

Interval Subroutine Library Mission*

George F. Corliss¹, R. Baker Kearfott², Ned Nedialkov³, John D. Pryce⁴, and
Spencer Smith⁵

¹ Marquette University

² University of Louisiana at Lafayette

³ McMaster University and Lawrence Livermore National Laboratory

⁴ Cranfield University, RMCS Shrivenham

⁵ McMaster University

Abstract. We propose the collection, standardization, and distribution of a full-featured, production quality library for reliable scientific computing with routines using interval techniques for use by the wide community of applications developers.

1 Vision – Why are we doing this?

The interval/reliable computing research community has long worked to attract practicing scientists and engineers to use its results. We use any of the terms interval, reliable, verified computation in the sense of producing rigorous bounds on true results [1, 2]. The Interval Subroutine Library (ISL) is a project to place interval tools into the hands of people we believe will benefit from their use by gathering and refining existing tools from many interval authors. We acknowledge that intervals carry a steep learning curve, and that they sometimes have been over-promised. The winning strategy for widespread adoption of interval technologies is the development of “killer applications” that are so much better (in some sense) than current practice that practicing scientists and engineers have no choice but to adopt the new technology.

The ISL team wants to see such killer applications appear, but producing them is not our mission. The routine use of interval techniques by practicing scientists and engineers is hampered by a lack of widely-used, comprehensive, quality interval software that is available on all major platforms (Linux, Mac OS, Unix, Windows). Once such software is available, use of interval techniques is likely to grow along at least three paths: small-scale applications by scientists/engineers in the course of their daily work; professionally built applications in a specific area, such as global optimization or curve graphing; and the almost invisible embedding of verified computing as a tool in commodity software such as spreadsheets or scientific data analysis and document preparation.

ISL can provide the infrastructure for such developments. ISL targets application developers, those who are developing the significant applications. Interval-based tools tailored for specific end-practitioner applications are developed by

* This work was supported in part by EPSRC Grant D033373/1.

applications developers with expertise in applications areas, but those developers are not finding interval tools they perceive as attractive for their applications. Currently, if we talk to a group of scientists or engineers about intervals and convince them of the value of interval techniques, when they ask, “Great! What software can I use?” there is a long, painful pause. We have many tools, packages, and research codes, but we have no CD that solves their problems with rigorous bounds.

Major obstacles to widespread adoption of interval techniques by developers of main-stream scientific and engineering applications include lack of speed and hardware support, lack of customer demand, lack of interoperability of interval tools, a steep learning curve, and many others. ISL, by itself, can solve **none** of those. ISL can help unify the work of the world-wide interval research community to ease the learning curve with easier to use, more portable software that interoperates better with other interval tools and with other common tools used in scientific computation. A central repository containing much of the high-quality work of the community helps attract customers, who, in turn, drive a broad-based demand for improved hardware support.

The ISL project itself does not author software; contributing authors do that. The goal of the ISL project is to gather and disseminate a library of high quality interval-based tools. The fundamental requirements is “Preserve containment.” Routines are expected to return an enclosure of the correct mathematical result or provide a suitable indication of failure. The qualities of interest for the ISL project include

- correctness,
- reliability,
- robustness,
- usability,
- comprehensiveness,
- performance,
- maintainability, and
- portability.

To achieve these qualities, the ISL project encourages its contributing authors to use software engineering sound principles, including documentation, good architecture, thorough testing, and coding standards. The documentation produced and the process of assembling the library also support the goal of achieving high quality. Documentation should be complete, consistent, correct, usable, verifiable, maintainable, and reusable. The development process should have the qualities of productivity, timeliness, and transparency.

The authors are embarking upon a plan for the cooperative development of such a library. This paper lays out the broad scope of the project.

1.1 Short-term goals

By the end of 2007, we expect to offer interval Basic Linear Algebra Subroutines (BLAS) levels 1 and 2 and a collection of problem-solving packages, mostly chosen from existing software, including linear systems, optimization, and differential equations. The collection may include utilities for automatic differentiation, Taylor models, and constraint propagation. For our plan to achieve this, see §3.1.

