THE FUTURE OF ELECTRICITY IS HERE. ARE YOU READY?

PRESENTING THE FIRST OF ITS KIND PLATFORM PRESENTING THE INTELLIGENT ELECTRICITY ECOSYSTEM

The 2015 IEEE-IEEMA INTELECT Conference and Exposition will include the first-ever 3-in-1 global platform that brings together a $10 billion business opportunity across industry verticals. It will feature:

- Interactive Display Pavilions
- A World Class Expo
- Global Conference: Smart Electricity for Emerging Markets

PLUG INTO THE FUTURE OF ELECTRICITY TODAY

www.ii-intelect.org
Guest Editors’ Introduction
Andy Terrel, Michael Tobis, and George K. Thiruvathukal
Scientific Software Communities

ResearchCompendia.org: Cyberinfrastructure for Reproducibility and Collaboration in Computational Science
Victoria Stodden, Sheila Miguez, and Jennifer Seiler
The authors outline three goals to consider in building cyberinfrastructure to support scientific research and dissemination. They also present ResearchCompendia, a project designed to facilitate reproducibility in computational science by persistently linking data and code that generated published findings to the article, and executing the code in the cloud to validate or certify those findings.

Reproducible Research as a Community Effort: Lessons from the Madagascar Project
Sergey Fomel
Instead of computational reproducibility being the responsibility of an individual author, it should become the responsibility of open source scientific-software communities. A dedicated community effort can keep a body of computational research alive by actively maintaining its reproducibility. The Madagascar open source software project offers an example of such a community.

Crowdsourcing Scientific Software Documentation: A Case Study of the NumPy Documentation Project
Aleksandra Pawlik, Judith Segal, Helen Sharp, and Marian Petre
Documentation crowdsourcing may be considered a way to address the issue of poor documentation of scientific software packages. Using the NumPy documentation project as a case study, the authors discuss how to leverage the knowledge about software that resides within the developer and user community. The conclusion suggests preliminary guidelines for those thinking about documentation crowdsourcing.

Run-Time Extensibility and Librarization of Simulation Software
Jed Brown, Matthew G. Knepley, and Barry F. Smith
Build-time configuration and environment assumptions are hampering progress and usability in scientific software. This situation, which would be utterly unacceptable in nonscientific software, somehow passes for the norm in scientific packages. The scientific software community needs reusable, easy-to-use software packages that are flexible enough to accommodate next-generation simulation and analysis demands.

STATEMENT OF PURPOSE
Computing in Science & Engineering (CiSE) aims to support and promote the emerging discipline of computational science and engineering and to foster the use of computers and computational techniques in scientific research and education. Every issue contains broad-interest theme articles, departments, news reports, and editorial comment. Collateral materials such as source code are made available electronically over the Internet. The intended audience comprises physical scientists, engineers, mathematicians, and others who would benefit from computational methodologies. All articles and technical notes in CiSE are peer-reviewed.

For more information on these and other computing topics, please visit the IEEE Computer Society Digital Library at www.computer.org/csdl.
ALSO IN THIS ISSUE

46 Simplified Pseudopotential Problems for the Classroom
Trisha Salagaram, Richard Charles Andrew, and Nithaya Chetty

The authors devise a range of simplified, easily programmable computational problems appropriate for the classroom, which can help teach advanced undergraduate students about particular theoretical and computational aspects of the electronic structure method. This article focuses on the pseudopotential, which is a centrally important concept in many modern ab initio methods.

54 Alternative Analysis Networking: A Multicharacterization Algorithm
Kevin Albarado, Timothy Ledlow, and Roy Hartfield

A neural network technique known as Kohonen unsupervised training is coupled with a modified version of the particle swarm optimization technique in an effort to develop an algorithm capable of finding multiple optimal solutions for a given problem. The results of five example problems of increasing difficulty validate the Alternative Analysis Networking algorithm's functionality.

COLUMNS

4 From the Editors
The Next Generation of Computational Science and Engineering
George K. Thiruvathukal

72 The Last Word
Modern Computing—in 1949
Charles Day

DEPARTMENTS

6 Books
A Different Perspective on Scientific Programming
Stephen Weppner

64 Scientific Programming
Vertical Integration
Matthew Turk

68 Reviewer Thanks

RESOURCES

7 AIP Membership Information
11 IEEE Computer Society Information

Erratum
In volume 16, number 6 of CiSE, the second reference on p. 7 incorrectly shows that L.K. Miller’s article was published in 2003. It was actually published in 2010.
The Next Generation of Computational Science and Engineering

Over the past few months, I’ve been thinking a lot about my career and how to maximize impact, by which I don’t necessarily mean scholarly impact. For me, it all started as a software developer with many interdisciplinary interests outside the purview of what might be termed my day job (a euphemism for what I get paid to do at any given time). Like many and not so many in the field, I’ve been pondering how richly blessed I am to be employed in a profession that also doubles as one of my hobbies. I’m even more blessed to be an editor of an excellent publication that values the intersection of computer science with other disciplines.

When I became EIC of CiSE, I was particularly delighted at all of the interdisciplinary content that makes CiSE what it is. This is in no small part to the excellent work of many of my predecessors who served as EIC, from Francis Sullivan (who was the EIC when I wrote my first CiSE article), to Norman Chonacky, to Isabel Beichl. These interdisciplinary editors were each able to see beyond their disciplinary training and think about how to leave their mark—or better put, imprimitur—on CiSE, so to speak. As I enter the second half of my time as EIC, I’m already coming to terms with the fact that this important and defining work of my career may well be drawing to a close. Crazy though it may sound, I’m not the least burned out as EIC and still feel like I have a lot of work to do.

And Now for Something Completely Different (or the Same)

As you all know by now, you’re either reading this editorial or not. You’re in a state similar to Schrödinger’s Cat, a paradox learned by most of us in the wonderful course known as Quantum Mechanics (which either leaves you loving physics or lost beyond hope). As well, you know that you’re now reading (or not reading) the digital version of CiSE, because unless you opted for a deal that you couldn’t refuse (or chose to refuse) you’re probably not getting the print version of CiSE. No matter what, you can’t actually be in both states (reading and not reading), for doing so would defy the basics of Boolean logic (True and not True, which is False). Anyway, I sure hope you’re reading, but I think reading is going to take on a different character in the future. In one scenario, folks will continue to read my editorial. In another scenario, folks will read whatever they want (or are directed) to read—say, through carefully placed keywords. But will they read my editorial? Time will tell.

So What Comes Next

It’s natural to wonder, what comes next? Will a space probe land on a comet, or will the EIC get back to the topic of his article?

Each EIC brings a different set of experiences and institutional strengths with him (or her). In my case, I’m a professor at a Jesuit Catholic university. No, you’re not about to get any sort of lecture on the Jesuits or Catholicism (and I’m not either, so I wouldn’t exactly be qualified to write about it). But I think in the nearly 15 years that I’ve been at Loyola University Chicago, I’ve become convinced of the great value of looking beyond your own discipline to understand other pressing issues in the world. I truly think CiSE needs to expand its focus twofold: by doing a complete review of what we do and how it can be more relevant. You’re going to see some of these changes coming down the pike.

With this issue, I’m planning to expand our focus on community. If there’s anything that makes CiSE great, it’s our community. More importantly, it’s an example of...
communities of practice (see http://en.wikipedia.org/wiki/Community_of_practice). This is in keeping with Jean Lave and Etienne Wenger’s idea that a community of practice involves “a group of people who share a craft and/or a profession.” I won’t spoil the guest editors’ introduction (even though I co-authored it), but the development of scientific and engineering software is something that deserves special, regular attention of our publication. It’s not always the sort of thing that journals and conferences want to focus on, but without real, working software, it’s hard to imagine our own existence. Toward this end, I’ll be working with my editorial board to expand this notion of communities of practice.

We have many such examples. Our recent issues on leadership computing are an example of communities of practice focusing on the high end of scientific/technical computing, mostly at national and international laboratories—and often using supercomputers. For a long time, I’ve felt that this community wasn’t much of a focus of CiSE, yet they were highly interested in what we do. After a few meetings with folks from Argonne National Laboratory, we decided to include this community of practice in our focus.

This takes me to another community of practice, which is well-represented in the current issue. There are many communities working on the development of scientific software. Not all of these folks work on supercomputing. In fact, many of the projects are used in the mainstream for non-scientific projects. For example, there are very well-known Python-based library projects, such as NumPy, SciPy, and so on. I’ve seen these projects used in a number of scientific and non-scientific projects alike, including projects for natural language processing, electronic music, and many others. Many of the folks doing these projects are outside of the academy and often work on them as a hobby, yet they’ve risen to real prominence. CiSE has an important place in these communities, and it may well be one of the only ways that their work can be recognized (beyond GitHub and Bitbucket, where they’re often hosted).

What other, more important things can we be doing? In my work to chair the IEEE Computer Society’s Special Technical Community on Broadening Participation, we’re working to expand our discussion of matters of representation (race, ethnicity, and gender, among criteria) in computing (and STEM in general). I’m working to ensure that these issues are covered in CiSE. It goes almost without saying that our field does suffer because of diversity issues, and we should be concerned with the lack of gender, racial, and ethnic diversity in many of the countries where our publication is received. As one of the cornerstones of my own university’s mission is to work for social justice and expand opportunity to everyone, I cannot think of any good reason not to address this issue in CiSE. If it’s good for our co-sponsoring organization, it’s also good for us. But more importantly, it will lead to a focus beyond the traditional disciplines we cover and, I hope, a corresponding increase in the board’s diversity and our overall way of thinking. Anyway, you’ll likely see a special issue and possible department forming around this and the aforementioned communities of practice.

Now, with that said, whether you’re still reading this (or not), please like and follow us on Facebook (https://www.facebook.com/CiSEmagazine) and Twitter (https://twitter.com/cisemag)!

George K. Thiruvathukal is a full professor in the Computer Science Department at Loyola University Chicago, where he also directs the Center for Textual Studies and Digital Humanities. He is also a guest faculty member at Argonne National Laboratory in the Mathematics and Computer Science Division, working on runtime environments and software to support computational biology applications. His research interests span multiple areas of computer science and interactions with science and the humanities. Thiruvathukal has a PhD in computer science from the Illinois Institute of Technology. He is the editor in chief of CiSE and an associate editor for Computing Now. Contact him at gkt@cisemagazine.org.

www.computer.org/cise
A Different Perspective on Scientific Programming

Stephen Weppner | Eckerd College

This is an unconventional computer science book, as it’s not a textbook, tutorial, or reference. It’s an attempt to categorize the algorithms of quantitative programming and then make further remarks using a number of examples from each category. The objective, I would presume, is to strengthen the ability of the reader to recognize that their own programming challenges can be reduced to a combination of approximate and statistical approaches that must be pragmatic (so that the time length of execution is reasonable). The coding illustrations demonstrate common techniques for the ordering of algorithms.

At the same time, it introduces the reader to Python, a language which is growing in popularity among engineers and scientists. The book excels as an introduction to the Python language for experienced programmers. It has an excellent chapter on parallel processing, an impressive random number generator discussion, and some fine biology and finance examples.

In chapter 2, there’s a good overview of the Python language in approximately 50 pages. It is admirable because it’s a thorough coverage of the subset of the language that’s missing from older high-level languages (help, list, tuple, dict, exceptions, lambda functions, overloading) like Fortran and C. Similar texts often have excess coverage on mundane conditional statements and formatted input/output. This author, aware that his audience may be seasoned programmers, eschews this style. NumPy, SciPy, and MatPlotLib libraries, which are utilized within and are indispensable for writing quantitative code easily, are also described.


Editorial note: The book under review was written by a member of the CiSE editorial board.

We then move to the central subject—algorithms—in chapter three. As promised, the author uses annotated code to go through the theories of orders of growth (how does the algorithm scale), recurrence, timing, data structures, trees, graphs, “greedy algorithms,” artificial intelligence, and long loops. This chapter is arranged in outline format, thus implying an underlying structure. All of these topics are important, but some are only valuable as subsets. Timing algorithms are of no importance to a quantitative researcher unless they’re used to extrapolate orders of growth or debug performance, and the author recognizes this. Likewise, data structures, trees, and graphs are all tools to organize data or structures, yet chapter three defines them as distinct algorithms.

At the end of the chapter, he mentions the infinities of Cantor and Gödel (in two pages). This chapter is the focus of my largest criticism—this might have made great class notes, but I would have preferred some refining and expanding (or exclusion) of the chapter as the author transitioned to book form. For example, his discussion of graph algorithms starts with a two-page list of formal definitions, which I’m unsure of the interested audience. In this chapter, he seems torn between formalizing theoretical constructs and annotating multidisciplinary examples, and I prefer when he does the latter. This is a matter of preference, but in 360 pages the book details the syntax of Python, develops and categorizes the theory of algorithms, and runs through a plethora of very good quantitative examples. J.M. Thijssen takes over 600 pages on a language-independent development of only graduate-level computational physics. Mark Newman takes over 500 pages to teach Python and develop a computational approach to undergraduate physics problems. Both of these authors only attempt an informal categorization by their choice of ordering the subject matter. This chapter by Di Pierro is too short to give the theory of algorithms justice, and at the same time avoid the pitfalls of most categorical systems.
Chapters four through seven contain the standard topics that are a part of most computational science textbooks, and there are some good examples within (but no exercises). The author does put emphasis on describing a variety of random number techniques and generators, which I found useful and is rare in a relatively small book like this.

The subtitle of this book is *with Applications in Physics, Biology, and Finance*. This is true; the biology examples include neural networks, genetics, and protein folding; option pricing, risk evaluation, income generation, capital budgeting, portfolio and trading theories are covered on the financial side. These are welcome. I’m always searching for that computational work that proffers less pure math and modern physics examples and replaces them with pragmatic problems in engineering, biology, and finance. One of the strengths of this book is its multidisciplinary approach. It does have a plethora of pure math and physics problems, but it does better than most in striving for a genuine balance.

Chapter eight, the last chapter, is on parallel processing; it’s excellent in theory and execution. The author really shows the power of parallel processing in his annotated examples (his research includes this topic, as described on the book’s back cover). He builds his own message passing interface (which he calls *Psim*) and explains broadcast, scatter, reduce, and collect algorithms. He then describes and uses old industry standards like MPI (as *mpi4py*) and newer GPU standards using *pyOpenCl*, which he uses to create a Laplace solver.

I bought this book to augment a science modeling class that I teach with examples outside of physics. Since it’s not a textbook, there are no “out of the box” exercises that are usable—but with a little work, this book has added some biology, finance, and parallel processing examples to the course. Though I am critical of aspects of this book that are nontraditional, I compliment the author for taking these risks to produce a nonstandard computational science book that’s distinctive from its competitors, and thus makes unique contributions to the field.

**References**


Stephen Weppner is a physicist at Eckerd College who does research in theoretical nuclear physics. He has used Python in the classroom and for research for a few years. He’s also the co-editor of the “Books” department for *CiSE*. Weppner has a PhD in physics from Ohio University. Contact him at wepnesp@eckerd.edu.

Selected articles and columns from IEEE Computer Society publications are also available for free at [http://Computing-New.computer.org](http://Computing-New.computer.org).
If simulation is the third tier of science, then the communities that build the simulation software are the engine of innovation. Yet the scientific community as a whole tends to avoid issues surrounding the building of software. With a preference for more traditional scientific achievements, such as experimental results or theoretical derivations, the average scientist has attributed the writing, maintaining, and distribution of software as a tax that must be paid rather than a process that’s rewarding in its own right. The scientific community as a whole, in its turn, neglects to reward producers of polished, shareable extensible software. The consequence is typically software that, while generally suitable for task, is brittle and problematic when viewed as an asset to the long-term needs of the scientific enterprise.

The importance of software to the modern world cannot be understated and software’s importance to science is no different. Whereas successful software efforts lead to a fruitful, celebrated career in industry, the scientific software writer is often forgotten. To highlight this uncherished group, this issue of CISE has been devoted to presenting the challenges and the collective efforts of scientific...
software communities. The scientific software community has produced so many huge innovations in our society that it's vital that we make the process of building such communities well understood and well supported.

The Community’s Role and Its Importance
In this issue, we hear two different approaches to reproducible software practices, an approach on maintaining documentation for important base libraries, and a discussion on ways of extending a software library's functionality to keep it relevant as a community evolves over time. These topics challenge the boundaries of what software can be for an individual and for a community. They expose weaknesses in our state-of-the-art practices with an eye towards a sustainable future. By using these techniques, we avoid numerous withdrawn results—a current crisis due to reliance on software without verification.

Our authors’ insights into these problems give us an occasion to pause and ask what the fundamental role of the scientific software community should be. We posit that writing simulation software is transitioning from an activity that can be accomplished by a few coders who learned on the job to a full profession requiring years of study. This transition has prompted a few institutions to build centers to give a scientific software writer a place to have a career in academe, but this trend is new and is only appearing after a massive drain of talent to private enterprise. Certainly, building a community that supports and sustains the careers of talent to private enterprise. Certainly, building a community that supports and sustains the careers of our novice software developers is critical to the path of science.

The Community's Role and Its Importance
In this issue, we hear two different approaches to reproducible software practices, an approach on maintaining documentation for important base libraries, and a discussion on ways of extending a software library's functionality to keep it relevant as a community evolves over time. These topics challenge the boundaries of what software can be for an individual and for a community. They expose weaknesses in our state-of-the-art practices with an eye towards a sustainable future. By using these techniques, we avoid numerous withdrawn results—a current crisis due to reliance on software without verification.

Our authors’ insights into these problems give us an occasion to pause and ask what the fundamental role of the scientific software community should be. We posit that writing simulation software is transitioning from an activity that can be accomplished by a few coders who learned on the job to a full profession requiring years of study. This transition has prompted a few institutions to build centers to give a scientific software writer a place to have a career in academe, but this trend is new and is only appearing after a massive drain of talent to private enterprise. Certainly, building a community that supports and sustains the careers of our novice software developers is critical to the path of science.

The transition of scientific software from an occasionally used skill to a demanding career also requires additional policies, practices, and structures to motivate excellence in the field. One often-used analogy for scientific software is the comparison to a piece of experimental equipment. While the physics experimental instruments will have hundreds of authors, even though the software has a similar numbers of contributors, our papers usually only list the very few and very dedicated authors. Additionally, these large efforts directly pursue a grand challenge that unites a large portion of the field, while software communities tend to be the foundation that must be strong to allow further growth in all sciences.

Perhaps the analogy to the university library is more apt. As the center of an institution's knowledge, libraries are the base of knowledge for an institution to retain and teach its pupils. Software communities similarly build tools and libraries that enable future generations. Additionally, as a librarian's role is to help researchers answer difficult questions through previously collected and stored knowledge, the scientific software community's job is to enable researchers to answer difficult questions through knowledge collected and stored in code. In this regard, the scientific software community’s role has become an extension of the librarians’ role, as our scientific knowledge has become a product of the code we write.

As the librarian stands as the guide for knowledge throughout the university, the scientific software developer stands as the trailblazer for new, computationally intensive intellectual enterprises. The trail is one that leads to further science results and a healthy dialog feeding more software innovation as a result. This feedback loop of science result to new idea to software implementation to contribution to software library is vital to the continued success of scientific innovation. The need for recognition and maintenance of our software communities is critical.

In This Issue
Articles in this issue provide some examples of how a more considered focus on the software development process can feed the development of science.

Victoria Stodden and her colleagues write about the needed infrastructure to support reproducible science in “ResearchCompendia.org: Cyberinfrastructure for Reproducibility and Collaboration in Computational Science.” They describe their ResearchCompendia effort, a Web portal for uploading and managing a compendium surrounding a scientific result. Addressing issues with the “ubiquity of errors” in science, including the specialized errors that occur in simulations, requires a complete pipeline that's documented, published, and managed. Such an effort requires a dedicated community to oversee the resource and help its adoption.

Next, in “Reproducible Research as a Community Effort: Lessons from the Madagascar Project,” Sergey Fomel gives us a perspective on reproducibility from leading a scientific software community. Admirably, the Madagascar project holds reproducibility as its foundational goal. This goal requires a dedicated team maintaining the previous work and using tools that automate the reproduction of the work. Fomel argues very well that while this burden often falls out of possibility for a single...
author, a community dedicated to lifting its members to their full potential benefits all of science.

In “Crowdsourcing Scientific Software Documentation: A Case Study of the NumPy Documentation Project,” Aleksandra Pawlik and her colleagues turn our attention to documentation, a task dreaded by all software developers but vital for community projects. A potential solution to this gap in needs of the community and dedication of developer time is to crowdsourced. The authors take us through the journey of creating infrastructure, maintaining the service, and engaging the user base to allow such a documentation procedure. By lowering the barriers of entry into contributing to the projects, we see the quality of the documentation grow and the community around the project grow as well.

Finally, Jed Brown and his colleagues take us on a thought experiment about what mass-market software would be like without run-time extensibility in “Run-Time Extensibility and Librarization of Simulation Software.” The nightmare scenario that would drive end users away from their browsers is offered as an analogy to the standard operating practice of scientific computation. They propose adopting methods of run-time extensibility, allowing code methods to progress from the ad hoc methods of a small project toward the development of infrastructure for sustaining the innovation of an entire community.

In summary, the articles in this issue show us the benefits of applying professional software development standards to scientific software. Software projects which build in best practices such as extensibility, reproducibility, deployment, and testing, encourage further productivity among their users and subsequent developers. The road to making software that can be tested, understood, reused, and extended without undue hardship helps all of science, even though the cost of development will initially be higher.

Analogously, developing an academic environment which rewards domain specialists for attention to the methods of good software will have costs. An academic environment which provides reliable and rewarding career paths for developers who have the rare overlap of skillsets of numerical, statistical, or combinatorial algorithmics as well as systematic, testable, and extensible software development will have costs, too, especially in a world where a subset of those skills is in great commercial demand.

