline{x}_1, \overline{x}_2) + \max(p_1 - p_2, 0) \cdot \max(\overline{x}_1, \underline{x}_2) +$$

$$\max(p_2 - p_1, 0) \cdot \max(\underline{x}_1, \overline{x}_2) + \min(1 - p_1, 1 - p_2) \cdot \max(\underline{x}_1, \underline{x}_2).$$

– Similar formulas can be produced for the cases when there is a strong correlation between $x_i$: namely, when $x_1$ is (non-strictly) increasing or decreasing in $x_2$.

**From Elementary Arithmetic Operations to General Algorithms: First Idea.** In general, we have an algorithm $f(x_1, \ldots, x_n)$ from $R^n$ to $R$, we have $n$ intervals $\mathbf{x}_i$ that contain the actual (unknown) values of $x_i$, and we have $n$ more intervals $\mathbf{E}_i$ that contain the expected values $E[x_i]$. Our goal is to find the range $\mathbf{y}$ of $y = f(x_1, \ldots, x_n)$ and the range $\mathbf{E}$ of possible values of $E[y]$.

To compute $\mathbf{y}$, we can use known interval computations techniques. How can we compute $\mathbf{E}$? Our first idea is to compute $\mathbf{E}$ along the same lines as straightforward interval computations:

– first, we parse the algorithm $f(x_1, \ldots, x_n)$, i.e., we represent this algorithm as a sequence of arithmetic operations;
– then, we replace each elementary operation on real numbers with the corresponding operation on quadruples $(\underline{x}, \underline{E}, \overline{E}, \overline{x})$.

At the end, as one can easily prove, we get an enclosure for the range $\mathbf{y}$ of $y$ *and* an enclosure for the range $\mathbf{E}$ of $E[y]$.

**From Elementary Arithmetic Operations to General Algorithms: Second Idea.** When we have a complex algorithm $f$, then the above step-by-step approach leads to excess width. How can we find the actual range of $E = E[y]$?

At first glance, the exact formulation of this problem requires that we use infinitely many variables, because we must describe all possible probability distributions on the box $\mathbf{x}_1 \times \ldots \times \mathbf{x}_n$ (or, in the independent case, all possible tuples consisting of distributions on all $n$ intervals $\mathbf{x}_1, \ldots, \mathbf{x}_n$). It turns out, however, that we can reformulate these problems in equivalent forms that require only finitely many variables:

**Theorem 10.** *For a general continuous function $f(x_1, \ldots, x_n)$, $\underline{E}$ is a solution to the following optimization problem: $\sum_{j=0}^{n} p^{(j)} \cdot f(x_1^{(j)}, \ldots, x_n^{(j)}) \to \min$ under the conditions*

$$\sum_{k=0}^{n} p^{(k)} = 1; \quad p^{(j)} \geq 0; \quad \underline{x}_i \leq x_i^{(j)} \leq \overline{x}_i; \quad \underline{E}_i \leq \sum_{j=0}^{n} p^{(j)} \cdot x_i^{(j)} \leq \overline{E}_i \quad (\text{for all } i, j),$$

*and $\overline{E}$ is a solution to $\sum_{j=0}^{n} p^{(j)} \cdot f(x_1^{(j)}, \ldots, x_n^{(j)}) \to \max$ under the same constraints.*

Thus, by using validated tools for solving the corresponding optimization problem, we can find the desired range of $E[y]$.

### 3.3 Open Problems

So far, we have provided explicit formulas for the elementary arithmetic operations $f(x_1, \ldots, x_n)$ for the case when we know the first order moments. What if, in addition to that, we have some information about second order (and/or higher order) moments of $x_i$? What will we be then able to conclude about the moments of $y$? Partial answers to this question are given in [37,57,62]; it is desirable to find a general answer.

## 4 Exact Real Arithmetic

In the last two decades, the theory of computability on (the *full* set of) real numbers developed very fast. Although there still are competing approaches, the Type-2-Theory of Effectivity (TTE) seems to be the most evolved [12,33, 63]. Several software packages for exact real arithmetic have been written based on the concepts from TTE or equivalent theories. They all allow functional or imperative programming with *atomic* real objects $x \in \mathrm{R}$, while still being fully consistent with real calculus. In this section, we will compare important aspects of these implementations.

### 4.1 Non-real Arithmetic

The starting point for any practical application of arithmetic is hardware-based fixed-size integer or floating point arithmetic (today usually 32 or 64 bit). For integer arithmetic, we have a canonical and rather intuitive semantics. This is not true for floating point numbers: Here the semantics is no longer canonical, but got so complicated with constructs like NaNs, infinities, +0, -0, directed roundings etc. that it was necessary to define the IEEE standards 754/854 to get a reliable behavior of this arithmetic.

Leaving hardware-based size arithmetic, there are the sets of integer or rational numbers, where again we have a canonical, 'mathematical' semantics that does not need to be standardized. At this time, the most prominent open source implementation on these sets surely is GMP [25], written in `C` with hand-optimized assembler parts.

Within the rational numbers, the multiple precision floating point numbers (i.e. generalizations of the hardware floating point numbers) play a special role: Again, there is no 'canonical', pure 'mathematical' semantics. Instead, the result of an operation like the division of 1 by 3 does not only depend on the arguments themselves, but also on additional parameters like the size available for the result, the chosen rounding mode etc. Some software packages, e.g. the important but older `Fortran77` MP package [11], do not even try to explicitly define the result of such operations, they only give a verified bound for the error. Others, namely

the MPFR package [66], again try to follow the spirit of the IEEE standards 754/854 as closely as possible.

Extending the set of rational numbers, there is the set of algebraic numbers, where still it is possible to implement a pure mathematical oriented semantics. The disadvantage of corresponding implementations is that the evaluation of deeply nested arithmetic expressions (like the solution of linear systems) becomes almost infeasible. See e.g. [42], page 116: *...you may have to wait a long time for the answer when the expression is complex.*

### 4.2 The Border of Decidable Equality

For all of the different types of arithmetic above, it was possible to decide whether two data structures $x$, $y$ depict the same number, i.e. a test on equality $\mathrm{val}(x) = \mathrm{val}(y)$ was decidable with more or less effort! In contrast to this, equality is not a decidable operation in TTE. This is the most obvious difference to the BSS model [9], where equality of real numbers was taken as a *basic* operation!

So an important question concerning the decidability of equality is: How far can equality be implemented in an usable manner beyond the algebraic numbers? To illustrate the problems arising we recall one still unproven attempt, the *Uniformity Conjecture* [56], in a slightly simplified version below. This conjecture tries to extend the decidability to expressions that just allow exponentiation and logarithm in addition to the algebraic operations:

Consider expressions $E$ built as follows:

- basic objects are integers in decimal form
- expressions may be built iteratively using $A{+}B$, $A{-}B$, $A{*}B$, $A/B$, $(A)$, $-A$, $e^A$, $\ln A$, $\sqrt[n]{A}$ from given expressions $A$ and $B$ (and integer constants $n$).

Now the conjecture is: If $E$ does not contain subexpressions $E' = e^{E''}$ where the value $val(E'')$ of $E''$ is bigger than 1 (i.e. if $E$ does not use exponentiation to create large numbers intermediately), then

$$\text{either} \quad val(E) = 0 \quad \text{or} \quad |val(E)| > 10^{-1.3 \cdot len(E)}$$

where the length $len(E)$ is simply the length of the string denoting $E$.