1.2 Long-term goals

In perhaps 6 to 8 years, we hope to offer a library of interval tools with coverage comparable to early releases of the IMSL or NAG libraries, SLATEC [3], the popular *Numerical Recipes* books [4, 5], the GNU Scientific Library (GSL) [6], or other comparable libraries. The library will be freely available, and we shall also encourage its appearance in commercial products. The library should be used by a significant number of applications widely used in their respective domains. For our plan to achieve this, see §3.2

1.3 History

To provide a comprehensive interval problem-solving library is an old idea, much of whose history in this section was kindly provided by an anonymous referee. In 1976, Ulrich Kulisch and H.W. Wippermann proposed developing language support and problem-solving routines in interval arithmetic. They raised funds and jointly with Nixdorf developed what finally became PASCAL-XSC. The language, the compiler, and the subroutine library are still available from www.xsc.de. The Russian translation of the language and the corresponding toolbox volume just appeared in its third edition.

When IBM became aware of this development, a close cooperation with the Kulisch institute at Karlsruhe was started in 1980. They jointly developed and implemented a Fortran extension called ACRITH-XSC, together with a large number of problem-solving routines with automatic result verification corresponding to the PASCAL-XSC development.

As a next step, a C++ class library called C-XSC was developed at the Kulisch institute at Karlsruhe in the early 1990s, with books [7] and a toolbox volume with problem-solving routines [8]. The software was recently updated, extended, and adapted to new versions of C and C++ at the institute of W. Krämer at Wuppertal. The software is publicly available, including source code, from www.math.uni-wuppertal.de/wrswt/xsc/cxsc_new.html. C-XSC runs on many platforms, and comes with a large number of fast and accurate elementary and special functions for real, complex, real interval, and complex interval data. They all come with proven error bounds. The problem-solving routines that come with C-XSC cover much of the problem space envisioned by the present ISL project.

The `filib++` package [9, 10], from Wuppertal, offers fast computation of guaranteed bounds for interval versions of a comprehensive set of elementary functions, and it adds an extended mode for exception-free computation mode relying on containment sets [11]. `Filib++` uses templates and traits classes to obtain an efficient, easily extendable, and portable C++ library. `Filib++` also come with an extensive set of problem-solving routines, and it is available from www.math.uni-wuppertal.de/wrswt/software/filib.html.

Another library providing guaranteed bounds is INTLAB [12] (based on Matlab) from the institute of S.M. Rump at Hamburg. R.B. Kearfott at Lafayette, Louisiana and W. Walter at Dresden offer other Fortran based libraries.

The development of killer applications to promote interval technologies is surely possible with existing interval environments, but they have not yet happened. By building on the work of our predecessors, ISL, if successful, will increase the penetration of interval techniques into the mainstream of scientific and engineering computation and lowering the barriers to the development of that killer application.

2 Product – What will we deliver?

To meet the needs of a wide community of applications developers in a broad cross-section of applications areas, we need a portable, comprehensive, production-quality library of interval tools solving most of the standard problems of scientific computation. The library should be available in a downloadable or CD form. The library should also have a clear licensing structure that protects authors, while still encouraging commercialization.

2.1 Contents

We envision a hierarchical library, with units organized into chapters roughly as suggested by Figure 1:

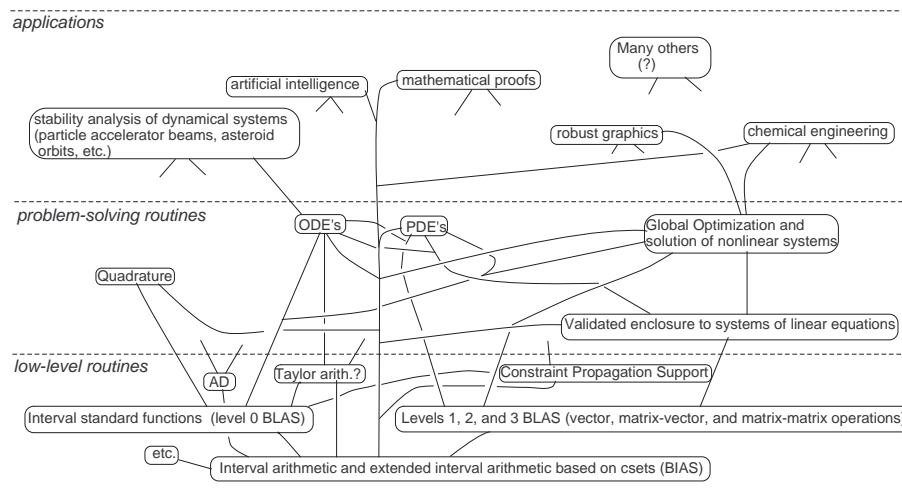


Fig. 1. A tentative hierarchical structure.

