Python Tools for Reproducible Research on Hyperbolic Problems

Randall J. LeVeque[1]

Abstract. Reproducible research in computational science is only possible if the computer codes used to generate published results are archived in a form that can later be used to regenerate the results and can be examined to determine details of the method used. Some difficulties in achieving this goal are discussed. A set of Python tools for facilitating reproducible research on finite volume methods for hyperbolic conservation laws using the Clawpack software are briefly surveyed and illustrated on a sample application.

1 Introduction

Within the world of science, computation is now rightly seen as the third vertex of a triangle, complementing experiment and theory. However, it has yet to reach the maturity of the experimental sciences in terms of the reproducibility of research.

Nowhere else in science can one so easily publish observations that are claimed to prove a theory, or illustrate the success of a technique, without giving a careful description of the methods used in sufficient detail that others can attempt to repeat the experiment. In most branches of science it is not only expected that publications contain such details, it is also standard practice for other labs to attempt to repeat important experiments soon after they are published. Even though this may not lead to significant new publications, it is viewed as a valuable piece of scholarship and a necessary component of the scientific method.

Scientific and mathematical journals are filled with pretty pictures of computational experiments that the reader has no hope of repeating. Even brilliant and well intentioned computational scientists often do a poor job of presenting their work in a reproducible manner. The methods are often very vaguely defined, and even if they are carefully specified they would normally have to be implemented from scratch by the reader in order to test them. Most modern algorithms are so complicated that there is little hope of doing this properly. Many computer codes have evolved over time to the point where even the person running a program and publishing the results knows little about some of the choices made in the implementation. And such poor records are typically kept of exactly which version of the code was used or of the parameter values chosen that even the author of a paper often finds it impossible to reproduce the published results at a later time. Regrettably, I speak from ample first hand experience.

As Buckheit and Donoho [3] point out in their classic paper on reproducible research, the scientific method and style of presenting experiments in publications that is currently taken for granted in the experimental sciences was uncommon before the mid-1800s. Now it is a required aspect of respectable research and experimentalists are expected to spend a fair amount of time keeping careful lab books, fully documenting each experiment, and writing their papers to include the details needed to repeat the experiments. A paradigm shift of the same nature may be needed in the computational sciences.

The idea of "reproducible research" in scientific computing is to archive and make publicly available all of the codes used to create the figures or tables in a paper, preferably in such a manner that the reader can download the codes and run them to reproduce the results. The program can then be examined to see exactly what has been done. As Buckheit and Donoho [3] put it,

> An article about computational science in a scientific publication is **not** the scholarship itself, it is merely **advertising** of the scholarship. The actual scholarship is the complete software development environment and the complete set of instructions which generated the figures.

[1]Department of Applied Mathematics and Department of Mathematics, University of Washington, Box 352420, Seattle, WA 98195-2420. `rjl@amath.washington.edu`
Version of June 29, 2008.

They present this as a distillation of the insights of Jon Claerbout, an exploration geophysicist who has been a pioneer in this direction since the early 90's (e.g., [30]).

The development of very high level programming languages has made it easier to share codes and generate reproducible research. Historically, many papers and text books contained pseudo-code, a high level description of an algorithm that is intended to clearly explain how it works, but that would not run directly on a computer. These days many algorithms can be written in languages such as Matlab or Python in a way that is both easy for the reader to comprehend and also executable, with all details intact.

Trefethen's book on spectral methods [33] is a good example of a textbook along these lines, in which each figure is generated by a 1-page Matlab program. These are all included in the book and nicely complement the mathematical description of the methods discussed. Trefethen makes a plea for more attention to short and elegant computer programs in his recent essay [34].

2 Objections to reproducible computational science

For the larger scale computer programs used to obtain most published results in computational science, there are many possible objections to making the code freely available in a form sufficient to reproduce the research. I will discuss three of these, perhaps the primary stumbling blocks.

