ERINGMATHEMATICALENGINEERINGMATHEMATI

Andreas Rauh · Ekaterina Auer
(Eds.)

Modeling, Design, and Simulation of Systems with Uncertainties

Springer

Modeling, Design, and Simulation of Systems with Uncertainties

Mathematical Engineering

For further volumes:
http://www.springer.com/series/8445

Andreas Rauh • Ekaterina Auer
Editors

Modeling, Design, and Simulation of Systems with Uncertainties

Springer

Editors
Andreas Rauh
University of Rostock
Chair of Mechatronics
Justus-von-Liebig-Weg 6
18059 Rostock
Germany
andreas.rauh@uni-rostock.de

Ekaterina Auer
University of Duisburg-Essen
Faculty of Engineering, INKO
Lotharstr. 63
47057 Duisburg
Germany
auer@inf.uni-due.de

ISBN 978-3-642-15955-8 e-ISBN 978-3-642-15956-5
DOI 10.1007/978-3-642-15956-5
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2011929791

Cover design: WMXDesign GmbH, Heidelberg

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

To describe the true behavior of most real-world systems with sufficient accuracy, engineers have to overcome difficulties arising from their lack of knowledge about parts of a process or from the impossibility to characterize it with absolute certainty. For example, measured parameters of (dynamical) systems cannot be determined exactly due to non-negligible equipment imprecision. Other sources of such model inaccuracies are order reduction techniques for complex systems used to simplify the design of their components and corresponding control algorithms. Therefore, both aleatory (due to randomness) and epistemological (due to the lack of knowledge) types of uncertainty have to be taken into account while developing techniques for a model-based analysis or synthesis of systems.

Depending on the application at hand, uncertainties in modeling and measurements can be represented in several different ways. For example, *bounded uncertainties* can be described by intervals, affine forms or general polynomial enclosures such as Taylor models. There are frameworks incorporating corresponding kinds of arithmetics to handle this type of uncertainty, which simultaneously provide verified results. This means that the results are enclosures guaranteed to contain the exact solution sets, assuming that the mathematical models and the corresponding ranges of uncertain quantities are correct.

Another situation arises if the uncertainty can be characterized in the form of probability distributions described, for example, by mean values, standard deviations and higher-order moments (*stochastic uncertainty*). In this case, Bayesian estimation frameworks offer a solution by propagating the corresponding probability density functions. These are handled in terms of either analytic or numeric representations, where the latter approach forms the basis of the well-known Monte Carlo methods.

For both bounded and stochastic uncertainties, there exist specific theoretic concepts and practical applications. The goal of this Special Volume on *Modeling, Design, and Simulation of Systems with Uncertainties* is to make the current research on techniques for uncertainty handling known to a broader circle of researchers and industry representatives. For this purpose, we have collected 16 articles from researchers from Canada, Russia, Germany, USA, France, Austria, Poland, Italy, and

Bulgaria dealing with this topic, from which five were presented at the Minisymposium on *Modeling, Design, and Simulation of Systems with Uncertainties* during the *16th European Conference on Mathematics for Industry ECMI* in Wuppertal, Germany, in July 2010.

The volume is subdivided into two parts. In the first we present works highlighting the theoretic background and current research on algorithmic approaches in the field of uncertainty handling, together with their reliable software implementation. The second part is concerned with real-life application scenarios from various areas including but not limited to mechatronics, robotics, and biomedical engineering.

Rostock,
Duisburg,
March 2011

Andreas Rauh
Ekaterina Auer

Acknowledgements

The editors of this Special Volume on *Modeling, Design, and Simulation of Systems with Uncertainties* would like to thank all authors for their high-quality contributions. Moreover, we would like to thank all researchers who supported us by reviewing the individual book chapters and by providing detailed comments and suggestions for improvements. Finally, we would like to thank Prof. Dr. Claus Hillermeier for giving us the opportunity to publish this collection of current research articles in the Springer book series on *Mathematical Engineering*.

Contents

List of Contributors

Haider Albassam
University of Duisburg-Essen, D-47048 Duisburg, Germany
e-mail: haider.albassam@uni-due.de

Matthias Althoff
Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA, USA
e-mail: malthoff@ece.cmu.edu

Felix Antritter
Automatisierungs- und Regelungstechnik, Universität der Bundeswehr München, Werner-Heisenberg-Weg 39, D-85579 Neubiberg, Germany
e-mail: felix.antritter@unibw.de

Harald Aschemann
Chair of Mechatronics, University of Rostock, D-18059 Rostock, Germany
e-mail: harald.aschemann@uni-rostock.de

Ekaterina Auer
University of Duisburg-Essen, D-47048 Duisburg, Germany
e-mail: auer@inf.uni-due.de

Neli S. Dimitrova
Institute of Mathematics and Informatics, Bulgarian Academy of Sciences, Acad. G. Bonchev Str. 8, 1113 Sofia, Bulgaria
e-mail: nelid@bio.bas.bg

Serena Doria
Department of Sciences, University G. D'Annunzio, Chieti-Pescara, Italy
e-mail: s.doria@dst.unich.it

Denis Efimov
IMS-lab, University of Bordeaux, 351 cours de la libération, 33405 Talence, France
e-mail: denis.efimov@ims-bordeaux.fr

Darya Filatova
UJK, ul. Krakowska 11, 25-027 Kielce, Poland and Analytical Centre of Russian Academy of Sciences, ul. Vavilova 40, 199911 Moscow, Russia
e-mail: daria_filatova@rambler.ru

Marek Grzywaczewski
Politechnika Radomska, ul. Malczewskiego 20A, 26-600 Radom, Poland
e-mail: mgrzyw@interia.pl