Level 0 – Basic Interval Arithmetic: Interval arithmetic including

1. Constructors,
2. Arithmetic operations,

3. Comparison operators,
4. Input/output, and
5. Elementary functions.

Level 0 should be consistent with a C++ interval arithmetic standard, such as that proposed by proposed by Brönnimann, Melquiond, and Pion (BMP) [13], should a suitable standard be adopted. Level 0 should have a large overlap with the functionality provided by many current interval arithmetic packages, including PROFIL/BIAS [12, 14, 15], `filib++` [9, 10], Boost [16], Gaol [17], C-XSC [7, 8, 18–20], and the Sun C++ compiler [21].

Level 1 – Utilities: Level 1 units are called by several units in Level 2 to provide capabilities including

1. Error handler,
2. Additional (non-basic) interval arithmetic features,
3. Vectors and matrix classes,
4. Level 1 and level 2 BLAS,
5. Automatic differentiation,
6. Taylor model arithmetic, and
7. Constraint propagation.

Level 2 – Problem-solving routines: Chapter outline will grow with time, initially, similar to the contents of many textbooks in numerical analysis:

1. Linear systems, eventually including sparse and eigensystems,
2. Nonlinear systems,
3. Optimization,
4. Quadrature,
5. Statistics,
6. Ordinary differential equations, and
7. Partial differential equations.

Level 3 – Applications: Not in the scope of ISL, but we strongly encourage applications developers to build on ISL.

Capabilities not listed here are by no means left out. For example, Level 1 may include multiple precision interval arithmetic with an API close to the Level 0, so that Level 2 and Level 3 units can easily switch from double precision intervals to intervals of higher precision, or vector and matrix classes using elliptical representations for multi-dimensional intervals. Eventually, each chapter should contain a variety of general- and special-purpose routines. Categorizing interval software in a structure roughly paralleling widely-used approximate libraries encourages interval researchers to consider gaps in interval coverage.

3 Plan – How will we accomplish that?

We have both short-term (two years) and long-term (3–10 years) plans.

3.1 Short-term plan: gather, organize, and disseminate

For perhaps two years, this is primarily a library project in the sense of identifying, collecting, organizing, and making available work that already exists.

Step 1 – Language standardization. ISL is a C++ library. Brönnimann, et al. have proposed to add intervals to the C++ language standard [13]. The ISL team is working for the strengthening and the adoption of this proposal. The BMP proposal can become the basis for our ISL Level 0 BIAS well before it is approved. Several existing implementations of intervals in C++ are reasonably close to the proposed standard, so multiple (almost) reference implementations are available.

Step 2 – Pilot inclusion into ISL. Select about three existing packages for initial inclusion into ISL. This gives us a chance to prototype policies, procedures, and practices for incorporating existing work. See the discussion of some issues in §3.3–3.9.

In particular, this paper is **not** a call for participation, as we are still working to refine how ISL will work.

Step 3 – Invite participation. Once some of the issues of policies, procedures, and practices for incorporating existing work have been refined, we will invite 6–10 researchers to submit their work for inclusion in ISL. At this stage, the number of packages we will invite remains modest, as we develop experience, participation, and visibility.

We hope for a very preliminary release including parts of Steps 1 and 2 by the end of 2006 and a release including about five ISL Level 2 problem-solving routines by the end of 2007.

3.2 Long-term plan

After we gain experience and visibility from the short-term “gather, organize, and make available” activities, we expect to expand the scope of the library by inviting contributions from the interval community. Work will be managed along the model of many successful open source projects. We anticipate releases each 1–2 years. We will continue development of a free version of ISL, while seeking a commercial partner such as NAG, Sun, IBM, Intel, or Microsoft.

Next, we turn our attention in subsections 3.3–3.9 to some of the issues that must be settled to ensure the success of ISL.

3.3 Language and environment

We do not wish to ignite religious warfare, but we must choose an appropriate computer language for ISL. In the short-term “gather, organize, and make available” activities, we can include packages in any language. Most existing interval software is in Matlab or some dialect of Fortran or C++. While it is attractive to suppose we can support all languages, with finite resources, our goals are served better by focusing on one language. **ISL is in C++** because there appears to be more existing interval software in C++ than in other candidates. Inter-language interoperability depends on support from the languages themselves.