If this conjecture turned out to be true, then we would be able to decide whether two values $a$ and $b$ are equal by approximating the value $a - b$ with a sufficient high precision (depending on the length of the expressions $A$ and $B$ yielding $a$ and $b$). But if we have a closer look on an example, we see that the conjecture would perhaps not be too helpful in practice: Just consider the following nonlinear iteration (sometimes called *logistic equation*) used as an important example e.g. by [36]

$$x_{i+1} = 3.75 \cdot x_i \cdot (1 - x_i) \quad , \quad x_0 = 1/2 \tag{8}$$

If we try the uniformity conjecture on this example, we get a sequence of expressions $E_i$ with $len(E_{i+1}) = 2 \cdot len(E_i) + 10$ (using $3.75 = 15/4$ with length

4), so already testing whether $x_{25} - x_{25}$ is equal to zero gives an expression $E$ with $len(E) \approx 870,000,000$; this would imply that we would need to evaluate $E$ to far more than one billion decimal digits to get a reliable answer. So today, we must face severe problems if we try to implement an arithmetic allowing decidable equality also for non-algebraic numbers.

### 4.3 Approximate Real Arithmetic

Deciding equality has already been dropped for a large class of computations: Interval arithmetic, either for hardware based on the standards IEEE 754/854 or for software solutions with variable size.

Two recent implementation in this area are `filib++` [39] (allowing IEEE 754/854 floats as interval borders) and `MPFI` [55], a multiple precision interval arithmetic library in `C`, based on `MPFR`. Of course, the use of interval software implies that the user 'thinks' in intervals, i.e. we have the look and feel of interval arithmetic.

A well-known approach by O. Aberth goes beyond this: In his *precise computation software* [1] he implemented an (almost) exact arithmetic, using 'range' arithmetic based on an own floating point software. This package, written in `C++`, is freely available on the internet (unfortunately, it does not compile cleanly with the recent `gcc3.x` compilers). It contains basic arithmetic, but extended with a calculus on elementary functions allowing $+$, $-$, $*$, $/$, $x^y$, sin, cos, tan, asin, acos, atan, $e^x$, $\ln x$, max, min as basic operations as well as e.g. integration and differentiation as higher level operators. Aberth uses a data type representing real numbers (constructible from the operations above, so we have the *look* of an exact real arithmetic). But the user still gets the *feel* of interval arithmetic: The implementation with range arithmetic is still essentially interval based, and these intervals may grow too large during computations leading to failures due to insufficient precision. An implementation of the sequence for the logistic equation (8) may look as follows:

```
long n, prec;
cin >> n; cin >> prec;  set_precision(prec);
real x=1/real(2);  real c=375/real(100);
for (i=1; i <= n; i++) {
  x=c*x*(one-x);
  if (i%10==0)  cout<<i<<" "<<x.str(2,20)<<endl;
}
```

If the second input parameter `prec` is too small for a first parameter $n$. the program fails. On the other hand, if the parameter is much too large, the computation time is unnecessarily high.

### 4.4 Implementations for Exact Real Arithmetic

The main part of this section is a comparison of the following packages:

- CRCalc (Constructive Reals Calculator, [10])
- XR (eXact Real arithmetic, [14])
- IC Reals (Imperial College Reals, [22])
- iRRAM (iterative Real RAM, [45])

Of course, there exist a lot more packages, e.g. the 'Manchester Reals' package by David Lester, which unfortunately is not available to the public at the moment. A test of this package together with the IC Reals, iRRAM, and a few others can be found in [7].

Some common basic concepts of exact real arithmetic are the following:

- Real numbers are atomic objects. The arithmetic is able to deal with (almost) arbitrary real numbers, but the usual entrance to R is Q.
- The implementations try to follow the theory of computability on the real numbers. This implies that computable functions are continuous, so tests of equality of numbers are not possible in general (they usually lead to an infinite loop if the arguments to be tested are equal).
- An important relaxation (called *multi-valued functions*) of the continuity restriction has been introduced by [13] and is implemented only in the iRRAM. A similar but less general concept are *lazy booleans* that first appeared in the IC Reals.

Two different basic methods of evaluation can be found in the packages:

- *Explicit computation diagrams*: During any computation, computation diagrams are built and maintained, leading to a quite high memory consumption. These diagrams are evaluated only at need using techniques like *lazy evaluation*, a concept primarily developed for functional programming languages. The evaluation of the diagrams usually is top-down, i.e. a recursive traversal from the root (giving the result) to the leaves (containing the arguments).
- *Implicit computation diagrams*: Instead of explicitly storing the diagrams (containing the full information on their real values), only snapshots of values are maintained. In addition, a small amount of relevant information (called decision history or multi-valued cache) is kept in order to be able to reconstruct better approximations at need. This in general implies that parts of a computation or even a whole computation have to be iterated. In addition, the evaluation of the computations could be called bottom-up, as it necessarily proceeds from the arguments of a computation to its result.

Before comparing the performance of the four packages, we would like to point out some characteristic properties for each of the packages:

- **CRCalc (Constructive Reals Calculator)**: This package by H. Boehm is a `JAVA` implementation. During a computation, it constructs explicit computation diagrams using methods from object oriented programming. At need, these diagrams are evaluated top down representing multiple precision numbers as scaled `BigInteger`. A sample program for the logistic equation sequence looks as follows:

```
CR one = CR.valueOf(1);
CR C = CR.valueOf(375).divide(CR.valueOf(100));
CR X = one.divide(CR.valueOf(2));
for (int i=1;i<param;i++){
   X= C.multiply(X).multiply(one.subtract(X));
   if (i%10==0) {
     { System.out.print(i); System.out.print(" ");
       System.out.println(X.toString(20));  }
}
```

– **XR (eXact Real arithmetic)**: Based on an extension `FC++` of `C++` towards functional languages, Keith Briggs implemented `XR`, where a real number $x$ is represented as a function $\lambda : \mathrm{Z} \to \mathrm{Z}$ in `FC++` via `typedef Fun1<int,Z> lambda`. This representation is defined as $x = \lim_{i\to\infty} \lambda(i) \cdot 2^{-i}$, i.e. $\lambda(i)$ is an approximation to $x$ with absolute error $2^{-i}$. Hence it is reasonable to implement the argument type of $\lambda$ as ordinary 32-bit-integers, but to use arbitrarily long integers as result type. The sample program looks like

```
int i;
XR c=QQ(375,100),x=QQ(1,2);
cout<<setprecision(20);
for (i=0; i<=param; i++) {
  x=c*x*(1-x);
  if (i%10==0) cout<<i<<" "<<x<<endl;
}
```

– **IC Reals (Imperial College)**: The previous examples were implementations, where the underlying representations for a real number $x$ essentially were normed Cauchy sequences $a_i$ with $|x - a_i| \leqslant 2^{-i}$. Errington, Krznaric, Heckmann, et al. used linear fractional transformations (e.g. [21]) instead to implement a `C`-package. These LFTs are a generalization of continued fractions, here a real number is represented by a sequence of integers $(a_i, b_i, c_i, d_i)$ (using GMP big integers) with $x_{i+1} = \dfrac{a_i x_i + b_i}{c_i x_i + d_i}$ and $x = \lim x_i$. Again we have explicit computation diagrams with a top-down lazy evaluation. One remarkable point is the use of lazy boolean predicates together with a multi-valued evaluation `realIF` that allows to implement non-continuous overlapping choices, like e.g.

```
y=realIF(2, x<1,a, x>-1,b)
```

to implement the following assignment

$$y := \begin{cases} a, \text{ if } x < 1 \\ b, \text{ if } x > -1 \end{cases}$$

– **iRRAM (iterative Real RAM)**: This C++-package written by Müller is the only package for exact real arithmetic that does not use explicit computation diagrams. Instead, it works with finite precision interval representations of real numbers similar to Aberth' package. A big difference is a built-in mechanism to repeat computations in case of failing interval operations. Additionally, many concepts from TTE have been implemented, like operators for the evaluation of limits of sequences or explicit multivalued functions as well as lazy boolean similar to the IC reals package. The example function here looks as follows:

```
REAL x = 0.5;  REAL c = 3.75;
for ( int i=1; i<=param; i++ ) {
  x= c*x*(1-x);
  if ( (i%10)==0 ) { rwrite(x,18); rprintf(" %d\n",i); }
}
```

In the following we will compare the different implementations using two examples: the sequence from the logistic equation (8) and the harmonic series.