Luc Jaulin
ENSIETA, OSM, Lab-STICC, 2 rue François Verny, 29806 Brest, France
e-mail: jaulinlu@ensieta.fr

Andrés Kecskeméthy
University of Duisburg-Essen, D-47048 Duisburg, Germany
e-mail: andres.kecskemethy@uni-due.de

Michel Kieffer
Laboratoire des Signaux et Systèmes - CNRS - SUPELEC - Univ Paris-Sud, 3 rue Joliot-Curie, F-91192 Gif-sur-Yvette cedex, on leave at LTCI - CNRS - Telecom ParisTech, 46 rue Barault, F-75013 Paris, France
e-mail: michel.kieffer@lss.supelec.fr

Marco Kletting
Multi-Function Airborne Radars (OPES22), Cassidian Electronics, Woerthstr. 85, D-89077 Ulm, Germany
e-mail: marco.kletting@cassidian.com

Georgy V. Kostin
Laboratory of Mechanics of Controlled Systems, Institute for Problems in Mechanics of the Russian Academy of Sciences, Pr. Vernadskogo 101-1, 119526 Moscow, Russia
e-mail: kostin@ipmnet.ru

Mikhail I. Krastanov
Institute of Mathematics and Informatics, Bulg Academy of Sciences
Acad. G. Bonchev Str. 8, 1113 Sofia, Bulgaria
e-mail: krast@math.bas.bg

Bruce H. Krogh
Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA, USA
e-mail: krogh@ece.cmu.edu

Wolfram Luther
University of Duisburg-Essen, D-47048 Duisburg, Germany
e-mail: luther@inf.uni-due.de

Mihály Csaba Markót
Fakultät für Mathematik, Universität Wien, Nordbergstr. 15, A-1090 Wien, Austria
e-mail: Mihaly.Markot@univie.ac.at

Mehrdad Moshir
Jet Propulsion Laboratory, California Institute of Technology, 4800 Oak Grove Dr., Pasadena, CA 91109, USA
e-mail: mehrdad.moshir@jpl.nasa.gov

Nedialko S. Nedialkov
Department of Computing and Software, McMaster University, Hamilton, Ontario, Canada, L8S 4K1
e-mail: nedialk@mcmaster.ca

Tarek Raïssi
IMS-lab, University of Bordeaux, 351 cours de la libération, 33405 Talence, France
e-mail: tarek.raissi@ims-bordeaux.fr

Andreas Rauh
Chair of Mechatronics, University of Rostock, D-18059 Rostock, Germany
e-mail: andreas.rauh@uni-rostock.de

Jöran Ritzke
Chair of Mechatronics, University of Rostock, D-18059 Rostock, Germany
e-mail: Joeran.Ritzke@uni-rostock.de

Vasily V. Saurin
Laboratory of Mechanics and Optimization of Structures, Institute for Problems in Mechanics of the Russian Academy of Sciences, Pr. Vernadskogo 101-1, 119526 Moscow, Russia
e-mail: saurin@ipmnet.ru

Hermann Schichl
Fakultät für Mathematik, Universität Wien, Nordbergstr. 15, A-1090 Wien, Austria
e-mail: Hermann.Schichl@univie.ac.at

Dominik Schindele
Chair of Mechatronics, University of Rostock, D-18059 Rostock, Germany
e-mail: Dominik.Schindele@uni-rostock.de

Sergey P. Shary
Institute of Computational Technologies SB RAS, 6 Lavrentiev ave., 630090 Novosibirsk, Russia
e-mail: shary@ict.nsc.ru

Olaf Stursberg
University of Kassel, Control and System Theory (FB16), Wilhelmshöher Allee 73, D-34121 Kassel, Germany
e-mail: stursberg@uni-kassel.de

Eric Walter
Laboratoire des Signaux et Systèmes - CNRS - SUPELEC - Univ Paris-Sud, 3 rue Joliot-Curie, F-91192 Gif-sur-Yvette cedex, France
e-mail: eric.walter@lss.supelec.fr

Ali Zolghadri
IMS-lab, University of Bordeaux, 351 cours de la libération, 33405 Talence, France
e-mail: ali.zolghadri@ims-bordeaux.fr

Part I
Theoretic Background and Software Implementation

In the first part of this book, we present works highlighting the theoretic background and current research on algorithmic approaches in the field of uncertainty handling together with their reliable software implementation. In Chapter 1, Nedialko S. Nedialkov presents techniques from literate programming which are used in the implementation of the verified ODE solver VNODE-LP. Chapter 2 authored by Sergey P. Shary is concerned with new methods for solving linear systems of equations with interval uncertainties. Andreas Rauh and Harald Aschemann describe techniques for the structural analysis of control and state estimation problems formulated as systems of differential-algebraic equations in Chapter 3. In Chapter 4, Matthias Althoff, Bruce H. Krogh, and Olaf Stursberg consider methods for reachability analysis of linear dynamic processes applicable to high-dimensional system models. A robustness analysis of different tracking control schemes in performed by Marco Kletting and Felix Antritter in Chapter 5. Approaches for set-membership state estimation are presented by Luc Jaulin in Chapter 6, whereas verified global optimization routines for parameter estimation of nonlinear models are discussed by Michel Kieffer, Mihály Csaba Markót, Hermann Schichl, and Eric Walter in Chapter 7. Chapter 8 by Darya Filatova and Marek Grzywaczewski deals with the theory applicable to the design of optimal control strategies for induction heating processes and a robustness evaluation of the obtained results. The first part of this volume is concluded by a contribution on coherent upper and lower conditional previsions authored by Serena Doria.

