Journal of
open research software

Choi, S-C T 2014 MINRES-QLP Pack and Reliable Reproducible Research via Supportable Scientific Software. *Journal of Open Research Software,* 2(1): e22, pp. 1-7, DOI: http://dx.doi.org/10.5334/jors.bb

ISSUES IN RESEARCH SOFTWARE

MINRES-QLP Pack and Reliable Reproducible Research via Supportable Scientific Software

Sou-Cheng (Terrya) Choi*

The MINRES-QLP Pack is a suite of standard and extended Krylov subspace methods for solving large linear systems and linear least-squares problems in which the coefficient matrices are potentially singular or ill-conditioned and possibly have special symmetries. Our purpose is to develop robust open-source MATLAB implementations of these algorithms that are faithful to the theory, following the philosophy of reproducible research (RR), and practicing development principles of what we call supportable scientific software (SSS) that promote reliable reproducible research (RRR). In this paper, we review key features in the ongoing theoretical and software development of our algorithms in the MINRES-QLP Pack. We highlight the most effective software engineering tools known to us that are potentially useful to other scientific research areas. We support open calls to create more incentives for practitioners of robust and reliable scientific software such as citations and grants for quality software. We encourage introducing principles of RRR via SSS to computational science students in advanced courses of scientific computing and to computational scientists through seminars, workshops, or conferences. To these ends, we started an experimental seminar course, "Reliable Mathematical Software" (IIT MATH-573) in our institution, and organized multiple sessions on "Reliable Computational Science" in the SIAM Annual Meeting 2014. We share our research practice and pedagogic experiences in this article.

1 Introduction

Reproducible research (RR) in computational sciences was pioneered by Claerbout in the Stanford Exploration Project (SEP) [1]. It has since been championed by Claerbout, Donoho, and their collaborators in their research areas of geophysics, signal and image processing [2, 3, 4]. The leading advocates for RR include Gentleman and Peng in biostatistics [5, 6], Koenkera in economics, LeVeque in numerical partial differential equations [7], and Stodden in legal frameworks that enable RR [8]. The defining principle of RR [3] is "When we publish articles containing figures which were generated by computer, we also publish the complete software environment which generates the figures." Clearly there is an inevitable dependency of RR to software, hardware, code, and data.

This article has two main messages. First, we contend that worthy RR in modern computational sciences have to be built upon strong mathematical, statistical, and approximation theories leading to rigorous analysis of statistical and numerical errors, as well as computational costs. Second, we establish our position that RR can be made substantially more reliable—hence reliable reproducible research (RRR)—by development of supportable scientific software (SSS), which we define as *a conceptual framework that encompasses a collection of software development methods for promoting the reliability of reproducing provably correct results in computational sciences.* We note that our discussion of the SSS framework is not restricted to any particular programming languages.

While the notions of RRR and SSS are not entirely new, they are not as often observed as hoped. Relatively few computational scientists seem to practice RR as reflected in recent survey results put together by Flemisch [9]. It is also plausible that the community is not as a whole well aware of the importance of reproducibility. When SSS is absent, RR risks remaining an ideal. On the other hand, conscientious adoption of the principles can lead to substantially more trustworthy research results.

More specifically, in terms of theory, we have a relatively new suite of algorithms we call MINRES-QLP Pack, for solving singular or ill-conditioned linear systems of equations or linear least-squares problems with guaranteed accuracy. In terms of practice, we do what we advocate; we borrow a number of strategies and tools from industrial software engineering that lead to more reliable RR. While this paper focuses on the MINRES-QLP Pack to illustrate how to achieve RRR via SSS, we and our collaborators have at least two more suites of accuracy-guaranteed algorithms developed using similar approaches: First is the Guaranteed Automatic Integration Library (GAIL) [10, 11] for numerical integration in one or many dimensions based on the recent groundbreaking work of Hickernell et al. [12, 13]. Second is the Open-Source CIM-EARTH

* NORC at the University of Chicago, Department of Applied Mathematics, Illinois Institute of Technology, USA
sctchoi@uchicago.edu

Framework (OSCEF) for solving large-scale computable general equilibrium (CGE) models [14, 15] built on the design of CIM-EARTH [16, 17, 18].