The maintenance of explicit computation diagrams can be very hard concerning memory consumption, nevertheless building the diagrams for the logistic sequence to e.g. $x_n$ with $n \approx 10000$ should pose no problems here. On the other hand, due to its recursive definition and its chaotic nature, the evaluation of $x_n$ is quite difficult. Ordinary floating point hardware delivers totally wrong values for $x_n$ from about $n \approx 100$. If $x_n$ is computed with interval methods, the sizes of the intervals grow almost with the involved factor of 3.75. So to get an approximation of $x_n$ with an error of $2^{-k}$, we need initial approximations for $x_0$, $x_1$ etc. with a precision of about $2^{-k-1.91n}$, i.e. for $n = 10000$ a precision of about 19000 bits is required. For the benchmarks, we tried to compute $x_n$ with about 20 significant decimals for different values of $n$, using the example programs from above.

A second example, where the loss of precision is much smaller, is the computation of the harmonic series $h(n) := \sum_{i=1}^{n} \frac{1}{i}$ We still have deeply nested operations, so this is a simple example to model e.g. the effects of basic linear algebra. We implemented $h(n)$ in all the packages and measured the time necessary to compute approximations to $h(n)$ with 10 decimals for rather large $n$.

The following timing results were obtained on a Pentium-3 with 1200 MHz; here "—" indicates that computations took longer than an hour or used more that 500 MB memory, so we canceled them.

| package | logistic sequence | | harmonic series | | |
|---|---|---|---|---|---|
| | n=1,000 | n=10,000 | n=5,000 | n=50,000 | n=5,000,000 |
| CRCalc | 1359 sec | — | 325 sec | — | — |
| XR2.0 | 423 sec | — | 2.48 sec | 2027 sec | — |
| IC-Reals | 1600 sec | — | 0.85 sec | — | — |
| Aberth | 0.5 sec | 1468 sec | ¡0.1 sec | 0.3 sec | 1835 sec |
| iRRAM | ¡0.1 sec | 17 sec | ¡0.1 sec | 0.1 sec | 8.5 sec |

The timings show that, at least for the two given problems, the advantage of the iterative approach compared to the explicit computation diagrams is so dramatic that the explicit approach seems to be unrealistic. In the package of Aberth, the error propagation seems to be done in an suboptimal way: The precision needed for the logistic sequence at $n = 10000$ was about 58000 bits instead of the sufficient 19000 bits. The same holds for the harmonic series, here a precision of about 14000 bits was necessary to compute $h(n)$ for $n = 5,000,000$. The iRRAM was able to do this using an internal precision of less than 50 bits.

The example of the harmonic series shows that the iRRAM is capable to deliver about 1 MFlops on the given CPU. As a comparison: The interval arithmetic `filib++` [39] based on hardware floats delivers about 8-22 MFlops on a Pentium IV with 2000 MHz. If we take into account the different speeds of the CPUs, then the exact arithmetic is just a factor 5 to 10 slower than a hardware based interval arithmetic, at least for cases where precision is not a critical factor.

To consider the influence of the necessary precision, we additionally used the iRRAM to compute approximations (with maximal error $2^{-50}$) of the inverse of the (bad conditioned) Hilbert matrix $H_n$ of size $n \times n$ using Gaussian elimination and compared this to the same computation applied to the well conditioned matrix $H_n + 1_n$ of the same size $n \times n$:

| $n$ | 50 | 100 | 150 | 200 | 250 | 500 |
|---|---|---|---|---|---|---|
| inversion of well conditioned matrix $H_n + 1_n$ of size $n \times n$ | | | | | | |
| bits | 100 | 100 | 100 | 100 | 100 | 162 |
| time | 0.7 s | 5.4 s | 19 s | 45s | 91 s | 1237 s |
| inversion of Hilbert matrix $H_n$ of size $n \times n$ | | | | | | |
| bits | 1037 | 2745 | 4372 | 5502 | 6915 | — |
| time | 3.2 s | 79 s | 457 s | 1200 s | 3052 s | — |

Obviously, the condition of the matrix has big influence on the internal precision that is maintained automatically by the iRRAM package, which explains the big differences in execution time between the two examples.

Similar computations were done using `octave` (a freely available high-level interactive language for numerical computations without interval arithmetic). `octave` is already unable to invert the Hilbert matrix of size 12. On the other hand, the inversion of the well-conditioned matrix $H_n + 1_n$ with $n = 500$ takes only 18.8 s, so here the iRRAM is about a factor of 65 slower.

As a last example, we compared the performance of a few trigonometric functions between `MPFR` (using software arithmetic with non-interval methods but with verified roundings) and an extension to `MPFR` (found in the iRRAM package) that uses subroutines from the iRRAM to compute those functions:

| | 10 decimals | | 10000 decimals | |
|---|---|---|---|---|
| x=$\sqrt{3}$ | MPFR | MPFR+iRRAM | MPFR | MPFR+iRRAM |
| e(x) | 0.0117 ms | 0.0577 ms | 201 ms | 1080 ms |
| ln(x) | 0.0388 ms | 0.0696 ms | 105 ms | 109 ms |
| sin(x) | 0.0427 ms | 0.0745 ms | 403 ms | 187 ms |
| cos(x) | 0.0270 ms | 0.0678 ms | 282 ms | 184 ms |

Again, the overhead due to the much more elaborate exact arithmetic in the iRRAM is remarkably small, in some cases the algorithms using exact arithmetic were even faster.

As a summary, we may say that exact real arithmetic is on its way to be useful either as a reference implementation or as a tool to handle precision critical computations.

## 5 Conclusions

Traditional design of numerical software with result verification is based on the assumption that we know the algorithm $f(x_1, \ldots, x_n)$ that transforms input $x_1, \ldots, x_n$ into the output $y = f(x_1, \ldots, x_n)$, and we know the intervals $\mathbf{x}_1, \ldots, \mathbf{x}_n$ of possible values of the inputs. Many real-life problems go beyond this paradigm:

- In some cases, we do not have an algorithm $f$, we only know some relation (constraints) between $x_i$ and $y$.
- In other cases, in addition to knowing the intervals $\mathbf{x}_i$ of possible values of $x_i$, we may have some additional information:
  - we may have known some relation *between* different quantities $x_i$;
  - we may also have some additional information about each of these quantities:
    - we may have some information about the *probabilities* of different values of $x_i$;
    - in some cases, we may even know the *exact* values of some of the inputs (e.g., we may know that $x_1 = \pi/2$).

To cover this additional information, in addition to traditional interval techniques, we must also have:

- techniques that translate known constraints between $x_i$ and $y$ into an algorithm that inputs the values $x_1, \ldots, x_n$ and computes the value(s) $y$ that satisfies all the given constraints (for given values of $x_i$);
- techniques that use the algorithm $f$, the ranges $\mathbf{x}_i$, *and* additional constraints between $x_i$ and $y$ to get a better estimate for the range of possible values of $y$;
- techniques that use the additional information about probabilities of different values of $x_i \in \mathbf{x}_i$ to come up with the information about the probabilities of possible values of $y = f(x_1, \ldots, x_n)$; and

– techniques that would enable us to deal with *exact* real numbers in addition to the numbers known with interval uncertainty.

In this paper, we describe the approaches for designing these techniques. The main remaining challenge is to *combine* these techniques into a single working tool.

