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

LNCS 2991

Numerical Software with Result Verification

International Dagstuhl Seminar
Dagstuhl Castle, Germany, January 2003
Revised Papers

Springer

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

Springer-Verlag is a part of Springer Science+Business Media

springeronline.com

Printed in Germany

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

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

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

Lecture Notes in Computer Science

For information about Vols. 1–2856

please contact your bookseller or Springer-Verlag

Vol. 2992: E. Bertino, S. Christodoulakis, D. Plexousakis, V. Christophides, M. Koubarakis, K. Böhm, E. Ferrari (Eds.), Advances in Database Technology - EDBT 2004. XVIII, 877 pages. 2004.

Vol. 2991: R. Alt, A. Frommer, R.B. Kearfott, W. Luther (Eds.), Numerical Software with Result Verification. X, 315 pages. 2004.

Vol. 2983: S. Istrail, M. Waterman, A. Clark (Eds.), Computational Methods for SNPs and Haplotype Inference. IX, 153 pages. 2004. (Subseries LNBI).

Vol. 2982: N. Wakamiya, M. Solarski, J. Sterbenz (Eds.), Active Networks. XI, 308 pages. 2004.

Vol. 2981: C. Müller-Schloer, T. Ungerer, B. Bauer (Eds.), Organic and Pervasive Computing – ARCS 2004. XI, 339 pages. 2004.

Vol. 2978: R. Groz, R.M. Hierons (Eds.), Testing of Communicating Systems. XII, 225 pages. 2004.

Vol. 2976: M. Farach-Colton (Ed.), LATIN 2004: Theoretical Informatics. XV, 626 pages. 2004.

Vol. 2973: Y. Lee, J. Li, K.-Y. Whang, D. Lee (Eds.), Database Systems for Advanced Applications. XXIV, 925 pages. 2004.

Vol. 2970: F. Fernández Rivera, M. Bubak, A. Gómez Tato, R. Doallo (Eds.), Grid Computing. XI, 328 pages. 2004.

Vol. 2964: T. Okamoto (Ed.), Topics in Cryptology – CT-RSA 2004. XI, 387 pages. 2004.

Vol. 2962: S. Bistarelli, Semirings for Soft Constraint Solving and Programming. XII, 279 pages. 2004.

Vol. 2961: P. Eklund (Ed.), Concept Lattices. IX, 411 pages. 2004. (Subseries LNAI).

Vol. 2958: L. Rauchwerger (Ed.), Languages and Compilers for Parallel Computing. XI, 556 pages. 2004.

Vol. 2957: P. Langendoerfer, M. Liu, I. Matta, V. Tsaoussidis (Eds.), Wired/Wireless Internet Communications. XI, 307 pages. 2004.

Vol. 2954: F. Crestani, M. Dunlop, S. Mizzaro (Eds.), Mobile and Ubiquitous Information Access. X, 299 pages. 2004.

Vol. 2953: K. Konrad, Model Generation for Natural Language Interpretation and Analysis. XIII, 166 pages. 2004. (Subseries LNAI).

Vol. 2952: N. Guelfi, E. Astesiano, G. Reggio (Eds.), Scientific Engineering of Distributed Java Applications. X, 157 pages. 2004.

Vol. 2951: M. Naor (Ed.), Theory of Cryptography. XI, 523 pages. 2004.

Vol. 2949: R. De Nicola, G. Ferrari, G. Meredith (Eds.), Coordination Models and Languages. X, 323 pages. 2004.

Vol. 2947: F. Bao, R. Deng, J. Zhou (Eds.), Public Key Cryptography – PKC 2004. XI, 455 pages. 2004.

Vol. 2946: R. Focardi, R. Gorrieri (Eds.), Foundations of Security Analysis and Design II. VII, 267 pages. 2004.

Vol. 2943: J. Chen, J. Reif (Eds.), DNA Computing. X, 225 pages. 2004.

Vol. 2941: M. Wirsing, A. Knapp, S. Balsamo (Eds.), Radical Innovations of Software and Systems Engineering in the Future. X, 359 pages. 2004.

Vol. 2940: C. Lucena, A. Garcia, A. Romanovsky, J. Castro, P.S. Alencar (Eds.), Software Engineering for Multi-Agent Systems II. XII, 279 pages. 2004.

Vol. 2939: T. Kalker, I. Cox, Y.M. Ro (Eds.), Digital Watermarking. XII, 602 pages. 2004.

Vol. 2937: B. Steffen, G. Levi (Eds.), Verification, Model Checking, and Abstract Interpretation. XI, 325 pages. 2004.

Vol. 2934: G. Lindemann, D. Moldt, M. Paolucci (Eds.), Regulated Agent-Based Social Systems. X, 301 pages. 2004. (Subseries LNAI).

Vol. 2930: F. Winkler (Ed.), Automated Deduction in Geometry. VII, 231 pages. 2004. (Subseries LNAI).