One natural objection is that it's a lot of work to clean up a code to the point where someone else can even use it, let alone read it. This is certainly true, but it is often well worth doing, not only in the interest of good science but also for the selfish reason of being able to figure out later what you did and build further on it.

Those of us in academia should get in the habit of teaching good programming, documentation, and record keeping practices to our students, and then demand it of them. We owe it to them to teach this set of computational science skills, ones that will certainly become increasingly necessary in academic research environments and that are already highly valued in industrial and government labs. It will also improve the chances that we will be able to build on the work they have done once they graduate, and that future students will be able to make use of their contributions rather than starting from scratch as is too often the case today.

While ideally all published programs would be nicely structured and easily readable with ample comments, as a first step it would be valuable simply to provide and archive the working code that produced the results in a paper. Even this takes more effort than one might think. It is important to begin expecting this as a natural part of the process so that people will feel less like they have to make a choice between finishing off one project properly or going on to another where they can more rapidly produce additional publications. The current system strongly encourages the latter.

Requiring it of our students may be a good place to start, provided we recognize how much time and effort it takes. Perhaps we should be more willing to accept an elegant and well documented computer program as a substantial part of a thesis, for example. This is not unreasonable. A thesis in mathematics, like a research paper in this field, typically contains long and detailed proofs of theorems that are unreadable to all but a handful of experts around the world. Many readers will be interested in the results without working through all the details, but it is expected that the details are provided. It is also expected that the student will spend considerable time perfecting these details and writing them up. Constructing a computer program is not so different from constructing a formal proof.

A second objection to publishing computer code is that a working program for solving a scientific or engineering problem is a valuable piece of intellectual property and there is no way to control its use by others once it is made publicly available. Of course if the research goal is to develop general software then it is desirable to have as many people using it as possible. However, for a scientist or mathematician who is primarily interested in studying some specific class of problems and has developed a computer program as a tool for that purpose, there is little incentive to give this tool away free to other researchers. This is particularly true if the program has taken years to develop and provides a competitive edge that could potentially lead to several additional publications in the future. By making the program globally available once the first publication appears, other researchers can potentially skip years of work and start applying the program to other problems immediately. In this sense providing a program is fundamentally different than carefully describing the materials and techniques of an experiment; it is more like inviting every scientist in the world to come use your carefully constructed lab apparatus free of charge.

This argument has considerable merit in some situations, but there are also several counter-arguments worth mentioning.

- It is true that a code that successfully solves a scientific problem has value as intellectual property, but who should own it? In many cases the development of the code has been supported by federal grants and hence the financial investment in the code is made in large part by the taxpayers, not by its authors. There is an increasing tendency for federal grant agencies to demand that the work they support be made openly available to the scientific community.

  The National Institutes of Health now require (as of April 7, 2008) that papers containing work they fund must be made available via the National Library of Medicine's PubMed Central within 12 months of publication in a scientific journal [25]. Agencies supporting code development are starting to take a similar attitude. For example, a recent request for proposals from the Department of Energy [6] states that

  > "Successful applicants of Enabling Technologies must ensure that source code is fully and freely available for use and modification throughout the scientific computing community via a preapproved open source process."

- It is notoriously difficult to take someone else's code and apply it to a slightly different problem. This is true even when people are trying to collaborate and willing to provide hands-on assistance with the code. It is often true even when the author of the code claims it is general software that is easy to adapt to new problems. It is particularly true if the code is obscurely written with few comments and the author is not willing to help out, as would probably be true of many of the research codes people feel the strongest attachment to.

- My own experience in computational science is that virtually every computational experiment leads to more questions than answers. There is such a wealth of interesting phenomena that can be explored computationally these days that any worthwhile code can probably lead to more publications than its author can possibly produce. If other researchers are able to take the code and apply it in some direction that wouldn't otherwise be pursued, that should be seen as a positive development, both for science and for its original author, provided of course that s/he gets some credit in the process. This is particularly true for computational mathematicians, whose goals are often the development of a new algorithm rather than the solution of specific scientific problems. Even for those not interested in software development *per se*, anything we can do to make it easier for others to use the methods we invent will be beneficial to our own careers.