**Acknowledgments.** L.G. was supported by PAI-Procope project. V.K. was supported by NASA grant NCC5-209, by the AFOSR grant F49620-00-1-0365, by NSF grants EAR-0112968 and EAR-0225670, by IEEE/ACM SC2001 and SC2002 Minority Serving Institutions Participation Grants, by a research grant from Sandia National Laboratories as part of the Department of Energy Accelerated Strategic Computing Initiative (ASCI), and by Small Business Innovation Research grant 9R44CA81741 to Applied Biomathematics from the National Cancer Institute (NCI), a component of NIH.

The authors are thankful to the organizers of the Dagstuhl meeting for their support and encouragement, and to the anonymous referees for valuable suggestions.

## References

1. Aberth, O.: Precise Numerical Methods Using C++, Academic Press, Boston (1998)
2. Apt, K.R.: The Role of Commutativity in Constraint Propagation Algorithms. ACM Transactions on Programming Languages and Systems **22**, No. 6 (2000) 1002–1036
3. Benhamou, F., Goualard, F. Universally Quantified Interval Constraints. In: Dechter, R. (ed.): Proceedings of of CP'2000, Principles and Practice of Constraint Programming, Springer Lecture Notes in Computer Science **1894** (2000) 67–82
4. Benhamou, F., Goualard, F., Granvilliers, L., and Puget, J.-F.: Revising Hull and Box Consistency. In de Schreye, D. (ed.): Proceedings of ICLP'99, International Conference on Logic Programming, Las Cruces, USA, MIT Press (1999) 230–244
5. Benhamou, F., McAllester, D., and Van Hentenryck, P.: CLP(Intervals) Revisited. In: Bruynooghe, M. (ed.): Proceedings of ILPS'94, International Logic Programming Symposium, Ithaca, USA, 1994. MIT Press (1994) 124–138.
6. Benhamou, F., Older, W.J.: Applying Interval Arithmetic to Real, Integer and Boolean Constraints. Journal of Logic Programming **32**, No. 1 (1997) 1–24
7. Blanck, J.: Exact Real Arithmetic Systems: Results of Competition, Springer Lecture Notes in Computer Science **2064** (2001) 389–394
8. Bliek, C., Neveu, B., Trombettoni, G.: Using Graph Decomposition for Solving Continuous CSPs. In: Maher, M., J.-F. Puget, J.-F. (eds.): Proceedings of CP'98, Principles and Practice of Constraint Programming, Springer Lecture Notes on Computer Science **1520** (1998) 102–116
9. Blum, L., Shub, M., Smale, S.: On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines, Bulletin of the AMS **21** (July 1989) 1
10. Boehm, H. Constructive Reals Calculator, `http://www.hpl.hp.com/personal/Hans_Boehm/new_crcalc/CRCalc.html`

11. Brent, R.P.: A Fortran multiple precision package, ACM Trans. Math. Software **4** (1978) 57–70
12. Brattka, V.: Recursive characterisation of computable real-valued functions and relations, Theoret. Comput. Sci. **162** (1996) 47–77
13. Brattka, V., Hertling, P.: Continuity and Computability of Relations, Informatik Berichte 164-9/1994, FernUniversität Hagen
14. Briggs, K. XR exact real arithmetic, `http://more.btexact.com/people/briggsk2/XR.html`
15. Christie, M., Languénou, E., Granvilliers, L.: Modeling Camera Control with Constrained Hypertubes. In: Van Hentenryck, P. (ed.): Proceedings of CP'2002, Principles and Practice of Constraint Programming, Springer Lecture Notes on Computer Science **2470** (2002) 618–632
16. Cleary, J.G.: Logical Arithmetic. Future Computing Systems **2**, No. 2 (1987) 125–149
17. Collavizza, H., Delobel, F., Rueher, M.: Comparing Partial Consistencies. Reliable Computing **5**, No. 3 (1999) 213–228
18. Collavizza, H., Delobel, F., Rueher, M.: Extending Consistent Domains of Numeric CSP. In: Proceedings of IJCAI'99, International Joint Conference on Artificial Intelligence, Morgan Kaufmann (1999) 406–413
19. Davis, E.: Constraint Propagation with Interval Labels. Artificial Intelligence **32** (1987) 281–331
20. Deville, Y., Jansen, M., Van Hentenryck, P.: Consistency Techniques in Ordinary Differential Equations. In: Proceedings of CP'1998, Principles and Practice of Constraint Programming, Springer Lecture Notes on Computer Science **1520** (1998) 162–176
21. Edalat, A., Heckmann, R.: Computing with real numbers: (i) LFT approach to real computation, (ii) Domain-theoretic model of computational geometry. In: Barthe, G., Dybjer, P., Pinto, L., and Saraiva, J. (eds): Springer Lecture Notes in Computer Science (2002)
22. Errington, L., Heckmann, R.: Using the IC Reals library, `http://www.doc.ic.ac.uk/~ae/exact-computation/ ic-reals-manual.pdf`
23. Ferson, S., Ginzburg, L., Kreinovich, V., Longpré, L., Aviles, M.: Computing Variance for Interval Data is NP-Hard, ACM SIGACT News **33** (2002) 108–118.
24. Fischer, X., Nadeau, J.-P., Sébastian, P., Joyot, P.: Qualitative Constraints in Integrated Design. In: Chedmail, P. Cognet, G., Fortin, C., Mascle, C., Pegna, J. (eds.): Proceedings of IDMME'2000, Integrated Design and Manufacturing in Mechanical Engineering Conference, Montréal, Canada, Kluwer Academic Publishers (2002) 35–42
25. Granlund, T.: GMP 4.1, `http://www.swox.com/gmp/`
26. Granvilliers, L.: On the Combination of Interval Constraint Solvers. Reliable Computing **7**, No. 6 (2001) 467–483
27. Granvilliers, L., Benhamou, F.: Progress in the Solving of a Circuit Design Problem. Journal of Global Optimization **20**, No. 2 (2001) 155–168
28. Granvilliers, L., Monfroy, E., Benhamou, F.: Symbolic-Interval Cooperation in Constraint Programming. In: Proceedings of ISSAC'2001, International Symposium on Symbolic and Algebraic Computations, ACM Press (2001) 150–166
29. Hansen, E.: Sharpness in interval computations, Reliable Computing **3** (1997) 7–29.
30. Jaulin, L. Kieffer, M., Didrit, O., Walter, E.: Applied Interval Analysis: With Examples in Parameter and State Estimation, Robust Control and Robotics, Springer, London (2001)