Vol. 2926: L. van Elst, V. Dignum, A. Abecker, Agent-Mediated Knowledge Management. XI, 428 pages. 2004. (Subseries LNAI).

Vol. 2923: V. Lifschitz, I. Niemelä (Eds.), Logic Programming and Nonmonotonic Reasoning. IX, 365 pages. 2004. (Subseries LNAI).

Vol. 2919: E. Giunchiglia, A. Tacchella (Eds.), Theory and Applications of Satisfiability Testing. XI, 530 pages. 2004.

Vol. 2917: E. Quintarelli, Model-Checking Based Data Retrieval. XVI, 134 pages. 2004.

Vol. 2916: C. Palamidessi (Ed.), Logic Programming. XII, 520 pages. 2003.

Vol. 2915: A. Camurri, G. Volpe (Eds.), Gesture-Based Communication in Human-Computer Interaction. XIII, 558 pages. 2004. (Subseries LNAI).

Vol. 2914: P.K. Pandya, J. Radhakrishnan (Eds.), FST TCS 2003: Foundations of Software Technology and Theoretical Computer Science. XIII, 446 pages. 2003.

Vol. 2913: T.M. Pinkston, V.K. Prasanna (Eds.), High Performance Computing - HiPC 2003. XX, 512 pages. 2003. (Subseries LNAI).

Vol. 2911: T.M.T. Sembok, H.B. Zaman, H. Chen, S.R. Urs, S.H. Myaeng (Eds.), Digital Libraries: Technology and Management of Indigenous Knowledge for Global Access. XX, 703 pages. 2003.

Vol. 2910: M.E. Orlowska, S. Weerawarana, M.M.P. Papazoglou, J. Yang (Eds.), Service-Oriented Computing - ICSOC 2003. XIV, 576 pages. 2003.

Vol. 2909: R. Solis-Oba, K. Jansen (Eds.), Approximation and Online Algorithms. VIII, 269 pages. 2004.

Vol. 2909: K. Jansen, R. Solis-Oba (Eds.), Approximation and Online Algorithms. VIII, 269 pages. 2004.

Vol. 2908: K. Chae, M. Yung (Eds.), Information Security Applications. XII, 506 pages. 2004.

Vol. 2907: I. Lirkov, S. Margenov, J. Wasniewski, P. Yalamov (Eds.), Large-Scale Scientific Computing. XI, 490 pages. 2004.

Vol. 2906: T. Ibaraki, N. Katoh, H. Ono (Eds.), Algorithms and Computation. XVII, 748 pages. 2003.

Vol. 2905: A. Sanfeliu, J. Ruiz-Shulcloper (Eds.), Progress in Pattern Recognition, Speech and Image Analysis. XVII, 693 pages. 2003.

Vol. 2904: T. Johansson, S. Maitra (Eds.), Progress in Cryptology - INDOCRYPT 2003. XI, 431 pages. 2003.

Vol. 2903: T.D. Gedeon, L.C.C. Fung (Eds.), AI 2003: Advances in Artificial Intelligence. XVI, 1075 pages. 2003. (Subseries LNAI).

Vol. 2902: F.M. Pires, S.P. Abreu (Eds.), Progress in Artificial Intelligence. XV, 504 pages. 2003. (Subseries LNAI).

Vol. 2901: F. Bry, N. Henze, J. Ma luszyński (Eds.), Principles and Practice of Semantic Web Reasoning. X, 209 pages. 2003.

Vol. 2900: M. Bidoit, P.D. Mosses (Eds.), Casl User Manual. XIII, 240 pages. 2004.

Vol. 2899: G. Ventre, R. Canonico (Eds.), Interactive Multimedia on Next Generation Networks. XIV, 420 pages. 2003.

Vol. 2898: K.G. Paterson (Ed.), Cryptography and Coding. IX, 385 pages. 2003.

Vol. 2897: O. Balet, G. Subsol, P. Torguet (Eds.), Virtual Storytelling. XI, 240 pages. 2003.

Vol. 2896: V.A. Saraswat (Ed.), Advances in Computing Science – ASIAN 2003. VIII, 305 pages. 2003.

Vol. 2895: A. Ohori (Ed.), Programming Languages and Systems. XIII, 427 pages. 2003.

Vol. 2894: C.S. Laih (Ed.), Advances in Cryptology - ASIACRYPT 2003. XIII, 543 pages. 2003.

Vol. 2893: J.-B. Stefani, I. Demeure, D. Hagimont (Eds.), Distributed Applications and Interoperable Systems. XIII, 311 pages. 2003.

Vol. 2892: F. Dau, The Logic System of Concept Graphs with Negation. XI, 213 pages. 2003. (Subseries LNAI).

Vol. 2891: J. Lee, M. Barley (Eds.), Intelligent Agents and Multi-Agent Systems. X, 215 pages. 2003. (Subseries LNAI).