Perhaps what's needed is an expanded copyright process for scientific codes, so that programs could be made available for inspection and independent execution to verify results, but with the understanding that they cannot be modified and used in new publications without the express permission of the author for some period of years. Permission could be granted in return for co-authorship, for example. In fact such a system already works quite well informally, and greater emphasis on reproducible research would make it function even better. It would be quite easy to determine when people are violating this code of ethics if everyone were expected to "publish" their code along with any paper. If the code is an unauthorized modification of someone else's, this would be hard to hide.

A third obstacle to making some programs freely available is that they are based on commercial, proprietary, or copyrighted software that cannot be redistributed. This is certainly a limitation if the proprietary code is in integral part of the program, but in many cases it is not. Often the part of the program that corresponds to the original research being published is the work of the author. Making it available can help the reader understand the research even if they cannot run it, while those who do have access to the proprietary code (e.g. by purchasing the required packages) can also run it. Certainly the large number of books and papers that include Matlab codes are valuable in spite of being written in a commercial language that many readers cannot afford.

Along the same lines, some codes only run on special hardware, such as massively parallel supercomputers that many readers may not have access to. But again the ability to inspect the code and determine what parameter or algorithmic choices were made is often much more important than the ability to re-run the code and re-create the results already published.

3 Clawpack software for hyperbolic problems

The remainder of this paper presents a brief case study to illustrate a set of tools for aiding in the presentation of reproducible research on wave propagation algorithms for solving hyperbolic partial differential equations. The reader need not be familiar with such problems to follow this discussion.

Hyperbolic partial differential equations (PDEs) model a variety of wave propagation and fluid flow problems, including acoustic and seismic waves, advective transport, shallow water theory, and compressible gas dynamics, to name just a few applications. In many case the equations are nonlinear systems of conservation laws whose solution contains shock waves or other discontinuities in the solution that are particularly difficult to capture with classical finite difference methods. Much of my work over the past 15 years has been devoted to trying to make it easier for myself, my students, and other researchers to perform computational scholarship in the development of numerical methods for hyperbolic PDEs, and also in a variety of application areas in science and engineering where these methods are used. This work has resulted in the Clawpack software [23] (Conservation Laws Package) and various extensions, open source Fortran code that has been freely available since 1994. More than 7000 users have registered to download this software since it first appeared, and it has been used on a wide variety of problems.

The details of the algorithms implemented are not important for our purposes here. The interested reader can learn more about them in the textbook [19], which was designed to be used in conjunction with the software. Virtually all of the figures in this book are reproducible, in the sense that the programs that generated them can be downloaded from the web and easily executed. Most figure captions contain a link to the corresponding web page where the code can be found, along with additional material not in the book, for example animations of the solution evolving in time. The problem-specific code for each example in the book is quite small and easy to comprehend and modify. The reader is encouraged to experiment with the programs and observe how changes in parameters or methods affect the results. These programs, along with others on the Clawpack website [23], can also form the basis for developing programs to solve similar problems.

Although the development of the Clawpack software was originally motivated by the desire to make a set of existing methods more broadly accessible, the availability of this software has also encouraged me to pursue new algorithmic advances that I otherwise might not have. I hope that the software will also prove useful to others as a programming environment for developing and testing new algorithms, and for comparing different methods on the same problems. Since the source code is available and the basic Clawpack routines are reasonably simple and well documented, it should be easy for users to modify them and try out new ideas. I encourage such use, and I certainly use it this way myself.