31. Kearfott, R.B.: Rigorous Global Search: Continuous Problems, Kluwer, Dordrecht (1996)
32. Kearfott, R.B., Kreinovich, V. (eds.): Applications of Interval Computations, Kluwer, Dordrecht (1996)
33. Ko, K.-I.: Complexity Theory of Real Functions, Birkhäuser, Boston (1991)
34. Kreinovich, V.: Probabilities, Intervals, What Next? Optimization Problems Related to Extension of Interval Computations to Situations with Partial Information about Probabilities, Journal of Global Optimization (to appear).
35. Kreinovich, V., Lakeyev, A., Rohn, J., Kahl, P.: Computational Complexity and Feasibility of Data Processing and Interval Computations, Kluwer, Dordrecht (1997)
36. Kulisch, U.: Memorandum über Computer, Arithmetik und Numerik, Universität Karlsruhe, Institut für Angewandte Mathematik (1996)
37. Kuznetsov, V.P.: Interval Statistical Models, Radio i Svyaz, Moscow (1991) in Russian
38. Lebbah, Y., Rueher, M., Michel, C.: A Global Filtering Algorithm for Handling Systems of Quadratic Equations and Inequations. In: Van Hentenryck, P. (ed.): Proceedings of CP'2002, Principles and Practice of Constraint Programming, Ithaca, NY, USA, Springer Lecture Notes in Computer Science **2470** (2002)
39. Lerch, M., Tischler, G., Wolff von Gudenberg, J., Hofschuster, W., Krämer, W.: The Interval Library filib++ 2.0 - Design, Features and Sample Programs, Preprint 2001/4, Universität Wuppertal (2001)
`http://www.math.uni-wuppertal.de/wrswt/literatur/lit_wrswt.html`
40. Lhomme, O.: Consistency Techniques for Numeric CSPs. In: Wahlster, W. (ed.): Proceedings of IJCAI'93, International Joint Conference of Artificial Intelligence, Chambéry, France, 1993. Morgan Kaufman (1993) 232–238
41. Lhomme, O., Gotlieb, A., Rueher, M.: Dynamic Optimization of Interval Narrowing Algorithms. Journal of Logic Programming **37**, No. 1–2 (1998) 165–183
42. Mehlhorn, K., Näher, S.: LEDA, Cambridge University Press (1999)
43. Moore, R.E.: Interval Analysis, Prentice-Hall, Englewood Cliffs, NJ (1966)
44. Moore, R.E.: Methods and Applications of Interval Analysis, SIAM, Philadelphia (1979)
45. Müller, N.Th.: iRRAM - Exact Arithmetic in C++,
`http://www.informatik.uni-trier.de/iRRAM/`
46. Nedialkov, N.S.: *Computing Rigorous Bounds on the Solution of an Initial Value Problem for an Ordinary Differential Equation.* PhD thesis, University of Toronto (1999)
47. Neumaier, A.: Interval Methods for Systems of Equations Cambridge University Press (1990)
48. Nivlet, P., Fournier, F., and Royer, J.: A new methodology to account for uncertainties in 4-D seismic interpretation, Proceedings of the 71st Annual International Meeting of the Society of Exploratory Geophysics SEG'2001, San Antonio, Texas, September 9–14 (2001) 1644–1647.
49. Nivlet, P., Fournier, F., Royer, J.: Propagating interval uncertainties in supervised pattern recognition for reservoir characterization, Proceedings of the 2001 Society of Petroleum Engineers Annual Conference SPE'2001, New Orleans, Louisiana, September 30–October 3 (2001) paper SPE-71327.
50. W. Older, W., Vellino, A.: Constraint Arithmetic on Real Intervals. In: Benhamou, F., Colmerauer, A. (eds.): Constraint Logic Programming: Selected Research, MIT Press (1993)

51. O'Sullivan, B.: Constraint-Aided Conceptual Design, PhD thesis, University College Cork (1999)
52. Puget, J.-F., Van Hentenryck, P.: A Constraint Satisfaction Approach to a Circuit Design Problem, Journal of Global Optimization **13**, No. 1 (1998) 75–93
53. Rabinovich, S.: Measurement Errors: Theory and Practice, American Institute of Physics, New York (1993)
54. Ratschan, S.: Continuous First-Order Constraint Satisfaction. In: Calmet, J., Benhamou, B., Caprotti, O., Henoque, L., Sorge, V. (eds.): Proceedings of AISC'2002, International Conference on Artificial Intelligence and Symbolic Computations, Springer Lecture Notes on Computer Science **2385** (2002) 181–195
55. Revol, N., Rouillier, F.: Motivations for an arbitrary precision interval arithmetic and the MPFI library Research report R 2002-27, LIP, École Normale Supérieure de Lyon (2002)
56. Richardson, D.: The Uniformity Conjecture, Proceedings of CCA2000, Springer Lecture Notes in Computer Science **2064** (2001) 253–272
`http://www.bath.ac.uk/~masdr/unif.dvi`
57. Rowe, N. C.: Absolute bounds on the mean and standard deviation of transformed data for constant-sign-derivative transformations, SIAM Journal of Scientific Statistical Computing **9** (1988) 1098–1113
58. Sam Haroud, D., Faltings, B.: Consistency Techniques for Continuous Constraints. Constraints **1** (1996) 85–118
59. Van Hentenryck, P., Mc Allester, D., Kapur, D.: Solving Polynomial Systems using a Branch-and-Prune Approach. SIAM Journal on Numerical Analysis **34**, No. 2 (1997) 797–827
60. Van Hentenryck, P., Michel, L., Deville, Y.: Numerica: a Modeling Language for Global Optimization. MIT Press (1997)
61. Vavasis, S.A.: Nonlinear Optimization: Complexity Issues, Oxford University Press, N.Y. (1991)
62. Walley, P.: Statistical Reasoning with Imprecise Probabilities, Chapman and Hall, N.Y. (1991)
63. Weihrauch, K.: Computable Analysis. An Introduction, Springer, Berlin (2000)
64. Yamamura, K., Kawata, H., Tokue, A.: Interval Analysis using Linear Programming. BIT **38** (1998) 188–201
65. Xiang, G.: unpublished manuscript.
66. Zimmermann, P.: MPFR: A Library for Multiprecision Floating-Point Arithmetic with Exact Rounding, 4th Conference on Real Numbers and Computers, Dagstuhl (2000) 89–90, `http://www.loria.fr/projets/mpfr/`

# Static Analysis-Based Validation of Floating-Point Computations

Sylvie Putot, Eric Goubault, and Matthieu Martel

CEA Saclay
91191 Gif-sur-Yvette Cedex, France
{sputot,egoubault,mmartel}@cea.fr

**Abstract.** Finite precision computations can severely affect the accuracy of computed solutions. We present a static analysis, and a prototype implementing this analysis for C codes, for studying the propagation of rounding errors occurring at every intermediary step in floating-point computations. The analysis presented relies on abstract interpretation by interval values and series of interval error terms. Considering all errors possibly introduced by floating-point numbers, it aims at identifying the operations responsible for the main losses of accuracy. We believe this approach is for now specially appropriate for numerically simple programs which results must be verified, such as critical instrumentation software.

## 1 Introduction

The manipulation of real numbers by computers is carried on using floating-point arithmetic, which relies on a finite representation of numbers. Although this approximation is accurate enough for most applications, in some cases results become irrelevant. And in critical software, these cases may not be acceptable.

Some work has already been done towards tools for evaluating the accuracy of computations in software. The most widely used, Cadna, relies on statistical methods, and gives most of the time a very sharp estimation of the relevance of computed results. But some errors can be underestimated, which is not satisfying for the verification of critical applications, which accuracy must be certified. Moreover, this method allows to study the result of a particular execution, and not for infinite sets of input values as is most of the time needed. Alternatively, most existing interval-based techniques, which are sure and consider sets of executions, aim at estimating tight bounds for the result of computations in infinite precision. This often supposes a rewriting of the code to be analyzed, and moreover does not address the problem of verifying the accuracy of existing software.

On the contrary, we are not interested in computing bounds for the real result of a given problem, but for the error committed using finite precision computations instead of real numbers computations[1]. Moreover, the origin of the main losses of precision is most of the time very localized, and we aim at pointing out which parts of the code are responsible for these losses. For that, we decompose

[1] This presentation follows earlier work by the authors, see [3],[6],[4]

R. Alt et al. (Eds.): Num. Software with Result Verification, LNCS 2991, pp. 306–313, 2004.

the error between the results of the same computation achieved respectively with floating-point and real numbers in a sum of error terms corresponding to the elementary operations of this computation. This modelisation of the propagation of errors, called concrete semantics, is the topic of section 2.

This semantics can not be used in an analyzer, because the errors are real numbers, that can not always be represented by floating-point numbers, even with higher precision. Thus we derive an abstract semantics, which is the implementable version of the concrete semantics : over-approximations of the values and errors are computed using intervals. These intervals also allow to consider sets of input values. Static analysis consists in computing all possible values of the variables on the nodes of a program without executing it. A considerable issue is the fixed point computation in loops. This is presented in section 3.