Chapter 1
Implementing a Rigorous ODE Solver Through Literate Programming

Nedialko S. Nedialkov

Abstract Interval numerical methods produce results that can have the power of a mathematical proof. Although there is a substantial amount of theoretical work on these methods, little has been done to ensure that an implementation of an interval method can be readily verified. However, when claiming rigorous numerical results, it is crucial to ensure that there are no errors in their computation. Furthermore, when such a method is used in a computer assisted proof, it would be desirable to have its implementation published in a form that is convenient for verification by human experts.

We have applied Literate Programming (LP) to produce VNODE-LP, a C++ solver for computing rigorous bounds on the solution of an initial-value problem (IVP) for an ordinary differential equation (ODE). We have found LP well suited for ensuring that an implementation of a numerical algorithm is a correct translation of its underlying theory into a programming language: we can split the theory into small pieces, translate each of them, and keep mathematical expressions and the corresponding code close together in a unified document. Then it can be reviewed and checked for correctness by human experts, similarly to how a scientific work is examined in a peer-review process.

1.1 Introduction

Interval numerical methods produce results that can have the power of a mathematical proof. Typically, such a method computes bounds that are guaranteed to contain the true solution of a problem, proves that a solution does not exists or it indicates

Nedialko S. Nedialkov
Department of Computing and Software, McMaster University, Hamilton, Ontario, Canada, L8S 4K1
e-mail: nedialk@mcmaster.ca

A. Rauh and E. Auer, *Modeling, Design, and Simulation of Systems with Uncertainties,* Mathematical Engineering 3, DOI 10.1007/978-3-642-15956-5_1,

that a solution cannot be found. For example, when computes an enclosure on the solution of an IVP in ODEs, an interval solver first proves that there exists a unique solution and then produces bounds that contain it [10]; when solving a nonlinear equation, an interval method can prove that a region does not contain a solution or computes bounds that contain a unique solution to the problem [30]. For an excellent, up-to-date survey of these methods, see [35].

To date, not much has been done to ensure that the implementation of such a method can be readily verified, and the bounds it computes are indeed rigorous. Showing that an implementation is correct is of paramount importance for these methods, as mathematical rigor cannot be claimed, if we miss to include even a single roundoff error in a computation. Furthermore, when interval software is used in a computer-assisted proof, it would be desirable to have the software published in a form that is convenient for inspection and verification by human experts.

The author released in 2001 VNODE [25, 28], Validated Numerical ODE, a C++ package for computing bounds on the solution of an IVP for an ODE. This package is carefully written and tested, and it had shown to be robust and reliable. While one can check the theory behind VNODE (e.g. in [25]), it would be difficult to show that its C++ translation does not contain errors. The same applies to the other packages for computing bounds in IVPs for ODEs: ADIODES [39], COSY [3], and VSPODE [20]. That is, it also would be difficult to establish the correspondence between underlying theory and source code in these packages. A notable exception is AWA [22], where there is a clear "match" between the theory and the program listing in [22]. Another well-documented implementation is the VODESIA package [5], but unfortunately it is not publicly available.

The above solvers have been used to compute rigorous bounds on solutions in IVP ODEs. For example, VNODE had been employed in applications such as rigorous multibody simulations [2], reliable surface intersection [24, 32], robust evaluation of differential geometry properties of a Bezier surface patch [18], computing bounds on eigenvalues [4], parameter and state estimation [12, 34], rigorous shadowing [7, 8], and theoretical computer science [1].

The author had always been concerned about possible errors in the implementation of VNODE. Obviously, if an error is present, then the works that have employed VNODE may contain invalid results. Moreover, how can one establish that the computed bounds are rigorous, and further, how others can be convinced that the implementation and the results are correct? This came as a major concern of the author of [1]: how one can trust the numerical results of VNODE? He needed a rigorous proof that an algebraic expression involving the solution of a highly nonlinear scalar ODE is less than one; otherwise his theorem would not hold. The strongest assurance argument was of the sort "VNODE has been accurate and reliable", but obviously this is not satisfactory. The value of this expression was approximately 0.999... in multiple precision in MAPLE, but it needed to be proved that it was always smaller than 1. With VNODE we showed that the exact value of this expression is always smaller than one, but still, we did not have an unquestionable proof.

This prompted the author to search for ways to show that not only the implementation is correct, but it can also be checked readily by others. Literate Programming

(LP) [16] was found particularly suitable for this purpose. Using LP, we can produce a *verifiable* implementation in the sense that it can be reviewed and examined for correctness, similarly to how a scientific work is reviewed by human experts in a peer-review process. This is in contrast to mechanical software verification, when a proof tool is applied to verify code against given specifications.

We reimplemented VNODE entirely with LP (along with some algorithmic improvements), which resulted in the VNODE-LP solver [27]. This paper gives an overview of VNODE-LP, elaborates on LP, and illustrates the process of employing it for carrying out a verifiable implementation.

Section 1.2 discusses LP. Section 1.3 presents an overview of VNODE-LP. Examples from its implementation, illustrating our approach using LP, are given in Section 1.4. Section 1.5 elaborates on relevant work. Section 1.6 summarizes our experience.

1.2 Literate Programming and VNODE-LP

Literate programming was introduced as a programming methodology by D. Knuth [14, 15]. Its essence can be captured as in [16, pg. 99]: "...instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do", and introducing concepts "...in an order that is best for human understanding, using a mixture of formal and informal methods that reinforce each other."