Careful direct comparisons of different methods on the same test problems are too seldom performed in the study of methods for hyperbolic problems, as in many computational fields. One reason for this is the difficulty of implementing other peoples' methods, so the typical paper contains only results obtained with the authors' method. Sometimes (not always) the method has been tested on standard test problems and the results can be compared with others in the literature, with some work on the reader's part, and assuming the reader is content with comparisons in the "eyeball norm" since many papers only contain plots of the computed solution and no quantitative results. Of course there are many exceptions to this, including papers devoted to careful comparisons of different methods, but these papers are still a minority. I hope that Clawpack might facilitate this process more in the future, and that other algorithms for hyperbolic PDEs might be provided in a Clawpack-style implementation that allows direct use and comparison by others. One example in this direction is the WENO-Claw software developed primarily by David Ketcheson [14] that implements a class of higher-order methods.

4 Python tools for Clawpack experimentation

Recently my students and I have been developing a set of Python tools to facilitate the use of Clawpack. We have several goals, including:

- To provide a wider range of graphics and visualization options for viewing results computed with this software. Traditionally Matlab has been used as the primary visualization tool and a number of Matlab scripts and functions are provided as part of Clawpack for visualizing results. While Matlab is familiar and convenient for many users, it is a commercial package that is not available to everyone and we wish to provide open source alternatives. Moreover, in three space dimensions Matlab graphics are not as powerful as other available packages, and in particular do not provide voxel graphics for volume rendering.

  The plots presented in this paper and on the webpage [22] were created using `matplotlib` in Python, available in SciPy [8], [31] and sufficient for many 1d and 2d plotting purposes. For three-dimensional data other packages must be used, and we are currently developing Python interfaces between Clawpack and other open source visualization tools, in particular the VisIt software being developed at Lawrence Livermore Laboratory [35], which supports a wide variety of tools for two- and three-dimensional data on adaptively refined grids.

- To provide literate programming tools for documenting code with mathematical expressions that are easy to read, and with links to other parts of the code, external documentation, or to web pages where more information can be found. The term *literate programming* was coined by Knuth [15] and refers to programs that are fully self documented in a manner that is readable by humans. Literate programming techniques can greatly assist in the development of reproducible research in computational science. This is discussed further in the Conclusions section of this paper.

- To provide a web-based interface to Clawpack, both to facilitate its use by students learning about hyperbolic problems (so they can easily experiment without having to work directly with the Fortran code and data files) and as a web portal so that these experiments can be conducted without having to download and install the Clawpack software locally. A preliminary version of this "Easy Access Graphical Laboratory for Exploring Conservation Laws" (EagleClaw) is available via [23].

- To provide templates for performing reproducible research. Many of the same tools needed for the web portal EagleClaw are useful also in developing scripts that run a series of tests and collecting the results of these tests into webpage or LaTeX documents. The remainder of this paper concerns this topic and contains a brief description of one case study.

Most of the Clawpack software is written in Fortran 77. The code reads in a set of parameters from ASCII text files with names like `claw2ez.data` (which contains parameters defining the computational domain, the number of grid cells, the method to be used, etc.) and `setprob.data` (which contains problem-specific parameters as defined by the user). Running the code creates a set of output files containing the solution at several specified times. These files (either ASCII or hdf binary) are then read into the visualization tool (Matlab or Python).

In designing a Python interface, we have retained this basic structure and developed tools that interact with the Fortran code by modifying the data files. These files have essentially the same form as in previous versions of Clawpack. This means that users can still work in the classical manner if they desire, and need not use Python at all if they prefer to modify the data files by hand. Previous applications will continue to run unchanged.

To illustrate these tools and their use, we consider the simplest possible hyperbolic equation in two space dimensions, the advection equation modeling the transport of a tracer in a specified velocity field. As a test problem we consider "solid body rotation", in which the velocity field is

$$u(x,y) = -2\pi y, \quad v(x,y) = 2\pi x,$$

corresponding to counterclockwise rotation about the origin with period 1. The initial data used can be found on the webpage [22], along with figures and animations of the numerical solution and all the source code used to generate these results and the table and log-log plot below.