All of these approaches have been proven on various occasions to have enormous payoffs in various computing domains. The articles in this issue testify to the proposition that science is no exception in this regard.

Our sincerest hope is that this issue gives visibility to some of the challenges that software communities encounter and benefits they provide in supporting the science. Whether tenured professor, scientific staff, or new research assistant, community members’ diligence in creating community-minded scientific software is critical to sustained innovation. Just as a river must have many feeding streams, scientific computation requires constant sources of ideas and implementations so that all of science reaps its benefit.

Andy Terrel is the Chief Science Officer at Continuum Analytics and President of the NumFOCUS Foundation, a foundation dedicated to sustaining scientific computing tools. His research includes utilizing supercomputers with Python and studying methods for speeding up computational fluid dynamics. Terrel has a PhD in computer science from the University of Chicago. He has contributed to numerous open source projects, notably the FENicS Project and Sympy. Contact him at andyterrel@gmail.com.

Michael Tobis is the editor-in-chief and cofounder of Planet 3.0, a site dedicated to scientifically informed conversations about sustainable technologies and cultures. His interests focus on the interface between science and public policy. Tobis has a PhD in atmospheric and oceanic sciences from the University of Wisconsin-Madison. Contact him at mtobis@gmail.com.

George K. Thiruvathukal is a full professor in the Computer Science Department at Loyola University Chicago, where he also directs the Center for Textual Studies and Digital Humanities. He is also a guest faculty member at Argonne National Laboratory in the Mathematics and Computer Science Division, working on runtime environments and software to support computational biology applications. His research interests span multiple areas of computer science and interactions with science and the humanities. Thiruvathukal has a PhD in computer science from the Illinois Institute of Technology. He is the editor in chief of CiSE and an associate editor for Computing Now. Contact him at gkt@cismeagazine.org.

Selected articles and columns from IEEE Computer Society publications are also available for free at http://ComputingNow.computer.org.
PURPOSE: The IEEE Computer Society is the world’s largest association of computing professionals and is the leading provider of technical information in the field.

MEMBERSHIP: Members receive the monthly magazine Computer, discounts, and opportunities to serve (all activities are led by volunteer members). Membership is open to all IEEE members, affiliate society members, and others interested in the computer field.

COMPUTER SOCIETY WEBSITE: www.computer.org

PURPOSE: To check membership status or report a change of address, call the IEEE Member Services toll-free number, +1 800 678 4333 (US) or +1 732 981 0060 (international). Direct all other Computer Society-related questions—magazine delivery or unresolved complaints—to help@computer.org.

CHAPTERS: Regular and student chapters worldwide provide the opportunity to interact with colleagues, hear technical experts, and serve the local professional community.

AVAILABLE INFORMATION: To obtain more information on any of the following, contact Customer Service at +1 714 821 8380 or +1 800 272 6657:

- Membership applications
- Publications catalog
- Draft standards and order forms
- Technical committee list
- Technical committee application
- Chapter start-up procedures
- Student scholarship information
- Volunteer leaders/staff directory
- IEEE senior member grade application (requires 10 years practice and significant performance in five of those 10)

PUBLICATIONS AND ACTIVITIES

Computer: The flagship publication of the IEEE Computer Society, Computer, publishes peer-reviewed technical content that covers all aspects of computer science, computer engineering, technology, and applications.

Periodicals: The society publishes 13 magazines, 16 transactions, and one letters. Refer to membership application or request information as noted above.

Conference Proceedings & Books: Conference Publishing Services publishes more than 175 titles every year.

Standards Working Groups: More than 150 groups produce IEEE standards used throughout the world.

Technical Committees: TCs provide professional interaction in more than 45 technical areas and directly influence computer engineering conferences and publications.

Conferences/Education: The society holds about 200 conferences each year and sponsors many educational activities, including computing science accreditation.

Certifications: The society offers two software developer credentials. For more information, visit www.computer.org/certification.

EDUCATIONAL ACTIVITIES

- Peer-reviewed technical content
- Conferences and publications
- Technical committees
- Standards working groups
- Educational activities
- Certification programs

IEEE BOARD OF DIRECTORS

President: Dejan S. Milojicic
President-Elect: Thomas M. Conte
Past President: David Alan Grier
Secretary: David S. Ebert
Treasurer: Charlene (“Chuck”) J. Walrad
VP, Educational Activities: Phillip Laplante
VP, Member & Geographic Activities: Elizabeth L. Burd
VP, Publications: Jean-Luc Gaudiot
VP, Professional Activities: Donald F. Shafer
VP, Standards Activities: James W. Moore
VP, Technical & Conference Activities: Cecilia Metra

2014 IEEE Director & Delegate Division VIII: Roger U. Fujii
2014 IEEE Director & Delegate Division V: Susan K. (Kathy) Land
2014 IEEE Director-Elect & Delegate Division VIII: John W. Walz

COMPUTER SOCIETY OFFICES

Phone: +1 202 371 0101 • Fax: +1 202 728 9614
Email: hq.ofc@computer.org
Los Alamitos: 10662 Los Vaqueros Circle, Los Alamitos, CA 90720
Phone: +1 714 821 8380
Email: help@computer.org

MEMBERSHIP & PUBLICATION ORDERS

Phone: +1 800 272 6657 • Fax: +1 714 821 4641 • Email: help@computer.org

Asia/Pacific: Watanabe Building, 1-4-2 Minami-Aoyama, Minato-ku, Tokyo 107-0062, Japan
Phone: +81 3 3408 3118 • Fax: +81 3 3408 3553
Email: tokyo.ofc@computer.org

IEEE BOARD OF DIRECTORS

President: J. Roberto de Marca
President-Elect: Howard E. Michel
Past President: Peter W. Staecker
Secretary: Marko Delimar
Treasurer: John T. Barr
Director & President, IEEE-USA: Gary L. Blank
Director & President, Standards Association: Karen Bartleson
Director & VP, Educational Activities: Saurabh Sinha
Director & VP, Membership and Geographic Activities: Ralph M. Ford
Director & VP, Publication Services and Products: Gianluca Setti
Director & VP, Technical Activities: Jacek M. Zurada
Director & Delegate Division V: Susan K. (Kathy) Land
Director & Delegate Division VIII: Roger U. Fujii

NEXT BOARD MEETING

26–30 January 2015, Long Beach, CA, USA
ResearchCompendia.org: Cyberinfrastructure for Reproducibility and Collaboration in Computational Science

Victoria Stodden | University of Illinois at Urbana-Champaign
Sheila Miguez and Jennifer Seiler | Columbia University

The authors outline three goals to consider in building cyberinfrastructure to support scientific research and dissemination. They also present ResearchCompendia, a project designed to facilitate reproducibility in computational science by persistently linking data and code that generated published findings to the article, and executing the code in the cloud to validate or certify those findings.

In 2004, Gentleman and Temple Lang introduced the concept of the research compendium: a new way of disseminating computational science results that delivers not only the article, but also the software tools and data required to reproduce the published findings. We describe a prototypical software infrastructure based on this idea, called ResearchCompendia, designed to implement two aspects of the compendium: persistently linking data and code that generated published findings to the article; and executing the code in the cloud to validate or certify those findings. Truly reproducible computational research not only improves the reliability of scientific findings, but permits the reuse of tools and data that generated the findings, facilitating downstream collaboration.

We outline three goals for cyberinfrastructure (CI) in support of scientific investigation and dissemination. We posit that CI should reinforce scientific norms—such as transparency and reproducibility—but however, CI should embed and encourage best practices in scientific research, and consider the entire discovery pipeline, even if focusing only on supporting a subset of the scientific workflow. In this article, we develop these ideas in the context of the ResearchCompendia project, and then include a discussion of the future vision of CI in support of science.

Reproducibility and ResearchCompendia.org

Reproducible computational science has attracted attention since Stanford Professor Jon Claerbout developed the idea of really reproducible manuscripts in 1991. Since then, a number of researchers have adopted reproducible methods or introduced them in their role as journal editors, and a body of scholarly literature is beginning to emerge. In May 2013, a workshop report on Knowledge Infrastructures was released, bringing attention to recent rapid changes in how “people create, share, and dispute” knowledge due to changes in communication and research technologies. They note that “new forms of collective discovery and knowledge production … are springing up within and across many academic disciplines” and call for a reexamination of scientific knowledge production, dissemination, and assessment. There have been numerous reports over the last few years from a variety of fields lamenting the irreproducibility of published scientific results and articles appearing in the popular press. A workshop was held in 2011 on computational tools for reproducible research called “Reproducible Research: Tools and Strategies for Scientific Computing” (see the CiSE special issue on this topic at http://scitation.aip.org/content/aip/journal/cise/14/4/10.1109/MCSE. 2012.82). In December 2012, a workshop called “Reproducibility in Computational and Experimental..."
Mathematics” was held at the Institute for Computational and Experimental Research in Mathematics (ICERM) at Brown University.5

Several demonstrated use cases and a rationale for reproducible research were given by David Donoho and his colleagues,6 reproducibility was advocated for the social science community by Gary King among others,7 and the topic was the subject of a 2011 special issue in the magazine Science, for example.8 Recent news articles in Nature and Science call for the release of academic code and data.9

Journals are beginning to require code and data to be made available to readers of their published articles, primarily on externally hosted sites.10

These issues are also being considered at the policy level. In February and then May 2013, the Obama administration issued an Executive Memorandum and an Executive Order, respectively. The Executive Memorandum requires that federal funding agencies submit plans for public access to publications and data—defined as “the digital recorded factual material commonly accepted in the scientific community as necessary to validate research findings including data sets used to support scholarly publications”—by August 2013. The Executive Order directs federal agencies to make government agency data publicly available. These two actions by the White House have had the effect of bringing data access to the fore in federal funding agency conversations and addressing new challenges about how to implement access to digital scholarly objects, including how to ensure reproducibility and how to persistently link together the data and code with the published article.

Computational researchers often make preprints and articles available in repositories such as the arXiv or SSRN, but no comparable convenient facility exists to disseminate the code and data associated with published scientific papers that links together the article, the data, and the code in a structured and persistent way. In this article, we describe a mechanism to knit these ideas into reproducible science, called ResearchCompendia.

We also present a roadmap for the incorporation of CI across the research landscape. This roadmap focuses on lodestars for technical development, and gives less consideration to the equally important issues of incentives, funding, and implementation strategies. We consider three principles to guide the development of computational infrastructure for science:

- Supporting scientific norms—not only should CI enable new discoveries, but it should also permit others to reproduce the computational findings, reuse and combine digital outputs such as datasets and code, and facilitate validation and comparisons to previous findings.
- Supporting best practices in science—CI in support of science should embed and encourage best practices in scientific research and discovery.
- Taking a holistic approach to CI—the complete end-to-end research pipeline should be considered to ensure interoperability and the effective implementation of 1 and 2.

As we’ll describe below, ResearchCompendia prototypes CI solutions for computational publications. Of course, there are myriad tools emerging to support reproducibility and the dissemination of scientific results, of which this is only one effort.11–15

The “Ubiquity of Error” and the ResearchCompendia Architecture

As one of us noted in 2009, “[t]he scientific method’s central motivation is the ubiquity of error—the awareness that mistakes and self-delusion can creep in absolutely anywhere and that the scientist’s effort is primarily expended in recognizing and rooting out error.”6 In the context of traditional empirical research, the response to the ubiquity of error employs standardized approaches such as statistical hypothesis testing and the reporting of information in the publication that enables reproducibility, principally through the materials and methods section.

Research that utilizes computational resources is subject to a new additional source of error, not captured by traditional publication standards. We believe that considering the computational aspects of an experiment as part of the experimental design itself will improve our ability to root error out of the scientific discovery process. For example, coding errors, the implementation or execution of algorithms or methods, or data filtering and cleaning decisions, could all be better checked with access to code and data.

ResearchCompendia is a website housing a collection of compendia pages. Each compendia page is associated with an externally available article, either published in a journal or made available in a preprint repository such as arXiv or SSRN. Figure 1 gives an example of such a webpage for a paper published in 2013.

A compendia page links to the webpage where the publication is available, or if the publication is open access, ResearchCompendia will host a copy and users can download it directly. In addition, author-provided code and data are available
for download by clicking the appropriate button, labeled “code” or “data.” These two terms are left somewhat ambiguous deliberately, to permit the author leeway in deciding the appropriate software steps and data to include for the computational findings to be reproduced on a different system by an independent researcher. In some contexts, data are a starting point and analysis is carried out on these data. In other cases the data may be generated by the scripts themselves (and so data dissemination may not be necessary for replication purposes). If datasets or code are small enough for ResearchCompendia to host locally, it will store a copy. ResearchCompendia links to larger datasets or code hosted in external repositories. We also wish to improve the persistence of these digital scholarly objects by respecting the LOCKSS principle—Lots of Copies Keeps Stuff Safe (see www.lockss.org/about/what-is-lockss). For these reasons, we deposit data and code in external repositories whenever appropriate. To encourage proper citation, when a user clicks to download code or data, a suggested citation appears. See Figure 2 for an example.

Notice also that the compendium page provides descriptions of the code and data itself, not merely the abstract from the research article. Contributing programmers or other project members can be associated with the research objects as authors—there’s no restriction to journal article authors alone. This permits flexibility in recognizing different types of contributions to research.

ResearchCompendia offers a larger infrastructure to support the creation and use of compendia pages. Prior to creating a compendium page, a user will create an account, and follow the steps to create a page. ResearchCompendia can fetch article DOI information to streamline the compendium page creation process, and DOIs are currently assigned by ResearchCompendia to all citable objects, such as code and data, in our labs pages at http://labs.researchcompendia.org/compendia/. Finally, it’s also possible to leave comments on the compendium page, for example, notifying page owners of a bug in their code, or perhaps authors would like to notify users of an updated version of their code. Because of the motivating factor of reproducibility, ResearchCompendia will continue to provide the versions of the code and data that replicate results, even if errors are found within. Users will be alerted to these errors on the compendium page and the new versions will be provided in addition to the originals associated with the publication.

A high-level illustration of the information stored by ResearchCompendia appears in Figure 3. For each article, we keep the following:

- the user that created the compendium page;
- the contributors, such as article authors, programmers, or data curators;
- the contributors, such as article authors, programmers, or data curators;
- DOI information for citable objects such as the article, code, and data;
- the abstract describing the code and data;
- the abstract from the article;
- bibliographic information;
- licensing information for all digital scholarly objects;
- any related URLs, such as for the published paper;
- taxonomic information; and
- code, data, and other associated files such as the article.

Each compendia page is created by an account holder and has contributors, and some information about each contributor is stored in a separate table documenting their role in the project. Information is also gathered on account holders (if they choose to provide it), including their name, biography, affiliation, country, as well as information on associated URLs, permissions for access to various parts of ResearchCompendia, their email address, and their username.

We made a philosophical and practical decision to develop the ResearchCompendia code base as an open source and collaborative project. The code is hosted by GitHub and available at https://github.com/researchcompendia/researchcompendia. We provide the source code in hope for collaboration, but also to permit others to stand up their own ResearchCompendia websites. We permissively license the code under the MITlicense and data under a Creative Commons license (CC0), following the licensing guidelines from previous work.16,17 Research code that is uploaded to ResearchCompendia has the MIT License as a default and with a different license upon request. Part of the rationale for default licensing is to simplify the upload process, and part is to maximize the ability of researchers to combine various codes into a new project.

Reliability, Executability, and Verification

As cyberinfrastructure becomes an increasingly important part of the scientific research pipeline, it can serve to encourage best practices in the scientific discovery process. As we described previously, ResearchCompendia seeks to encourage good practices such as the citation of code and data when reused. A second goal for ResearchCompendia is to certify or validate published findings, bringing our efforts more in line with the vision of Gentleman and Temple Lang described at the outset of this article. In our lab pages at http://labs.researchcompendia.org/compendia/, ResearchCompendia extends compendium page functionality to include execution of the research codes to verify the results in the publication, using the data provided by the author. On these “executable compendium pages,” users have the additional option of running the code in the cloud directly through the compendium page. The creation of these pages requires ResearchCompendia to check the code submitted by the researchers (that is, compatibility, CPU requirements, computing time, verifying constraints on input parameters, and so on) and ensure that it replicates the figures from the original article. (See http://labs.researchcompendia.org/compendia for pilot executable compendium pages that permit the user to verify the computational findings.) Running the author’s code can be very quick or take significant time. After successfully executing the code, we generally offer access to cached results, but users are free to run the code on their own independent platforms as well. In this sense, ResearchCompendia acts as a certifier of research results. See Figure 4 for an example executable compendium page. In addition to the code and data download buttons, notice the “verify” button and documentation of verification runs.

For the creation of executable compendium pages, we’ve developed technology using lightweight virtual machines, commonly called containers, to create a local environment that executes the code. This is done for security reasons (to keep these code executions and any user input separate from the rest of ResearchCompendia) and to ensure that the necessary software and libraries are installed so that the codes will execute.

Figure 5 augments Figure 3 by showing the additional information collected for executable compendium pages, including the time of the verification run, whether or not it was successful, any standard output or error, parameter information, and archives output files and information.
Prioritizing the Complete Research Pipeline

Factors that are accelerating the implementation of reproducible research include the open source software movement (permitting sharing, reusing, and using good practices) and the adoption of cloud computing in scientific research (permitting the launch of complex jobs on thousands of cores with a single click, and providing common environments for code execution). The computations used in research today can be very complex, involving sometimes immense amounts of code to merge and clean datasets, and implement ambitious algorithms, but complexity also arises in research workflows combining various codes implementing these data processing or algorithmic steps. To illustrate ongoing trends, Figure 6 gives the staggering increase in lines of code submitted to the CALGO repository for the ACM Transactions on Mathematical Software (TOMS) from 1960 through 2013. The total number of lines of code submitted increases steadily on a log scale, from 875 lines in 1960 to nearly 5 million in 2012 (the proportion of total publications with associated code remained roughly constant at approximately 1/3, with standard error of about .12, and the journal consistently publishing around 35 articles each year).

The evolution of computing infrastructure will include the documentation of research pipelines with workflow tools, as researchers chain together complex software from different sources, scripts, and codes in different languages. Researchers will “build on the shoulders of giants” through repurposing code and data from other authors. Even if good software practices are followed and each algorithm is well documented, computing infrastructure and communication standards will need to incorporate an additional notion of a research workflow that documents published computational findings so that they can be reproduced.

Discussion and Next Steps for ResearchCompendia

Future development of ResearchCompendia falls into two categories—short term and long term. In the short term, we’d like to extend the executability demonstrated in the labs to the entire website. We also plan to evolve the compendium webpage to interact with the user—to give users an opportunity to run the code with different inputs, such as alternative parameter settings or updated datasets (or other datasets uploaded to ResearchCompendia, perhaps those associated with different compendium pages).

We believe this will help research both for validation purposes and as a tool of stability. The
concept of validation is well known in the scientific computing and simulation-oriented disciplines. It refers to “[t]he process of determining the accuracy with which a model can predict observed physical events (or the important features of a physical reality).”18 In other words, “[t]he process of determining the degree to which a model (and its associated data) is an accurate representation of the real world from the perspective of the intended uses of the model.”19 Validation is often carried out via comparison with independently generated results, or other sources of real-world data not used in the current model. ResearchCompendia offers an alternative avenue for validation. As it collects data from studies, these datasets themselves may be used to validate previous results. For example, we could imagine findings based on very small sample sizes (perhaps considered large at the time) being validated over much larger databases as they’re submitted to ResearchCompendia.

ResearchCompendia can also facilitate the understanding of stability in scientific findings—the notion that the variability of estimators is bounded when the underlying data is perturbed in well-understood ways.20 Code that runs in a system that’s frequently augmented with new data sources could give an opportunity to test the stability of models using different datasets not included in the original study. ResearchCompendia develops the ability to validate findings on much larger datasets as it collects models and data from different articles investigating the same related scientific hypotheses. For example, code implementing a model on a small dataset could be executed on much larger datasets contributed to ResearchCompendia from other related studies.21

As we continue to execute code/data combinations, we’ll gather the information about what makes code easy or difficult for us to run, and use that information to create a set of guidelines for code submission to ResearchCompendia. ResearchCompendia is a testbed for the implementation of the three ideas regarding scientific CI presented here, and we can monitor usage to discover successes and failures in this approach as researchers increasing share myriad types of data and code.

In the longer term, we expect scientific publishing to move away from the publication as a standalone object in .pdf format. The published findings are of course fixed at the time of publication, but so should be the versions of the code and data that generated those findings. Some barriers to overcome include: versioning of code and data to maintain reproducibility of published results; persistently connecting article, data, and code (including workflow information) in the Gentleman and Temple Lang spirit of a research compendium; and maximizing interoperability of code and data for reuse while minimizing the burden on the computational researcher, and safeguarding privacy and confidentiality concerns in the data.

Interoperability includes reuse of code on different projects and on different systems. At this point, ResearchCompendia’s initial execution of the code is carried out manually when creating the executable compendium page, but using the container approach described above will permit the service to scale. We anticipate that ResearchCompendia will be able to directly accept containers from researchers that house the fully functional software and data, thereby automating availability as an executable compendium page. Eventually, ResearchCompendia will supply the script or image for these containers to authors, who will then ensure that their results reproduce computationally in that sharable environment.