3.4 Organizational structure

Quality, comprehensive libraries are not compiled by a single person or small group of people over a short time. There are many models we can follow of software development by large, loosely-coupled teams over several years, including the LAPACK project [22], PETSc [23–25], and many open source projects such as the GSL [6].

The ISL project is coordinated by a steering committee, currently, the authors of this paper. We meet occasionally as a group, and subsets meet as possible at conferences. The steering committee sets directions and policies, such as those outlined in this paper. In the long-term steady state, the role of the steering committee is somewhat like that of the editorial board of a major journal, overseeing the work of authors, referees, and the publication process.

3.5 Adding value

There are several good packages for interval arithmetic corresponding to our proposed Level 0 BIAS, there are many interval-based problem-solving routines corresponding to our Levels 1 and 2, and there are a few comprehensive projects such as Karlsruhe XSC Toolbox books [8] and Neumaier’s COCONUT [26]. Kreinovich does an admirable job of capturing and maintaining pointers to many interval projects at [27]. What value does ISL add?

We return to our initial premise. Although many interval tools are available, there is no single source, a web site or a CD offering a standardized, portable, peer-reviewed suite of tools that install and work together. In the long term, we envision a comprehensive, universally used library. This is in contrast to offering general languages, such as in the COCONUT project or General Algebraic Modeling System (GAMS), or offering graphical user interfaces, such as in commercial packages such as Maple. We view the effort as promoting standardization, portability, and reuse. In the short term, ISL works with contributing authors to gather existing interval tools, standardize their installation and interfaces, perform peer review acceptance and comparative testing, provide examples, and make these tools available from a single source.

3.6 Quality Assurance

All interval code has to have the quality of correctness. The code must obey the rule “Preserve containment.”

Contributions are peer refereed. To be considered for inclusion in ISL, normally we expect the algorithm to have been the subject of at least one peer refereed journal paper. Codes, testing, and documentation are also refereed, similar to the standards for an Association for Computing Machinery (ACM) Algorithm. We intend that publications and programs associated with ISL be held to the highest academic and software engineering standards.

We strongly recommend that contributing authors follow modern software engineering practices, recognizing that “modern software engineering practices”

encompass methods proposed by Parnas [28], Literate Programming [29, 30], agile methods such as Extreme Programming [31] and Test-Driven Development [32], and others. Generally, we favor the more formal methods because specifications for, say, a linear equation solver, are not expected to change significantly while development is being done.

The ideal contribution to ISL consists of the following parts.

A specification of the software requirements, including the mathematical statement of the problem and information on the required inputs and possible output values. With respect to the inputs, the specification indicates clearly any constraints that exist on the data. Where necessary, a flag shall be specified whose values indicate the reason for failure when a solution cannot be determined. All contributions to ISL share the goal of achieving the qualities listed in the introduction, especially the requirement, “Containment is preserved.” However, it is difficult to write validatable specifications of non-functional requirements. For instance, validating correctness is challenging for scientific computing problems, because formal proofs of correctness are difficult and often overly conservative, although non-formal proofs with rigor comparable to proofs in the mathematical literature **are** often appropriate. Therefore, rather than specify the requirements, the approach is taken of describing the final software product, typically including statements of the form, “It finds an enclosure of correct solution if the input lies in set Y .” This description is given in the software validation report, discussed below.

A design specification. The ideal specifications includes an API or function signature plus semantics, often modeled on specifications for corresponding packages for approximate solutions.

A software validation report. The software validation report is about a combination of observed scope, tightness and speed (plus maybe memory load), and the observed interplay among them. It characterizes the problems that are successfully solved.

The contributing authors are asked to provide the evidence that the software meets the stated requirements and to describe the level to which the software meets the software quality goals. The software validation can consist of informal and formal analysis, testing and a summary of important software metrics. Techniques for informal and formal analysis include code walkthroughs, code reviews, and inspection. Techniques such as literate programming can be employed so that confidence can be built on the correctness of the code, in a similar sense to how confidence is built by mathematicians inspecting a mathematical proof. The summary of testing also builds confidence by showing the test cases that were passed and that any user can download and run for themselves. The descriptions set expectations for the behavior of the program in similar situations. The descriptions can be tested for lies; for instance, the validation report might assert, “the software was run over a given range of inputs on machines x and y and the program terminated in 5 seconds or less with an enclosure of the correct answer in 87% of cases, terminated in 5 seconds or less with a failure indicator

in 10% of cases, and had not terminated in 5 seconds in the remaining 3% of cases.”