The outline of this paper is as follows. The next section gives a compressed description of the algorithms in the MINRES-QLP Pack. Section 3 recapitulates principles and benefits of RR, including examples of high-quality software and associated highly cited publications. In Section 4, we draw from our computational research experience and highlight a number of strategies and tools for robust software package development. Section 5 covers some existing but rare incentives and our initiatives on conducting a weekly seminar course and organizing themed meetings, which serves to educate or exchange ideas with computational researchers about RRR and SSS. In the last section, we conclude with thoughts for future work.

2 MINRES-QLP Pack

Most Krylov methods for solving large linear systems or linear least-squares problems have pervasive applications in science and engineering fields. Nonetheless, they usually assume nonsingular or full-rank matrices (or linear operators that are not explicitly represented as matrices). These methods are generally divided into two classes: Those for symmetric matrices (e.g., conjugate gradient, MINRES, SYMMLQ) and those for unsymmetric matrices (e.g., BCG, GMRES, QMR, BICGSTAB, LSQR, and IDR(s)). Such a division is largely due to the fact that historically the most prevalent matrix structure from practical applications is symmetric. However, other types of symmetry structures, notably complex symmetric, skew symmetric, and skew Hermitian matrices, are becoming increasingly common in modern applications. Often, these are treated as general unsymmetric matrices. In contrast, our algorithms take advantage of the symmetries to minimize computational costs such as memory and arithmetic operations.

On a singular linear system such as

$$\begin{bmatrix} 1 & 1 \\ 0 & \varepsilon \end{bmatrix} x = \begin{bmatrix} 1 \\ 1 \end{bmatrix},$$

where ε is the machine precision, the pseduoinverse solution is

$$x^{\dagger} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$

Most of MATLAB's Krylov solvers would abort or return an exploding solution. To carefully handle singularity and exploit various symmetry structures, we have designed a suite of MINRES-QLP [19, 20, 21, 22] algorithms, which can constructively reveal (numerical) singularity and compatibility of a given linear system of equations; users do not have to know these properties a priori. (We are also currently developing two related algorithms known as GMRES-QLP and GMRES-URV for *unsymmetric* singular square systems; see [19, 23].)

The key automatic steps of the MINRES-QLP algorithm are: **First**, detect special symmetry with an efficient statistical test. **Second (optional)**, construct a symmetric positive definite preconditioner M whose number of distinct nonzero eigenvalues $M^{-1/2}AM^{-1/2}$ is less than that of A and hence number of iterations required in the next stage are reduced. Without loss of generality, let M be the identity matrix in our subsequent discussion. **Third**, in the kth iteration, for $k = 1,\ldots,\min\{p, K\}$, where p is the smallest positive integer such that $\{b, Ab,\ldots, A^p b\}$ is linearly dependent, and K is a user-input positive integer, the algorithm searches for kth in the kth Krylov subspace $K_k(A, b) = (b, Ab,\ldots, A^{k-1}b)$, where x_k is the minimum-length element of minimal-residual solutions for the problem

$$\min_{x \in k_k(A,b)} \|Ax - b\|_2 .$$

It is known that x_k exists and is unique. If either residual norms $||r_k||_2 = ||b - Ax_k||_2$ or $||Ar_k||_2$ is smaller than a scalar multiple of a user-given tolerance, then the algorithm returns x_k as an approximant; otherwise the third step is repeated with k incremented by one. We note that the square root $M^{-1/2}$ is not explicitly constructed and all the quantities such as x_k, $||r_k||_2$, and $||Ar_k||_2$ are accurately and efficiently computed by short recurrences.