When developing a literate program, we break down an algorithm into smaller, easy-to-understand parts, and explain, document, and implement each of them in an order that is more natural for human comprehension, versus order that is suitable for compilation. In a literate program, documentation and code are in one source. The program is an interconnected "web" of pieces of code, referred to as *sections* [14,16] or *chunks* [11,37], which can be presented in any sequence. They are assembled into a program ready for compilation in a *tangle* process, which extracts the source code from the LP source. The documentation is "weaved" in a *weave* process, which prepares it for typesetting [16, 17].

We developed VNODE-LP using the CWEB literate programming tool [17] and its `ctangle` and `cweave` utilities. CWEB enables the inclusion of documentation and C++ code in a CWEB source file, which is essentially a LaTeX file with additional statements for dealing with source code.

From a CWEB file, `cweave` generates a LaTeX file; `cweave` takes care of page layout, indentation, suitable fonts, pretty printing of C/C++ code, and generates extensive cross-index information. Originally, CWEB could deal with TeX input only. The LaTeX `cweb` [36] class allows using LaTeX; the `cweb-hy` class [37], an extension of `cweb`, allows structuring of a LaTeX document in chapters, sections, subsections, etc., and also provides automatic generation of hyperlinks, which are convenient for navigation through the code in the resulting, e.g., PDF file.

The ctangle utility extracts the source code and writes C/C++ files. It also includes line information in the generated files so that handling errors when compiling and debugging can be done in terms of CWEB source files, and not the generated C/C++ files. That is, when syntax errors or warnings are encountered, a compiler gives line numbers in web files, and similarly, when runtime errors are detected, a debugger gives line numbers in web files.

Developing a literate program reduces to writing an article or a book: we present the program in an order that follows our thought process and strive to explain our ideas clearly in a document that should be of publishable quality. For each algorithm in [27], we present its theory first, and then translate parts of it, where the division is such that the code in each part is not difficult to inspect. During development, if errors in compilation or execution occur, we can review the manuscript and update accordingly the CWEB files, without looking into the generated program files (they are for compiler consumption). Similarly, when inspecting VNODE-LP, we can work only with the LP document [27].

This article and [27] are created with CWEB and the cweb-hy class. The latter is composed like a book: with a table of contents, list of figures, hierarchical structure of the presentation, index, and bibliography. This document contains everything related to VNODE-LP: user guide, theory, documentation, source code, example, test cases, makefiles, and gnuplot files used for generating the plots in [27]

All the theory of VNODE-LP is included in [27]. Our goal was to have a self-contained, detailed presentation, so a reviewer would need only [27] when evaluating VNODE-LP. Since all the pieces for verifying the theory and implementation are in [27], if their correctness is confirmed by human experts like in a peer-review process, we may trust, or at least have high-confidence, in the correctness of the implementation of VNODE-LP and accept the bounds it computes as rigorous. When claiming rigor, however, we presume that the operating system, compiler, and the packages VNODE-LP uses do not contain errors affecting its execution.

1.3 Overview of VNODE-LP

We introduce interval arithmetic (IA), state the IVP that is the subject of this work (§1.3.1), and discuss briefly the methods in VNODE-LP and the packages it uses (§1.3.2).

1.3.1 The IVP VNODE-LP *Solves*

The VNODE-LP software builds on IA as defined below. Denote the set of closed (finite, nonempty) intervals on $\mathbb{R}$ by

$$\mathbb{IR} = \left\{ \boldsymbol{a} = [\underline{a}, \overline{a}] \mid \underline{a} \le x \le \overline{a},\ \underline{a}, \overline{a} \in \mathbb{R} \right\}.$$

If $\boldsymbol{a}$ and $\boldsymbol{b} \in \mathbb{IR}$ and $\bullet \in \{+,-,\times,/\}$, then the IA operations are defined as

$$\boldsymbol{a} \bullet \boldsymbol{b} = \big\{x \bullet y \mid x \in \boldsymbol{a},\ y \in \boldsymbol{b}\big\},$$

where division is undefined if $0 \in \boldsymbol{b}$.

Now consider the IVP

$$y'(t) = f(t,y), \quad y(t_0) = y_0, \qquad t \in \mathbb{R},\ y \in \mathbb{R}^n. \tag{1.1}$$

where $f : \mathbb{R} \times \mathbb{R}^n$ is sufficiently smooth. As a consequence, the code list of f should not contain for example branches, abs, or min. For more details see [10, 25–29].

Denote the set of n-dimensional interval vectors by $\mathbb{IR}^n$. Given $\boldsymbol{y}_0 \in \mathbb{IR}^n$ and $t_{\text{end}} \neq t_0$ $(t_{\text{end}} \in \mathbb{R})$, VNODE-LP tries to compute a $\boldsymbol{y}_{\text{end}} \in \mathbb{IR}^n$ at t_{end} that contains the solution to (1.1) at t_{end} for all $y_0 \in \boldsymbol{y}_0$. If VNODE-LP cannot reach t_{end}, for example the bounds on the solution become too wide, bounds at some t^* between t_0 and t_{end} are returned.

1.3.2 *Methods and Packages*

Denote by $y(t_j;t_0,y_0)$ the solution to (1.1) with an initial condition y_0 at t_0, and denote by $\boldsymbol{y}_j$ an enclosure of the solution at t_j. That is,

$$y(t_j;t_0,y_0) \in \boldsymbol{y}_j \quad \text{for all } y_0 \in \boldsymbol{y}_0.$$

This solver proceeds in a one-step manner from t_0 to t_{end}, where it computes bounds at (adaptively) selected points $t_j \in (t_0,t_{\text{end}}]$. On a step from t_j to t_{j+1}, VNODE-LP computes first a priori bounds $\widetilde{\boldsymbol{y}}_j$ such that

$$y(t;t_j,y_j) \in \widetilde{\boldsymbol{y}}_j \quad \text{for all } t \in [t_j,t_{j+1}] \quad \text{and all } y_j \in \boldsymbol{y}_j.$$

Then it finds tight bounds $\boldsymbol{y}_{j+1}$ at t_{j+1} such that

$$y(t_{j+1};t_0,y_0) \in \boldsymbol{y}_{j+1} \quad \text{for all } y_0 \in \boldsymbol{y}_0;$$

see Figure 1.1. To compute these bounds, we use IA, Taylor series expansion of the solution to (1.1) at each integration point, and various interval techniques.