More detailed quality assurance policies and procedures are being developed based on our experiences in Step 2 of the Short Term Plan outlined in §3.1.

3.7 Licensing

Especially in view of Sun being granted several interval-related patent applications, the interval community is increasingly aware of the importance of the protection of intellectual property. ISL needs a carefully considered license which

- protects rights and reputations of authors,
- provides for free distribution, and
- encourages commercialization.

To help us in that, we are gathering and evaluating examples including

- GNU Lesser General Public License (LGPL), Modified Berkeley, Boost, MIT, and other Open Source licenses;
- licenses of various interval packages; and
- intellectual property policies of some (possibly) participating universities.

3.8 Publications

Since many potential participants in the ISL project are academics, the project will not succeed without clear publication opportunities:

- Continuing publication of incremental and innovative development of interval software;
- Identification of omissions in coverage (holes) as development targets;
- Comparative testing of similar packages in the spirit of Enright and Hull [33] or Mazzia, Iavernaro, and Magherini [34] for approximate ODE solvers or Pryce [35, 36] for Sturm-Liouville solvers; and
- Suites of test problems for interval problem-solving routines, e.g., Corliss and Yu for interval arithmetic operations and elementary functions [37]. Interval test suites should include many problems from existing test suites for approximate solvers and also problems intended to test existence and containment properties.
- Software engineering publications related to the development of scientific software with respect to appropriate process models, methodologies and documentation. Software engineering has mostly ignored scientific software and placed most of the emphasis on research on safety critical systems and information systems. There is room to make contributions by looking at the issues that are specific to scientific software.

For academic researchers, release of software to ISL in addition to journal publications offers an external, peer-reviewed process for recognizing research contributions and wider dissemination than links from the authors’ web site.

While valuing the role of *Reliable Computing* as **the** core journal in this research field, we encourage contributing authors to publish in a wide variety of journals, especially journals in applications areas, to help bring the message of intervals to as wide an audience as possible. Intervals are much closer to the main stream of scientific computing than many of us realize, as new applications and researchers using interval techniques are published regularly. We help more people learn about intervals by publishing in the outlets they read.

In the long-term steady state, having code accepted for inclusion in ISL may be viewed as equivalent to a journal paper, probably contributing more to the overall advance of the infrastructure of science than many journal papers.

3.9 Funding

While it would be welcome if someone wanted to provide large funding, that is unlikely. If we look at the models of LAPACK or most open source projects, there may be modest funding somewhere for overall leadership and organization, but the developers are on their own to secure their own funding. One would hope that contributing to a large, well-organized, well-publicized international effort might help many interval researchers get our own work funded.

Similarly, it would be welcome if a large software company provided the leadership and modest funding for the champion to lead an open source project. In other fields of study, with more obvious customers, several firms have made significant contributions to various open source projects.

4 Partners – How can you help?

Clearly, the long-term goal is ambitious, requiring the work of many people over many years. This section outlines our vision of an ideal partnership of contributing authors, chapter architects, referees, and ISL steering committee.

4.1 Contributing author

A contributing author contributes any of

- Code unit for the library to solve some well-defined problem of scientific computing, e.g., constraint propagation, linear systems, optimization;
- Functionality or performance improvements, corrections, or extensions to an existing unit of the library;
- Test suites;
- Documentation; and
- API architecture for a chapter of the library.

For a new unit for the library, a contributing author should submit

- User Guide: installation, requirements, examples;
- Maintenance documentation: system architecture, detailed design, test plan and report;

- Source code; and
- Journal article (with quality of TOMS article and algorithm).

A typical interaction might be

1. Contributing author contacts (or is contacted by) the ISL steering committee.
2. They discuss a suitable problem of scientific computing, specifications, licensing, etc.
3. Contributing author submits a suitable research publication to a journal.
4. Contributing author submits to ISL source; installation instructions; documentation of the problem, description of algorithms, examples of use, references, etc.; acceptance and other tests; copies of journal papers, etc.
5. ISL or chapter editor sends submitted materials to referees.
6. Usual iterations with editors, referees, and authors.
7. ISL accepts or declines the submission.
8. After acceptance, ISL maintains discussions with the contributing author to ensure that updates to the original work are reflected in the library.