In the MINRES-QLP Pack, our goals are to have MATLAB routines well-documented, well-tested, optimized for speed, accompanied by examples, and stored in a repository [24] where they can be modified, extended, and re-tested as needed by the originators or any public members. Over time we will work to improve existing algorithms and add new methods in the package. If resources permit, we will implement them in other languages such as Fortran 90/95 [21] and PETSc [25], which is a parallel mathematical library capable of peta-scale computations.

3 Reproducible research and challenges

RR for modern computational sciences as originally proposed by Claerbout is not simply a form of ideological rhetoric. It actually has rather specific instructions. Claerbout encouraged every author of computational research papers to mark each figure with the tags [**ER**] for "easily reproducible," [**CR**] for "conditionally reproducible," or [**NR**] for "non-reproducible" to indicate the extent of reproducibility. To qualify for [**ER**], a figure is generated and accompanied by programs with parameters and input data, as well as "makefiles executable on Linux and portable to other operating systems." A conditionally reproducible figure is often time consuming to reproduce from scratch and depends on the availability of computational resources such as memory, commercial applications, or large input datasets. Non-reproducible figures may include images drawn or photographs taken—not computed—to illustrate scientific ideas. Today there is no reason not to extend the tags to other computational results or online resources such as interactive figures, video clips, tables, or online databases produced or referenced by a report [26].

Paraphrasing Claerbout, Buckheit and Donoho [3] stated that "an article about computational result is *advertising*, not scholarship. The actual **scholarship** comprises the **full software environment**, **code** and data, that produced the result." Donoho et al. produced multiple MATLAB packages, WaveLab (1995), BeamLab (2004),