![Figure 6. The increase in the number of lines of code submitted to the CALGO repository associated with the ACM Transactions on Mathematical Software (TOMS) journal, 1960–2013. The best-fit line shows the dramatic increase in code complexity and size, represented on a log scale. In recent years, the variability in the number of lines of code has increased. The journal published about 35 articles per year consistently throughout this time period, and about a third of the articles submitted code to the CALGO repository. These data also appear in Figure 1 of Victoria Stodden’s article, “Reproducing Statistical Results,” in the journal Annual Review of Statistics and Its Application, forthcoming in January 2015.](image-url)
We plan to continue open source development of ResearchCompendia. Not only is this philosophically consistent with our larger goals of achieving greater transparency in scientific research and communication, but it permits a community to grow around these efforts and contribute back to the project. It also allows others to duplicate and extend the infrastructure in other contexts. We hope that such downstream use contributes to discoverability of code and data. As mentioned previously, ResearchCompendia hosts the data, code, and articles it makes available (barring size and/or legal barriers to doing so), and points to external copies when it doesn’t. ResearchCompendia isn’t meant to replace other forms of dissemination, such as supplemental material sections in journals or dedicated data repositories, for example, but to augment their efforts by providing a centralized, discoverable, and persistent way of linking the digital objects that comprise computational scholarship. To our knowledge there’s no other service that focuses uniquely on reproducibility, and hence, the article, data, and code—the Research Compendium—as the appropriate unit of scholarly communication for computational science.

Acknowledgments
This research was supported by Alfred P. Sloan Foundation award number PG004545 “Facilitating Transparency in Scientific Publishing.” The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

References

Victoria Stodden is an associate professor in the Graduate School of Library and Information Science at the University of Illinois at Urbana-Champaign. Her current research interests include statistical methods for large data applications and reproducible computational research. Stodden has a PhD in statistics from Stanford University and an MLS from Stanford Law School. Contact her at victoria@stodden.net.

Sheila Miguez is a research scientist in the Department of Statistics at Columbia University. She enjoys helping people at Python workshops and Python office hours at her local hackerspace. Miguez has a BS in computer science and psychology from the University of Maryland College Park. Contact her at shekay@gmail.com.

Jennifer Seiler is a postdoctoral researcher in the Department of Statistics at Columbia University. Her current research interests include open science and astrophysics. Seiler has a PhD in physics from the Max Planck Institute for Gravitational Physics. Contact her at jenn.seiler@gmail.com.

Selected articles and columns from IEEE Computer Society publications are also available for free at http://ComputingNow.computer.org.
Reproducible Research as a Community Effort: Lessons from the Madagascar Project

Sergey Fomel | University of Texas at Austin

Instead of computational reproducibility being the responsibility of an individual author, it should become the responsibility of open source scientific-software communities. A dedicated community effort can keep a body of computational research alive by actively maintaining its reproducibility. The Madagascar open source software project offers an example of such a community.
development environment and the complete set of instructions which generated the figures.

In Claerbout’s own words,\textsuperscript{15} It is a big chore for one researcher to reproduce the analysis and computational results of another [...] I discovered that this problem has a simple technological solution: illustrations (figures) in a technical document are made by programs and command scripts that along with required data should be linked to the document itself [...] This is hardly any extra work for the author, but it makes the document much more valuable to readers who possess the document in electronic form because they are able to track down the computations that lead to the illustrations.

The initial efforts by Claerbout and his students to create a reproducible research framework were only partially successful.\textsuperscript{12} The first implementation employed the Unix make utility\textsuperscript{16} to automate different tasks. This framework turned out to be somewhat over-complicated. It consisted of multiple make rules, Shell scripts, Perl scripts, LaTeX macros, and so on, and was hardly portable outside of the original research group.\textsuperscript{17} Even more disturbingly, although the initial reproducibility of each paper was carefully tested,\textsuperscript{18} it was never checked again. As a result, reproducibility of archived research papers was discovered to deteriorate quickly after their initial publication.

Based on this experience, I started the Madagascar project in 2003 with the explicit goal of building a convenient and portable framework for reproducible research. In June 2006, Madagascar was publicly released at a workshop in Vienna and became an open source project, with a repository hosted by SourceForge. Starting with a handful of initial users and developers, the Madagascar community has grown to dozens of active contributors and thousands of installations. As of December 2014, more than 80 people have contributed to the project.

What Is Madagascar?
The collection of open source codes in the Madagascar package exists on several levels.\textsuperscript{11} On the top level, there’s a growing compendium of (currently more than 200) published research papers and book chapters, which are included in the package with their LaTeX sources. The next level contains scripts needed to reproduce figures (results of computational experiments) described in the papers. We adopt SCons, a convenient Python-based tool, for managing figure-generating workflows.\textsuperscript{19} SCons is an enhanced modern replacement for the Unix make.\textsuperscript{20} Its configuration files (SConstruct scripts) are written in Python. The lowest level contains “Lego blocks”— programs used in the scripts for low-level number-crunching computations. These programs are combined together using the Unix philosophy\textsuperscript{21} and can be written in almost any language. While the majority of them are written in C and C++, the core Madagascar library also provides interfaces for Java, Fortran-77, Fortran-90, Python, and Matlab. Naturally, the source codes for these programs are also included in the package.

The three levels described above (programs, workflow scripts, and papers) correspond to three common activities of a computational scientist: implementing numerical algorithms, conducting computational experiments, and publishing the results of these experiments.

In addition to the source code included in the package, the Madagascar project maintains a repository for publicly available research data (used as input to some of the documented experiments) and a separate version-controlled repository for generated figures (used for testing reproducibility, as described below).

Madagascar is designed to provide general-purpose tools for analysis of large-scale multidimensional data. Because of its origin and current support in the exploration geophysics community, most of the current examples and publications come from geophysical research and focus on seismic imaging and seismic data processing. Many of the low-level programs are tailored for these applications as well.

Lessons from the Madagascar Project
The lessons about reproducibility maintenance learned in more than 10 years of Madagascar development and the preceding 10 years of the reproducible research development at SEP can be summarized as follows.

Reproducibility in itself isn’t the goal. Direct replicability of computational experiments is difficult to achieve, especially when computations involve floating-point numbers and round-off errors. There are many other cases where exact replicability is impossible or even undesirable.\textsuperscript{22} In cases like that, it’s important to remember that the true goal of reproducible research isn’t achieving full computational replicability in itself but in making it easier for the reader to verify and extend published research.
As an example, consider a large-scale computational experiment that uses proprietary data or expensive hardware. Scientists outside of the author’s research lab may have a hard time trying to reproduce a computation like that. However, this shouldn’t be used as an excuse for hiding computational details, such as software codes and experimental parameters. A lack of unambiguous documentation in the form of open source code would make it significantly harder for others to verify and extend the work.23

The main beneficiary is the author. From its description, reproducible research sounds like a service that the author performs for the scientific community, possibly at a sacrifice of personal research productivity or academic competitiveness. In fact, our experience shows the opposite: investing in reproducibility is an excellent way to boost one’s research productivity. In the words of Jon Claerbout,24

We found that although the effort seems to be directed to helping other people stand up on your shoulders, the principal beneficiary is generally the author herself. This is because time turns each one of us into another person, and by making effort to communicate with strangers, we help ourselves to communicate with our future selves.

In addition to helping authors reproduce their own work, sharing research with others also opens up many possibilities for research collaboration that would be closed otherwise. Guiding graduate students, postdocs, and other collaborators to a research project is never easier than when they can start by reproducing your previous work.

Each computation is a test. Test-driven development (TDD) is a well-established agile technique in software engineering.25 Scientific computing doesn’t always follow the patterns required by TDD, because scientific computations are often exploratory, when the result of a computational experiment isn’t known beforehand. Nevertheless, we can safely say that any computation becomes a test after the experiment is completed, because when the same computation is repeated in the future, we should expect the results to reproduce.

When comparing different numerical algorithms to one another in terms of their efficiency or accuracy, it’s often difficult to make a fair experimental comparison. Reproducible research makes such comparisons easy and straightforward. In fact, any documented experiment implemented in a reproducible research framework, such as Madagascar, turns immediately into both a regression test (for verifying its reproducibility in the future) and a computational benchmark (for trying to achieve similar results with more efficiency or accuracy).

Reproducibility requires maintenance, maintenance requires an open community. Without continuous maintenance, the reproducibility of computational results tends to deteriorate surprisingly fast. Inevitable changes in the software environment (libraries, compilers, operating systems, and so on) cause reproducibility to break, often fairly soon after the initial publication. Without dedicated maintenance, published computational experiments lose their reproducibility and, as a result, they also lose their credibility and usefulness.

Is it feasible to require each author to actively maintain the reproducibility of published results? Probably not. Researchers can change jobs and research interests. Graduate students and postdocs can move from academic environments to the industry. The published research that they leave behind should be picked up by a scientific community. Through continuous testing, the community can make sure that all relevant results stay reproducible and that the associated scientific knowledge continues to grow.

Building an open community is a difficult task, which requires a dedicated effort and considerable patience.26 Techniques from general open source software communities27 don’t always apply to scientists, who have higher concerns about getting credit and professional recognition. To facilitate personal recognition, part of the Madagascar source tree is organized by contributor name. The paper directory is ordered by research organization. Despite considerable progress, keeping a community spirit in the Madagascar project is an ongoing struggle.

Reproducibility Maintenance in Madagascar
As of this writing, Madagascar’s growing collection contains more than 800 reproducible scripts (GConscript files) generating more than 8,000 figures. A Madagascar reproducibility test amounts to rerunning the script and comparing the newly computed figure with the one stored in the repository. The comparison tool that we use (sfvplotdiff by Joe Dellinger) allows for some floating-point tolerance and tries to avoid flagging visually indistinguishable results that differ only because of different precision of computation on different computers.
Let's look at the following example SConstruct file to illustrate how our testing environment works.

```python
from rsf.proj import *

# Download data
Fetch('wz.25.H','wz')

# Convert format and window
Flow('data','wz.25.H',
    dd form=native |
    window min2=-2 max2=2 |
    put label1=Time label2=Offset |
    unit1=s unit2=km
    ***
)

# Time-power correction
Flow('tpow','data','pow pow1=2')

# Display
Plot('data',
    'grey title="(a) Original"')
Plot('tpow',
    'grey title="(b) Corrected"')
Result('tpow','data tpow',
    'SideBySideAniso')

End()
```

The script uses five basic commands defined in rsf.proj module.

- **Fetch** specifies a rule for downloading data from the data server. In this case, the input data is a seismic record.28
- **Flow** specifies a rule for transforming a data file into another data file. In this case, the transformations involve converting formats and windowing (implemented in a pipeline of elementary "Lego blocks" programs), and a time-power correction, commonly used in seismic reflection data analysis.
- **Plot** specifies a rule for transforming a data file into an image file.
- **Result** is similar to Plot but indicates that the image file is a computational result to be included in the publication.
- **End** specifies the end of the workflow and sets rules for default targets. Running the `scons` command on the command line generates the default target, which is the image file specified by `Result`:

```bash
bash$ scons -Q
retrieve(['wz.25.H'], [])
< wz.25.H /usr/bin/sfdd form=native | \
/usr/bin/sfwindow min2=-2 max2=2 | \n/usr/bin/sfput label1=Time label2=Offset \nunit1=s unit2=km > data.rsf
< data.rsf /usr/bin/sfpow pow1=2 > tpow.rsf
< data.rsf /usr/bin/sfgrey \ntitle="(a) Original" > data.vpl
< tpow.rsf /usr/bin/sfgrey \ntitle="(b) Corrected" > tpow.vpl
/usr/bin/vppen yscale=2 vpstyle=n \ngridnum=2.1 data.vpl tpow.vpl > Fig/tpow.vpl
```

The image is shown in Figure 1.

Suppose that we modify one of the parameters in the SConstruct file—for example, changing `pow1=2` to `pow1=1`. Running `scons` again will regenerate only those targets that are affected by the change:

```bash
bash$ scons -Q
scons: 'data.rsf' is up to date.
< data.rsf /usr/bin/sfpow pow1=1 > tpow.rsf
scons: 'data.vpl' is up to date.
< tpow.rsf /usr/bin/sfgrey \ntitle="(b) Corrected" > tpow.vpl
/usr/bin/vppen yscale=2 vpstyle=n \ngridnum=2.1 data.vpl tpow.vpl > Fig/tpow.vpl
```

This behavior is the main reason why SCons scripts are more convenient than straight Shell or Python scripts.

![Figure 1. Image from the example workflow shows a seismic record before and after a time-power correction.](image-url)
Once we're done experimenting and are satisfied with the final result, we can run `scons lock` (Madagascar extension), which installs all results in a safe directory.

```
bash$ scons -Q lock
 Install file: "Fig/tpow.vpl" as "/usr/share/figs/book/paper/project/tpow.vpl"
```

After that, all intermediate files generated by the experiment can be removed with `scons -c`.

When a figure such as Figure 1 appears in an online publication on the Madagascar website, it's followed by a link to the `SConstruct` script that produced it, which in turn links to programs and data used in the script (Figure 2). This documentation is generated automatically using customized `LaTeX2HTML` and implements our approach to reproducible research.

Suppose that, at a later date, we want to check whether the result is still reproducible. We can do that by running `scons test` (a Madagascar extension), which regenerates the figure and compares it with the stored one.

```
bash$ scons -Q test
test([".test_tpow"], ["Fig/tpow.vpl"])
Comparing /usr/share/figs/book/paper/project/tpow/tpow.vpl and Fig/tpow.vpl
```

If, for some reason, the reproducibility is broken, `scons test` will report an error. We can then investigate the change (for example, by visually flipping between the two images with `scons tpow.flip`) and try to fix the error.

We have set up dedicated servers that run hundreds of automated reproducibility tests every time somebody commits a change to the Madagascar repository, in the fashion of continuous integration technology. Running tests with SCons makes sure that no effort is wasted if the test is unaffected by the change.

What happens when reproducibility of a particular paper breaks? Fixing the problem becomes the community's responsibility. If nobody in the community cares enough to fix it, the paper is removed from the compendium. Several papers like that have been declared “dead” and moved to the special Grave directory. While other papers in the Madagascar collection continue reproducing, they will be kept “alive” and serve as a possible foundation for new computational research.

Using the experience of the Madagascar open source software project, I discussed the issue of maintaining computational reproducibility. Our experience shows that long-term maintenance of reproducible results is possible when it becomes the responsibility of an open source scientific community rather than the
responsibility of the original author. Reproducible papers included in the Madagascar collection have been kept “alive” (in some instances for more than 20 years) by maintaining their reproducibility through repeated testing and continuous integration.

The core design of the Madagascar package is fairly general. So far, our user community has been mostly limited to exploration geophysicists. In the future, it may reach out to other fields, where there is a need for analyzing large-scale multidimensional data using reproducible workflows. I also hope that the lessons of reproducible research learned by us during the last 20 years will help other scientists in creating their own open source software communities.

Acknowledgments

No dedicated funding has been provided to the Madagascar project so far. The work has been supported through volunteer effort and through funding from various research projects. Dozens of different people have contributed to the Madagascar project (see http://sourceforge.net/p/rsl/library was particularly valuable, because it enables our Biondo Biondi, and Bob Clapp. Joe Dellinger's Vplot Jon Claerbout, Dave Hale, Stew Levin, Rick Ottolini, Joe most important contributions coming from Rob Clayton, and researchers contributed to SEPlib, with some of the maintained by Bob Clapp. Generations of SEP students all-time contributors. The Madagascar project started by implementing the functionality of SEPlib, a package from the Stanford Exploration Project (SEP), currently reimplemented the functionality of SEPlib, a package from the Stanford Exploration Project (SEP), currently maintained by Bob Clapp. Generations of SEP students and researchers contributed to SEPlib, with the most important contributions coming from Rob Clayton, Jon Claerbout, Dave Hale, Stew Levin, Rick Ottolini, Joe Dellinger, Steve Cole, Dave Nichols, Martin Karrenbach, Biondo Biondi, and Bob Clapp. Joe Dellinger's Vplot library was particularly valuable, because it enables our style of reproducibility testing.

References


Sergey Fomel is a professor at the Jackson School of Geosciences, the University of Texas at Austin. His research interests are exploration and computational geophysics, with a focus on seismic imaging and geophysical data analysis. Fomel has a PhD in geophysics from Stanford University. Contact him at sergey.fomel@beg.utexas.edu.

Selected articles and columns from IEEE Computer Society publications are also available for free at http://ComputingNow.computer.org.
In his new book, David Alan Grier tells the stories that technical papers omit. Moving beyond the stereotypes of nerds and social misfits, *The Company We Keep* explores the community of people who build, use, and govern modern computing technology. The essays are both insightful and intimate, showing the impact of technology and the human character behind it.

978-0-7695-4764-0 • September 2012
Paperback • 280 pages • $19.95
An IEEE Computer Society Press Publication

New Title From IEEE CS Press

**The Company We Keep**

David Alan Grier

To Order

Order Online
Enter code
DZWE5FVE
for 15% off!*

Also available on
http://amazon.com

*Note: discount code only valid on createspace.com

Save 15%
Crowdsourcing Scientific Software Documentation: A Case Study of the NumPy Documentation Project

Aleksandra Pawlik | University of Manchester
Judith Segal, Helen Sharp, and Marian Petre | The Open University

Documentation crowdsourcing may be considered a way to address the issue of poor documentation of scientific software packages. Using the NumPy documentation project as a case study, the authors discuss how to leverage the knowledge about software that resides within the developer and user community. The conclusion suggests preliminary guidelines for those thinking about documentation crowdsourcing.

Research in many, if not most, scientific disciplines requires scientists to not only use but also to develop software to advance their research. Some software applications used in science are highly specialized open source packages developed and maintained by scientists themselves—for example, cross-disciplinary NumPy and SciPy (see www.scipy.org), rOpenSci (see http://ropensci.org), or the more discipline-specific Madagascar (see www.ahay.org). Ideally, documentation should be an inseparable part of software development and maintenance. Without documentation, software becomes difficult to use and to maintain. However, despite its importance, scientific software documentation is often incomplete or absent.¹,²

The issue of poor documentation is not exclusive to scientific software. Professional software developers aren’t fond of writing documentation,³ and scientists seem to be no different.⁴ The reason behind a lack of documentation is usually simple: priorities. Writing documentation is more boring and less satisfying than coding. What adds to the issue is that, while scientist-developers might enjoy software development per se, this activity remains secondary to their main goal, which is advancing their research.⁵ Documentation has even lower priority. And resources follow priorities. While in commercially developed scientific software there’s likely to be a team and budget dedicated to writing documentation, in open source projects for scientific software most likely there’s neither.

How does software find its way into use despite poor documentation? Communities develop around a shared need and the software tools that address it, and the knowledge about the software resides within the community. The users and developers introduce each other to the software, and they share advice on fora, mailing lists, and other means/channels of online communication, or in informal conversations over coffee.⁶ Scientists put a considerable trust in their peers’ recommendations and opinions.⁶

It seems that there’s potential within the communities that develop around scientific software to compensate for documentation shortcomings. The question is: how can we harness this potential and channel the informal, unstructured, and often volatile exchange of knowledge about the software into standardized and reusable documentation? To answer this question, we conducted a case study of the NumPy documentation project in which the community members (consisting primarily of users) documented the software they used. We explored how the community organized and handled the process, what benefits it brought to them, and what challenges they faced.
Background on NumPy
The NumPy library is a “fundamental library needed for scientific computing with Python” (see www.scipy.org/about.html) providing efficient implementation of multidimensional arrays together with a wide range of mathematical functions to operate on them. NumPy interfaces with well-developed, stable, and widely used linear algebra packages: Basic Linear Algebra Subprograms (BLAS) and Linear Algebra Package (LAPACK), as well as fast-Fourier Transformation Package (FFTPACK). Hence, both the use and development of NumPy require advanced knowledge and understanding of numerical methods and their applications.

NumPy was preceded by three other projects of related scope. The first one was Matrix Object Package for Python, developed by Jim Fulton starting in 1994. The second one was Numeric (or Numerical Python), whose initial code base was mostly written by Jim Hugunin with contributions from the community known as the Python Matrix-SIG (see https://mail.python.org/pipermail/matrix-sig). The third one was Numarray (a specialized fork of Numerical Python), whose development was led by Perry Greenfield, Rick White, and Todd Miller. In 2005, Travis Oliphant started the unification work involving together Numeric and Numarray. For a short period, this new package was called scipy.core, until it was eventually renamed NumPy, to distinguish it both from the previous two projects and from SciPy. It should be noted that the name SciPy now also refers to a "Python-based ecosystem of open-source software for mathematics, science, and engineering" (see www.scipy.org) and to the community in which it's used. SciPy also denotes the community's conferences, workshops, and meetings.

The growing NumPy code base was scarcely documented. Poor documentation made the software extremely difficult to use, especially for new users. At the same time, the scientist-developers didn’t have time to write documentation. They were more interested in developing the code itself rather than in its documentation. Several members of the community came up with the idea that the users could write documentation. It’s not clear who exactly first suggested crowdsourcing documentation. They were more interested in documenting the library rather than in writing the documentation. As one of the people interviewed for our study put it, “it just happened organically.”

Python supports the docstrings, specifically-formatted string literals, similar to code comments, which can be used for displaying help information about a particular piece of code. It’s a common convention to use docstrings as documentation reference in Python projects. Many docstrings for methods in the NumPy library were incomplete or missing and so writing them became the main task for the documentation project. In our study, we focused on the process of editing existing and writing new docstrings.

Data Collection and Analysis
We collected the data for this case study from four different sources. The first source was the publicly available archives of three mailing lists: numpy-discussion, scipy-dev, and scipy-user. Using the mailing list search engine we filtered out 2,230 posts between the beginning of the archives (2000 and 2001, depending on the list) and April 2012, including words stemming from “doc*.” We then manually screened the filtered posts to select the messages that discussed the documentation process. This eventually left us with 477 posts, which we then analyzed in detail.

The second source of data was semistructured interviews with nine members of the SciPy community, all of them key stakeholders involved in the documentation project. The interviews lasted between 30 and 90 minutes; most of them took just under one hour.

The third source of data was progress reports on the SciPy Documentation Project published in the SciPy Conference Proceedings,7,8 a technical overview paper published in the SciPy Conference Proceedings,9 and the project website (see docs.scipy.org).