A prototype implements this model, it is intended to cope with real problems. Special care was attached to the design of a graphic interface, that makes the large amount of information computed easily exploitable. The user can make sure that the floating-point computations are accurate enough, and identify the operations responsible for the main losses of accuracy.

## 2 Concrete Semantics to Interpret Arithmetic Operations

Let us first examine an introductory example in which we consider a simplified set $\mathbb{F}$ of floating-point numbers composed of a mantissa of four digits written in base 10. We consider $a$ and $b$ two intermediate computation results that are not computed exactly, and we note

$$a = 621.3 + 0.055\varepsilon_{\ell_1},\ \ b = 1.287 + 0.00055\varepsilon_{\ell_2}\ .$$

In this definition, $a \in \mathbb{R}$ and $b \in \mathbb{R}$ are the values that would be got from an infinite precision computation. The floating-point execution of the same computation gives $a_{\mathbb{F}} = 621.3 \in \mathbb{F}$ and $b_{\mathbb{F}} = 1.287 \in \mathbb{F}$, and an error of 0.055 was committed at point $\ell_1$ on the computation of $a$, and an error of 0.00055 was committed at point $\ell_2$ on the computation of $b$. The symbols $\varepsilon_{\ell_1}$ and $\varepsilon_{\ell_2}$ are formal variables related to the control points $\ell_1$ and $\ell_2$.

We now consider the product $c = a \times b$, at point $l_3$. The exact result of the product of the floating-point numbers is $a_{\mathbb{F}} \times b_{\mathbb{F}} = 799.6131$, but the nearest floating-point number, supposing the current rounding mode is to the nearest, is $c_{\mathbb{F}} = 799.6$. A rounding error, defined by $a_{\mathbb{F}} \times b_{\mathbb{F}} - c_{\mathbb{F}} = 0.0131$, is thus committed. The computation $a \times b$ in real numbers intended by the programmer, is then

$$a \times^{\ell_3} b = c_{\mathbb{F}} + 0.0131\varepsilon_{\ell_3} + 0.055 \times 1.287\varepsilon_{\ell_1} + 0.00055 \times 621.3\varepsilon_{\ell_2} + 0.055 \times 0.00055\varepsilon_{\ell_1}\varepsilon_{\ell_2}\ .$$

We keep only one term gathering the errors of order higher than one, and rewrite

$$a \times^{\ell_3} b = c_{\mathbb{F}} + 0.070785\varepsilon_{\ell_1} + 0.341715\varepsilon_{\ell_2} + 0.0131\varepsilon_{\ell_3} + 0.00003025\varepsilon_{hi}\ .$$

The initial errors are amplified or reduced by further computations, thus the error on $c$ is mainly due to the initial error on $b$. This result is quite obvious

on this very simple example, but would be much more difficult and tedious to establish by hand on larger programs. We aim at designing an automatic tool providing this kind of information.

We now introduce formally this semantics [6], that details the contribution to the global error of the first order error terms, and globally computes the higher order errors, which are most of the time negligible. Let $\mathbb{F}$ be either the set of simple or double precision floating-point numbers. Let $\uparrow_\circ : \mathbb{R} \to \mathbb{F}$ be the function that returns the rounded value of a real number $r$, with respect to the rounding mode $\circ$. The function $\downarrow_\circ : \mathbb{R} \to \mathbb{F}$ that returns the roundoff error is defined by

$$\forall f \in \mathbb{R},\ \downarrow_\circ (f) = f - \uparrow_\circ (f)\ . \tag{1}$$

Assume that the control points of a program are annotated by unique labels $\ell \in L$, and that $\mathcal{L}$ denotes the union of $L$ and the special word $hi$ used to denote all terms of order higher or equal to 2. A number $x$ is represented by

$$x = f^x + \sum_{\ell \in \mathcal{L}} \omega_\ell^x \boldsymbol{\varepsilon}_\ell\ . \tag{2}$$

In equation (2), $f^x$ is the floating-point number approximating the value of $x$. A term $\omega_\ell^x \boldsymbol{\varepsilon}_\ell$ denotes the contribution to the global error of the first-order error introduced by the operation labeled $\ell$, $\omega_\ell^x \in \mathbb{R}$ being the value of this error term and $\boldsymbol{\varepsilon}_\ell$ a formal variable labelling the operation $\ell$.

The result of an arithmetic operation $\diamond^{\ell_i}$ contains the combination of existing errors on the operands, plus a new roundoff error term $\downarrow_\circ (f^x \diamond f^y)\boldsymbol{\varepsilon}_{\ell_i}$. For addition and subtraction, the errors are added or subtracted componentwise :

$$x +^{\ell_i} y = \uparrow_\circ (f^x + f^y) + \sum_{\ell \in \mathcal{L}} (\omega_\ell^x + \omega_\ell^y)\boldsymbol{\varepsilon}_\ell + \downarrow_\circ (f^x + f^y)\boldsymbol{\varepsilon}_{\ell_i}\ .$$

The multiplication introduces higher order errors, we write :

$$x \times^{\ell_i} y = \uparrow_\circ (f^x f^y) + \sum_{\ell \in \mathcal{L}} (f^x \omega_\ell^y + f^y \omega_\ell^x)\, \boldsymbol{\varepsilon}_\ell + \sum_{\ell_1 \in \mathcal{L},\ \ell_2 \in \mathcal{L}} \omega_{\ell_1}^x \omega_{\ell_2}^y\, \boldsymbol{\varepsilon}_{hi} + \downarrow_\circ (f^x f^y)\boldsymbol{\varepsilon}_{\ell_i}\ .$$

The semantics for the division is obtained by a power series development :

$$(y)^{-1^{l_i}} = \uparrow_\circ \left(\frac{1}{f^y}\right) - \frac{1}{(f^y)^2} \sum_{\ell \in \mathcal{L}} \omega_\ell^y \boldsymbol{\varepsilon}_\ell + \frac{1}{f^y} \sum_{n \geq 2} (-1)^n \left(\sum_{\ell \in \mathcal{L}} \frac{\omega_\ell^y}{f^y}\right)^n \boldsymbol{\varepsilon}_{hi} + \downarrow_\circ \left(\frac{1}{f^y}\right) \boldsymbol{\varepsilon}_{\ell_i}\ .$$

## 3 Static Analysis and Abstract Interpretation

Static analysis consists in computing some properties of a program without executing it, and for possibly large or infinite sets of inputs. We want here to compute all possible values $f$ and errors $\omega_\ell$ for each variable, valid for any iteration of the loops, on the nodes of the programs to analyze. Interval computations [8] are used to get computable supersets of these coefficients, in an abstract interpretation framework [2]. They allow on one hand to consider sets of execution, and on the other hand to include the rounding errors committed by the analysis.

### 3.1 Abstract Semantics to Interpret Arithmetic Operations

Let us consider again the multiplication introduced in section 2. The errors are real numbers, they are not always representable by floating-point numbers. Thus we define the abstract semantics for the operation, that implements the concrete semantics, using intervals as computable supersets of the real coefficients. We suppose the numbers used for the analysis have a mantissa of five digits in base 10, then the multiplication of $a$ and $b$ with the abstract semantics writes :

$$a \times^{\ell_3} b = [c_{\mathbb{F}}, c_{\mathbb{F}}] + [0.070785, 0.070785]\boldsymbol{\varepsilon}_{\ell_1} + [0.34171, 0.34172]\boldsymbol{\varepsilon}_{\ell_2} \\ +[0.0131, 0.0131]\boldsymbol{\varepsilon}_{\ell_3} + [0.00003025, 0.00003025]\boldsymbol{\varepsilon}_{hi} \ .$$

The floating-point result is still the result $c_{\mathbb{F}}$ of the multiplication $a_{\mathbb{F}} \times b_{\mathbb{F}}$, rounded to the nearest floating-point number, with the precision of the floating-point number analyzed. This results simulates the floating-point execution.
The errors are computed using classical interval arithmetic, that is with outward rounding, to include the errors coming from the analysis which uses itself finite precision numbers. Using a higher precision for the computation of these error intervals allows to estimate them more tightly. Here, an extended precision to six digits would be enough to compute exactly the error, without the use of intervals. But it would be too costly to extend the precision for each additional operation. Moreover, some errors can not be represented by extended precision floating-point numbers, for example in some cases of divisions.