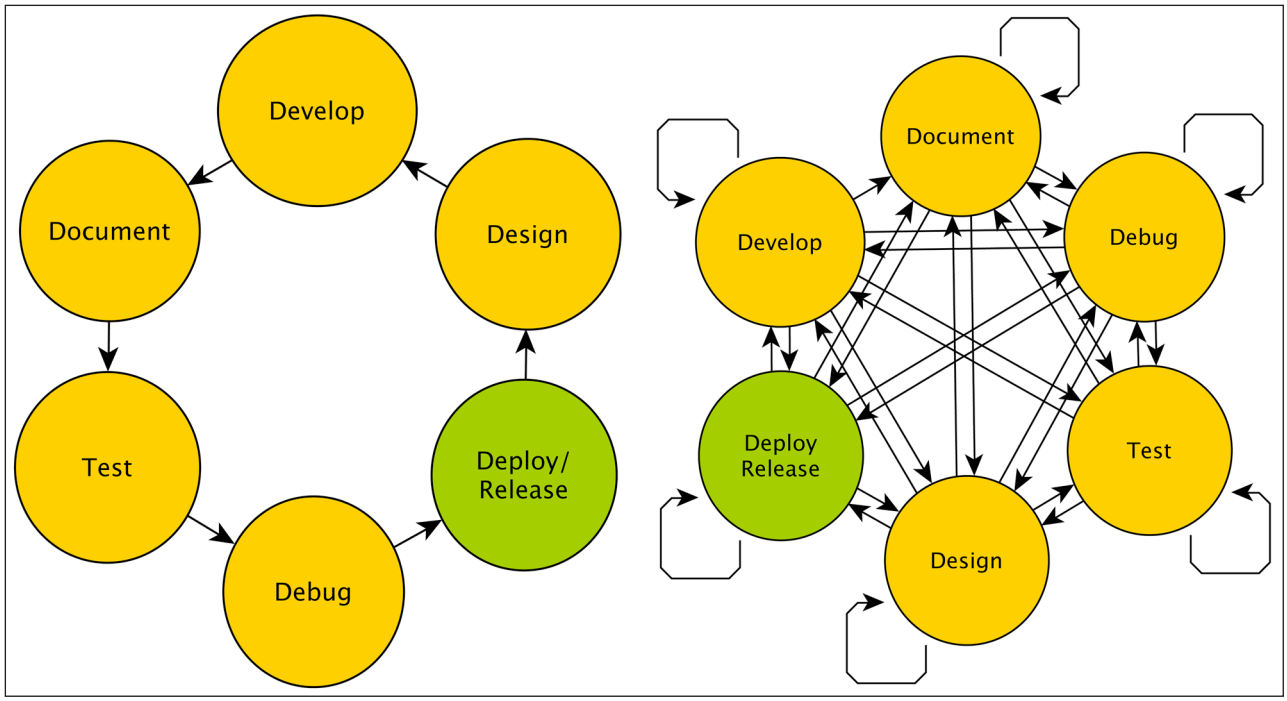

Figure 1: Left: A standard software development cycle. **Right:** Software development cycles in reality.

SparseLab (2007), and MCALab (2008). Some of the most highly cited papers Donoho co-authored such as [27, 28] are accompanied by well-crafted MATLAB software. It is reasonable to posit that reproducible research with quality code contributes significantly to the citation statistics of the associated papers.

Nevertheless, we ask in practice, how reliable is RR? It would be at most as dependable as the strength and availability of its underlying theories, data, code, software, and hardware. Obviously, it is a "decreasing function" of time due to emergence of new software (versions), phasing out of old computer architecture, unaffordability of commercial software, or lack of quality software. Often, authors release only partial data, code fragments, or incomplete environment specification. It is not uncommon to see loss of data or code due to insufficiently frequent backup or unstable transfer processes. Algorithms of suboptimal design, with insufficient documentation, inadequate testing, and infrequent release of bug fixes—all understandable due to limited human resources or expertise—may render the computational software practically unusable even to the original creators not too much later in time. Modern complex applications with big data and big code, as well as emphasis on global team collaboration only add to these difficulties.

The process of software development by itself is often exceedingly challenging for programmers with moderate or even abundant experience. A common software development cycle such as the one shown on the left of **Figure 1** is oversimplified. In practice, the "disorder" on the right of the same figure depicts much more accurately the potential multitude of cycles and transitions among the key phases. In the next section, we show how to restore partial order to the process via what we call the SSS principles.

4 Supportable scientific software: principles and tools

Flemisch's 2013 survey [9] on reproducible research received feedback from only about 10% of targeted recipients. Among those who responded, more than 70% reported that the efforts to reproduce own or others' computational results are between average to very high. The survey results also convey that RR is perceived as "unrewarding" hard work. The survey suggested the central roles of strategic tools such as the use of a repository (e.g., SVN, CVS, GIT, Mercurial). The survey results are in line with our understanding and experiences. Despite the burden of RR, we maintain our position and belief in the importance of enhancing its reliability.

We define supportable scientific software (SSS)[1] as a conceptual framework that encompasses a collection of software development methods for promoting the reliability of reproducing provably correct results in computational sciences. There are four critical elements of SSS [29]:

- Justified: Scientific software should be based on theoretical guarantees that specify sufficient preconditions leading to success in satisfying error tolerance or uncertainty level. There are lower and upper bounds of computational cost functions of input size n in terms of memory or storage, as well as computational and communication time (these functions are preferably low-degree polynomials of n).
- Efficient: In theory, complexity analysis is an important tool for gauging efficiency. Nevertheless, its nature is asymptotic. Even polynomial-time algorithms can be costly if they have large leading coefficients or are of high degrees. In practice, a profiler can be utilized to measure and pinpoint code lines that consume the most resources, such as time and memory. Code reuse

is another measure of efficiency; it can be achieved through refactoring of duplicate code patterns into functions to save development time.

- Robust/Durable: A robust software package is tested under a wide variety of conditions and ranges of inputs. The design of a robust software package should ideally be durable under changing computing hardware and software environments. In addition, we recommend persisting and maintaining source code and research data via repositories with version control.
- Intelligible: At the minimum, we expect clear documentation of key algorithmic function and steps, inputs and outputs, parameters, usage examples, and related references.