The fourth source of data was logs from the server that hosted the infrastructure developed for the purpose of crowdsourcing documentation. The logs from the server included quantitative information such as the number of contributor accounts registered on the system; total number of edited, proofed, and revised posts in different time periods; and so on.

Observations
The data analysis revealed a number of aspects related to the organization, challenges, and benefits of documentation crowdsourcing in the scientific software context. These findings are not only a collection of observations about documentation crowdsourcing, but should be informative for those considering or already implementing such a process. The outcomes of this case study help to inform, guide, and provide insights into potential caveats and benefits that might not seem obvious or expected beforehand.

How to Organize Documentation Crowdsourcing
Organizing NumPy documentation crowdsourcing included dealing with both technical and human aspects. The former was related to developing and maintaining the infrastructure, together
with designing and implementing standards for writing documents. The latter included finding a project champion who both had sufficient technical knowledge and was able to engage the community and manage the contributors’ efforts. There was also a financial aspect, which had its influence, in particular at the beginning of the project.

**Infrastructure.** The key considerations for setting up the documentation crowdsourcing process were to allow a wide community to access the code base, while ensuring that the source code remained safe from any potential damaging changes. The NumPy source code was kept under version control (using Apache Subversion) at the time. The code was freely available for anyone to use, but only a selected group of core developers had write-access to the repository. This meant that any changes in the code (including docstrings) by those who didn’t have write-access had to be actively reviewed by the developers and then applied as patches. This obviously was a barrier for crowdsourcing documentation, as it would only add more work for the developers who were already at the limit of their capacity.

On the other hand, giving write-access to the repository to anyone who wanted to write documentation was too risky. The risk of purposely malicious injections to the source code was estimated to be rather low. However, increasing the number of people who had direct write-access would also increase the probability of introducing accidental errors. In addition, the contributors who could write documentation would be limited to those who were able to use version control. Therefore, the main goal was to develop an infrastructure for writing documentation that kept the code safe, but was easy to use by a potentially large number of contributors.

The infrastructure developed for the documentation project was eventually inspired by two tools, which were developed independently: one was a wiki, and the second was LiveDocs. The advantage of the wiki was that everyone could register and edit its content. The disadvantage was that it wasn’t connected with the code, and hence, changes made on the wiki didn’t have any effect on the docstrings. In addition, the wiki wasn’t automatically updated with any changes to the source code (including the docstrings). LiveDocs, setup by the original creator of NumPy, ran on a server and provided hierarchically-indexed documentation using docstrings. The advantage of this system was that it showed the most up-to-date docstrings. The main disadvantage was that it was read-only.

The solution was to combine wiki and LiveDocs functionalities, allowing as many contributors as possible to edit docstrings and at the same time providing full control over the write-access to the code repository. This documentation system was developed by one of the core NumPy developers with help from a few others. The documentation editor allowed two-way documentation updating: the changes made by the contributors in the editor were turned into a patch and applied to the code back in the Subversion repository, and the documentation in the editor itself was updated with any changes.

The documentation editor supported the workflow, which assumed that a docstring might have more than one status: needs editing, being written, needs review, needs work (reviewed), reviewed (needs proof), proofed, and unimportant. The docstrings that went through a number of iterative changes and that were classified as correct made it into a patch. This removed the burden from the developers of checking the docstrings before applying them to the code.

**Stylistic guidelines.** Even though some bits of NumPy code were documented using docstrings, there was no comprehensive or recommended documentation standard or style. However, for crowdsourcing, such guidelines were essential and needed to be clearly laid out and explained to the contributors. The discussion about the standards and guidelines for contributors took place mainly on the mailing list. All subscribers were openly encouraged to express their opinions.

Initially, one of the NumPy core developers proposed that the guidelines follow the style in which he had written docstrings for several years. According to him the docstrings should include: function inputs, outputs, algorithm, authors (of the given piece of code), examples, tables, references, and additional notes. The standard was then in a plain text file named HOW_TO_DOCUMENT.txt that was made publicly available online (the updated version is available at https://github.com/numpy/numpy/blob/master/doc/HOW_TO_DOCUMENT.rst.txt).

The standards were also reflected in the infrastructure of the documentation system. First, following a suggestion of one of the community members, a professional technical writer, all of the documentation that needed writing was split into small bits. As the technical writer argued on the mailing list: “A short document has less opportunity
to be poorly structured than a long one. A collection of short documents can be assembled into a helpful organization as it evolves." The documentation system allows the contributors to edit a small section of documentation. The sections are also divided according to the standards proposed earlier: parameters, returned values, examples, and so on. It’s interesting that, according to one of the developers interviewed, this standard was then picked up in other scientific Python packages.

**Funding.** One factor that sped up the start of the documentation project was the funding, which one of the NumPy users secured and was used for employing a full-time documentation writer, who would also lead and coordinate the efforts of other contributors. The user, a professor at one of the US universities that provided the funding, had wanted students to use NumPy during one of his courses. Due to poor documentation, the students struggled and couldn't complete assignments. Frustrated with this situation, the professor managed to allocate some of the funding to employ one of the NumPy developers for a year to write documentation. It quickly became clear that the documentation task was much too large for one person to complete. The documentation writer’s role then evolved to encompass managing and leading the work done by the others.

Two people acted as writer and leader and were employed at different times, with about a two-month gap between them, for a total time of two and a half years. The money came from a grant that didn't specifically mention documentation writing. The flexibility of the grant made it possible to spend some of the funds on the documentation. However, it also meant that the funding was limited and that the documentation project needed a sustainability model.

**Project leaders.** Both project leaders were employed full-time to write and coordinate the documentation. Both were experienced developers. However, while the first was one of the core NumPy developers, the second didn’t contribute code to this particular library. The in-depth knowledge of implementation details of NumPy was useful but not essential for documenting the code.

The task of documenting NumPy was too big for one person. Writing documentation every day all day was also described by one of the leaders as incredibly mundane and simply boring at some stages. Crowdsourcing the documentation lifted some of that burden from the full-time documentation writers but also added other responsibilities. The project leaders monitored and reported back to the community on the progress with documentation writing; motivated the contributors to carry on with their work; and proposed changes and improvements for the process or the infrastructure.

**Engaging the community.** Engaging many members of the user community was a high priority and was essential to the project’s success. Calls for contributors were sent to the mailing list regularly, encouraging everyone to write documentation. All community members were welcomed to contribute documentation: from advanced users who might have been developers themselves, to beginners who only recently started using the library and had limited experience as developers. In one of the early emails, one of the developers explicitly encouraged a user who started writing documentation:

> Being a ‘newbie’ is maybe the best time to write documentation—while you still know what new users need to know, and how to explain it to them simply. Don’t feel shy to contribute.

The community members were not only encouraged to write documentation but also to provide feedback on the whole project. The mailing lists were the place to discuss the infrastructure as well as the standards and formats. The emphasis on community spirit and a shared ownership of the documentation was evident from the beginning of the project.

The custom-made infrastructure for collaborative documentation writing was a low-level entry point. It was easy to set up an account and then to write bits of documentation. The registration (setting up an account) didn't automatically give permission to edit content; first, the registered users had to send an email to the developers’ mailing list providing their login, and they would automatically be given permission by one of the documentation system administrators. There was no background check, and everyone who registered was given write-access to the docstrings in the system. The only reason behind the extra step was to prevent potential spammers from setting up accounts. Figure 1 shows the cumulative number of accounts registered in the system over the period of the case study. It should be noted that these numbers don’t equal the number of actual contributors, as some accounts were never activated, and some
people registered and activated their accounts but never actually contributed to the project.

When interviewed, the professor who secured the funding for employing the full-time documentation writer admitted that he was rather skeptical about whether the community would engage in the project. He actually made bets about the number of people who would sign up:

I think the threshold was about 30 people and I told them that I don’t think that we are going to have that many signups for the first 2 months. I think the bet was on ice cream. I lost that bet. Lots of people signed up. The response was very good. Now we’ve just got hundreds of people out there.

Challenges with Documentation Crowdsourcing

Inevitably, the process of documentation crowdsourcing didn’t proceed without some challenges, which were largely related to review, expectations, and momentum.

Endorsing stylistic guidelines and workflow. The discussions on the mailing list about the standards for documentation were lengthy but eventually led to a consensus and guidelines for contributors. To ensure that the contributed documentation actually met the standards, reviewers checked the docstrings. Reviewers were registered documentation-system users whose main role was to revise edited docstrings before they were approved and applied on the repository source code.

It turned out that the reviewing and proofing steps in documentation crowdsourcing were difficult to complete. Our interviewees weren’t sure why this was the case: perhaps the role and responsibilities of reviewers weren’t clearly defined, or the reviewing itself wasn’t an interesting task.

Expectations concerning the contributors. Even though the main assumption of crowdsourcing documentation was to harness the community’s potential and knowledge, there were some concerns about opening up to contributions from anyone. These concerns were raised on the mailing list during the discussions about the standards, format, and infrastructure for documentation crowdsourcing.

The main argument was that there should be some minimum set of skills and knowledge expected of the potential contributors. Some community members were questioning on the mailing list whether contributors who, for example, were unable to use some tools commonly thought of as basic for scientists (such as LaTeX) could be trusted with documenting complex libraries like NumPy. However, the prevailing view was that knowledge of word processing tools shouldn’t be the criterion for selecting contributors. The decision was made that anyone could make a contribution, and that further down the line these contributions would be checked for quality before they were included.

Keeping up the momentum. The project kicked off with considerable enthusiasm from the community. The contributors (also referred to as editors) started registering and writing documentation in the online editor. The challenge was keeping up the momentum, ensuring that the initial enthusiasm didn’t wear off.

An early idea was the Documentation Marathon (see http://wiki.scipy.org/Developer_Zone/DocMarathon2008) announced on the mailing list in May 2008. The name “Marathon” referred to coding sprints (short, few-day meetings when developers get together to write code, usually to develop a particular feature or functionality). “Marathon” also clearly indicated an ongoing effort lasting months rather than days (unlike typical coding sprints). Documentation Marathon

Figure 1. The cumulative number of registered contributor accounts in the documentation system. It should be noted that these numbers don’t equal the number of actual contributors, as some accounts were never activated, and some people registered and activated their accounts but never actually contributed to the project.
goals included: “Produce complete docstrings for all numpy functions and as much of scipy as possible,” and “Check everything into the sources by 1 August 2008 so that the Packaging Team can cut a release and have it available in time for fall 2008 classes.”

Another idea was to award all contributors who edited at least 1,000 words with a T-shirt. The names of contributors were also included in a progress report, which was published and presented at the annual SciPy conference. Both were tokens of appreciation and recognition for the work done for the community. The progress report presented a comprehensive summary of how much work contributors had achieved.

The threshold for the T-shirt award was set arbitrarily, but it turned out to be relatively good approximation of the cut-off point for the most engaged contributors. Almost 70 percent of contributors wrote or edited less than 1,000 words. The biggest contributions, over 10,000 words, were actually made by only 7 percent of editors (which included the two full-time documentation writers). Figure 2 shows the amounts of words either written or edited in the documentation system by the registered contributors.

Benefits of Documentation Crowdsourcing
It might seem that the primary benefit of documentation crowdsourcing is obvious—that is, more complete documentation. However, our case study showed that documentation crowdsourcing brought other benefits: expanding the community and setting up and promoting standards. These benefits were perceived as being at least as important as improving the documentation.

Documentation improvement. When the documentation project started around mid-2008, the total number of words in the NumPy reference pages was 8,658. By July 2012, this number reached 140,000 words. Although more doesn’t necessarily equal better, the increase in documentation didn’t happen in a random process. The documentation was written to guidelines and standards, and the new documentation, or at least a majority of it, was discussed and reviewed.

All interviewed participants of this study said that the documentation improved immensely. As evidence, they cited the growing number of NumPy users. These new users started using the library on their own, and didn’t have to rely on someone guiding them through and explaining to them how to use the code. Such guidance was almost a necessity before the documentation was crowdsourced.

However, evaluating whether documentation crowdsourcing actually improved NumPy documentation is not straightforward. It should be noted that the increased uptake of NumPy could be also related to the increasing popularity of Python in the scientific community in general. It’s possible that, as more scientists write their software in Python, NumPy is used more often. Evaluating the documentation quality wasn’t the goal of our study, but the insights into documentation crowdsourcing presented here can help develop measures for such a study.

Expanding the community. Documentation crowdsourcing helped to expand the SciPy community. The improved documentation meant that the library was easier to access, which in turn enabled more new users to use NumPy in the software they develop. However, it was not only the new users who made the addition to the community.

Writing documentation empowered those who, before the documentation project was launched, felt they were unable to contribute. They didn’t have sufficient skills and experience in software development that would allow them to write and review the code. Despite the fact that they were
SCIENTIFIC SOFTWARE COMMUNITIES

willing to help, they were unable to do so. As one of the interviewees put it:

There is a human dynamic in that which is very appealing. You want to get engaged. Obviously there are people who simply want to get things for themselves but most decent human beings when they are treated with generosity, there is a natural response to at least return a little bit. But the problem is, if in order to return in kind, you have to cross very high barriers, you may end up giving up. I think the point of creating a system like this was let’s give anyone who wants to give back a way that is important, which is helping to improve the documentation, let’s make it very easy for them to give back. Let’s lower the unnecessary barriers so that they can easily contribute back.

Interestingly, one of the interviewees said that writing documentation gave her enough confidence to start developing the source code. She was a part of the community before the documentation project, and she used the library, but she felt that her skills weren’t sufficient to develop and commit software. The work she did while documenting, and the feedback she received, encouraged her to join one of the software projects related to NumPy as a developer.

Discussion: Is Documentation Crowdsourcing a Good Fit for Scientific Software?

In scientific software, documentation typically needs to cover two things: information about the implementation and the science it represents. The software users need to have the relevant knowledge about both of these areas. The case of NumPy showed that the user community represented different levels of knowledge about the code that they documented, in a continuum from advanced user/developers, to naive users with limited knowledge of implementation. For the former, the library was a white box with mechanisms they understood well, and for the latter it was more of a black box, as what concerned them was the science underlying the methods they used. The case study demonstrated that users across the continuum were capable of making useful contributions to the project.

The documentation editor based on the wiki system lowered the entry barrier, allowing contributors to write documentation almost immediately without needing to use complex tools for access and editing. Other examples, such as public version control hosting services such as GitHub (see http://github.com), where NumPy is currently hosted, or BitBucket (see http://bitbucket.org) provide additional evidence that, by removing the overhead of learning how to make contributions, software projects can benefit from the knowledge of wider communities. Even small drive-by contributions add up to bigger improvements and could potentially lead to new collaborations.

The users who contributed to documentation were scientists, mostly working in an academic environment. Their everyday jobs required explaining and instructing others. These academic skills could potentially help them write documentation. On the other hand, academics’ writing training focuses mainly on writing academic papers in which they construct clear arguments, set up hypotheses, and draw conclusions. Software documentation is a different form of writing, as it’s regimented in terms of style, format, and content, and it doesn’t allow for individual approaches or solutions.

The main issue with software documentation writing done by scientists in academia is the lack of recognition. Academic reputations and careers are built on publications. Developing scientific software, even software that’s widely popular and useful for the community, brings scientists few career benefits apart from making them more employable outside of academia. Recognition for developing scientific software tends to be low in academia, let alone recognition for documentation writing. Documenting doesn’t add anything to an academic resume, and it’s not a skill valued in research institutions. Therefore, the rewards to contributors arise from the community to which they contribute, or from their intrinsic satisfaction with contributing.

Guidelines: What to Consider When Planning Documentation Crowdsourcing?

The outcomes of our study provide a basis for forming guidelines that are useful to take into account when considering crowdsourcing documentation.

Technical infrastructure should support both the community members who want to contribute and the community champions who manage the collaborative effort. The infrastructure might reinforce the inclusive or exclusive nature of the crowdsourcing project. The simpler the infrastructure is to use, the more people are likely to use it and get involved. On the other hand, if use of the infrastructure requires knowledge of some specific tools and skills, it might help to filter the community for the desired contributors who have the selected skill set.

Stylistic guidelines help with writing consistent documentation and enhancing its usability. The
Documentation guidelines should clearly state what to do and how to do it. For software documentation, the guidelines should focus specifically on the information that needs to be captured, the level of detail, the language, the order of information, the format, and so on. The stylistic guidelines should be readily available for anyone wanting to contribute.

Clear and sustainable workflow turns crowdsourcing from a chaotic process into a process resembling a system of gears that move together with precision. The workflow can help arrange the tasks that are involved in documentation into a sequence that prevents clashes and deadlocks. The workflow helps new contributors joining the project to more quickly begin work.

The project leader coordinates the process of crowdsourcing. The leader’s role isn’t limited to executing the workflow. The coordinator might need to use a divide and conquer approach, breaking larger and more complex goals into self-contained tasks, which then are assigned to the community members. The leader might need to set up and revise the milestones. One of the most challenging responsibilities could be keeping up the momentum, ensuring that, when the novelty and enthusiasm wears out, the collaborative effort doesn’t die down.

Motivation and recognition is much needed to keep the project running. Showing the contributors the outcomes of their effort helps them believe that what they do makes sense and has value. It’s important to make the return of investment visible. Providing regular updates on progress and giving tokens of recognition show the contributors that their investment isn’t wasted and doesn’t go unnoticed.

Crowdsourcing documentation might in some cases be a solution to the challenge of providing documentation for open-source scientific software. However, it’s a solution that requires investment in its implementation. While crowdsourcing can lessen at least some of the burden of documenting for the scientist-developers, it adds the burden of planning, organizing, and running the project for those driving the process. The approach also has its own challenges, especially in terms of maintaining community momentum long enough to establish an effective body of documentation. Seeing the effects of their work, the contributors are likely to be motivated to continue their efforts. But intrinsic motivation must be complemented with explicit recognition. Recognition for writing good documentation—not just by the community but also by academia and employers—is as much needed as recognition for developing good scientific software.

Crowdsourcing documentation can have significant advantages in addition to the generation of necessary documentation. Removing technical barriers from the process of writing documentation helps build the user community and can help users grow into developers. Future research in this area could explore what other steps could be undertaken to empower users keen to contribute back to the software they use. Writing and improving documentation is certainly not the only way to do that.}

References
Now available: A video introducing the IEEE Computer Society’s new OnlinePlus™ publication model for Transactions. Viewers will see an overview of the great features and benefits included with an OnlinePlus™ subscription and will take a tour of the user-friendly interface included on the accompanying disc.

Go to www.computer.org/onlineplus to view the video and learn all about it today.

Aleksandra Pawlik works at the University of Manchester in the Software Sustainability Institute where she leads the training activities and supports community development. Her research interests include scientific software development. She promotes and teaches computing skills essential for researchers from different domains, helping them be more innovative and productive in their work. Pawlik has a PhD in computing from The Open University. Contact her at aleksandra.pawlik@manchester.ac.uk.

Judith Segal is a visiting research fellow at The Open University. Her research interests include the organizational and contextual aspects of scientific software development. Contact her at judith.segal@open.ac.uk.

Helen Sharp is a professor of software engineering at The Open University. Her research interests include the study of professional software practice with a focus on human and social aspects of software development. She is the associate editor for Transactions on Software Engineering; a member of the IEEE Software advisory board; a member of ACM, IEEE Computer Society, and the British Computer Society; and a Higher Education Academy (HEA) Fellow. Contact her at h.c.sharp@open.ac.uk.

Marian Petre is a professor of computing at The Open University. Her research interests include expert behavior and reasoning in the design of complex systems. Petre has a PhD in computer science from University College London. She held a Royal Society Wolfson Research Merit Award, in recognition of her research on the nature of expertise in software design, and her empirical studies on reasoning and representation in software development. Contact her at m.petre@open.ac.uk.

Selected articles and columns from IEEE Computer Society publications are also available for free at http://ComputingNow.computer.org.
Take the CS Library wherever you go!

IEEE Computer Society magazines and Transactions are available to subscribers in the portable ePub format.

Just download the articles from the IEEE Computer Society Digital Library, and you can read them on any device that supports ePub, including:

- Adobe Digital Editions (PC, MAC)
- iBooks (iPad, iPhone, iPod touch)
- Nook (Nook, PC, MAC, Android, iPad, iPhone, iPod, other devices)
- EPUBReader (Firefox Add-on)
- Stanza (iPad, iPhone, iPod touch)
- ibis Reader (Online)
- Sony Reader Library (Sony Reader devices, PC, Mac)
- Aldiko (Android)
- Bluefire Reader (iPad, iPhone, iPod touch)
- Calibre (PC, MAC, Linux)
  (Can convert EPUB to MOBI format for Kindle)

www.computer.org/epub
Run-Time Extensibility and Librarization of Simulation Software

Jed Brown | Argonne National Laboratory and University of Colorado Boulder
Matthew G. Knepley | University of Chicago
Barry F. Smith | Argonne National Laboratory

Build-time configuration and environment assumptions are hampering progress and usability in scientific software. This situation, which would be utterly unacceptable in nonscientific software, somehow passes for the norm in scientific packages. The scientific software community needs reusable, easy-to-use software packages that are flexible enough to accommodate next-generation simulation and analysis demands.
simulation while raising the level of abstraction to risk-aware design and decision problems. This evolution unavoidably involves deeper software stacks and the cooperation of distributed teams from multiple disciplines. Meanwhile, each application area continues to innovate and can be characterized as much by its forms of extensibility—such as boundary conditions, geometry, subgrid closures, analysis techniques, data sources, and inherent uncertainty/bias—as by the underlying equations. Original authors can no longer foresee all the use cases for their software. Many common configuration and extensibility approaches create artificial bottlenecks that impede science goals, and the only sustainable approach is to defer all configuration and extensibility to run-time. Doing this effectively pushes applications to minimize the assumptions made about their environment, resulting in applications that are more like libraries—better suited to coupling with other models and performing advanced analysis.