ISL should be more than a listing of web links to contributing authors' pages. That requires some process, at least semi-formal. The short-term “gather, organize, and make available” phase of the project will be used to find an appropriate balance of a formal process to ensure quality and a light-weight process all can use effectively. For example, there is no need to duplicate refereeing work already performed for journal publication.

4.2 Chapter architect/editor

In the short term, the ISL steering committee are the architects of the library and the editors for contributed units. As the scope grows, each chapter of the library (see §2.1) has an architect/editor responsible for

- External architecture of the chapter, problem coverage, consistent API;
- Internal architecture, shared utilities; and
- Collaboration with contributing authors for this chapter.

4.3 Referee

The referee contributes to the overall quality of the library by providing an external assessment. The referee reviews materials submitted by contributing authors including source, installation instructions, documentation of the problem, description of algorithms, examples of use, references, acceptance and other tests. The referee is assessing the library materials as they affect application developers who use the library, rather than the more academic concerns of a traditional journal referee. We anticipate that some referees are anonymous, and some are collegial.

The ISL refereeing process may be modeled on the process for refereeing Association for Computing Machinery (ACM) Algorithms. We encourage ISL

contributing authors to submit their work as ACM Algorithms, in which case, the ISL refereeing is sharply truncated.

We anticipate that some refereeing work leads to publishable careful comparative testing of similar packages and compilation of sets of standard test problems for interval problem-solving routines.

4.4 Applications development

The goal of the ISL project is to get quality, portable, uniform, comprehensive interval tools into use by developers of applications used by practicing scientists and engineers. Our target audience includes

- Developers in the interval community, our contributing authors. For example, authors of interval differential equations or optimization solvers benefit from automatic differentiation, constraint propagation, and linear solvers in ISL.
- Scientists from applications areas developing more reliable software than that currently available.
- Small commercial companies seeking the competitive advantage of high reliability software in their market niche.
- Large commercial companies who develop market-leading software packages in industry segments, such as chemical engineering, structural engineering, financial modeling, chip and circuit design, supply chain planning, industrial process engineering, etc.

ISL targets the developers of software for use in these areas.

5 Will ISL Succeed?

It is natural to ask why this ISL project might be more successful than its predecessors described in §1.3. We seek not to disparage the work of others but to highlight opportunities open to the community as a whole to increase the penetration of interval techniques into the common practice of scientific and engineering computation.

Stand on the shoulders of giants – ISL should incorporate as much as possible existing work in the underlying packages for interval arithmetic and the higher-level problem-solving routines.

C++ Standard – If intervals are standardized in C++, intervals become available in the daily software toolbox of most developers of scientific and engineering software, without even a need to download. A standard makes it much easier for several packages to inter-operate, removing the current impediments of slight inconsistencies in the interval arithmetic provided by for example by C-XSC, `filib++`, PROFIL/BIAS, and Sun.

Interoperability – Interval problem-solving routines available with C-XSC, `filib++`, PROFIL/BIAS, and Sun each have strengths. If we can leverage a C++ interval standard for the underlying interval operations and migrate existing problem-solving routines toward more consistent interfaces, it becomes easier for developers to use the best routines from several authors.

Hardware support – Broader adoption of interval software is much more attractive if interval arithmetic is supported directly by hardware, a position advanced by Professor Kulisch for many years, most recently in [38].

Customers – Major hardware vendors will support intervals when there is customer demand, and customers do not demand what they cannot experience. More widely available, portable, easy to use interval software facilitates experiencing the certainty afforded by interval techniques by a broader audience.

Open source – The success of many broad-based open source software development projects demonstrates an alternative to the Not Invented Here model of development in single institutes by leveraging the strengths of many researchers, rather than putting the resources of the research community into competing projects.

World-wide – ISL should involve researchers from Japan, China, India, Brazil, and others as well as traditional seats in Europe, Canada, and US.

Connections with point-based algorithms – Approximate algorithms using pure floating-point arithmetic are becoming more reliable, e.g., Baron for global optimization. A wider network of developers can leverage connections with point numerical analysis and applications developers. Advances from approximate algorithms can improve the speed of interval algorithms, and interval insights can improve the reliability of approximate algorithms at critical steps.

Ease the steep learning curve – Compared with approximate algorithms, most interval algorithms require a deeper understanding of the problem, of the method of solution being used, and of the coding of the problem. We reduce that gap by very careful attention to ease of use and by promoting more consistent interfaces across a wider variety of problem-solving routines.