VNODE-LP is based on Taylor series and the Hermite-Obreschkoff [25] methods. It is a fixed-order, variable-stepsize solver. The stepsize is varied such that an estimate of the *local excess* per unit step is below a user-specified tolerance. Typical values for the order (for efficient integration) can be between 20 and 30 [26]; the default order is set to 20.

Generally, VNODE-LP is suitable for computing bounds on the solution of an IVP ODE with point initial conditions or interval initial conditions with a sufficiently small width. If the initial condition set is not small enough and/or long time integration is desired, the COSY package [3] of Berz and Makino can produce tighter

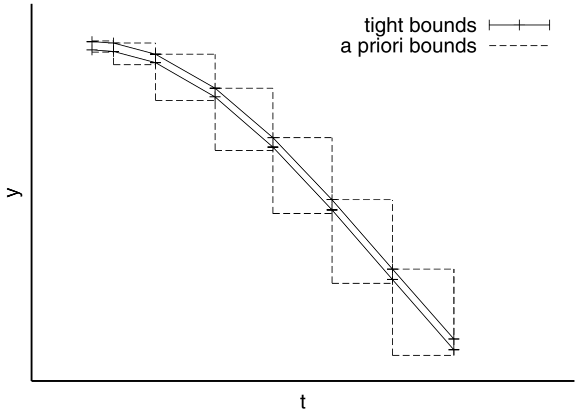

Fig. 1.1: A priori and tight bounds. For this visualization, the tight bounds are connected with lines, which do not necessarily enclose the true solution

bounds than VNODE-LP. Alternatively, one can subdivide the initial interval vector (box) $\boldsymbol{y}_0$ into smaller boxes, perform integrations with them as initial conditions, and build an enclosure of the solution at the desired t_{end}.

We tried to avoid advanced C++ constructs and tried to minimize the dependence of VNODE-LP on the IA package. The present distribution of VNODE-LP compiles with either of the IA packages PROFIL/BIAS [13] or FILIB++ [19]. Recently, the IA package GAOL [6] was used as the IA package in VNODE-LP [9].

The interface to an IA package is encapsulated in 26 small (most of them single line), inline wrapper functions that call functions from it. We aimed at keeping this interface as small as possible, such that another IA package can be incorporated easily by implementing these wrapper functions. For this reason, we do not use, for example, the matrix and vector classes of PROFIL/BIAS, but implement our own matrix and vector operations through the C++ standard template library.

A major component of our solver is the tool for generating Taylor coefficients and Jacobians of Taylor coefficients through automatic differentiation (AD). This is done using the FADBAD++ [40] AD package. We also use LAPACK and BLAS for computing an approximate matrix inverse, which is needed for enclosing the inverse of an interval matrix.

1.4 Examples from VNODE-LP

We illustrate typical steps when developing VNODE-LP: we give examples of two simple functions (§1.4.1) and an example of translating an expression that is part of a function (§1.4.2). We also present a simple program for integrating the Lorenz system (§1.4.3).

1.4.1 Computing h such that $[0,h]\,\mathbf{a} \subseteq \mathbf{b}$

The following problem is from the VNODE-LP implementation: given finite machine intervals $\boldsymbol{a}$ and $\boldsymbol{b}$, where $0 \in \boldsymbol{b}$, find the largest $h \geq 0$ such that $[0,h]\,\boldsymbol{a} \subseteq \boldsymbol{b}$. Here, for $\boldsymbol{x},\boldsymbol{y} \in \mathbb{IR}$, $\boldsymbol{x} \subseteq \boldsymbol{y}$ iff $\underline{x} \geq \underline{y}$ and $\overline{x} \leq \overline{y}$.

We derive a formula for h and then produce the C++ code. By $\bigtriangledown(x/y)$, we denote the rounded towards $-\infty$ result of x/y.

1. If $\underline{a} = \overline{a} = 0$, then $[0,h]\,\boldsymbol{a} = [0,0] \subseteq \boldsymbol{b}$ for any h, and we set $h = \textbf{numeric\_limits}\langle\textbf{double}\rangle::max()$, the largest double precision number. Below we assume $\boldsymbol{a} \neq [0,0]$.
2. If $\underline{a} \geq 0$, then $\overline{a} > 0$ and $[0,h]\,\boldsymbol{a} = [0,h\overline{a}] \subseteq [\underline{b},\overline{b}]$ when $h \leq \overline{b}/\overline{a}$. We set $h = \bigtriangledown(\overline{b}/\overline{a})$.
3. If $\overline{a} \leq 0$, then $\underline{a} < 0$ and $[0,h]\,\boldsymbol{a} = [h\underline{a},0] \subseteq [\underline{b},\overline{b}]$ when $h \leq \underline{b}/\underline{a}$. We set $h = \bigtriangledown(\underline{b}/\underline{a})$.
4. If $\underline{a} < 0 < \overline{a}$, then $[0,h]\,\boldsymbol{a} = [h\underline{a},h\overline{a}] \subseteq [\underline{b},\overline{b}]$ when $h = \min\{\underline{b}/\underline{a},\overline{b}/\overline{a}\}$. We set $h = \min\big\{\bigtriangledown(\underline{b}/\underline{a}),\ \bigtriangledown(\overline{b}/\overline{a})\big\}$.