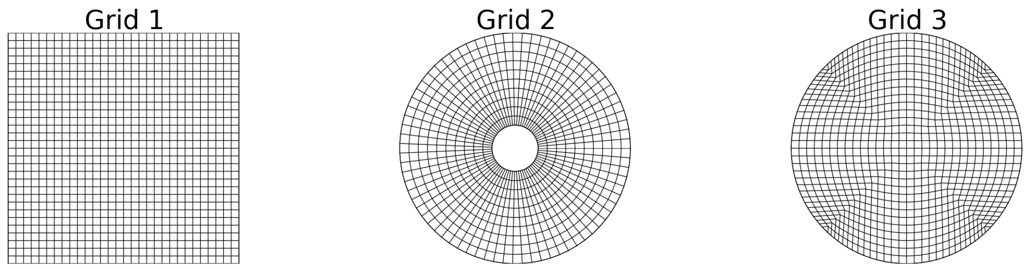

Figure 1: The three types of grids used for the advection test.

We compare the behavior of the algorithms implemented in Clawpack on three different types of computational grids, as illustrated in Figure 1:

Grid 1: Cartesian grids with square grid cells,
Grid 2: Polar grids in an annulus,
Grid 3: Quadrilateral grids of the type described in [4].

For each grid type, we compute the solution at four different grid resolutions in order to estimate the order of accuracy of the method. We also compare two different numerical methods. The first (with `Limiter = 0`) is a variant of the Lax-Wendroff method. This method should be second order accurate on smooth solutions and smooth grids, but is dispersive and often produces nonphysical oscillations in the numerical solution. The other method we test is one of the high-resolution limiter methods also implemented in Clawpack (with `Limiter = 3`, a particular choice known as the monotonized centered limiter [19]). This method is no longer formally second order accurate but often performs much better in practice, particularly on problems with discontinuous solutions such as the shock waves that often arise in solving nonlinear hyperbolic equations.

Tables 1–3 display the results from these 24 test cases (4 resolutions on each of 3 grid types, comparing 2 different numerical methods in each case). The error at time $t = 0.75$ is computed by comparing the numerical solution in each grid cell to the true solution at the cell center. The sum of the absolute value of all errors is scaled by the average cell area to give a value that approximates the 1-norm of the error. The value "Ave Δx" listed in the tables is the square root of the average cell area for the grid. We also present the errors from all 24 tests in a single log-log plot in Figure 2 to more easily compare the accuracy of the methods on different types of grids. A slope of 2 in the log-log plot corresponds to second order accuracy.

We see that roughly second order accuracy is achieved on all three grids with either choice of limiter. Errors are smallest in polar coordinates, which is not surprising since the grid is aligned with the flow and the advection equation reduces to one-dimensional advection in θ.

We also see that the error on Grid 3 is only slightly larger than on Grid 1, the Cartesian grid, verifying that these highly skewed quadrilateral grids can indeed be used with the algorithms implemented in Clawpack. Finally, we observe that on all three grids use of the limiter improves the 1-norm of the error by a factor of 3 or more.

5 Python tools

The Clawpack code is easily set up to run this problem and the traditional form of the code allows one to change data parameters in input files that are read by the Fortran code. One can run the 24 tests described above by hand and collect the results. With the newly developed Python tools for Clawpack, it is much easier to automate this entire process in a way that can be archived and easily duplicated at a later time.

Figure 3 shows a simplified version of the Python script to run the numerical tests, one that runs 16 tests using only grids of type 1 and 3. The polar grids require a different domain in r and θ coordinates and the specification of periodic boundary conditions, which is easily accomplished by setting some of the parameters differently. See the `clawtest.py` module on the webpage [22] for full details.