“Deeper” analysis – We have problem-solving routines covering a wide variety of problems, but there are opportunities to improve the set of problems each can handle and their scalability to larger problems.

“Broader” analysis – We have solved the general case for linear and non-linear systems, differential equations, and optimization repeatedly. A centralized repository makes it easier to recognize the need for special-purpose solvers, e.g., sparse systems, linear ordinary differential equations, linear programming problems, and many more.

Structure for academic recognition for software development – Most members of the interval community are academic researchers, rewarded for publishing, not for coding. ISL may come to provide in the long term a structure for recognition of peer-reviewed software development, leading to increased funding opportunities.

It will take a long time. It will take lots of people. And it will take money.

6 Conclusions

Initially, ISL intends to be a single source for as large a body of existing interval routines as resources allow. In the longer term, ISL offers a quality, portable,

uniform, comprehensive, problem-solving library. Eventually, we aspire to be seamlessly integrated with tools and libraries for approximate computation.

Acknowledgement

An initial draft of this paper was prepared during a visit of the first author to Tibor Csendes at the Department of Applied Informatics, University of Szeged, Hungary, January 16 - 20, 2006. We thank Tibor for his kind hospitality.

We thank anonymous referees for contributing significantly, especially for providing much of the content of §1.3.

References

1. Moore, R.E.: *Methods and Applications of Interval Analysis*. SIAM, Philadelphia, PA (1979)
2. Neumaier, A.: *Interval Methods for Systems of Equations*. Cambridge University Press, Cambridge (1990)
3. Fong, K., Jefferson, T., Suyehiro, T., Walton, L.: *Guide to the SLATEC common mathematical library*. Technical report, [netlib.org](http://www.netlib.org) (1990) See <http://www.netlib.org/slatec/>.
4. Press, W.H., Flannery, B.P., Teukolsky, S.A., Vetterling, W.T.: *Numerical Recipes in Fortran: The Art of Scientific Computing*, Second ed. Cambridge University Press, Cambridge (1992) Also available for Fortran 90, C, and C++.
5. Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P.: *Numerical Recipes in C++: The Art of Scientific Computing*, Second ed. Cambridge University Press, Cambridge (2002)
6. GSL: GNU Scientific Library (1996 - June 2004) <http://www.gnu.org/software/gsl/>.
7. Klätte, R., Kulisch, U., Wiethoff, A., Lawo, C., Rauch, M.: *C-XSC – A C++ Library for Extended Scientific Computing*. Springer-Verlag, Heidelberg (1993)
8. Hammer, R., Hocks, M., Kulisch, U., Ratz, D.: *Numerical Toolbox for Verified Computing I – Basic Numerical Problems*. Springer-Verlag, Heidelberg (1993)
9. Lerch, M., Tischler, G., von Gudenberg, J.W., Hofschuster, W., Krämer, W.: *filib++*, a fast interval library supporting containment computations. *ACM Transactions on Mathematical Software* **32**(2) (2006)
10. 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, Wuppertal, Germany (2001)
11. Pryce, J.D., Corliss, G.F.: *Interval arithmetic with containment sets*. Computing (accepted) Technical Report DoIS/TR01/06, Department of Information Systems, Shrivensham Campus, Cranfield University, Swindon SN6 8LR, UK, 2005. Available at <http://homepage.ntlworld.com/j.d.pryce/isloct05/prycecorliss06.pdf>.
12. Rump, S.M.: *INTLAB interval toolbox, version 5.2 (1999–2006)* <http://www.ti3.tu-harburg.de/intlab.ps.gz>.
13. Brönnimann, H., Melquiond, G., Pion, S.: *A proposal to add interval arithmetic to the C++ standard library*. Technical Report N1843-05-0103, CIS Department, Polytechnic University, New York, and Laboratoire de l'Informatique du Parallélisme, École Normale Supérieure de Lyon, and INRIA Sophia Antipolis (2005–2006)