We translate the above cases into:

1 $\langle h$ such that $[0,h]\boldsymbol{a} \subseteq \boldsymbol{b}$ (intervals) 1$\rangle \equiv$

```
#include <climits>
  using namespace std;
  using namespace v_bias;
  inline double compH(const interval &a, const interval &b)
  {      /* inf(a) returns a̲; sup(a) returns a̅ */
    if (inf(a) ≡ 0 ∧ sup(a) ≡ 0) return numeric_limits⟨double⟩::max();
    round_down();      /* set rounding mode to −∞ */
    if (inf(a) ≥ 0) return sup(b)/sup(a);
    if (sup(a) ≤ 0) return inf(b)/inf(a);
    return std::min(inf(b)/inf(a), sup(b)/sup(a));
  }
```

This code is used in chunk 2

This is a *chunk* of code. It is identified by its name, here "h such that $[0,h]\boldsymbol{a} \subseteq \boldsymbol{b}$ (intervals)". The `ctangle` program, when extracting the code, orders the chunks based on their names. Each chunk is numbered by `cweave`, and these numbers are convenient for referencing them in the LP document.

A nice feature of `cweave` is that it typesets the code in a very readable form, while the code that is typed in a web file does not even need to be indented. Mathematics can be included in a LaTeX form as a comment, and **if** conditions are typeset more like math, rather than C++.

Now, given interval vectors $\boldsymbol{a}$ and $\boldsymbol{b}$, with each component of $\boldsymbol{b}$ containing 0, we wish to find the largest representable $h \geq 0$ such that $[0,h]\boldsymbol{a} \subseteq \boldsymbol{b}$. We write

2 $\langle h$ such that $[0,h]\boldsymbol{a} \subseteq \boldsymbol{b}$ (interval vectors) 2$\rangle \equiv$
  $\langle h$ such that $[0,h]\boldsymbol{a} \subseteq \boldsymbol{b}$ (intervals) 1$\rangle$

```
double compH(const iVector&a, const iVector&b)
{
  double hmin = compH(a[0], b[0]);
  for (unsigned int i = 1; i < sizeV(a); i++) {
    double h = compH(a[i], b[i]);
    if (h < hmin) hmin = h;
  }
  return hmin;
}
```

This chunk includes the previous one and calls *compH* on each two components to find h.

1.4.2 Translating Expressions

A method in VNODE-LP can be broken down into a sequence of formulas, and each formula must be implemented carefully, to ensure that all truncation and roundoff errors in a computation are included in the resulting bounds. To achieve this, each formula (or a few formulas) is translated into a chunk. The resulting chunks are put together by `ctangle`, thus obtaining an implementation of the complete method.

Here is another simple example from VNODE-LP's implementation. When propagating bounds on the global excess [25, 27], we need to evaluate

$$\boldsymbol{r}_{j+1} = (A_{j+1}^{-1}\boldsymbol{A}_{j+1})\boldsymbol{r}_j + A_{j+1}^{-1}\boldsymbol{v}_{j+1},$$

where $\boldsymbol{r}_j$ and $\boldsymbol{v}_{j+1}$ are interval vectors, $\boldsymbol{A}_{j+1}$ is an interval matrix, and A_{j+1} is a nonsingular point matrix. The chunk implementing this formula (we omit the declarations of objects and variables) is:

3 $\langle \boldsymbol{r}_{j+1} = (A_{j+1}^{-1}\boldsymbol{A}_{j+1})\boldsymbol{r}_j + A_{j+1}^{-1}\boldsymbol{v}_{j+1}\ \ 3\rangle \equiv$ /*

$$\begin{array}{r}
\textit{trial_solution}\rightarrow A = A_{j+1} \\
A \supseteq \boldsymbol{A}_{j+1} \\
v \supseteq \boldsymbol{v}_{j+1} \\
\textit{solution}\rightarrow r \supseteq \boldsymbol{r}_j \\
\hline
\end{array}$$

$$\begin{aligned}
\textit{Ainv} &\ni A_{j+1}^{-1} \text{ if } \textit{ok} \\
\textit{temp} &\supseteq A_{j+1}^{-1}\boldsymbol{v}_{j+1} \\
M &\supseteq A_{j+1}^{-1}\boldsymbol{A}_{j+1} \\
\textit{trial_solution}\rightarrow r &\supseteq (A_{j+1}^{-1}\boldsymbol{A}_{j+1})\boldsymbol{r}_j \\
\textit{trial_solution}\rightarrow r &\supseteq \boldsymbol{r}_{j+1} = (A_{j+1}^{-1}\boldsymbol{A}_{j+1})\boldsymbol{r}_j + A_{j+1}^{-1}\boldsymbol{v}_{j+1}
\end{aligned}$$

*/

```
bool ok = matrix_inverse→encloseMatrixInverse(Ainv, trial_solution→A);
if (ok) {
   multMiVi(temp, Ainv, v);
   multMiMi(M, Ainv, A);
   multMiVi(trial_solution→r, M, solution→r);
   addViVi(trial_solution→r, temp);
}
```

In the comment above the horizontal line, we state informally where the vectors and matrices are stored before executing the code: *trial_solution→A* stores[1,2] A_{j+1}, v contains $\boldsymbol{v}_{j+1}$, A contains $\boldsymbol{A}_{j+1}$, and *solution→r* contains $\boldsymbol{r}_j$. After the horizontal line, we state each step of the computation, so we can easily check the code that follows against it.