From the practice of reproducible research in our work, we affirm that *RRR via SSS can be done*. Many rudimentary examples already exist even though it is probably extremely hard to find "a general solution" for a given scientific field. The tools we highlight in **Table 1** are available with MATLAB and similar counterparts can be found with other modern programming languages such as C++, Java, and Python, enabling researchers to develop functional and polished algorithms.

We recommend the use of a code repository for every project from the beginning. We also employ Integrated Development Environment (IDE) software. In the case of MATLAB, it has its own IDE built in Java, with visually appealing automated reports on code dependency, documentation evaluation, and performance analysis, which together can help pinpoint at line level or character position where the code base can be improved.

We emphasize on the importance of designing simple application programming interfaces (APIs) from the outset. If we were allowed to communicate only one thing to expert or application users about our implementation, our choice would be its API, which consists of function name, required and optional inputs, outputs, and our parsing schemes. Our algorithms parse user inputs and ensure that they fall in the correct ranges; in some cases we supply default values for missing optional parameters and gently correct invalid input values. We craft detailed and readable documentation in both text and searchable HTML formats with specification of APIs, syntax, default input values, examples of usage, and further references.

Lastly, to ensure the correctness of computational results from our software and examples in the documentation, we create unit tests [30], doctests [31], and stress tests. An effective unit test or doctest would be demonstrative of API usage, and run in less than a small fraction of a second. Without these tests in place, our experience is that even minor changes to our code or documentation could quickly and unknowingly depart from the original design and theory. These two kinds of tests effectively safeguard most changes we make to our algorithms. We adhere to the following test practices:

Every time before code is checked into the repository, unit tests and doctests are run.
Every time a bug is found, unit tests are expanded.

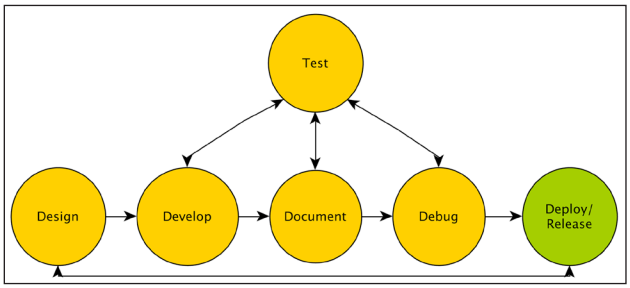

Figure 2: Test-driven software development cycles.

All unit tests and doctests have to pass at 100% success rate for each code check-in and software release. The diligent execution of an expanding test suite serves to verify and strengthen the software functions, and to simplify the software development cycles; see **Figure 2**. On the other hand, stress tests check if the software performs under load.

5 Incentives and educational efforts for RRR and SSS

Practitioners of RRR and SSS need more incentives in terms of specialized software citation [32], comprehensive journal review policies for software publication such as ACM TOMS' [33], and research funding schemes such as NSF's SSE & SSI program [34] that gives significant weightage to quality scientific software infrastructure at all scales. We urge members of the scientific community to cite high-quality software and online resources; the following are two examples of referencing the software GAIL [10] and MINRES-QLP community website [24] in BibTeX format:

```
@misc{CDHJZ14gail,
   title={{GAIL}: {G}uaranteed {A}utomatic {I}
   ntegration {L}ibrary (Version 1.3),
   {MATLAB} Software},
   author={Choi, Sou-Cheng T. and Ding, Yuhan and
   Hickernell, Fred J. and Jiang, Lan and Zhang,
   Yizhi},
   year={2014},
howpublished={\url{http://code.google.com/p/gail/},
}

@misc{MinresqlpWebsite,
   author={Choi, Sou-Cheng T.},
   title={{MINRES-QLP} Project and Community
   Website},
howpublished={\url{http://code.google.com/p/
minres-qlp/}},
   year={2013},
   notes={Website},
}
```

To maximize quality software's exposure and usage, we support free software and open source movement, but are also open to having our software included in quality commercial software. There are also viable open source packages sustained by provision of commercial services.