The script shown in Figure 3 employs a number of functions from the `clawtools` and `clawtest` modules that will not be displayed here but that can be found on the webpage. The `ClawData` class is used to store data values that will be written into the data files for use in the Fortran code. For example, the file `setprob.data` contains a line

```
1 =: igrid   # which type grid to use
```

The Python commands

```
data = clawtools.ClawData()
data.igrid = 2
data.write('setprob.data')
```

would cause the value 1 in the data file to be replaced by 2. All other lines of the data file are unaltered. The only change made to the data files from previous versions of Clawpack is the introduction of the assignment operator `=:` in the portion of each line that is ignored by the Fortran code (which typically only reads a single number from each line).

Table 1: Errors on Grid 1

			Limiter = 0		Limiter = 3	
mx	my	Ave Δx	Error	Observed order	Error	Observed order
30	30	0.0591	0.3263	nan	0.1490	nan
60	60	0.0295	0.1336	1.29	0.0404	1.88
120	120	0.0148	0.0389	1.78	0.0088	2.20
240	240	0.0074	0.0105	1.89	0.0020	2.14

Table 2: Errors on Grid 2

			Limiter = 0		Limiter = 3	
mx	my	Ave Δx	Error	Observed order	Error	Observed order
10	75	0.0634	0.0917	nan	0.0408	nan
20	150	0.0317	0.0297	1.63	0.0112	1.86
40	300	0.0159	0.0084	1.82	0.0033	1.78
80	600	0.0079	0.0023	1.89	0.0008	1.97

Table 3: Errors on Grid 3

			Limiter = 0		Limiter = 3	
mx	my	Ave Δx	Error	Observed order	Error	Observed order
30	30	0.0591	0.3843	nan	0.1610	nan
60	60	0.0295	0.1567	1.29	0.0492	1.71
120	120	0.0148	0.0456	1.78	0.0123	1.99
240	240	0.0074	0.0123	1.89	0.0034	1.86

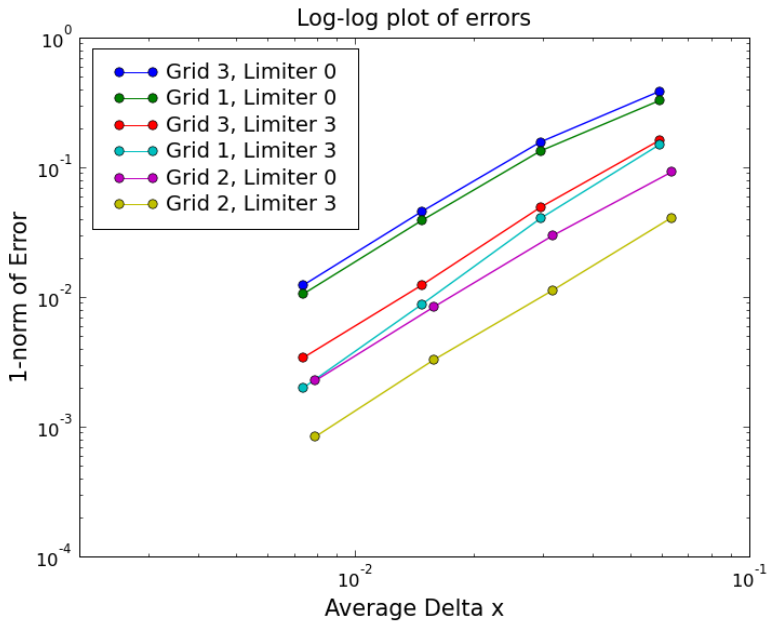

Figure 2: Log-log plot of the errors from 24 test cases as described in the text.

6 Conclusions and additional thoughts

The example just given illustrates one modest attempt at providing tools to ease the task of doing reproducible research in the development of numerical methods for hyperbolic problems. Use of these tools requires little overhead or infrastructure beyond the classical use of Clawpack. Python is ideally suited to this purpose and is available on virtually every operating system. The tools could easily be adapted to other contexts, for use with any code that reads parameters from a text file to set up a particular run.