Compile-Time Configuration
Many applications, especially those written in Fortran, perform configuration in the build system. (Alternatives were limited prior to Fortran 2003’s ISO C bindings and now TS29113, which is slated for Fortran2015.) The motivation for configuration in the build system stems from various efficiency concerns (often ill-founded or fixable by adjusting interface granularity), software tool limitations (such as in algorithmic differentiation), poor language support, perceived implementation complexity, and short-term value assessment. Once a package chooses compile-time configuration, the build system becomes a public API used by scripts that perform higher-level analysis. Ad hoc public APIs inhibit software evolution by imposing an unintentionally high cost on change as well as dilution of effort to meet short-term deliverables.

In applications that rely on build-time code generation or pragma-based specialization and optimization, or those written in C++ with heavy template use, the possible combinations must be enumerated at compile-time. Although templates aren’t exclusive (you can compile several variants in the same application), it’s common to see a combinatorial explosion of variants as well as a direct exposure of templates in public interfaces. Because developers can’t compile all combinations into one application, any analysis or testing that explores a large or unpredictable part of the combinations space must include recomilation. Attempting to push the size limits leads to

- error-prone workarounds such as -mcmodel=large (a compiler option that affects linking/compatibility);
- processes spanning more than one NUMA node (degrading memory locality); and
- the inability to run the application on low-memory architectures that might otherwise suit it.

Compute nodes often don’t have access to compilers, making all build-system and compile-time decisions inaccessible to online analysis. A given application might be unable to run in both configurations on different nodes or on different MPI communicators. This limits analysis capability, requires frequent recompilation, and increases user errors resulting from accidentally using the wrong compiled version. The batch queues’ length exacerbates the issue, sometimes requiring days between compiling an application and actually running it. Every compatibility that must be maintained by hand is another opportunity for mistakes, some of which the user might not realize prior to publication.

Some applications create sophisticated scripts for maintaining consistency through the compilation and batch submission process. These scripts must be ported to each architecture, increasing the complexity both of application debugging and of reproducing problems encountered on particular architectures.

Integration tests often must be submitted to batch systems. If different integration tests require that different dependencies be compiled differently, those different versions must be built in advance and kept straight through the test submission and run. When many configurations are needed, the multiple required compilations tend to take a long time and burn through the disk quota.

Advanced Analysis
As models mature in each application area, emphasis shifts from qualitative and subjective interpretation of model output to quantitative analysis of accuracy, reliability, and parameter influence on the target quantities. Correspondingly, today’s models are increasingly used as both forward models and as the target of advanced analysis techniques such as stochastic optimization, risk-aware decisions, and stability analysis. The forward model must then expose an interface for each form of modification that the analysis levels can explore. An interface requiring build-time modification shifts an unacceptable complexity burden to the analysis software and is algorithmically constraining—limiting parallelism, introducing artificial bottlenecks, and preventing some algorithms.
In lieu of tractable deterministic techniques for calibrating empirical phenomenological models, tremendous expert time must be spent tuning parameters. In fields such as climate, earthquakes, and molecular dynamics, this calibration is notoriously sensitive to numerical methods, temporal and/or spatial resolution, and other simulation models. Yet, when faced with this extreme uncertainty and volatility, these parameters are often hard-coded in the source, thwarting reasonable attempts to automate the calibration or model comparisons.

**Model Coupling**

Visionary scientists operating in a single domain have produced a large fraction of successful scientific software. Such visionaries predicted many important model configurations and analysis types, and the community has been largely content to explore within their fuzzy scopes. Each package has been king of its own environment and thus choices were often made without concern for interoperability or impact on other packages. However, the gaping holes in our scientific understanding and engineering capability lie increasingly in the gaps not covered by these mature packages.

Rarely do multiple models operate on identical spatial and temporal scales with similar model and parameter uncertainties. Thus, coupling often requires grappling with multiscale phenomena and high-variance statistics, each an algorithmic challenge in its own right. When components make excessive assumptions about their environment, attempts to couple are either written off or algorithmic quality falls by the wayside, leading to nominally coupled simulations that are unreliable at best and, in most cases, effectively nonconvergent.

The most powerful and pragmatic software approach we know of is to formulate models as libraries with a clean interface hierarchy that lets the external client compose the key capabilities into a coupled model without the higher-level parts that would algorithmically constrain a coupled model. This approach has repeatedly demonstrated its effectiveness outside of scientific computing in areas traditionally dominated by standalone applications, such as compilers (LLVM), Web browsers (KHTML/WebKit), and SQL databases (SQLite). Although process isolation can be useful for security (as in qmail and postfix), reliability (Web browser tabs), and distribution (remote databases), it’s easier to add isolation upon library interfaces than to add composition/embedding atop process separation, especially in HPC environments for which oversubscription is usually catastrophic.

**Provenance and Usability**

Reproducibility and provenance are perpetual challenges of computational science that become more acute as the software stack deepens and a larger number of models, each of greater complexity, are coupled. How can we capture the state of all configuration knobs so that a computational experiment can be reproduced? Compare the complexity of a single configuration file to be read at run-time with that of a heterogeneous configuration consisting of multiple build systems, files passed from earlier stages of computation, and run-time configuration. Provenance is simplified if we use each package without modification, compile them in a standard way, and control them entirely via run-time options. This implies that any libraries the application uses (transitively) must be responsible libraries that adhere to the principles discussed here and elsewhere.² For both maintenance and provenance reasons, custom components needed for a given computational experiment are better placed in version-controlled plug-ins instead of being implemented by modifying upstream sources. To support a coherent top-level specification in a system with build-time or source-level choices, those configuration options must be plumbed through all the intermediate levels, often resulting in another layer of “workflow” scripts and bloated, brittle high-level interfaces.

**Big Data**

Workflows that involve multiple executables usually pass information through the file system. It takes an hour to read or write the contents of volatile memory to global storage on today’s top machines, assuming that peak I/O bandwidth is reached. The largest allocations (as in INCITE or ALCC awards) are on the order of tens of millions of core hours, which means the entire annual compute budget can be burned in a few reads and writes. Global storage as an algorithmic mechanism is dead: where out-of-core algorithms were used in the past, today’s scientists can simply run on more cores, up to the entire machine; but, if the entire machine doesn’t have enough storage, the allocation simply doesn’t have the budget to run an out-of-core algorithm.

If a different application or different application version must be used for the simulation/analysis pipeline’s next stage, data must be dumped to the file system. In situ analysis provides an excellent opportunity to increase efficiency by reducing dependence on the file system, but it’s viable only if the more varied analysis workflow can be performed in
the same application. Interfaces for exchanging data in-memory between different software components could be the same as those used to describe data sets for parallel IO.

Some of today’s simulations support a large and diverse community that analyzes the output. Transitioning to in situ analysis will require dynamic and extensive analysis interfaces to support varied analysis demands. Unlike most parts of mature simulation software, the analysis code often changes with each question a scientist asks and thus is highly volatile and doesn’t benefit from the same amount of testing.

**Nested Dependencies**

Some library dependencies are indirect (transitive) via some intermediate interface that the application actually intends to depend on. A key software engineering principle is that of *encapsulation*, allowing clients to depend only on interfaces that it uses directly, rather than on implementation concerns. Encapsulation isn’t possible if a transitive dependency must be reconfigured for each use case, and combining uses into one application can cause conflicts. The build system for any “library” that requires use-specific configuration effectively becomes a public API that top-level components must interact with, even when the library is used only indirectly.

A single library can be used by multiple components in the same executable. This might be rare when a library is first being developed, but it’s common among popular and versatile libraries. If a library has mutually incompatible configurations, the entire executable can use only one version unless the library developer has taken great care (this is often impractical, especially when linking statically—an unfortunate necessity on many HPC architectures). Even in the best case, needing to use multiple versions complicates the installation and debugging process, invariably leading to a degraded user experience and increased support workload for library maintainers.

**User Modifications**

Software project fragmentation is notoriously expensive and should be avoided when possible. Maintaining local modifications with no plan for upstreaming is a recipe for divergent design—technical debt that must be paid off to combine the features developed in each fork. Fragmentation is especially toxic for libraries that might be used by multiple higher-level packages combined by the overall experiment.

**Packaging and Distribution**

Software developers often underestimate the challenge of installing their own packages. From the user experience perspective, it hardly matters if an installation failure was caused by a user’s broken environment (a circumstance all too familiar to maintainers of popular packages). Upgrading an OS can break existing package installs if the underlying system libraries change. The most reliable way to distribute packages that will always be in sync with the OS is to have them packaged by many common OSs, such as Debian APT, RedHat RPM, MacPorts, and so on. Configure-time options are package distribution’s bane because each variant must be named and conflicts between the variants resolved. Packagers for binary distributions (which are most convenient for users) are justifiably paranoid about the binary interface and hence will be reluctant to package software with fragmented configuration options.

**Implementation and Recommendations**

To manage these workflow challenges, application developers must think more like library developers and control namespaces; avoid global state; relinquish top-level control; control the parallelism’s scope; localize memory allocation; localize complexity so that it doesn’t “bubble up” to the top level; and pay attention to the completeness, generality, stability, and extensibility of all public interfaces. Our suggestions are shaped by experience developing and supporting the Portable Extensible Toolkit for Scientific computation (PETSc) as well as other packages, from low-level libraries to end-user applications. Developers have implemented similar ideas for extensibility and run-time configuration in applications such as Multiphysics Object-Oriented Simulation Environment (MOOSE) and PyLith.

**Resource Allocation**

To localize configuration, allocating resources such as memory should be done locally, with reference counting when appropriate. Contrary to urban legend, static memory allocation offers no tangible performance advantage (so long as dynamic allocations are amortized) and unavoidably ties the workflow into the build system while committing the sin of needless global variables. Different malloc implementations have varying performance, especially in multithreaded scenarios. If necessary, fast implementations like TCMalloc can be recommended,

www.computer.org/cise
but it’s better to contain this complexity in favor of good performance with any `malloc`. Performant allocation can be achieved by associating memory pools or work arrays with algorithm objects, so that `malloc` isn’t called in inner loops.

**Plug-ins**

Source-level dependencies on an implementation (such as directly instantiating a derived class) rather than a generic interface cause choices from deep in the stack to “bubble up” via brittle interfaces that plumb the user’s configuration to the appropriate component. Plug-ins provide a strong way to identify interfaces that can be extended by users and distributed separately from the core package. For example, every class in PETSc has a plug-in architecture, from base linear algebra components to preconditioners, nonlinear solvers, and adaptive controllers for time integration. A plug-in can provide any of these components, which will be indistinguishable from a PETSc native component. Plug-ins consist of a registration function called via `dlopen()`—a creation function called when the plug-in is activated (such as instantiating an object implemented in the plug-in)—and any supporting functions that will be exposed via the object’s methods. Historically, Fortran’s type system and inability to store function pointers have conspired against plug-in implementations, but the new standard provides the necessary tools.

Plug-ins also provide a mechanism to invert dependencies without creating dependency loops. For example, suppose `libB` depends on `libA`, but we would like to provide an optional implementation of an interface in `libB` that depends on `libA`. We can’t put it in `libA` because this would make a cyclic dependency, but it’s unrelated to `libB`’s public interface, so it doesn’t belong there either. We can create `libA-`plugin that depends on both `libA` and `libB`, registering itself as a plug-in of `libA` and calling into `libB` in its implementation. Plug-ins can also be used for optional interfaces to third-party libraries. It’s best to have plug-in search paths from which plug-ins are loaded by `dlopen`, so that they can be distributed independently from the base system without requiring relinking. Shared libraries should be versioned (such as on most POSIX systems, and `current_version` and `compatibility_version` on OSX) to make this distribution more reliable and to assist the layers built on top. (More information on shared library versioning and controlling symbol visibility is available elsewhere.)

Although distribution via shared libraries is convenient for users and packagers, some important HPC execution environments don’t support shared libraries. If you must use such antiproductive environments, the plug-in structure can be preserved, but the build system must ultimately be able to link everything statically. For an application, this typically means that plug-in source trees are placed in a location that the build system picks up; code to call the registration function is then generated and everything is linked together. For a library, plug-ins either must be compiled into a single static archive or the user must explicitly link the plug-ins (in the correct order). The linking interface is a public interface, so changing it shouldn’t be taken lightly. The library can either distribute a tool that determines which plug-ins are available and generates a suitable link line, or it can create a static archive containing all plug-ins. Unfortunately, the `pkg-config` tool is not sufficient to manage multiple configurations and optional dependencies, so many libraries must have their own executable. Wrapper compilers are exclusive (only one library can use a wrapper compiler), and thus they should be avoided.

**Inversion of Control, Recursive Configuration, and the Options Database**

Software libraries’ primary purpose is to contain complexity. Public interfaces should be as simple as possible (but no simpler), meaning that transitive complexity must not be a mandatory part of the public interface. Furthermore, extensible components aren’t known at compile-time (indeed, they might not have been written yet) and thus would be rendered useless if implementation complexity leaked into the public interface. It should be possible to instantiate the same plug-in (implementation unknown to client code) at different locations in the object graph, each with its own configuration. Because the client doesn’t know how to configure the object, some inversion of control is necessary. PETSc’s approach is similar to service locator, but new projects should consider several variations. In PETSc, multiple objects’ instances are distinguished by a `prefix` in the options database, allowing conflict-free, run-time configuration. For example, a multiphysics solver might use a block decomposition and geometric-algebraic multigrid with choices and diagnostics for each block and at each level of one or more multigrid solves, each instance of which we distinguish by prefix. The basic principle is to choose good defaults and defer precise configuration to the run-time interface. Some packages take dynamic extensibility further by embedding a
Turing-complete programming language such as Lua, JavaScript, or Scheme.

PETSc also acknowledges that some users take active control over method configuration, adapting it in response to the physical regime or other factors. Such control is more naturally implemented and debugged with an object-based run-time interface; thus, any run-time configuration exposed via the options database is also exposed via the object-oriented interface. The most challenging compromise in this scenario occurs when an algorithm adaptively configures recursive levels, but the client wants to actively configure portions. Solutions include fine-grained interfaces for “forcing” (in the lazy functional programming sense) certain parts of the setup and callbacks to configure portions when reached. Neither is completely satisfactory.

Object-Oriented Design

We turn now to some contentious issues in object-oriented with which we’re less than enamored with the oft-repeated recommendations.

Partial implementation. Some people believe that all errors should be compile-time errors; thus, any incompatibility must be visible to the compiler. Unfortunately, this approach leads to extremely complicated and fragile type hierarchies. For example, a Matrix is a linear transformation on finite-dimensional vector spaces. Should a Matrix have computable entries? Should the diagonal be extractable? Can the transpose be applied? Are Neumann subproblems available (that is, matrices with certain properties whose sum equals the original matrix)?

Although matrix entries can be computed in principle, the space and time complexity can be so unaffordable as to render that representation useless. Meanwhile, other operations that are unaffordable for explicitly stored matrices might be fast for matrices with special structure. Different preconditioners (which might reside in plug-ins) can require different functionality from the Matrix. Any type system that can guarantee full implementation of a given Matrix interface will end up conflating the desired generic interface with implementation-specific semantics, especially when the Matrix type is also extensible, leading to undesirable dependencies and leakage of transitive complexity. Moreover, the “not implemented” run-time error is likely to be more understandable than a type mismatch error.

Changing the run-time implementation. PETSc has found it useful for major objects to change implementations—such as from multigrid to a direct solve—at run-time. One object can have many dependencies/references and be referenced by many other objects. If the implementation can be changed only at object creation, the user ends up holding factory objects (or the equivalent) solely to recreate “similar” objects. Someone must be responsible for keeping track of these factory objects and rewiring the dependencies when replacing an existing object. This turns out to be messy and error-prone: PETSc thus chose to absorb the “factory” functionality into the object itself, allowing reconfiguration of any sort, at any time. This also removes the need for special interfaces to pass a factory object around to all components that should have a say in that new object’s configuration.

Controlling the binary interface. Time spent recompiling code is nothing but wasted productivity. Implementation concerns such as private variables and new (virtual) methods should never require client code recompilation. PETSc uses a delegator pattern (also known as a “pointer to implementation”7 or bridge8 pattern) to keep such implementation concerns out of the binary interface, thus minimizing recompilation and enabling binary distribution of shared library5 upgrades. This is idiomatic in C, where “objects” are typically implemented via opaque pointers, but often under-utilized in C++ because it entails a bit more boilerplate than the native object model that reveals the classes’ private contents. Delegator incurs an additional static function call, but tests with classic virtual methods and delegator indicate that the main function call overhead (several cycles) comes from the indirect call (virtual function) rather than the static call to the delegator, thus the incremental cost of using the delegator pattern is usually less than two cycles. An ancillary benefit of the delegator pattern is that there’s a unique place to set a debugging breakpoint for each function (rather than having to choose the correct virtual function) and a common place for input validation.

It’s increasingly popular to expose libraries through more dynamic environments such as Python or Julia. Because different languages have different type systems, it’s easier and more reliable to develop language bindings with a simple type system and stable binary interface. Naturally, static methods and opaque pointers are simpler than struct definitions and template-based systems.
Just-in-Time Compilation

With fine-grained composition (such as that in material models and Riemann solvers) and fusion of memory-intensive operations, the number of possible compositions grows combinatorially; in any specific run, however, only a few are important. Precompiling and dispatching (via C++ templates or other inlining techniques) every combination leads to large compile times, bloated executables, confusing debugging, and compromises about which combinations will be made available.

Although a dynamic interface is far more maintainable, the performance overhead is unacceptable for certain applications. When the interface granularity can’t be increased to amortize the overhead of dynamicism, just-in-time (JIT) compilation is an attractive approach to preserve strong encapsulation and debuggability. We expect technologies such as LLVM and OpenCL to become ubiquitous, allowing judicious use of JIT for dynamic kernel fusion and plug-in-style packaging of fine-grained components without sacrificing performance. This might involve tighter integration with languages like Julia and the Numba package for Python, or language extensions to support JIT within traditionally compiled languages.

Upstreaming, Distribution, and Community Building

To provide attractive alternatives to forking, maintainers must be diligent in creating a welcoming environment for upstream contributions. The maintainers should nurture a community that can review contributions, advise about new development approaches, and test new features, while recognizing all forms of contribution. In a transparent community, paper reviewers can easily determine who did the work to implement a new feature; thus any attempt to “scoop” a result based on new capability is easily spotted. We believe that scooping is a purely social problem and that the secrecy inherent in any technical solution is so costly as to rarely be justified. Several major tech companies have famously underestimated this cost when forking open source packages such as the Linux kernel for internal use, later repaying the technical debt to reintegrate with upstream. In science, it’s exceedingly difficult to obtain funding to pay off the technical debt incurred by forking, leading to a wasteland of abandoned forks. This is contrary to the interests of stakeholders, ranging from the program managers and taxpayers to other scientists in the field.

In addition to community building, developers should provide versatile extension points so that contributions can be made without compromising existing functionality and without degrading package maintainability. Developers should see this as a technical prerequisite for maintainable extension rather than private forking. Such extensions must be accompanied by tests lest they break as interfaces evolve. It’s far easier to write tests for dynamic configuration sets than to add new build-time configurations. Additionally, compilers and static analysis tools can check combinations that are not actively used. In contrast, conditional compilation (#ifdef) is not checked, invariably leading to more frequent breakage by other developers (in the test suite, if covered; otherwise the breakage will be found by users and other developers).

Configuration and environment design decisions made by developers of today’s scientific libraries and applications are often disproportionately harmful to usability, productivity, and capability. In such cases, the most effective way to increase scientific or engineering value is to design and refactor software using best practices for extensible library development.

Acknowledgments

Jed Brown and Barry F. Smith were supported by the US Department of Energy, Office of Science, Advanced Scientific Computing Research (contract DE-AC02-06CH11357). Matthew G. Knepley was partially supported by the US DOE (contract DE-AC02-06CH11357) and the US National Science Foundation (grant OCI-1147680).

This article was created by UChicago Argonne, LLC, Operator of Argonne National Laboratory (“Argonne”). Argonne, a US DOE Office of Science laboratory, is operated under contract DE-AC02-06CH11357. The US Government retains for itself, and others acting on its behalf, a paid-up nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government.

References


**Jed Brown** is an assistant computational mathematician at Argonne National Laboratory and an assistant professor adjunct at the University of Colorado Boulder. His research interests include scalable solvers for implicit multiphysics, high-order partial differential equation (PDE) discretization in complex geometry, compatible discretizations for heterogeneous flows, and PDE-constrained optimization. Brown has a PhD in glaciology from ETH Zürich. He’s a member of SIAM, the American Geophysical Union (AGU), ACM Special Interest Group on High-Performance Computing (SIGHPC), and Consortium for Mathematics in the Geosciences (CMG++). He received the 2014 SIAM Activity Group on Supercomputing Junior Scientist Prize and the 2014 IEEE Technical Committee on Scalable Computing Young Achievers Award. Contact him at jedbrown@mcs.anl.gov.

**Matthew G. Knepley** is a senior research associate at the Computation Institute at the University of Chicago. His research interests focus on scientific computation, including fast methods, parallel computing, software development, numerical analysis, and multicores architectures. He’s an author of Argonne National Laboratory's Portable Extensible Toolkit for Scientific computation (PETSc) library for scientific computing and principal designer of the PyLith library for solving dynamic and quasi-static tectonic deformation problems. Knepley has a PhD in computer science from Purdue University, and won the R&D 100 Award in 2009 as part of the PETSc team. Contact him at knepley@ci.uchicago.edu.