14. Knüppel, O.: PROFIL/BIAS – A fast interval library. *Computing* **53**(3–4) (1994) 277–287 http://www.ti3.tu-harburg.de/profil_e.
15. Knüppel, O.: PROFIL/BIAS v 2.0. Bericht 99.1, Technische Universität Hamburg-Harburg, Harburg, Germany (1999)
16. Brönnimann, H., Melquiond, G., Pion, S.: The Boost interval arithmetic library (2006) <http://www.cs.utep.edu/interval-comp/main.html>.
17. Goulard, F.: Gaol, not just another interval library (2006) <http://www.sourceforge.net/projects/gaol/>.
18. Hofschuster, W.: C-XSC – A C++ Class Library web page (2004) <http://www.math.uni-wuppertal.de/wrswt/xsc/cxsc.html>.
19. Hofschuster, W., Krämer, W.: C-XSC 2.0: A C++ library for extended scientific computing. In: Numerical Software with Result Verification. Number 2991 in Lecture Notes in Computer Science. Springer-Verlag, Heidelberg (2004) 15–35 Also appeared as Preprint BUW-WRSWT 2003/5, Universität Wuppertal, 2003.
20. Hofschuster, W., Krämer, W., Wedner, S., Wiethoff, A.: C-XSC 2.0: A C++ library for extended scientific computing. Preprint BUGHW-WRSWT 2001/1, Universität Wuppertal (2001)
21. Microsystems, S.: C++ interval arithmetic programming reference (2004–2006) <http://docs.sun.com/db/doc/806-7998>.
22. Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., Sorensen, D.: LAPACK User's Guide, 3rd edn. SIAM, Philadelphia, PA (1999) Certain derivative work portions have been copyrighted by the Numerical Algorithms Group Ltd. See <http://www.netlib.org/lapack/>, <http://www.nacse.org/demos/lapack/>.
23. Balay, S., Buschelman, K., Eijkhout, V., Gropp, W.D., Kaushik, D., Knepley, M.G., McInnes, L.C., Smith, B.F., Zhang, H.: PETSc users manual. Technical Report ANL-95/11 - Revision 2.1.5, Argonne National Laboratory (2004)
24. Balay, S., Buschelman, K., Gropp, W.D., Kaushik, D., Knepley, M.G., McInnes, L.C., Smith, B.F., Zhang, H.: PETSc Web page (2001) <http://www.mcs.anl.gov/petsc>.
25. Balay, S., Gropp, W.D., McInnes, L.C., Smith, B.F.: Efficient management of parallelism in object oriented numerical software libraries. In Arge, E., Bruaset, A.M., Langtangen, H.P., eds.: Modern Software Tools in Scientific Computing, Birkhäuser Press (1997) 163–202
26. Neumaier, A.: COCONUT Web page (2001–2003) <http://www.mat.univie.ac.at/~neum/glopt/coconut>.
27. Kreinovich, V.: Interval Computations (2006) <http://www.cs.utep.edu/interval-comp/main.html>.
28. Parnas, D.L.: Software Fundamentals: Collected Papers by David L. Parnas. Addison-Wesley (2001)
29. Knuth, D.E.: Literate programming. *The Computer Journal* **27**(2) (1984) 97–111
30. LiterateProgramming: Literate Programming Web page (2000–2005) <http://www.literateprogramming.com/>.
31. Burke, E.M., Coyner, B.M.: Java Extreme Programming Cookbook. O'Reilly, Sebastopol, CA (2003)
32. Beck, K.: Test-Driven Development: By Example. Addison Wesley (2003)
33. Hull, T., Enright, W., Fellen, B., Sedgwick, A.: Comparing numerical methods for ordinary differential equations. *SIAM J. Numer. Anal.* **9** (1972) 603–637
34. Mazzia, F., Iavernaro, F., Magherini, C.: Test set for IVP solvers, release 2.2 (2003) <http://pitagora.dm.uniba.it/~testset/>.

35. Pryce, J.D.: A test package for Sturm-Liouville solvers. *ACM Trans. Math. Software* **25**(1) (1999) 21–57
36. Pryce, J.D.: Algorithm 789: SLTSTPAK, a test package for Sturm-Liouville solvers. *ACM Trans. Math. Software* **25**(1) (1999) 58–69
37. Corliss, G.F., Yu, J.: Testing COSY's interval and Taylor model arithmetic. In Alt, R., Frommer, A., Kearfott, R.B., Luther, W., eds.: *Numerical Software with Result Verification: Platforms, Algorithms, Applications in Engineering, Physics, and Economics*. Number 2992 in *Lectures Notes in Computer Science*. Springer, Heidelberg (2004) 91–105
38. Kirchner, R., Kulisch, U.W.: Hardware support for interval arithmetic. *Reliable Computing* **12**(3) (2006) 225–237