The use of these tools makes it much easier to archive the environment used to generate a series of tests. Rather than requiring the retention of 24 sets of data files, one for each test case, a single Python script clearly shows the parameter choices used for each test case and allows them all to be easily recreated later if needed.

Of course one must also archive the computer code used to perform the tests. If the code changes in the future then having the data sets preserved will be insufficient to reproduce the tests. For the purposes of this paper a webpage [22] has been created with a full copy of all the Clawpack code actually used for this example, which in this case is quite small. For on-going research projects, it is desirable to use versioning software such as Subversion [5], [32] in order to easily preserve the state of the code at any point in time without the need to save full copies of the entire code. By saving the revision number of the code it is possible to recreate any past state. The Subversion repository for recent versions of Clawpack can be accessed via [23].

The use of versioning software should be mandatory among computational scientists at this point. It is easy to use and extremely beneficial even for a single individual who is developing programs and writing papers based on the results, and invaluable for any kind of collaborative work.

There is an additional difficulty with attempting to perform reproducible research that I have not yet addressed, namely the fact that even if the code and the data are preserved, the computer, operating system, or computer language may change in ways that render the code unusable or unable to reproduce previous results. As one example particularly relevant to the tools described here, the Python language is undergoing revision and Python 3.X will not be backward compatible with the Python 2.5 used for the tools recently developed. Hence in addition to archiving the tools, one should really archive the current state of the language as well, or at least assume this is being done somewhere (as is the case for

Python). Archiving the computer operating system and the computer itself is of course more difficult.

Looking further down the road, there is great uncertainty about the durability of digital archives of any sort. While books and papyrus can last millennia, we often find it is impossible to read data or computer codes from a few years ago. This is obviously a concern that many professionals are working hard to address, but in the meantime should not be used as an excuse to avoid attempting to achieve some level of reproducibility with the available technology.

The webpage for this paper [22] also provides a glimpse of the manner in which Python tools can be developed to advance the goals of literate programming as laid out by Knuth [15]. The hope is to illuminate, for the interested reader, the algorithms that produced the published results within the documentation of the computer code. For documenting mathematical programs it is convenient to be able to include mathematical descriptions in the comments of the code, written in LaTeX and readable (as typeset mathematics) along with the code.

A general Python script `mathcode2html.py`, available from [21], can been used to convert source code in many languages (and also data files) into html documents, with any comment statements delimited by `begin_html` and `end_html` treated as html code. When used in conjunction with `jsMath` [13], this allows simple LaTeX equations to be included in the comments and properly typeset in the resulting html version of the code. Some wiki-like formatting tools allow links to be included in the computer code in a manner that is fairly readable in the raw code and yet easily converted into the appropriate links in the html version. A more specialized version of this script, `clawcode2html.py`, is included with the latest Clawpack bundle and is used for documenting and cross referencing the code and examples.

Literate programming and reproducible research in computational science often go hand in hand, and a number of other approaches and systems have been developed for implementing some combination of these goals. A few notable projects are the CWEB system of Knuth and Levy [16], Noweb [12], [27], [28], Sweave [18], AMRITA [26], and Madagascar [24]. Other recent papers on literate programming and reproducible research include [1], [2], [7], [9], [10], [11], [17], [29], [30].

The tools described in this paper have one advantage, I believe, over the more ambitious approaches described in some of these references. They require relatively little infrastructure and can be be added on to existing projects in an incremental manner rather than requiring a fresh start. While ideally all software would be designed from the beginning in a literate and reproducible manner, this is not likely to happen soon and legacy codes from decades ago will be with us for some time to come. With appropriate tools I believe even these can be greatly enhanced, as I have attempted to illustrate with Clawpack.