Now consider the same multiplication where the floating-point value of $a$ is no longer a single value, but any possible value in an interval : for example we take $a' = [610, 630] + [0.055, 0.055]\boldsymbol{\varepsilon}_{\ell_1}$. We get:

$$a' \times^{\ell_3} b = [785.1, 810.8] + [0.070785, 0.070785]\boldsymbol{\varepsilon}_{\ell_1} + [0.3355, 0.3465]\boldsymbol{\varepsilon}_{\ell_2} \\ +[-0.05, 0.05]\boldsymbol{\varepsilon}_{\ell_3} + [0.00003025, 0.00003025]\boldsymbol{\varepsilon}_{hi} \ .$$

Indeed, $610 \times 1.287 = 785.07$, rounded to the nearest gives 785.1, and $630 \times 1.287 = 810.81$, rounded to the nearest gives 810.8. Thus the floating-point part of the result can be any floating-point value in the interval $[785.1, 810.8]$. The error coming from point $\ell_1$ and the error of order higher than 1 are unchanged. The error coming from point $\ell_2$ belongs to the result of the interval multiplication (with outward rounding) of $a'_{\mathbb{F}}$ and the error 0.00055, that is $[0.3355, 0.3465]$. The roundoff error introduced by the multiplication can only be bounded by the largest set of values which added to the floating-point result, do not affect its value in floating-point arithmetic, that is the interval $[-0.05, 0.05]$.

In the general case, we get the abstract semantics by interpreting the operation over error series with interval coefficients, using rounding to the nearest for the computation of the floating-point part, and outward rounding and possibly more precision for the propagation of the existing errors. For the division, we must compute an over-approximation of the sum of the terms of order higher or equal to two. For that, we note that the error committed by approximating

$(1+u)^{-1}$ by the first-order development $1-u$ is $g(u) = (1+u)^{-1}u^2$. And we can easily bound $g(u)$ for $u = (f^y)^{-1}\sum_{\ell\in\mathcal{L}}\omega_\ell^y$ by studying function $g$.

Most of the time, the new roundoff error introduced by an operation can only be bounded. Suppose the floating-point result of an operation is in the interval $[a,b]$, and note $r = max(|a|,|b|)$. The roundoff error due to this operation is bounded by $[-ulp(r)/2, ulp(r)/2]$, where $ulp(r)$ is the unit in the last place of $r$, that is the smallest number which, added to the floating-point number $r$, does affect its value. If the floating-point parts of the operands are intervals reduced to points ($a = b$), the error can be bounded more accurately using (1), by the difference of the floating-point result, and the interval result of the same operation achieved with outward rounding and the precision of the analysis.

### 3.2 Computations in Loops

When encountering a loop, the analyzer will try to produce an *invariant*, i.e. a property which holds true before or after some instruction in the body of the loop, regardless of the number of loops already executed. As an example, look at the program:

```
int i=1;
while (i<100)
  (1): i++;
(2):
```

suitably annotated with labels `(1)` (respectively `(2)`), locating the control point at the beginning of the body of the loop, just before `i++` takes place (respectively, after the loop). A correct invariant at `(1)` is `i` in $S = \{1,2,\ldots,99\}$, because each time the control flow goes through `(1)`, `i` takes its value in $S$. Notice that $S' = [0,100]$ is also an invariant, but less precise. The most precise invariant at `(2)` is `i` equals 100.

If we represent the "effect" of one iteration of the loop on variables' values by a function $f$ (its "semantics"), then calculating `(1)` amounts to finding the least fixed point of $f$, above some initial set of values $X_0$. Equivalently (when $f$ is "continuous"), the invariant $i_{(1)}$ at `(1)` - only concerning variable `i` here - is given by Kleene's theorem:

$$i_{(1)} = X_0 \cup f(X_0) \cup f^2(X_0) \cup \ldots \cup f^n(X_0) \cup \ldots$$

This gives an immediate algorithm for computing the invariant, called the *fixed point iteration sequence*, in which we start with $i^0_{(1)} = X_0 = [1,1]$ and carry on by defining *(it):* $i^{n+1}_{(1)} = i^n_{(1)} \cup f\,(i^n_{(1)})$, the limit of which being the least fixed point in question.

In our example, $f(S) = ([1,1] \cup (S+[1,1])) \cap ]-\infty, 99]$. The iteration sequence is then $i^0_{(1)} = [1,1]$, $i^1_{(1)} = [1,2]$, ..., $i^j_{(1)} = [1, j+1]$ and finally, $i^{99}_{(1)} = i^{98}_{(1)} = [1,99]$ (the fixed point). This algorithm is not very efficient in general. One may like to extrapolate the iteration sequence, by replacing the union operator in equation *(it)* by a so-called *widening* operator $\nabla$. It can be defined axiomatically as an

operator which always over-approximates the union, such that there is no infinite increasing sequence in such iterations. This ensures finite time response of a static analyzer in practice.

A very simple and classical widening operator on intervals of values is the one for which $[a,b]\nabla[c,d]$ is $[e,f]$ with

$$e = \begin{cases} a & \text{if } c \geq a \\ -\infty & \text{otherwise} \end{cases} \qquad f = \begin{cases} b & \text{if } b \geq d \\ \infty & \text{otherwise} \end{cases}$$

This operator extrapolates the max bound by $\infty$ if the max bound seems to increase from one iteration to the other (respectively, the min bound by $-\infty$ if the min bound seems to decrease from one iteration to the other). In our example, applying the widening operator in place of the union after step 1 of the iteration sequence, we get $i^2_{(1)} = [1,\infty[$ which is a correct invariant, although overapproximated, since $f(i^2_{(1)}) = [1,99] \subseteq [1,\infty[$. One more iteration gets us to the least fixed point indeed.

Static analysis is interesting for computing efficiently some properties over sets of executions. Consider the toy example

```
void f(int n) {
  float x = 2;
  for (i=0 ; i<n ; i++)
   (1): x = x/(n+1) + 1;  }
```

Static analysis allows to tell in two iterations, that for all possible value of $n \in [0,\infty]$, the value of $x$ at point (1) in the loop belongs to $[1,2]$. Indeed, $x_0 = 2$; $x_1 = 2/(n+1) + 1 \cup 2 \in [1,2]$; $x_2 \in [1,2]$, the fixed point is reached.

The case of numerical computations in loops requires particular care : the classical fixed point iteration carried out without precaution will underline possibly infinite errors for most stable loops. We have had to design some special fixed point iteration strategies in order to get tighter estimations, but these are beyond the scope of this paper.

### 3.3 Other Semantics

The interpretation of the results for large programs can be facilitated by choosing different levels of error points (C lines, blocks of lines, functions, etc), and refining locally the result in the functions that have the most important errors. Grouping error points can also be used during the computation to reduce the memory and computation time of the analysis [6].