Vol. 2890: M. Broy, A.V. Zamulin (Eds.), Perspectives of System Informatics. XV, 572 pages. 2003.

Vol. 2889: R. Meersman, Z. Tari (Eds.), On The Move to Meaningful Internet Systems 2003: OTM 2003 Workshops. XIX, 1071 pages. 2003.

Vol. 2888: R. Meersman, Z. Tari, D.C. Schmidt (Eds.), On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE. XXI, 1546 pages. 2003.

Vol. 2887: T. Johansson (Ed.), Fast Software Encryption. IX, 397 pages. 2003.

Vol. 2886: I. Nyström, G. Sanniti di Baja, S. Svensson (Eds.), Discrete Geometry for Computer Imagery. XII, 556 pages. 2003.

Vol. 2885: J.S. Dong, J. Woodcock (Eds.), Formal Methods and Software Engineering. XI, 683 pages. 2003.

Vol. 2884: E. Najm, U. Nestmann, P. Stevens (Eds.), Formal Methods for Open Object-Based Distributed Systems. X, 293 pages. 2003.

Vol. 2883: J. Schaeffer, M. Müller, Y. Björnsson (Eds.), Computers and Games. XI, 431 pages. 2003.

Vol. 2882: D. Veit, Matchmaking in Electronic Markets. XV, 180 pages. 2003. (Subseries LNAI).

Vol. 2881: E. Horlait, T. Magedanz, R.H. Glitho (Eds.), Mobile Agents for Telecommunication Applications. IX, 297 pages. 2003.

Vol. 2880: H.L. Bodlaender (Ed.), Graph-Theoretic Concepts in Computer Science. XI, 386 pages. 2003.

Vol. 2879: R.E. Ellis, T.M. Peters (Eds.), Medical Image Computing and Computer-Assisted Intervention - MICCAI 2003. XXXIV, 1003 pages. 2003.

Vol. 2878: R.E. Ellis, T.M. Peters (Eds.), Medical Image Computing and Computer-Assisted Intervention - MICCAI 2003. XXXIII, 819 pages. 2003.

Vol. 2877: T. Böhme, G. Heyer, H. Unger (Eds.), Innovative Internet Community Systems. VIII, 263 pages. 2003.

Vol. 2876: M. Schroeder, G. Wagner (Eds.), Rules and Rule Markup Languages for the Semantic Web. VII, 173 pages. 2003.

Vol. 2875: E. Aarts, R. Collier, E.v. Loenen, B.d. Ruyter (Eds.), Ambient Intelligence. XI, 432 pages. 2003.

Vol. 2874: C. Priami (Ed.), Global Computing. XIX, 255 pages. 2003.

Vol. 2871: N. Zhong, Z.W. Raś, S. Tsumoto, E. Suzuki (Eds.), Foundations of Intelligent Systems. XV, 697 pages. 2003. (Subseries LNAI).

Vol. 2870: D. Fensel, K.P. Sycara, J. Mylopoulos (Eds.), The Semantic Web - ISWC 2003. XV, 931 pages. 2003.

Vol. 2869: A. Yazici, C. Şener (Eds.), Computer and Information Sciences - ISCIS 2003. XIX, 1110 pages. 2003.

Vol. 2868: P. Perner, R. Brause, H.-G. Holzhütter (Eds.), Medical Data Analysis. VIII, 127 pages. 2003.

Vol. 2866: J. Akiyama, M. Kano (Eds.), Discrete and Computational Geometry. VIII, 285 pages. 2003.

Vol. 2865: S. Pierre, M. Barbeau, E. Kranakis (Eds.), Ad-Hoc, Mobile, and Wireless Networks. X, 293 pages. 2003.

Vol. 2864: A.K. Dey, A. Schmidt, J.F. McCarthy (Eds.), UbiComp 2003: Ubiquitous Computing. XVII, 368 pages. 2003.

Vol. 2863: P. Stevens, J. Whittle, G. Booch (Eds.), "UML" 2003 - The Unified Modeling Language. XIV, 415 pages. 2003.

Vol. 2860: D. Geist, E. Tronci (Eds.), Correct Hardware Design and Verification Methods. XII, 426 pages. 2003.

Vol. 2859: B. Apolloni, M. Marinaro, R. Tagliaferri (Eds.), Neural Nets. X, 376 pages. 2003.

Vol. 2857: M.A. Nascimento, E.S. de Moura, A.L. Oliveira (Eds.), String Processing and Information Retrieval. XI, 379 pages. 2003.

Table of Contents

Languages

Software Systems and Tools

New Verification Techniques Based on Interval Arithmetic

Applications in Science and Engineering

Novel Approaches to Verification

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. ∇ 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 $\nabla; \underline{z} = \nabla(\underline{x} + \underline{y}); \overline{z} = \nabla(-\nabla(-\overline{x} - \overline{y}))$; 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`$\leftrightarrow$ `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;
  }
  // ...
```