The *encloseMatrixInverse* function computes an interval matrix, output argument *Ainv*, which encloses A_{j+1}^{-1}. If this function computes an enclosure (A_{j+1} is nonsingular and not badly conditioned), then we evaluate the expression. Here *Mi* and *Vi* stand for interval matrix and interval vector, respectively. Obviously, it is not difficult to establish the validity of this code.

Remark 1.1. One may find the explanations here and in [27] containing too much detail. However, our goal is to provide as much detail as possible such that one can readily verify all the steps when going from theory to code.

Remark 1.2. For better understanding, the author has found it helpful to write in comments what is computed, in addition to the exposition before a chunk. We could comment separate lines of code, but it becomes less readable.

1.4.3 Integrating the Lorenz System

We give an example illustrating basic integration with VNODE-LP and showing in more detail how LP works. More examples are given in [27].

With VNODE-LP, the user has to specify first the right side of an ODE problem and then provide a main program. An ODE must be given by a template function for evaluating $y' = f(t, y)$ of the form

4 ⟨template ODE function 4⟩ ≡

```
template⟨typename var_type⟩
void ODEName(int n, var_type *yp, const var_type *y, var_type t,
        void *param)
{
        /* body */
}
```

[1] For readers not familiar with C++, the operator → selects a field in a structure when a pointer is being used.

[2] Since *trial_solution→A*, *A*, *v*, *trial_solution→r*, *Ainv*, *temp*, and *M* are C++ objects, they do not appear in bold font, as they are typeset by `cweave` as code.

Here n is the size of the problem, t is the time variable, y is a pointer to input variables, yp is a pointer to output variables, and $param$ is a pointer to additional parameters that can be passed to this function.

Consider the Lorenz system

$$\begin{aligned} y_1' &= \sigma(y_2 - y_1) \\ y_2' &= y_1(\rho - y_3) - y_2 \\ y_3' &= y_1 y_2 - \beta y_3, \end{aligned}$$

where σ, ρ, and β are constants. This system is encoded in the *Lorenz* function below. The constants have values $\sigma = 10$, $\beta = 8/3$, and $\rho = 28$. We initialize *beta* with the interval containing $8/3$: **interval**(8.0) creates an interval with endpoints 8.0, and **interval**(8.0)/3.0 is the interval containing $8/3$.

5 ⟨Lorenz 5⟩ ≡

```
template<typename var_type>
void Lorenz(int n, var_type *yp, const var_type *y, var_type t,
        void *param)
{
  interval sigma(10.0), rho(28.0);
  interval beta = interval(8.0)/3.0;
  yp[0] = sigma * (y[1] - y[0]);
  yp[1] = y[0] * (rho - y[2]) - y[1];
  yp[2] = y[0] * y[1] - beta * y[2];
}
```

This code is used in chunk 6

We give a simple main program and develop its parts.

6 ⟨simple main program 6⟩ ≡

```
⟨Lorenz 5⟩
int main()
{
  ⟨set initial condition and endpoint 7⟩
  ⟨create AD object 8⟩
  ⟨create a solver 9⟩
  ⟨integrate (basic) 10⟩
  ⟨check if success 11⟩
  ⟨output results 12⟩
  return 0;
}
```

This code is used in chunk 13

The initial condition and endpoint are represented as intervals in VNODE-LP. In this example, they are all point values stored as intervals. The components of *iVector* (interval vector) are accessed like a C/C++ array is accessed.

7 ⟨set initial condition and endpoint 7⟩ ≡
```
const int n = 3;
interval t = 0.0, tend = 20.0;

iVector y(n);
y[0] = 15.0;
y[1] = 15.0;
y[2] = 36.0;
```
This code is used in chunk 6

Then we create an AD object of class `FADBAD_AD`. It is instantiated with data types for computing Taylor coefficients (TCs) of the ODE solution and TCs of the solution to its variational equation, respectively [25]. To compute these coefficients, we employ the FADBAD++ package [40]. The first parameter in the constructor of `FADBAD_AD` is the size of the problem. The second and third parameters are the name of the template function.

8 ⟨create AD object 8⟩ ≡
```
AD *ad = new FADBAD_AD(n, Lorenz, Lorenz);
```
This code is used in chunk 6

Now, we create a solver:

9 ⟨create a solver 9⟩ ≡
```
VNODE *Solver = new VNODE(ad);
```
This code is used in chunk 6

The integration is carried out by the *integrate* function. It attempts to compute bounds on the solution at *tend*. When *integrate* returns, either $t = tend$ or $t \neq tend$. In both cases, *y* contains the ODE solution at *t*.

10 ⟨integrate (basic) 10⟩ ≡
```
Solver->integrate(t, y, tend);
```
This code is used in chunk 6

We check if an integration is successful by calling *Solver→successful*( ):

11 ⟨check if success 11⟩ ≡
```
if (¬Solver->successful())
        cout << "VNODE-LP␣could␣not␣reach␣t␣=␣" << tend << endl;
```
This code is used in chunk 6

Finally, we report the computed enclosure of the solution at *t* by

12 ⟨output results 12⟩ ≡
```
cout << "Solution␣enclosure␣at␣t␣=␣" << t << endl;
printVector(y);
```
This code is used in chunk 6

The VNODE-LP package is in the namespace *vnodelp*. The interface to VNODE-LP is stored in the file `vnode.h`, which must be included in any file using VNODE-LP. We store our program in the file `basic.cc`.