Other variations lead to "relational" analyses : an idea is to use the linear correlations between variables in order to reduce the over-estimation of errors, somehow like what is done in affine interval arithmetics [1]. Suppose there is one $\boldsymbol{\varepsilon}_i$ per node of the control flow graph of the program, a variable $x$ can be written

$$x = f^x + \sum_{\ell \in L} t^x_\ell . \gamma_\ell \boldsymbol{\varepsilon}_\ell + \omega^x_{os} \boldsymbol{\varepsilon}_{os} \ , \tag{3}$$

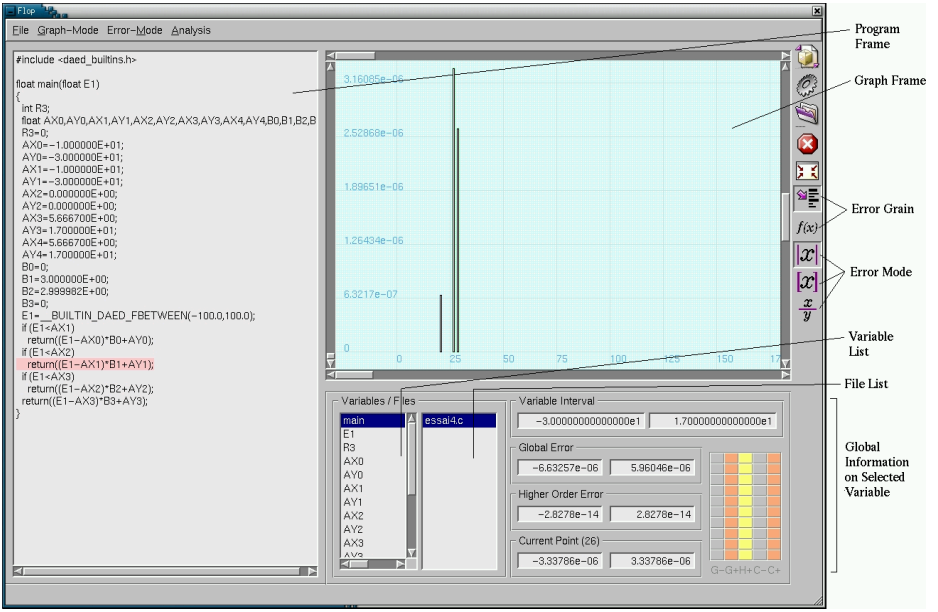



**Fig. 1.** Main window of the analyzer.

where $f^x \in \mathbb{F}$ is the computed floating-point value, $\gamma_\ell \in \mathbb{R}$ is the error committed at point $\ell$, and $t^x_\ell \in \mathbb{R}$ expresses the propagation of this error on variable $x$. When abstracting the coefficients $\gamma_\ell$ and $t^x_\ell$ by intervals, the linear correlations between variables are expressed in the $t^x_\ell$, and allow some error balancing, which was not possible with only an interval error that lost a part of these correlations. The error $\gamma_\ell$, which value is a priori unknown but can be bounded, is represented by an interval, but is seen as a formal variable that takes one particular value in this interval. And we can write for example the addition in the following way :

$$z = x +^{\ell_i} y = \uparrow_\circ (f^x + f^y) + \sum_{\ell \in L} (t^x_\ell + t^y_\ell).\gamma_\ell \varepsilon_\ell + (\omega^x_{os} + \omega^y_{os})\varepsilon_{os} + \downarrow_\circ (f^x + f^y)\varepsilon_{\ell_i} \ .$$

In this expression, the error $\gamma_{\ell_i}$ is $\downarrow_\circ (f^x + f^y)\varepsilon_{\ell_i}$, and, at point $\ell_i$, the propagation coefficient is $t^z_{\ell_i} = 1$. Other variations using correlations between values and errors, for example by means of relative error, could also be used.

## 4 The Fluctuat Tool

A prototype [4] implements this abstract interpretation, for the analysis of C programs. The multi-precision library MPFR [5] (based on GMP) is used to compute tight bounds on the errors. As shown in Fig. 1, the main window of the analyzer displays the code of the program being analyzed, the list of variables in the program, and a graph representation of the error series related to the selected

variable, at the last control point of the program. The operations are identified with their line number in the program, displayed on the X-axis. The bars indicate the maximum of the absolute values of the interval bounds. Clicking on an error bar makes the code frame emphasize the related program line and conversely.

In the example of Fig. 1, a program typical of an instrumentation software is being analyzed. It consists basically in an interpolation function with thresholds. One can see from the graph that the sources of imprecision for the return result of the function are (variable `main` selected): the floating-point approximation of constant B2 = 2.999982, the 2nd `return`, and the 3rd `return`, the last two being the more important. Using the assertion `__BUILTIN_DAED_FBETWEEN`, we imposed that `E1` is between -100 and 100. Then the control flow can go through all `return`. But in the first and last `return`, there is no imprecision committed. Thus, to improve the result, we can improve the computation of the 2nd and 3rd `return`. One way is to use `double E1` to improve the accuracy of the subtractions.

## 5 Conclusion and Future Work

We have presented some ideas about what static analysis can do for programs using floating-point numbers. A part of the work consists in modeling the results and the losses of accuracy using finite precision computations. The model used, looks like affine interval arithmetics, but is used with a very different intention : the coefficients have a meaning (floating-point value, influence of a part of the program on the global error), and are not used only to improve the precision like in affine arithmetics. Some work can still be done to improve the accuracy of this modelisation. But a consequent and difficult part is related to static analysis : efficient algorithms for fixed point computations in loops must be designed, and implementing a static analyzer for real problems is a heavy work. Our first concern is the analysis of instrumentation software, but we hope to be able to go slowly towards numerically more complex programs.

## References

1. J. L. D. Comba and J. Stolfi. Affine arithmetic and its applications to computer graphics. In SIBGRAPI'93, Recife, PE (Brazil), October 20-22, 1993.
2. P. Cousot and R. Cousot. Abstract interpretation frameworks. Journal of Logic and Symbolic Computation, 2(4):511–547, 1992.
3. E. Goubault. Static analyses of the precision of floating-point operations. In Static Analysis Symposium, SAS'01, number 2126 in LNCS, Springer-Verlag, 2001.
4. E. Goubault, M. Martel, and S. Putot. Asserting the precision of floating-point computations : a simple abstract interpreter. In ESOP'02, LNCS, Springer 2002.
5. G. Hanrot, V. Lefevre, F. Rouillier, P. Zimmermann. MPFR library. INRIA, 2001.
6. M. Martel. Propagation of roundoff errors in finite precision computations : a semantics approach. In ESOP'02, number 2305 in LNCS, Springer-Verlag, 2002.
7. M. Martel. Static Analysis of the Numerical Stability of Loops. In SAS'02, number 2477 in LNCS, Springer-Verlag, 2002.
8. R. E. Moore. Interval Analysis. Prentice-Hall, Englewood Cliffs, NJ, 1963.

# Author Index


Alefeld, Götz 191
Alt, René 250
Auer, Ekaterina 132

Beelitz, Thomas 198
Bischof, Christian 198
Borovac, Stefan 226
Braems, Isabelle 124
Bühler, Katja 160

Corliss, George F. 91

Dyllong, Eva 160

Fausten, Daniela 206

Goubault, Eric 306
Granvilliers, Laurent 274
Grimmer, Markus 64

Haßlinger, Gerhard 206
Heindl, Gerhard 226
Henni, Abderrezak 250
Hofschuster, Werner 15

Jaulin, Luc 124

Kearfott, R. Baker 36
Kecskeméthy, Andrés 132
Kieffer, Michel 107, 124
Krämer, Walter 15
Kreinovich, Vladik 274

Lang, Bruno 198
Lester, David 259
Luther, Wolfram 160

Martel, Matthieu 306
Mayer, Günter 191
Müller, Norbert 274

Neher, Markus 36

Oishi, Shin'ichi 36
Oussena, Baya 250

Petras, Knut 64
Putot, Sylvie 306

Revol, Nathalie 64
Rico, Fabien 36

Schichl, Hermann 243
Schulte Althoff, Klaus 198

Tändl, Martin 132
Traczinski, Holger 132

Walter, Eric 107, 124
Wolff von Gudenberg, Jürgen 1

Yu, Jun 91