In Summer 2013, we started a team devoted to development of a MATLAB research software package GAIL [11] for numerical integration with guarantee of accuracy. Our members role play a team of software engineers with specializations in three of the critical software functions, namely testing, documentation, and release. We make use of a private wiki to strengthen team communication, recording key steps of important scientific routine process and procedures; maintaining checklists of day-to-day

	Justified	Efficient	Durable/Robust	Intelligible
Repository			✓	
Editor (IDE)			✓	✓
Code Analyzer		✓	✓	✓
Help Report, Contents Report				✓
Searchable HTML documentation				✓
Doctest			✓	
Unit Testing Framework			✓	
Debugger	✓	✓	✓	
Profiler		✓	✓	

Table 1: Tools compatible with or available in MATLAB 2014a that can be employed to enhance the defining elements of SSS.

tasks; and sharing of intermediate results, knowledge, and expertise. We prepare our students of applied mathematics to become practitioners of RRR via SSS. As a beneficial side effect, as they gain software engineering skills that are usually obtainable only in information-technology driven companies, they become more attractive job candidates. In fact, one of our five student members was employed as a part-time software test engineer the following semester.

We say, "*Those who can, do and teach*." Principles and novel ideas behind the research work would ideally be propagated in beginning and advanced courses in scientific computing, whereas the practices of engineering supportable software promoted to computational scientists and researchers through workshops such as WSSSPE [35, 36] To this end, we started an experimental seminar course, "Reliable Mathematical Software" (IIT MATH-573) [29] in our home institution. We drew teaching materials from exemplary mathematical software such as Chebfun (http://www2.maths.ox.ac.uk/chebfun). The class met weekly for 75 minutes for a total of 10 weeks; each time we demonstrated one or two tools in **Table 1** with motivating scenarios and finished the last two weeks with each student presenting a research algorithm implemented with some of the SSS principles. After the course, a master student completed her thesis and five doctoral students are developing research papers following the principles of RRR via SSS. In addition, we serve the computational community by organizing a two-session minisymposium "Reliable Computational Science" in the upcoming SIAM Annual Meeting 2014 (http://meetings.siam.org/program.cfm?CONFCODE=AN14) and co-authoring a comprehensive summary report for WSSSPE1 [35, 36].

6 Conclusions

We agree with the importance of reproducible research at all scales, and recommend to strengthen it with supportable scientific software development practices. In the short term, RRR via SSS may seem more time consuming. In the medium or long run, the investment of time and efforts gives valuable software a better chance to "survive" and make a greater impact in the research area. Some researchers may actually find that our approach of RRR via SSS provides to some extent a structured course for investigating and attacking complex computational problems. More reliable incremental research can be built on supportable software packages, leading to stably upward spirals of progress in scientific knowledge. As a community, we need more time to reflect, brainstorm, and experiment specific operations. We also need more ongoing dialogue on the topic in the community.

Acknowledgments

The author thanks David Donoho and Ron Yang, from whom she was introduced to the key principles and best practices of reproducible research in production of BeamLab [37] and robust enterprise software development, respectively. She is grateful to Michael Saunders and Chris Paige, her co-authors of MINRES-QLP [20, 21]. She also thanks Fred Hickernell for introducing her to Doctest [31], co-developing the definition of SSS and the principles of RRR via SSS during the development of GAIL [10, 11] and the course *Reliable Mathematical Software* [29]. The author is thankful for the comments and feedback for an earlier version of this work from the reviewers of the JORS and the First Workshop on Sustainable Software for Science [35, 36].

Notes

[1] SSS originally stands for *staunch scientific software* in [29].