13 ⟨`basic.cc` 13⟩ ≡

#include `<ostream>`
#include `"vnode.h"`
  using namespace std;
  using namespace vnodelp;
  ⟨simple main program 6⟩

When compiled and executed, the output of this program is

```
Solution enclosure at t = [20,20]
14.30[38161600956570,44725513004334]
9.5[785946141093152,801346480733898]
39.038[2374138960486,4119183796657]
```

It is interpreted as

$$y(20) \in \begin{pmatrix} [14.3038161600956570, 14.3044725513004334] \\ [\ \ 9.5785946141093152,\ \ 9.5801346480733898] \\ [39.0382374138960486, 39.0384119183796657] \end{pmatrix}. \tag{1.2}$$

These results are produced using PROFIL/BIAS, and the output format is due to the output format of this package. (The platform is x86 Linux with the GCC compiler.) For comparison, if we integrate the Lorenz system with MAPLE using `dsolve` with options `method=taylorseries and abserr=Float(1,-18)`, and with `Digits := 20`, we obtain

$$y(20) \approx \begin{pmatrix} 14.304146251277895001 \\ 9.5793690774871976695 \\ 39.038325167739731729 \end{pmatrix},$$

which is contained in the bounds (1.2).

Needless to say, one can write application programs without LP. In Figure 1.2, we show the code of the above example written in “plain” C++.

Remark 1.3. Here, the chunks are presented in a consecutive order, but as mentioned earlier, they can be in any order.

1.5 Relevant Work

A comprehensive collection of resources on LP, including extensive bibliography is [21]; annotated bibliography of LP until 1991 is [38]. To the best of the author’s knowledge, VNODE-LP is the first LP implementation of an interval package, and the only other implementation of non-trivial *numerical* software appears to be [33].

```
#include <ostream>
#include "vnode.h"
using namespace std;
using namespace vnodelp;

template<typename var_type>
void Lorenz(int n,var_type*yp, const var_type*y,
            var_type t, void*param) {
  interval sigma(10.0), rho(28.0);
  interval beta = interval(8.0)/3.0;

  yp[0] = sigma*(y[1]-y[0]);
  yp[1] = y[0]*(rho-y[2])-y[1];
  yp[2] = y[0]*y[1]-beta*y[2];
}

int main() {
  const int n = 3;
  interval t = 0.0,tend = 20.0;
  iVector y(n);
  y[0] = 15.0;
  y[1] = 15.0;
  y[2] = 36.0;
  AD *ad= new FADBAD_AD(n,Lorenz,Lorenz);
  VNODE *Solver= new VNODE(ad);
  Solver->integrate(t,y,tend);
  if(!Solver->successful())
    cout<<"VNODE-LP could not reach t = "<<tend<<endl;
  cout<<"Solution enclosure at t = "<<t<<endl;
  printVector(y);
  return 0;
  }
```

Fig. 1.2: “Plain” C++ code for the Lorenz example

In [23], LP is used to facilitate the verification of a network security device. The authors propose in [23] that LP techniques are used to “document the entire assurance argument.” According to their experience, rigorous arguments, including machine-generated proofs of theory and implementation, “did not significantly improve the certifier’s confidence” in their validity. One of the main reasons is that specifications and proofs were documented in a manner to facilitate acceptance by mechanical tools rather than humans. Essentially, the authors conclude that LP greatly facilitates the development of assurance arguments that would be more naturally understood by (human) certifiers than descriptions of machine-generated proofs.

A notable methodology for inspecting an implementation is the program function tables approach of D. Parnas [31]. Before considering LP, the author assessed this approach for inspecting VNODE. However, program function tables are suitable when the relation between input and output arguments is represented by a relatively simple function, which is hardly the case with VNODE.

1.6 Summary of Experience

Developing a non-trivial literate program can be time consuming, which manifests itself into a substantial "up-front" investment of time: we focus on writing a high-quality, well-structured, and easy-to-understand document. This requires paying attention to detail and ensuring that no errors are present. Since this process is inherently slow, one is "forced" to write code carefully, reducing the likelihood of errors.

Once the effort is put into writing a good LP document, then little time goes into debugging and testing—instead of trying to discover errors through them, we simply proofread the LP document. Moreover, theory and code can be cross checked against each other, and error in one may be revealed in the other. In addition, since documentation and code are in one source, they can be naturally kept in sync.

In the author's opinion, if one shows that (a) the theory of a method is correct and (b) its implementation is a provably correct translation of the theory, then minimal testing is required. From the author's experience, if he had implemented the original VNODE solver through LP, then less time would have been spent on checking the implementation, debugging, and testing. More importantly, the confidence in the implementation would have been much higher.

There are 14 tests in the distribution of VNODE-LP. Their main purpose is to ensure that the IA package and VNODE-LP are installed properly. Indeed, the few problems reported to the author about VNODE-LP not being able to execute a test successfully were all related to problems in the installation of the underlying IA package.

It does not appear appropriate to use LP at early stages of program development, when prototyping and experimenting with algorithms, design, and interfaces. When a design is settled, and no major changes are anticipated, then one can "cement" the implementation with LP. In our case, VNODE was in a stable state, and no experimenting was needed before investing into VNODE-LP.

The number of C/C++ lines (without comments) in VNODE-LP is 2,030. This is not a large package, but complex "per line of code." The LP document [27] is 218 pages. For much larger programs, LP may not be an attractive option, especially when a software product must be delivered on time. In academia, researchers rarely go beyond prototype, research codes and releasing software packages, let alone devoting a substantial amount of time into producing a book-like manuscript (which may not count as a publication). At least for the above two reasons, LP is not ubiquitous, even though it has existed for more than 25 years.

Although LP may appear prohibitively time consuming, the author believes that the cumulative effort for producing and maintaining a complex program is smaller using LP compared to "traditional" program development. The author also believes