**Barry F. Smith** is a senior computational mathematician at Argonne National Laboratory’s Mathematics and Computer Science Division. His research interests include numerical algorithms for linear algebra and partial differential equations, and software for high-performance computing. He’s a developer of PETSc. Smith has a PhD in applied mathematics from New York University’s Courant Institute of Mathematical Sciences, and is a fellow of SIAM and a member of ACM. He was co-recipient (with Lois Curfman McInnes) of the US Department of Energy’s 2011 Ernest Lawrence Award for outstanding contributions in research and development. Contact him at bsmith@mcs.anl.gov.
Simplified Pseudopotential Problems for the Classroom

Trisha Salagaram, Richard Charles Andrew, and Nithaya Chetty | University of Pretoria, South Africa

The authors devise a range of simplified, easily programmable computational problems appropriate for the classroom, which can help teach advanced undergraduate students about particular theoretical and computational aspects of the electronic structure method. This article focuses on the pseudopotential, which is a centrally important concept in many modern ab initio methods.

The 1964 paper by Hohenberg and Kohn on Density Functional Theory (DFT) and many of the publications describing the key approximations related to the implementation of DFT are among the most cited physics papers of all time. The number of citations for these fundamental papers in DFT is set to grow because of the increasing number of scientists across many different scientific and engineering disciplines that are using DFT codes in their research today.

In the 80’s, researchers often developed their own density functional codes and many graduate students were intimately involved in all aspects of code development. The situation today cannot be more different. We have access to slick commercial codes and even some excellent academic codes that are automated and optimized to run on some of the fastest and highly parallel computing architectures. These codes are written by teams of dedicated professionals whose job it is to automate these versatile codes for the growing practitioner community. The practitioner has nothing more to do than choose from pull-down menus and then to turn the proverbial computational knobs to launch computations that were simply not possible barely a decade ago. Consequently, we’re producing students who are adept at using computational code in their applications to real physical systems, but the vast majority have no inkling about the algorithms’ detail for key components of these codes and the theoretical framework that underpin these components.

These students don’t develop the necessary computational skills that make them transferable to different scientific working environments, and this should be a worry for any educator. We fear that this problem is true of other computational communities as well, for example, computational astrophysics, computational fluid dynamics, and so on.

Our aim is to develop computational projects in electronic structure methods that attempt to address these shortcomings, albeit at a fairly basic level appropriate to advanced undergraduate studies. Figure 1 shows the key steps in the self-consistent solutions of the Kohn-Sham equations. The challenge for us is to develop standalone problems that only focus on the essential physics and computing aspects of Figure 1.

For example, pseudopotentials are generated in three dimensions, in the context of self-consistency involving many interacting electrons and particular choices for the exchange-correlation functionals. Pseudopotentials were first used in neutron scattering and atomic physics to simplify complex problems. The essential idea of the pseudopotential can be stripped down to a small...
subset of constructs that we can easily implement in 1D quantum mechanical problems such as the harmonic oscillator. In this way, the problems that we're developing mimic particular key aspects of real electronic structure codes.

Here, we focus on developing simplified pseudopotential problems for the classroom. Specifically, we're focusing on the so-called ab initio pseudopotentials as different from the so-called empirical pseudopotentials.\(^7-^9\) We believe that the advanced undergraduate student working through a range of problems of this nature develops a good pedagogical understanding of this key aspect of the plane-wave pseudopotential method. There are other basis sets that we can use to represent the electronic energy states of a solid state system, such as localized atomic orbitals. However, plane waves are the simplest, as well as elegant, pedagogical, and appropriate for developing the concept of the pseudopotential in the classroom.

Creating Norm-Conserving Pseudopotentials for 1D Systems

In real atomic systems, the interaction between valence electrons and the stationary ions cause the high-level valence electron wave functions to oscillate rapidly in the core region. This makes a numerical study of a solid state system difficult as it requires a great many plane waves to accurately represent this rapid oscillatory behavior. One way

---

**Figure 1.** Self-consistent solution of the Kohn-Sham equations within the plane-wave pseudopotential method.
in which this might be overcome is by replacing the actual Coulombic potential with a suitable pseudopotential that accurately describes the interaction between the ionic core and valence electrons, and which makes a numerical solution easier because the pseudo wave function is smoother.

The implementation of the pseudopotential method in atomic systems is rather intricate. However, we can capture the essential principles in a simpler class of problems. In this article, we create norm-conserving pseudopotentials for simple, noninteracting 1D systems, such as the infinite square well (ISW), finite square well (FSW), simple harmonic oscillator (SHO), and the radial solution of the 3D hydrogen atom (only one electron so the complications that arise from including many-electron interactions don’t apply), to illustrate the pseudopotential method.

There are several ways of creating pseudopotentials. In this article, we focus on the method used by G.P. Kerker. The first step in creating a norm-conserving pseudopotential is to create a pseudo wave function, $F(x)$, for a chosen quantum state, $n$, of the system being studied. The pseudo wave function is created to be nodeless to enable even parity solutions, and to have a single node for odd parity solutions for ISW, FSW, and SHO. For the radial solution of hydrogen, the pseudo wave function is constructed to be nodeless. For all the 1D systems considered, except hydrogen, this is accomplished by first choosing two cut-off lengths, $-x_c$ and $x_c$, that lie between the first and last maximum of the actual wave function $\Psi(x)$. This preserves the symmetry of the system. The pseudo wave function is then created by interpolating between $-x_c$ and $x_c$. The interpolation can be numerical, or we can use an analytical function such as a polynomial, exponential, Gaussian, and so on. For $x \leq -x_c$ and $x \geq x_c$, the pseudo wave function exactly coincides with the actual wave function. To create norm-conserving pseudo wave functions for these 1D systems, we choose the following interpolating function for the ISW, FSW, and SHO:

$$F(x) = x^n \exp(ax^6 + bx^4 + cx^2 + d),$$  \hspace{1cm} (1)

Because these systems’ potentials are symmetric, even functions of $x$, the wave functions have either even or odd parity. We tailor a pseudo wave function for an even or odd state such that it resembles the ground state for the respective even or odd case. Therefore a pseudo wave function for an even state will be nodeless, whereas that for an odd state will consist of one node. The exponent $\eta$ in Equation 1 is equal to 0 for the even states and 1 for the odd states of the ISW, FSW, and SHO.

For the hydrogen atom, a nodeless pseudo wave function is obtained by interpolating from $x = 0$ to a single cut-off radius $x_c$, which lies between the last node and maximum of the real wave function. The interpolating function used is

$$F(x) = x^n \exp(ax^4 + bx^3 + cx^2 + d),$$  \hspace{1cm} (2)

where $\eta$ is equal to $(l + 1)$, where $l$ is the angular momentum quantum number. This is in keeping with Kerker’s method for atomic systems.

The constants $a$, $b$, $c$, and $d$ in Equations 1 and 2 are determined by applying the following constraints:

- The pseudo wave function is equal to the real wave function at $x_c$:
  $$F(x_c) = \Psi(x_c).$$  \hspace{1cm} (3)

- The first and second derivatives of the pseudo wave function are equal to those of the real wave function at $x_c$:
  $$F'(x_c) = \Psi'(x_c),$$  \hspace{1cm} (4)
  $$F''(x_c) = \Psi''(x_c).$$  \hspace{1cm} (5)

- The real and pseudo system have the same eigenvalue for the quantum state $\nu$ being psuedized:
  $$\varepsilon_{\nu} = \varepsilon_{\nu}.$$  \hspace{1cm} (6)

- The norm of the real and pseudo system within the pseudized range is conserved:
  $$\int_{-\infty}^{\infty} |F(x)|^2 \, dx = \int_{-\infty}^{\infty} |\Psi(x)|^2 \, dx,$$  \hspace{1cm} (7)
  with
  $$\int_{0}^{\infty} |F(x)|^2 \, dx = \int_{0}^{\infty} |\Psi(x)|^2 \, dx$$  \hspace{1cm} (8)
  for hydrogen.

For the ISW, FSW, and SHO, Equations 3–5 at $-x_c$ are satisfied by symmetry. For the hydrogen atom the cut-off region is $0 \leq x \leq x_c$ and the conditions are satisfied at $x_c$.

The cut-off region referred to thus far is the region where the wave function is relegated nodeless subject to norm conservation, and beyond which there is no change to the wave function. This mimics...
the situation in real chemical systems where the tail end of the wave functions participate in chemical bonding, but the core remains essentially inert.

The pseudopotential, \( V_{ps} \), is determined by inverting the 1D Schrödinger equation for the pseudo system, which in terms of Rydberg atomic units where \( h/2m = 1 \), is given by

\[-F''(x) + V_{ps} F(x) = \varepsilon_n F(x).\]

This yields

\[V_{ps}(x) = \varepsilon_n + \frac{F''(x)}{F(x)} \]  

(9)

We deliberately chose \( F(x) \) so that the term \( F''(x)/F(x) \) in Equation 9 is finite for all \( x \) including at \( x = 0 \) where \( F(0) = 0 \).

The condition in Equations 7 or 8 ensures that the pseudo wave function is properly normalized. It also ensures that the norm of the real and pseudo system are the same between –\( x_c \) and \( x_c \), (0 and \( x_c \) for hydrogen), and consequently the scattering properties of the core of the real potential are transferred to that of the pseudopotential. The scattering phase shifts of quantum systems are related to the first energy derivative of the logarithmic derivative of the wave function. Therefore, the transferability of a pseudopotential can be determined by plotting the logarithmic derivative of the pseudo wave function as a function of energy and comparing this to that of the real wave function. This is done numerically by integrating the discrete version of the Schrödinger equation for the pseudo system

\[
\frac{\Delta F_{i+1} - 2F_i + F_{i-1}}{(2\Delta x)^2} + V_{ps}^i F_i = EF_i,
\]

to calculate the wave function \( F_{i+1} \) over a range of energies \( E \),

\[F_{i+1} = (V_{ps}^i - E)\Delta x^2 F_i + 2F_i - F_{i-1},\]

where \( i \) is the index for the spatial grid, and \( \varepsilon_n - \delta < E < \varepsilon_n + \delta \). Thereafter, we calculate the logarithmic derivative of the wave function at a particular diagnostic length \( x_D \) using the following finite difference expression:

\[
x_D \frac{d}{dx} \ln F \big|_{x_D} = \frac{x_D (F_{i+1} - F_{i-1})}{F_i (2\Delta x)},
\]

where \( x_D \) is the spatial index of the diagnostic length \( x_D \). In this study, we’ve chosen \( x_D = x_c \). We use similar equations to calculate the logarithmic derivative of the real wave function. The energy range for which these curves coincide for the real and pseudo wave functions is a direct measure of the energy range for which the pseudopotential is transferable. In a solid, the atomic energy levels spread into bands of energies, and so norm conservation ultimately guarantees that the single particle states in a solid are accurately determined within the pseudopotential framework.

**Results and Discussion**

We present results for the various systems studied in this section.

**Infinite Square Well, Finite Square Well, and Simple Harmonic Oscillator**

Real wave functions for the second and third excited states of the ISW and the corresponding pseudo wave functions are presented in Figures 2a and 2d, respectively. The pseudo wave functions are obtained by satisfying conditions in Equations 3–7. The pseudo wave function of the second excited state is nodeless, smooth, and continuous with the real wave function in the cut-off region, as desired. The pseudo wave function of the third excited state has a single node at the origin as desired, and is also smooth and continuous with the real wave function in the interpolated interval. These results illustrate that the chosen interpolating function from Equation 1 is a suitable one for the ISW.

We calculate the corresponding pseudopotentials using Equation 9. These are plotted together with the real potentials in Figure 2b for the second excited state and Figure 2e for the third excited state of the ISW. The pseudopotentials are equal to the actual potentials for \( x \leq -x_c \) and \( x \geq x_c \). We determine the transferability of these pseudopotentials by comparing the logarithmic derivatives of the real and pseudo wave functions as a function of energy. These results are presented in Figure 2c and Figure 2f for the second and third excited states of the ISW, respectively.

For the second excited state the transferability of the pseudopotential appears to be reasonable as the logarithmic derivatives show observable deviation around the energy level of interest, as indicated by the arrow in Figure 2c. To clarify this result an ancillary test, which involves comparing the eigen energy of the fourth excited state of the real ISW potential with the first excited state of the pseudopotential, is performed. These energies differ by more than 10 percent, which suggests that more work is needed to improve on the interpolation scheme to derive a more transferable potential. This is also exemplified in Figure 2f, where in the case of the third excited state the logarithmic
derivatives show significant deviation around the energy level of interest. We leave it to the reader to explore improved interpolation schemes to increase the transferability of the potential.

We obtained similar results for the FSW (see Figure 3) and SHO (see Figure 4), where we used Equation 1 to create a pseudo wave function for the second excited state of both systems. The corresponding pseudopotentials obtained using Equation 9 aren't transferable to other excited eigenstates, as there are significant deviations between the logarithmic derivatives for the real and pseudo wave functions (in Figures 3c and 4c) around the energy levels of interest.

Hydrogen

We obtain nodeless pseudo wave functions for the $2s$, $3p$, and $4d$ states of hydrogen using the interpolating function (see Equation 2), which has to satisfy the conditions in Equations 3–6, and 8. These are presented in the first row of Figure 5. The second row of Figure 5 shows the pseudopotentials plotted together with the real potentials. These are finite at the origin, which is a desirable feature, especially in the computational study of more complex solid state systems, as it reduces the number of plane waves required to accurately describe the system.

The third row of Figure 5 presents the logarithmic derivatives of the real and pseudo wave functions versus energy. The result for the $2s$ state shows excellent curve tracking for the logarithmic derivatives of the real and pseudo wave functions over a wide energy range. The energies of the first excited state of the pseudopotential and the $n = 3$ excited state differ by less than 1 percent. This is indicative of a highly transferable pseudopotential to more complex chemical environments\textsuperscript{13,15} involving hydrogen. The pseudopotential has a repulsive core, which simply reflects the repulsion due to the Pauli exclusion principle (orthonormality of the states in this context means that the higher level states for a given orbital momentum quantum number are pushed out).

The results obtained for the $3p$ and $4d$ states of hydrogen are reasonable, however they suggest that the transferability of pseudopotential decreases as the energy of the state increases.

Other interpolating functions could be used to create nodeless pseudo wave functions to obtain more transferable pseudopotentials, which would make for useful exercises for students.

---

Figure 2. Wave functions, potentials, and logarithmic derivatives for the real (solid line) and pseudo (dashed line) infinite square well (ISW). The top row consists of results for the second excited state of the ISW, which is an even parity state, and the bottom row consists of results for the third excited state of the ISW, which is an odd parity state.
We created norm-conserving pseudopotentials for several noninteracting 1D quantum mechanical systems. The purpose of this endeavor was to develop computational exercises to teach undergraduate and graduate students particular theoretical and numerical aspects of the pseudopotential method which is an essential aspect of modern ab initio codes and which many graduate students don’t understand, in spite of them using these ab initio codes for their research.

Using the method described in this article, we obtained norm-conserving pseudopotentials for all the systems considered. The transferability of the

---

**Figure 3.** (a) Wave functions, (b) potentials, and (c) logarithmic derivatives for the real (solid line) and pseudo (dashed line) finite square well. The second excited state is pseudized.

**Figure 4.** (a) Wave functions, (b) potentials, and (c) logarithmic derivatives for the real (solid line) and pseudo (dashed line) simple harmonic oscillator. The second excited state for this system is pseudized.
Pseudopotentials varied from highly transferable for the 2s state of the hydrogen atom to untransferable for the ISW (third excited state), FSW, and SHO. These exercises are simple and require reasonably uncomplicated algorithms to produce a working numerical solution. However, a student will have to thoroughly understand all aspects (theoretical, numerical, and programming) of the problem to produce a correct numerical solution. Therefore such exercises can be used to enhance students understanding of the pseudopotential method.

Acknowledgments
We thank the National Research Foundation and the University of Pretoria for providing funding for this work. Nithaya Chetty is grateful to the National Institute for Theoretical Physics for support. We also thank Wolfgang Christian for useful discussions.

References

Figure 5. Wave functions (first row), potentials (second row), and logarithmic derivatives (third row) for the real (solid line) and pseudo (dashed line) hydrogen atom. The 2s, 3p, and 4d states are pseudized.


---

Trisha Salagaram is a lecturer in the Department of Science, Mathematics, and Technology Education at the University of Pretoria. Her research interests include computational physics education research, and theoretical and computational studies of liquid-phase sintering of silicon nitride and diamond synthesis. Salagaram has a PhD in physics from the University of Kwazulu-Natal. Contact her at trisha.salagaram@up.ac.za.

Richard Charles Andrew is a postdoctoral fellow of physics at the University of Pretoria. His research interests include 3D alloys, 2D systems, and electronic structure computational projects for the classroom. Andrew has a PhD in physics from the University of Pretoria. Contact him at richard.andrew@up.ac.za.

Nithaya Chetty is a full professor of physics at the University of Pretoria, and is deputy CEO:Astronomy at the National Research Foundation. His research interests include first principle quantum mechanical studies of silicon carbide, carbon nanotubes, graphene, boronitrene, and transition metal alloys using plane wave pseudopotential methods; and computational physics education research. Chetty has a PhD in theoretical and computational solid state physics from the University of Illinois at Urbana-Champaign. Contact him at nithaya.chetty@up.ac.za.
A neural network technique known as Kohonen unsupervised training is coupled with a modified version of the particle swarm optimization technique in an effort to develop an algorithm capable of finding multiple optimal solutions for a given problem. The results of five example problems of increasing difficulty validate the Alternative Analysis Networking algorithm’s functionality.
where $x_i$ is the $i$th component of the initial set of inputs, and $z$ is the normalized input.

2. Randomly initialize weights, $w_i$, for the desired number of neurons and normalize weights according to the following:

$$w_i = \frac{w_i}{\sqrt{\sum_{j=1}^{N} w_j^2}}.$$ (1)

3. One input pattern at a time, apply the inputs to each neuron:

$$\text{net} = \sum_{i=1}^{N} z_i w_i = z w^T.$$  

4. Update the weight of the winning neuron ($W_k$) with the highest net value:

$$W_k = W_k + \alpha Z,$$

where $\alpha$ is a learning constant.

5. Renormalize new weights using Equation 1.

6. Repeat steps 3, 4, and 5 for each input pattern.

Two important properties of note for this classification method are that during the training, some neurons will never take part in learning, which will lead unoccupied neighborhoods, and that there’s no restriction on a neighborhood’s size. In a 2D sense, the Kohonen technique divides the domain into unequally sized slices of pie, and whichever slice of pie a particle falls within dictates to which neighborhood that particle belongs. This is demonstrated in Figure 1. From the figure, we see how seemingly unclustered data can be clustered into distinct groups. These individual groups will form neighborhoods of particles that will be applied to the optimization schemes to be discussed. Having the ability to define the number of neighborhoods effectively gives the user the freedom to estimate the number of local optima to search.

**Optimization**

After setting up the neighborhoods, three distinct optimization phases take place: base-level optimization within the individual neighborhoods, a second-level optimization among the top performers from each neighborhood, and a gradient-based routine for solution refinement. This last point is directly inspired by the successes demonstrated by Jenkins and Hartfield.\(^3\)

**Base-Level Optimization**

We used the particle swarm technique for base-level optimization. We limited particles in communication to only the particles within a given neighborhood. The equations of motion for a particle were unchanged from those presented by Sudhanshu Mishra\(^6\) in the repulsive particle swarm optimizer (RPSO):

$$v_{i+1} = -\gamma (\hat{x}_m - x_i) + R_2 \beta (\hat{x}_i - x_i) + \omega v_i + R_3,$$

$$x_{i+1} = x_i + v_{i+1}.$$  

We used the Mishra RPSO algorithm due to its robustness as compared to the basic particle swarm optimizer. In particular, the Mishra algorithm is more effective in especially complex design spaces, and has been tested on a wide array of functions.

**Second-Level Optimization**

We implemented the second-level optimization so that the neighborhoods could move throughout the design space. Without a second-level optimization in place, each neighborhood would locate an optimum within the local domain, but all collectively could fail to find the optimal region in the entire design space. The second-level optimization provides a way for the neighborhoods to move as a whole. The method implemented in this study was an atomic attraction/repulsion model. We chose this model somewhat arbitrarily, but it proved to be successful. Future work would involve investigating other models for this phase of optimization. The attraction/repulsion model is represented as

$$x_{i+1} = -\gamma (\hat{x}_m - x_i) \left( \frac{\gamma}{(\hat{x}_m - x_i)^2} - \frac{\omega}{(\hat{x}_m - x_i)^2} \right).$$
where \( \hat{x}_m \) represents the global best member, \( x \) represents the positions of the neighborhood best particles, and \( \gamma \) and \( \omega \) are tuning parameters.

**Gradient-Based Optimization**

We added a gradient-based minimization routine to refine the best members of each neighborhood. This improves the probability of finding true local optima. The minimization scheme was the pattern search method developed by Robert Hooke and Terry Jeeves.\(^7\) This method is robust and doesn’t require an exact calculation of the gradient. Rather, the pattern search method evaluates the gradient in each direction and performs a collective pattern move in the direction of steepest descent. We implemented the pattern search method into the Alternative Analysis Network-ing algorithm on the basis of results observed by Jenkins and Hartfield.\(^3\) At the end of each base-
and second-level maneuver, a pattern search is performed on the best performing members of each neighborhood.

**Results**

We performed five test cases, each with increasing complexity. The first test case is a simple 2D mathematical optimization problem that is unconstrained but fraught with numerous local optima. The second test case is a constrained tension/compression spring design problem with four input variables. The third case is a rocket motor design problem, which presents numerous constraints and is fraught with numerous local and global optima. The fourth case is a multiobjective clutch brake design problem with numerous constraints. The final case is a highly constrained multiobjective robotic gripper optimization design problem, where the solution space consists of large infeasible regions with small areas of feasible solutions scattered throughout.