References

1. **Claerbout, J** REPRODUCIBLE COMPUTATIONAL RESEARCH: A history of hurdles, mostly overcome. Available at: http://sepwww.stanford.edu/data/media/public/sep/jon/reproducible.html [Last accessed 21 April 2014].
2. **Fomel, S** and **Claerbout, J F** 2009 Guest Editors' Introduction: Reproducible Research. *Computing in Science and Engineering,* 11(1): 5–7. DOI: http://dx.doi.org/10.1109/MCSE.2009.14
3. **Buckheit, J B** and **Donoho, D L** 1995 Wavelab and reproducible research. Springer.
4. **Donoho, D L, Maleki, A, Rahman, I U, Shahram, M** and **Stodden, V** 2009 Reproducible Research in Computational Harmonic Analysis. *Computing in Science and Engineering,* 11(1): 8–18. DOI: http://dx.doi.org/10.1109/MCSE.2009.15
5. **Robert, G** 2005 Reproducible Research: A Bioinformatics Case Study. *Statistical Applications in Genetics and Molecular Biology*, 4(1): 1–25.

6. **Peng, R D** 2011 Reproducible Research In Computational Science. *Science,* 334(6060): 1226–1227. DOI: http://dx.doi.org/10.1126/science.1213847
7. **LeVeque, R J** 2009 Python Tools for Reproducible Research on Hyperbolic Problems. *Computing in Science and Engineering,* 11(1): 19–27. DOI: http://dx.doi.org/10.1109/MCSE.2009.13
8. **Stodden, V** 2009 The Legal Framework for Reproducible Scientific Research: Licensing and Copyright. *Computing in Science and Engineering,* 11(1): 35–40. DOI: http://dx.doi.org/10.1109/MCSE.2009.19
9. **Flemisch, B** 2013 Online Survey: Reproducibility in Computational Science and Engineering. Available at: SIAM-CSE@siam.org
10. **Choi, S C T, Ding, Y, Hickernell, F J, Jiang, L** and **Zhang, Y** 2014 GAIL: Guaranteed Automatic Integration Library (Version 1.3). MATLAB Software. Available at: http://code.google.com/p/gail/
11. **Choi, S C T, Ding, Y, Hickernell, F J, Jiang, L** and **Zhang, Y** 2013 GAIL: Guaranteed Automatic Integration Library (Version 1.0). MATLAB Software. Available at: http://code.google.com/p/gail/
12. **Clancy, N, Ding, Y, Hamilton, C, Hickernell, F J** and **Zhang, Y** 2014 The Cost of Deterministic, Adaptive, Automatic Algorithms: Cones, Not Balls. *Journal of Complexity*, 30(1): 21–45. DOI: http://dx.doi.org/10.1016/j.jco.2013.09.002
13. **Hickernell, F J, Jiang, L, Liu, Y** and **Owen, A B** 2014 Guaranteed Conservative Fixed Width Confidence Intervals via Monte Carlo Sampling. Available at: http://arxiv.org/abs/1208.4318
14. **Choi, S C T** and **Munson, T S** 2014 OSCEF 1.1 APIs. Available at: http://code.google.com/p/oscef/
15. **Choi, S C T** and **Munson, T S** 2014 OSCEF (Open-Source CIM-EARTH Framework), Version 1.1. Available at: http://code.google.com/p/oscef/
16. **Elliott, J, Foster, I, Judd, K, Moyer, E** and **Munson, T** 2010 *CIM-EARTH: Community Integrated Model of Economic and Resource Trajectories for Humankind*. Argonne, Illinois: Argonne National Laboratory. ANL/MCS-TM-307.
17. **Choi, S C T** and **Munson, T S** 2013 *OSCEF: The Open-Source CIM-EARTH Framework User Manual for Version 1.0.* IL: Computation Institute, University of Chicago. ANL/MCS-TM-339.
18. **Choi, S C T** 2014 A Complementarity Approach to Solving Computable General Equilibrium Models. *Computational Economics,* (in review).
19. **Choi, S C T** 2006 *Iterative Methods for Singular Linear Equations and Least-Squares Problems*. ICME, Stanford University.