**Case 1: Unconstrained Math Problem**

The unconstrained math problem provides a validation that the optimization scheme works correctly. The answer is known beforehand, and can be verified quickly. In the case of classifying multiple optimal solutions, the unconstrained math problem still needs to be somewhat complex. Equation 2 results in the surface shown in Figure 2, which is fraught with numerous local minima and maxima:

\[
\varepsilon = 25(\sin^2 x + \sin^2 y) + x^2 + y^2,
\]  

where \( x \) and \( y \) lie in the domain of \([-5; 5]\). The surface has 8 local minima, 1 global minimum, 12 local maxima, and 4 global maxima. The global minimum lies at the origin with the global maxima at the corners of the domain. Table 1 lists the global and local minima and their associated output.

<table>
<thead>
<tr>
<th>Minima number</th>
<th>X</th>
<th>Y</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global minima</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Major local minima</td>
<td>(\pm 3)</td>
<td>0</td>
<td>9.5</td>
</tr>
<tr>
<td>Major local minima</td>
<td>0</td>
<td>(\pm 3)</td>
<td>9.5</td>
</tr>
<tr>
<td>Minor local minima</td>
<td>(\pm 3)</td>
<td>(\pm 3)</td>
<td>19</td>
</tr>
</tbody>
</table>

**Table 1. Global and local minima, locations, and output values.**
the particle swarm to 0.3, and the optimizer iteration limit to 25. This run was repeated 100 times to gauge how well the optimizer performs at identifying all local minima near the global minimum. Table 2 gives the statistics from the 100 runs. The global minimum was found in every run.

Table 2. Final results for the unconstrained math problem.

<table>
<thead>
<tr>
<th>Minima found</th>
<th>%</th>
<th>Total found</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Two major</td>
<td>97</td>
<td>5</td>
<td>93</td>
</tr>
<tr>
<td>Three major</td>
<td>77</td>
<td>6</td>
<td>62</td>
</tr>
<tr>
<td>Two minor</td>
<td>62</td>
<td>7</td>
<td>27</td>
</tr>
<tr>
<td>Three minor</td>
<td>19</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

From Table 2, we see that the minor local minima were found less often than the major local minima. This was expected because as the solution develops, the major local minima draw particles at minor local minima away due to their lower function cost. We expected that finding three of the four minor local minima shouldn’t happen nearly as often as finding three out of the four major local minima. The optimizer will always favor local minima with a better function cost over others because the second-level optimization scheme will draw the neighborhoods toward the global minimum. Also from Table 2, five minima were found 93 percent of the time. Although not listed, at least four minima were found in every run, including the global minima in every run. This example demonstrates the optimizer’s ability to simultaneously seek out local optima but also jump from minor local optima to major local optima and global optima. Demonstrating this ability is a major step forward from a conventional optimizer, which is capable of either only finding global optima or only finding local optima, toward an optimizer that can do both, and in some cases simultaneously.

Case 2: Tension/Compression Spring Design

The tension/compression spring design problem has been optimized previously by Carlos A. Coello and Xiaohui Hu and his colleagues. This problem represents a highly constrained engineering design problem for which the near optimal solution has been reported already. There have been no reported local optima to the problem. Mathematically, the spring design problem is formalized as minimization of the weight of a tension compression spring via Equation 3:

\[ f(X) = (N + 2)Dd^2 \]  

where \( D \) is the mean coil diameter, \( d \) is the wire diameter, and \( N \) is the number of active coils. The spring design is subject to the following constraints:

\[
\begin{align*}
G(1) &= 1 - \frac{D^4 N}{71,785d^2} \leq 0 \\
G(2) &= \frac{4D^3 - dD}{12,566(Dd^3 - d^3)} + \frac{1}{5108d^2} \leq 0 \\
G(3) &= 1 - \frac{140.45d}{D^2} \leq 0 \\
G(4) &= \frac{D + d}{1.5} - 1 \leq 0
\end{align*}
\]

Because no local optima exist for this problem, the end goal for the optimizer should be to find the single global optima that meets the constraints. We executed the optimizer using 150 members with 150 generations with four neighborhoods. Table 3 presents the final results given by Carlos A. Coello and Xiaohui Hu and his colleagues alongside the results for this study.

The optimizer developed in this study retained the ability to find the global optimum (and as it turns out, a better answer for this problem than previously reported) when no local optima exist, despite the highly constrained design space. This demonstrates the communication ability that’s available between the neighborhoods.

Case 3: Solid Rocket Motor Design Problem

The solid rocket motor design problem is associated with a highly complex design space typical of what
would be observed in a real-world engineering design scenario. The solid rocket motor program uses nine geometric variables to describe a solid rocket motor. The specific geometric inputs used for this study are described in full detail elsewhere.\textsuperscript{10,11} The goal for the optimizer in this case study was to find as many motor geometries as possible that can produce a chamber pressure of 300 pounds per square inch for exactly 20 seconds. The performance metric for this problem is the root-mean-square (RMS) error between the output pressure–time profile and the neutral pressure–time curve that's desired.

This is a common solid rocket motor problem that presents a number of challenges. There are a total of nine independent variables, which makes the design space complex and difficult to visualize. Two of the independent parameters are integers. The first integer parameter, the number of star points, has a large effect on the profile of the pressure versus time curve (progressive, regressive, neutral, and so on). The second integer parameter, propellant selection, is a series of choices and has no meaningful gradient, as some propellant choices are similar while others are different. There are numerous geometric constraints, some of which can be handled implicitly by nondimensionalizing some of the independent variables in an intelligent manner.

We executed the optimizer with 100 particles for 50 generations with six neighborhoods. The results of the optimization were four unique solutions, shown in Figures 3 and 4. While the pressure–time curves weren’t perfect, it’s important to remember that the optimizer is looking for global and local optima. The results in Figure 3 should be considered local optima, as they’re close (10 percent RMS) but not exact. From Figure 4, clearly the optimizer was searching vastly different areas of the design space. While the fuel type selection was the same for all four solutions, and the number of star points varied,

Figure 3. Pressure–time profile for four distinct solutions from solid rocket motor optimization. (a) Motor 1, (b) motor 2, (c) motor 3, and (d) motor 4.
points for each motor was between seven and nine, the lengths of each motor are drastically different. This example effectively demonstrates the optimizer’s full capabilities at locating unique, local optima in a complex and constrained design space.

**Case 4: Multiple Clutch Brake Design**

The multiple clutch brake design problem is a multiobjective optimization problem from Andrzej Osyczka. Figure 5 shows the diagram of the brake design under consideration, including the parameters that define the geometry of the brake. These parameters can be changed by the optimizer, and include the inner radius ($R_i$), the outer radius ($R_0$), the disc thickness ($A$), the number of friction surfaces ($Z$), and the actuating force ($F$). These five variables as well as the respective minimum and maximum bounds used in this optimization process can be seen in Table 4. The objective of this design problem is to minimize the brake’s mass while also minimizing the brake’s stopping time. The mass of the brake (in kilograms) can be found by

$$f_1(x) = \pi (R_0^2 - R_i^2) A (Z + 1) \rho,$$  \hspace{2cm} (4)

while the stopping time of the brake (in seconds) can be found by

$$f_2(x) = t_s = \frac{J_f \omega}{M_s + M_f},$$  \hspace{2cm} (5)

where the braking torque $M_s$ is shown by

$$M_s = \frac{2}{3} \mu F Z (R_0^3 - R_i^3).$$

We determined the total fitness by summing the weighted and normalized fitness values from Equations 4 and 5. The fitness values $f_1(x)$ and $f_2(x)$ were normalized due to the fitness values’ differing magnitudes, which causes the optimizer to be biased and only minimize the fitness value with the larger magnitude. The equation for total fitness follows:

$$F_{\text{total}} = W_{f_1} \left( \frac{f_1(x)}{0.35} + (1 - W_{f_1}) \frac{f_2(x)}{12} \right),$$  \hspace{2cm} (6)

where $W_{f_1}$ is the weighted value that controls the bias toward $f_1(x)$, where $W_{f_1} = 1$, or toward $f_2(x)$, where $W_{f_1} = 0$. We set this value at 0.5 for this work.

Table 5 shows the parameters that were held constant for this optimization process. These values were set to be the same as those used in Osyczka’s analysis to ensure accurate comparison of the optimization results.

![Figure 4. Motors geometries 1 through 4 (left to right) based on nine independent variables. While the fuel type selection was the same for all four solutions, and the number of star points for each motor was between seven and nine, the lengths of each motor are drastically different.](image-url)

![Figure 5. Multiple clutch brake schematic.](image-url)

**Table 4. Decision variable bounds.**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Minimum bound</th>
<th>Maximum bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_i$</td>
<td>55 mm</td>
<td>90 mm</td>
</tr>
<tr>
<td>$R_0$</td>
<td>75 mm</td>
<td>110 mm</td>
</tr>
<tr>
<td>$A$</td>
<td>1.5 mm</td>
<td>3.0 mm</td>
</tr>
<tr>
<td>$Z$</td>
<td>1</td>
<td>9</td>
</tr>
<tr>
<td>$F$</td>
<td>500 Newtons (N)</td>
<td>1,000 N</td>
</tr>
</tbody>
</table>
The brake design problem is subject to the following constraints:

\[
\begin{align*}
G(1) &= R_i - R_{i_{\text{min}}} \geq 0 \\
G(2) &= R_{0_{\text{max}}} - R_0 \geq 0 \\
G(3) &= R_0 - R_i - \Delta R \geq 0 \\
G(4) &= A - A_{\text{min}} \geq 0 \\
G(5) &= A_{\text{max}} - A \geq 0 \\
G(6) &= L_{\text{max}} - (Z + 1)(A + \delta) \geq 0 \\
G(7) &= Z_{\text{max}} - (Z + 1) \geq 0 \\
G(8) &= Z - 1 \geq 0 \\
G(9) &= P_{\text{max}} - P_{\text{min}} \geq 0 \\
G(10) &= P_{\text{max}} V_{\text{f_{max}}} - P_{\text{max}} V_{j} \geq 0 \\
G(11) &= V_{\text{f_{max}}} - V_{j} \geq 0 \\
G(12) &= t_{\text{max}} - t_{\delta} \geq 0 \\
G(13) &= M_{\theta} - kM_{\theta} V_{j} \geq 0 \\
G(14) &= t_{\delta} \geq 0 \\
G(15) &= F \geq 0 \\
G(16) &= F_{\text{max}} - F \geq 0,
\end{align*}
\]

where

\[
P_{\text{min}} = \frac{F}{S}, \\ S = \pi \left( R_0^2 - R_i^2 \right),
\]

and

\[
V_{j} = \frac{\pi R_0 \omega}{30}, \\ R_{\delta} = \frac{2 \left( R_0^2 - R_i^2 \right)}{3 \left( R_0^2 - R_i^2 \right)}.
\]

We conducted the optimization run such that the number of function evaluations in this work were equivalent to the number of function evaluations in Osyczka’s analysis. This resulted in a population size of 200 members, which were each evaluated for a total of 123 generations.

Figure 6 shows the Pareto front that we obtained in this work (circles) compared to the Pareto front obtained in Osyczka’s work (triangles). We can see that the particle swarm with neighborhood obtained solutions that are either better than or equivalent to the results from Osyczka’s work in the same number of function evaluations.

**Table 5. Multiple clutch brakes parameters.**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>( R_{i_{\text{min}}} )</td>
<td>55</td>
<td>mm</td>
</tr>
<tr>
<td>( R_{0_{\text{max}}} )</td>
<td>110</td>
<td>mm</td>
</tr>
<tr>
<td>( \Delta R )</td>
<td>20</td>
<td>mm</td>
</tr>
<tr>
<td>( A_{\text{min}} )</td>
<td>1.5</td>
<td>mm</td>
</tr>
<tr>
<td>( A_{\text{max}} )</td>
<td>3.0</td>
<td>mm</td>
</tr>
<tr>
<td>( \delta )</td>
<td>0.5</td>
<td>mm</td>
</tr>
<tr>
<td>( L_{\text{max}} )</td>
<td>30</td>
<td>mm</td>
</tr>
<tr>
<td>( \rho )</td>
<td>0.0000078</td>
<td>kg/mm³</td>
</tr>
<tr>
<td>( P_{\text{dop}} )</td>
<td>1</td>
<td>Megapascals (MPa)</td>
</tr>
<tr>
<td>( Z_{\text{max}} )</td>
<td>10</td>
<td>–</td>
</tr>
<tr>
<td>( V_{\text{f_{max}}} )</td>
<td>10</td>
<td>Meters/second</td>
</tr>
<tr>
<td>( k )</td>
<td>1.5</td>
<td>–</td>
</tr>
<tr>
<td>( F_{\text{max}} )</td>
<td>1,000</td>
<td>N</td>
</tr>
<tr>
<td>( t_{\text{max}} )</td>
<td>15</td>
<td>Seconds</td>
</tr>
<tr>
<td>( \omega )</td>
<td>250</td>
<td>Revolutions per minute</td>
</tr>
<tr>
<td>( M_{\theta} )</td>
<td>40</td>
<td>Newton-Meters (Nm)</td>
</tr>
<tr>
<td>( M_{f} )</td>
<td>3</td>
<td>Nm</td>
</tr>
<tr>
<td>( J_{z} )</td>
<td>55</td>
<td>kg * m²</td>
</tr>
</tbody>
</table>

**Case 5: Robot Gripper Optimization**

The robot gripper design optimization problem is also a multiobjective optimization problem from Osyczka. This problem is more complex than the multiple clutch brake design problem due to the highly constrained nature of the problem and the increase in the number of decision variables. The geometry for the robotic gripper under consideration can be seen in Figure 7, and can be described using seven different variables. Table 6 shows these seven variables along with their respective upper
and lower bounds that we used for this work. This design problem’s objective is to minimize the sum of the length of all gripper elements, as seen by

\[ f_1(x) = a + b + c + e + f + l, \]

and to minimize the force transmission ratio between the gripper actuator and the gripper ends

\[ f_2(x) = \frac{P}{\min F_i(x, z)}, \]

where the force exerted by the gripper ends is defined as

\[ F_i = \frac{P \cdot b \cdot \sin(\alpha + \beta)}{2 \cdot c \cdot \cos(\alpha)}, \quad (7) \]

where \( P \) is the force from the gripper actuator, which is set to be constant at \( P = 100 \, \text{N} \). Equation 7 shows that the force applied by the gripper ends is also dependent upon the angles \( \alpha \) and \( \beta \) (shown in Figure 7). We can find these angles using geometric relations.

The Pythagorean theorem can be applied to find the length of distance \( g \) as shown by

\[ g = \sqrt{(l - z)^2 + e^2}. \]

We can then use this value in the Law of Cosines to determine the angle \( \alpha \) as

\[ \alpha = \cos^{-1} \left( \frac{a^2 + g^2 - b^2}{2 \cdot a \cdot g} \right) + \phi, \]

and the angle \( \beta \) as

\[ \beta = \cos^{-1} \left( \frac{b^2 + g^2 - a^2}{2 \cdot b \cdot g} \right) - \phi, \]

where \( \phi \) is represented by

\[ \phi = \tan^{-1} \left( \frac{e}{l - z} \right). \]

We found the total fitness for the robot gripper optimization problem using an equation similar to the one used in the multiple clutch brake design problem:

\[ F_{\text{total}} = W_{f1} \left( \frac{f_1(x)}{100} \right) + (1 - W_{f1}) \left( \frac{f_2(x)}{4} \right), \quad (8) \]

with the only difference between Equations 6 and 8 being the values in the denominator used to normalize the fitness functions \( f_1(x) \) and \( f_2(x) \). The value \( W_{f1} \) is once again the weighted value that controls the bias toward \( f_1(x) \), where \( W_{f1} = 1 \), or toward \( f_2(x) \), where \( W_{f1} = 0 \), and set at 0.5 for this work.

Figure 7. Robot gripper geometry.\(^{12}\)

<table>
<thead>
<tr>
<th>Table 6. Decision variable bounds.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable</td>
</tr>
<tr>
<td>---------</td>
</tr>
<tr>
<td>( a )</td>
</tr>
<tr>
<td>( b )</td>
</tr>
<tr>
<td>( c )</td>
</tr>
<tr>
<td>( e )</td>
</tr>
<tr>
<td>( f )</td>
</tr>
<tr>
<td>( l )</td>
</tr>
<tr>
<td>( \delta )</td>
</tr>
</tbody>
</table>

The robot gripper design problem is subject to the following constraints:

\[ G(1) = Y_{\text{min}} - y(x, Z_{\text{max}}) \geq 0 \]
\[ G(2) = y(x, Z_{\text{max}}) \geq 0 \]
\[ G(3) = y(x, 0) - Y_{\text{max}} \geq 0 \]
\[ G(4) = Y_G - y(x, 0) \geq 0 \]
\[ G(5) = (a + b)^2 - F - e^2 \geq 0 \]
\[ G(6) = (l - Z_{\text{max}})^2 + (a - e)^2 - b^2 \geq 0 \]
\[ G(7) = l - Z_{\text{max}} \geq 0 \]
\[ G(8) = FR - (\text{max}F_1(x, z) - \text{min}F_1(x, z)) \geq 0, \]

where the displacement of the gripper ends \( y(x, z) \) can be found by

\[ y(x, z) = 2 \left[ e + f + c \cdot \sin \left( \beta + \delta \right) \right]. \]

Table 7 shows the values for the constants \( Y_{\text{min}}, Y_{\text{max}}, Y_G, Z_{\text{max}}, \) and \( FR \) that we used in the constraints.

We conducted the optimization run using a population size of 400 members, each of which we evaluated for a total of 1,000 generations.
Figure 8 shows the comparison between the Pareto front found in this work and the Pareto front found in Osyczka’s work. We can see that the particle swarm optimizer with neighborhooding techniques (circles) found greatly improved solutions in the same number of function calls as in Osyczka’s work (triangles). The highly constrained nature of the solution space resulted in a much thinner Pareto front. The most restrictive constraint was found to be constraint $G(8)$ because of the low FR value that we used.

Figure 8 shows the comparison between the Pareto front found in this work and the Pareto front found in Osyczka’s work. We can see that the particle swarm optimizer with neighborhooding techniques (circles) found greatly improved solutions in the same number of function calls as in Osyczka’s work (triangles). The highly constrained nature of the solution space resulted in a much thinner Pareto front. The most restrictive constraint was found to be constraint $G(8)$ because of the low FR value that we used.

The development of population based optimization routines brought about the capability to locate solutions in a complex and constrained design space. These stochastic schemes typically only develop a single solution, usually the global optimum. In most instances, however, it’s desirable to find multiple optimal solutions. The algorithm described in this article is capable of accomplishing this feat. The Alternative Analysis Networking algorithm has the following advantages:

- From the unconstrained mathematical problem, the algorithm is capable of jumping from minor local optima toward major local optima and the global optima. This fact is important in verifying that the optimizer won’t simply find a local optima within a local neighborhood domain, but instead will make some attempt to improve.
- The constrained tension/compression spring example demonstrated the algorithm’s ability to navigate a complex design space and constraints as well as the algorithm’s ability to find a global optima when no local optima are known to exist.
- The solid rocket motor example proved that the algorithm can be effective in practical real-world engineering design problems.
- The multiple clutch brake design problem and the robot gripper design problem proved that the algorithm is effective at solving problems in which multiple objective functions are to be optimized.
- The Kohonen unsupervised training technique provides the user with the ability to define the number of desired optimal solutions to search.

While the development thus far shows great promise, some improvements can still be made to make the algorithm more efficient. A study of optimization techniques should be performed for the base- and second-level optimization phases to determine which combination of optimization schemes are most efficient for a wide range of problems. The gradient-based scheme can also be improved by switching to a more robust and efficient routine.

References
3. R. Jenkins and R. Hartfield, “Hybrid Particle Swarm-Pattern Search Optimizer for Aerospace


Kevin Albarado is an engineer at Dynetics. His research interests include optimization method development, multimodal optimization, evolutionary computing, neural networks, and optimization applications to missile system design/analysis. Albarado has an MS in aerospace engineering from Auburn University. Contact him at albarkm@gmail.com.

Timothy Ledlow is pursuing an MS in aerospace engineering at Auburn University. His research interests include the development and application of optimization methods to complex engineering problems such as missile system design and grid fin optimization. Ledlow has a BS in aerospace engineering from Auburn University. Contact him TWL0001@auburn.edu.

Roy Hartfield is a Walt and Virginia Wolotsz Professor of Aerospace Engineering in the Department of Aerospace Engineering at Auburn University. His research interests include engine development, aerodynamic measurement technology, high-speed air breathing propulsion, rocket propulsion, vehicle aerodynamics, missile system modeling, optimization of aerospace systems, and risk assessment for unmanned aerial vehicles. Hartfield has a PhD in mechanical and aerospace engineering from the University of Virginia. He’s an associate fellow of the American Institute of Aeronautics and Astronautics and a member of the AIAA High Speed Air Breathing Propulsion Technical Committee. Contact him at rjh@eng.auburn.edu.

Kevin Albarado is an engineer at Dynetics. His research interests include optimization method development, multimodal optimization, evolutionary computing, neural networks, and optimization applications to missile system design/analysis. Albarado has an MS in aerospace engineering from Auburn University. Contact him at albarkm@gmail.com.

Timothy Ledlow is pursuing an MS in aerospace engineering at Auburn University. His research interests include the development and application of optimization methods to complex engineering problems such as missile system design and grid fin optimization. Ledlow has a BS in aerospace engineering from Auburn University. Contact him TWL0001@auburn.edu.

Roy Hartfield is a Walt and Virginia Wolotsz Professor of Aerospace Engineering in the Department of Aerospace Engineering at Auburn University. His research interests include engine development, aerodynamic measurement technology, high-speed air breathing propulsion, rocket propulsion, vehicle aerodynamics, missile system modeling, optimization of aerospace systems, and risk assessment for unmanned aerial vehicles. Hartfield has a PhD in mechanical and aerospace engineering from the University of Virginia. He’s an associate fellow of the American Institute of Aeronautics and Astronautics and a member of the AIAA High Speed Air Breathing Propulsion Technical Committee. Contact him at rjh@eng.auburn.edu.