20. **Choi, S C T, Paige, C C** and **Saunders, M A** 2011 MINRES-QLP: A Krylov subspace method for indefinite or singular symmetric systems. *SIAM Journal on Scientific Computing*, 33(4): 1810–1836. DOI: http://dx.doi.org/10.1137/100787921
21. **Choi, S C T** and **Saunders, M A** 2014 Algorithm 937: MINRES-QLP for symmetric and Hermitian linear equations and least-squares problems. ACM Transactions on Mathematical Software. 2014; 40(2).
22. **Choi, S C T** 2013 *Minimal Residual Methods for Complex Symmetric, Skew Symmetric, and Skew Hermitian Systems*. Chicago, IL: CI, University of Chicago. ANL/MCS-P3028–0812.
23. **Choi, S C T** 2014 Generalized Minimal Residual Methods for Singular Linear Systems or Linear Least-Squares Problems. (working).
24. **MINRES-QLP** 2013 MINRES-QLP Project and Community Website. Available at: http://code.google.com/p/minres-qlp/
25. PETSc: Portable, Extensible Toolkit for Scientific Computation. Available at: http://www.mcs.anl.gov/petsc/ [Last accessed 21 April 2014].
26. **Claerbout, J** and **Karrenbach, M** 1992 Electronic Documents Give Reproducible Research a New Meaning. Available at: http://sepwww.stanford.edu/doku.php?id=sep:research:reproducible:seg92
27. **Chen, S S, Donoho, D L** and **Saunders, M A** 1998 Atomic decomposition by basis pursuit. *SIAM Journal on Scientific Computing,* 20(1): 33–61. DOI: http://dx.doi.org/10.1137/S1064827596304010
28. **Starck, J L, Candès, E J** and **Donoho, D L** 2002 The curvelet transform for image denoising. *Image Processing, IEEE Transactions on,* 11(6): 670–684.
29. **Choi, S C T** and **Hickernell, F J** 2013 IIT MATH-573 Reliable Mathematical Software [Seminal Course]. Available at: http://mypages.iit.edu/~schoi32/MATH573Slides.pdf
30. MATLAB Unit Testing Framework, 2013. Available at: http://www.mathworks.com/help/
31. Doctest, 2010. Available at: http://www.mathworks.com/matlabcentral/
32. Purdue Online Writing Lab, 2013. Available at: http://owl.english.purdue.edu/owl/resource/560/10/.
33. ACM Algorithms Policy. Available at: http://toms.acm.org/AlgPolicy.html [Last accessed 21 April 2014].
34. **NSF 13–525** 2013 Software Infrastructure for Sustained Innovation - SSE and SSI. Available at: http://www.nsf.gov/pubs/2013/nsf13525/nsf13525.htm
35. Workshop on Sustainable Software for Science: Practice and Experiences, 2013. Available at: http://wssspe.researchcomputing.org.uk/
36. **Katz, D S, Choi, S C T, Lapp, H, Maheshwari, K, Löffler, F, Turk, M**, et al 2014 Summary of the First Workshop on Sustainable Software for Science: Practice and Experiences (WSSSPE1). *Journal of Open Research Software,* 2(1):e6. DOI: http://dx.doi.org/10.5334/jors.an
37. **Choi, S C, Donoho, D L, Flesia, A G, Huo, X, Levi, O** and **Shi, D** 2002 About BeamLab—a Toolbox for New Multiscale Methodologies. Available at: http://www-stat.stanford.edu/~beamlab/

How to cite this article: Choi, S-C T 2014 MINRES-QLP Pack and Reliable Reproducible Research via Supportable Scientific Software. *Journal of Open Research Software,* 2(1): e22, pp. 1-7, DOI: http://dx.doi.org/10.5334/jors.bb

Published: 9 July 2014

]u[*Journal of Open Research Software* is a peer-reviewed open access journal published by Ubiquity Press.

OPEN ACCESS