Kevin Albarado is an engineer at Dynetics. His research interests include optimization method development, multimodal optimization, evolutionary computing, neural networks, and optimization applications to missile system design/analysis. Albarado has an MS in aerospace engineering from Auburn University. Contact him at albarkm@gmail.com.

Timothy Ledlow is pursuing an MS in aerospace engineering at Auburn University. His research interests include the development and application of optimization methods to complex engineering problems such as missile system design and grid fin optimization. Ledlow has a BS in aerospace engineering from Auburn University. Contact him TWL0001@auburn.edu.

Roy Hartfield is a Walt and Virginia Wolotsz Professor of Aerospace Engineering in the Department of Aerospace Engineering at Auburn University. His research interests include engine development, aerodynamic measurement technology, high-speed air breathing propulsion, rocket propulsion, vehicle aerodynamics, missile system modeling, optimization of aerospace systems, and risk assessment for unmanned aerial vehicles. Hartfield has a PhD in mechanical and aerospace engineering from the University of Virginia. He’s an associate fellow of the American Institute of Aeronautics and Astronautics and a member of the AIAA High Speed Air Breathing Propulsion Technical Committee. Contact him at rjh@eng.auburn.edu.

Kevin Albarado is an engineer at Dynetics. His research interests include optimization method development, multimodal optimization, evolutionary computing, neural networks, and optimization applications to missile system design/analysis. Albarado has an MS in aerospace engineering from Auburn University. Contact him at albarkm@gmail.com.

Timothy Ledlow is pursuing an MS in aerospace engineering at Auburn University. His research interests include the development and application of optimization methods to complex engineering problems such as missile system design and grid fin optimization. Ledlow has a BS in aerospace engineering from Auburn University. Contact him TWL0001@auburn.edu.

Roy Hartfield is a Walt and Virginia Wolotsz Professor of Aerospace Engineering in the Department of Aerospace Engineering at Auburn University. His research interests include engine development, aerodynamic measurement technology, high-speed air breathing propulsion, rocket propulsion, vehicle aerodynamics, missile system modeling, optimization of aerospace systems, and risk assessment for unmanned aerial vehicles. Hartfield has a PhD in mechanical and aerospace engineering from the University of Virginia. He’s an associate fellow of the American Institute of Aeronautics and Astronautics and a member of the AIAA High Speed Air Breathing Propulsion Technical Committee. Contact him at rjh@eng.auburn.edu.

Kevin Albarado is an engineer at Dynetics. His research interests include optimization method development, multimodal optimization, evolutionary computing, neural networks, and optimization applications to missile system design/analysis. Albarado has an MS in aerospace engineering from Auburn University. Contact him at albarkm@gmail.com.

Timothy Ledlow is pursuing an MS in aerospace engineering at Auburn University. His research interests include the development and application of optimization methods to complex engineering problems such as missile system design and grid fin optimization. Ledlow has a BS in aerospace engineering from Auburn University. Contact him TWL0001@auburn.edu.

Roy Hartfield is a Walt and Virginia Wolotsz Professor of Aerospace Engineering in the Department of Aerospace Engineering at Auburn University. His research interests include engine development, aerodynamic measurement technology, high-speed air breathing propulsion, rocket propulsion, vehicle aerodynamics, missile system modeling, optimization of aerospace systems, and risk assessment for unmanned aerial vehicles. Hartfield has a PhD in mechanical and aerospace engineering from the University of Virginia. He’s an associate fellow of the American Institute of Aeronautics and Astronautics and a member of the AIAA High Speed Air Breathing Propulsion Technical Committee. Contact him at rjh@eng.auburn.edu.

Kevin Albarado is an engineer at Dynetics. His research interests include optimization method development, multimodal optimization, evolutionary computing, neural networks, and optimization applications to missile system design/analysis. Albarado has an MS in aerospace engineering from Auburn University. Contact him at albarkm@gmail.com.

Timothy Ledlow is pursuing an MS in aerospace engineering at Auburn University. His research interests include the development and application of optimization methods to complex engineering problems such as missile system design and grid fin optimization. Ledlow has a BS in aerospace engineering from Auburn University. Contact him TWL0001@auburn.edu.

Roy Hartfield is a Walt and Virginia Wolotsz Professor of Aerospace Engineering in the Department of Aerospace Engineering at Auburn University. His research interests include engine development, aerodynamic measurement technology, high-speed air breathing propulsion, rocket propulsion, vehicle aerodynamics, missile system modeling, optimization of aerospace systems, and risk assessment for unmanned aerial vehicles. Hartfield has a PhD in mechanical and aerospace engineering from the University of Virginia. He’s an associate fellow of the American Institute of Aeronautics and Astronautics and a member of the AIAA High Speed Air Breathing Propulsion Technical Committee. Contact him at rjh@eng.auburn.edu.
Vertical Integration

Matthew Turk | National Center for Supercomputing Applications

The only Web browser tab I always have open is Gmail. It serves as the center of my digital life. When I receive a new email with a Google Docs document linked in it, there’s a preview of that document—when I send one to someone else, Gmail first checks to see if the person receiving it has permission to view it and lets me know if they don’t. If I have pictures uploaded to my account, I can insert them into an email without re-uploading. The apps that Google provides share data effectively (thanks to a controversial terms-of-service change from a few years ago) and this enables a lower overhead to deploying new features and disseminating those features to users. My friends that use iOS products find similar smoothness of integration between different apps provided by Apple, but Apple also deploys a strategy like this by developing the hardware, operating system, and software that are used by their devices.

The term vertical integration refers to a situation where one vendor controls many different components of the same supply chain. In Google’s case, the chain is mostly software-based, whereas in Apple’s case the chain is from assembly to use of a product. While these two companies in particular are often criticized for their vertical integration strategies, as it can also be a form of vendor lock-in, it provides measurable value for them in ensuring users are retained in their existing and future ecosystem.

Development of scientific software is increasingly seen as closer to conventional business models than was previously realized. Books like Marketing for Scientists by Marc Kuchner, social media outreach for scientific software projects, and a newfound focus on design and outreach have underscored the relationship between mindshare and developing strong scientific collaborations. The trend toward vertical integration, while perhaps not always recognized as such, has long been present in scientific software—although often termed feature creep, rather than a conscious strategy.

In my own day-to-day work, I see this, too. (And calling it vertical integration instead of feature creep makes me feel better about it when I’m the one doing it!) Particularly when prototyping a new feature, the barrier to entry to adding that new feature to an existing codebase is much smaller than the barrier of creating a new project. On top of this, the problem of dissemination is drastically reduced. As long as software packaging remains the challenge it is today, this will be a strong motivator for feature creep.

Along these lines, a few months ago my co-editor Konrad Hinsen posted a blog entry describing the changes he found in NumPy, a numerical Python library, over the years of its existence (see http://khinsen.wordpress.com/2014/09/12/the-state-of-numpy). Recently, NumPy dropped support for a particular, somewhat obsolete API from early in its history; this has resulted in broken code that needs to be updated. NumPy benefits from being a library, from being decoupled from the updates of packages that rely upon it. Contrast this with an integrated set of packages, where the dependencies and the underlying software were released in lockstep. In that case, either the necessary APIs would likely not be removed, or their removal would be accompanied by an upgrade to all dependent packages. Yet, if every application developer had chosen to write his/her own implementation of an array library, the NumPy/SciPy ecosystem would simply not exist—sharing data between libraries would be nearly impossible. If you’ve attempted to share data between libraries in C/C++ or Java, you may have experienced forms of this difficulty, as different libraries assume different conventions for arrays.

So Why Should We Vertically Integrate?

The Sage project (sagemath.org) ships a bundle of all of the dependent software packages in a single suite; this enables them to control the versions distributed, avoid breakages, and perhaps most importantly it essentially solves the problem of distributing a diverse set of packages to all major platforms. If they need to develop a custom version of something, they can ship a modified version of the package with their distribution. This strategy will help them control the user’s experience, ensuring consistency. This particular example somewhat blurs the line between bundling and vertical integration.
Jamie Zawinski famously said, “Every program attempts to expand until it can read mail.” As the infrastructure of a program grows, it becomes very tempting to bolt on new features. In the life of an imaginary scientific program, someone might develop helper functions for arrays—why manually loop over elements, when this function can be overloaded and written only once? And while you’re at it, why not add on some derivative functions, and then even routines to easily serialize these arrays? Each of these will be tuned to your exact use case, with a minimum of features, and then new developers to your project will have access to them as a given, as part of the core routines they can build on.

This lends itself to dividing the broad types of software into applications (mainly meant to be used by an end user), libraries (mainly meant to be linked into other applications and combined with other libraries to create an application), and what could broadly be called platforms or environments, within which applications and libraries can be assembled. The vertically integrated stack is well-suited to the realm of environments, like programming languages (Python, for instance, is known for coming “batteries included”) and domain-specific dialects (Sage), where the ecosystem is somewhat controlled, somewhat centrally defined, and where inserting new components into that ecosystem can be moderately challenging.

Providing individuals in an “environment” with the ability to call upon a wide variety of standard, integrated, and deeply coupled components encourages them to invest in that environment, that ecosystem. This encourages them to continue to grow that software ecosystem.

And yet, libraries are often the worst choice to be vertically integrated—libraries by their very nature are meant to be components in broader ecosystems, to be the components out of which application developers compose their applications. Feature creep brings with it the danger of conflicts, and perhaps most challengingly, the risk of preferred solutions to non-essential functionality. As libraries begin to make decisions—for instance, a networking library deciding in what form bits should be stored on disk—that impacts other areas of the calling code’s responsibilities, so that impedance mismatches and minor conflicts may creep in along with those features. This can have secondary effects, but it might also cause application developers to reconsider the heavyweight approach in favor of something considerably lighter. Often this will lead to application developers taking the vertical integration approach on their own!

So Why Shouldn’t We Vertically Integrate?

In my mind, the biggest threat that comes with vertical integration—which has plagued Apple, Python, and Google—is that some components are just kind of low-quality, or worse, unreliable. If one component, upon which individuals have come to depend, is prone to failure, or fails to meet standards and needs, it can result in individuals getting frustrated and either loudly disparaging or even outright abandoning the ecosystem. As an example, the Python standard library—long held up as a shining example of the batteries-included approach Python took to programming—has in recent memory been criticized as stagnant, or as needing much faster turn-around times than are available with the Python core language release schedule. Finding that something exists, and the disappointment that comes from finding out it won’t work for a particular use case, is discouraging.

Furthermore, particularly in scientific software, projects suffer from a dearth of resources. The creeping temptation to vertically integrate, to add on new components and continue to develop outward from a software ecosystem’s core, runs the risk of spreading yourself further and further. The counterexample, of course, is when the new features end up injecting a considerable amount of energy into a community—witness the growth of IPython (now Jupyter) following the development and widespread deployment of the Web notebook that now widely drives adoption. The greater the number of subprojects, components, and elements that are “insourced” to a project, the deeper the commitment from contributors and developers required to continue the advancement of the project as a whole.

Is Scientific Software “Different”?

Two attributes of scientific software producers and consumers provide some distinctions from traditional business or consumer software. Scientific software producers and scientific software consumers are in general much closer in skillsets and interests than industrial software developers are with traditional consumers; this means that individuals presented with a poorly performing or otherwise rickety stack of software are more likely to look at it and say, “Well, I could do better, I don’t need this.” The buy-in cost is much different for a piece of scientific software than for, say, an operating system. This can work either for or against the notion of vertical integration; a well-designed, smooth stack of software is less likely to push people away, but the larger the stack, the more overwhelming and off-putting it can be.

This leads into the other looming difference: motivations and rewards for scientific software developers are centered around mindshare and attribution, and much less around revenue streams. Seeking mindshare requires both being of value and being deeply ingrained, but not so ingrained as to be invisible. And yet, when revenue is relevant—for instance, licensing software or services for scientific software to universities—it can create incentive for developers and discourage for consumers to use the software.

In reading over this article, as I prepare to submit it, I realize that it’s really an exploration of the choices available, more than a direct opinion piece. There’s no easy answer, no quick discussion of the right way to develop software, particularly
scientific software, but there are many different viewpoints to consider. I have pursued vertical integration in some of my software projects—and occasionally, this was successful. In other projects, it furthered distraction and resulted in (to paraphrase Larry Page) less wood behind more arrows. But perhaps writing this will give me perspective to avoid such mistakes? Time will tell.

These are a few of my thoughts, but I invite readers to discuss them further, among their collaborations, or to send me your thoughts directly. What are the tradeoffs between providing a full stack, versus components? What are the drawbacks?

Matthew Turk is a research scientist at the National Center for Supercomputing Applications (NCSA) at the University of Illinois and a research assistant professor in the Astronomy department at the University of Illinois, Urbana-Champaign. His scientific research interests are focused on the way the first stars in the Universe formed, how they interacted with and transformed their environments, and how the fundamental mechanism by which stars form changed from that first generation to subsequent generations. His interests in computational science and discovery are focused on enabling deeper questions to be asked of data, and how to broaden the reach of software communities and collectives in science. Turk has a PhD in physics from Stanford University. Contact him at matthewturk@gmail.com.

Selected articles and columns from IEEE Computer Society publications are also available for free at http://Computing-Now.computer.org.
NEW TITLE FROM IEEE CS Press

SCIENTIFIC FREEDOM & HUMAN RIGHTS
Scientists of Conscience During the Cold War
by Jack Minker

This memoir is a first-hand account of computer scientists working to safeguard the human rights of scientists throughout the world.

“It is not very often that solidarity among scientists is brought to the public eye, and it is certainly not common for people outside science to associate scientists with heroic struggles for human rights, freedom, and dignity. Jack Minker’s new book will change this perception.”
—Professor Judea Pearl, UCLA

978-0-7695-4660-5 • June 2012
Paperback • 562 pages • $29.95
An IEEE Computer Society Press Publication

To Order

Save 15%

Order Online
Enter code DZWE5FVE
for 15% off!* 

Also available on
http://amazon.com

*Note: discount code only valid on createspace.com
The articles appearing in CiSE are the result of hard work by many people. We deeply appreciate the efforts of everyone who reviewed the many articles submitted to CiSE last year. The peer review process helps maintain the magazine’s revered quality. All of us in the computational science community owe gratitude to people who participate in this crucial service. Readers who would like to contribute as reviewers can visit computer.org/cise-reviewers to find out how they can get involved.

—CiSE’s Editorial Board and Staff

A
Andreas Adelmann
Yuri Alekseev
Ann Almgren
Rommie Amaro

B
Ramesh Balakrishnan
Wolfgang Bangerth
Blaise Barney
Steven Barrett
Klaus Bartschat
Andrew Benson
Sarah Benziane
David Bernholdt
Martin Berzins
Allen Brookes

D
Romaric David
Carleton De Tar
Thomas Deboni

E
Stephen Elbert
Thomas Epperly

F
Johannes Feist
Hal Finkel
Juliana Freire
REVIEWER THANKS

Robert Sugar
Yuji Sugita
Godehard Sutmann
Alexender S. Szalay

V
Alvaro Vazquez-Mayagoitia
Venkat Vishwanath

T
Wei Tan
Kristin Tolle
Matthew Turk

W
Bei Wang
Fusheng Wang
Timothy Williams

U
Tom Uram

Y
Chen Yung-Fu

IEEE Intelligent Systems provides peer-reviewed, cutting-edge articles on the theory and applications of systems that perceive, reason, learn, and act intelligently.
Focus on Your Job Search

**IEEE Computer Society Jobs** helps you easily find a new job in IT, software development, computer engineering, research, programming, architecture, cloud computing, consulting, databases, and many other computer-related areas.

**New feature:** Find jobs recommending or requiring the IEEE CS CSDA or CSDP certifications!

Visit [www.computer.org/jobs](http://www.computer.org/jobs) to search technical job openings, plus internships, from employers worldwide.

http://www.computer.org/jobs

---

The IEEE Computer Society is a partner in the AIP Career Network, a collection of online job sites for scientists, engineers, and computing professionals. Other partners include Physics Today, the American Association of Physicists in Medicine (AAPM), American Association of Physics Teachers (AAPT), American Physical Society (APS), AVS Science and Technology, and the Society of Physics Students (SPS) and Sigma Pi Sigma.
Modern Computing—in 1949

Computational scientists have always pushed machines to their processing limits, and not just for the heady thrill of it all. A calculation performed today on an old and slow machine might have already been done by someone else back when the same machine was new and fast.

To meet the need for speed, computational scientists have striven to squeeze the most out of their machines—which raises the question, What can we learn from the computational scientists of the past, whose puny machines demanded ingenious and frugal programming?

For an answer to that question, I searched Physics Today’s back issues for articles about computing. The oldest I found was Robert Richtmyer and Nicholas Metropolis’s “Modern Computing,” which appeared in August 1949, eight months before the magazine’s second anniversary.

Both men were pioneers of computational science. The theorem that Richtmyer derived with Peter Lax defines the conditions under which finite-difference methods converge. The most famous application of the Monte Carlo method bears Metropolis’s name.

Perhaps because computers were so new and rare in 1949, Richtmyer and Metropolis focused on explaining how computers work and what they can do, rather than on teaching Physics Today’s readers how to use one. Remarkably, if you discount the authors’ reference to vacuum-tube technology, their opening paragraph retains its relevance:

The intricacies of automatic computing methods have been popularized by pictures, visual and verbal, of complicated wiring diagrams, great banks of electron tubes, and dramatic control boards, as well as by certain romantic analogies between the machines and the human brain. There remains, however, a need for defining the limits of computing machine operation, as well as its promise.

In the 75 years since Richtmyer and Metropolis wrote their article, the frontiers of computational science have expanded greatly but not limitlessly. Operational constraints imposed by the bulky size of vacuum tubes have been superseded by operational constraints imposed by the tiny sizes of interconnects and capacitors. Communicating effectively with a computer continues to be a challenge. The first high-level programming language written for an electronic computer, Short Code, was conceived by John Mauchly in 1949 and implemented the following year by William Schmitt. The concept of a programming language is not mentioned in Richtmyer and Metropolis’s article, but it’s foreshadowed by their discussion of basic vocabularies, degrees of inflection, and variable addresses.

As you might guess, Richtmyer and Metropolis end with speculations about the future of computational science. Two of their predictions were far out of reach in 1949, yet not so visionary that they were inconceivably difficult to realize. Solving algebraic equations on a computer was achieved in the 1960s by Tony Hearn, Martinus Veltman, and other pioneers of symbolic manipulation. And in 1969, satisfying Richtmyer and Metropolis’s hope that computers be applied to biological systems, Michael Levitt and Arieh Warshel computed the energy landscape of a single protein for the first time.

Charles Day is the Web editor at Physics Today.
COMPSAC is the IEEE Signature Conference on Computers, Software, and Applications. It is a major international forum for academia, industry, and government to discuss research results, advancements and future trends in computer and software technologies and applications.

With the rapidly growing trend in making computation and data both mobile and cloud-based, such systems are being designed and deployed worldwide. However, several challenges arise when they are applied to different domains or across domains. COMPSAC 2015 will provide a platform for in-depth discussion of such challenges in emerging application domains such as smart and connected health, wearable computing, internet-of-things, cyber-physical systems, and smart planet.

Important Dates:
January 17, 2015: Paper submissions
March 15, 2015: Paper notifications
April 28, 2015: Camera-ready version and registration due

Contact ITIP Symposium organizers at cs_compsac-ITIP-2015@iastate.edu

CALL FOR PAPERS
July 1-5, 2015
Tunghai University
Taichung, Taiwan

COMPSAC 2015 Symposium on IT in Practice (ITIP)

This new COMPSAC symposium on IT in Practice (ITIP) is directed towards the interests and needs of IT practitioners, application developers, IT managers, and CXOs from business, industry, and government, as well as academics and researchers. The symposium, co-sponsored by IT Professional, is an interactive forum for sharing and gaining valuable insights and experiences and for active engagement among technical professionals around the globe. It aims to inspire professionals, executives, and researchers to focus on diverse elements necessary to successfully architect, design, engineer, configure, and field a range of IT applications in several domains. Topics of interest include the following:

- Innovative & challenging IT applications
- Deploying and managing large scale IT systems
- Real world issues, concerns & challenges
- Development & operational experiences
- IT application design & validation
- Emerging trends in enterprise IT
- Strategic alignment of IT
- IT workforce & IT workplace
- Managing IT in practice
- Opportunities & challenges of C-level positions
- Operational issues & insights
- Crowd sourcing problems & solutions
- The research-practice nexus
- IT-enabled business innovation
- Case studies
- Future perspectives

Authors are invited to submit original, unpublished research work and novel computer applications in full-paper format. Simultaneous submission to other publication venues is not permitted. The review and selection process for submissions is designed to identify papers that break new ground and provide substantial support for their results and conclusions as significant contributions to the field. Submissions will be selected that represent a major advancement in the subject of the symposia to which they are submitted. Authors of submissions with a limited contribution or scope may be asked to revise their submissions into a more succinct camera-ready format; e.g., workshop paper, fast abstract, or poster to the main conference.

ITIP Symposium Co-Sponsored by:

www.compsac.org
3D Printing Will Actually Change the World! Are You Ready?

Every company needs to prepare and implement 3D printing in order to remain relevant in their industry! No one can sit this phenomenon out!

Get ready at Rock Stars of 3D Printing, a one-day event featuring the experts, early adopters, and visionaries that are driving this revolution.


Here’s a list of other Rock Star speakers for Rock Stars of 3D Printing:

- Paul Brody, Vice President and Global Industry Leader of Electronics, IBM
- Brian David Johnson, Futurist and Director, Future Casting and Experience Research, Intel
- Cliff Waldman, Council Director and Senior Economist, Manufacturers Alliance for Productivity and Innovation

17 March 2015
The Fourth Street Summit Center
San Jose, CA

REGISTER NOW
Early Discount Pricing Now Available!

computer.org/3dprinting