Acknowledgments. The introductory part of this article is based on an invited talk and paper that appeared in the proceedings of the International Conference of Mathematicians [20], and numerous people provided valuable feedback during the preparation of that paper and in response to it. I would also like to thank the editor and anonymous reviewers of this paper, who further helped to improve this paper. Thanks are due also to Kyle Mandli for assistance in developing some of the Python and web portal tools being used with Clawpack. This work was supported in part by NSF grant DMS-0609661.

References

[1] G. Baiocchi. Reproducible research in computational economics: guidelines, integrated approaches, and open source software. *Computational Economics*, 30:19–40, 2007.

[2] N. H. F. Beebe. A bibliography of literate programming. `http://www.literateprogramming.com/litprog-bib.pdf`, 2002.

[3] J. B Buckheit and D. L. Donoho. WaveLab and reproducible research. `http://www-stat.stanford.edu/~donoho/Reports/1995/wavelab.pdf`, 1995.

[4] D. A. Calhoun, C. Helzel, and R. J. LeVeque. Logically rectangular finite volume grids and methods for “circular” and “spherical” domains. *SIAM Review*, 2008.

[5] B. Collins-Sussman, B. W. Fitzpatrick, and C. M. Pilato. Version control with Subversion, 2004. `http://svnbook.red-bean.com/`.

[6] Department of Energy. SciDAC request for proposals. `http://www.er.doe.gov/grants/FAPN06-04.html`, 2006.

[7] D. L. Donoho and X. Huo. BeamLab and reproducible research. *Int. J. Wavelets, Multires. and Inf. Proc.*, to appear, 2005.

[8] Enthought Python Distribution. `http://www.enthought.com/products/epd.php`.

[9] R. Gentleman and D. Temple Lang. Statistical analyses and reproducible research. Bioconductor Project Working Papers. Working Paper 2. `http://www.bepress.com/bioconductor/paper2`, 2004.

[10] R. C. Gentleman et al. Bioconductor: open software development for computational biology and bioinformatics. *Genome Biology*, 5:R80.1–R80.16, 2004.

[11] R. H. F. Jackson, P. T. Boggs, S. G. Nash, and S. Powell. Guidelines for reporting results of computational experiments. Report of the ad hoc committee. *Math. Programming*, 49:413–425, 1991.

[12] A. L. Johnson and B. C. Johnson. Literate programming using `noweb`. *Linux Journal*, October:64–69, 1997.

[13] `jsMath` software. `http://www.math.union.edu/~dpvc/jsMath/`.

[14] D. I. Ketcheson and R. J. LeVeque. WENOCLAW: A higher order wave propagation method. In *Hyperbolic Problems: Theory, Numerics, Applications, Proc. 11'th Intl. Conf. on Hyperbolic Problems*, page to appear, 2006.

[15] D. E. Knuth. Literate programming. *The Computer Journal*, 27:97–111, 1984.

[16] D. E. Knuth and S. Levy. *The CWEB System of Structured Documentation.* Addison-Wesley, Reading, MA, 1993. `http://www-cs-faculty.stanford.edu/~knuth/cweb.html`.

[17] J. Kovačević. How to encourage and publish reproducible research. *Proc IEEE Int. Conf. Acoust. Speech, and Signal Proc.*, pages IV:1273–1276, 2007.

[18] F. Leisch. Sweave: Dynamic generation of statistical reports using literate data analysis. In W. Härdle and B. Rönz, editors, *Compstat 2002 — Proceedings in Computational Statistics*, pages 575–580. Physica Verlag, Heidelberg, 2002. ISBN 3-7908-1517-9.

[19] R. J. LeVeque. *Finite Volume Methods for Hyperbolic Problems.* Cambridge University Press, 2002.

[20] R. J. LeVeque. Wave propagation software, computational science, and reproducible research. In *Proc. Int. Cong. Math.*, pages 1227–1254, 2006. `http://www.amath.washington.edu/~rjl/pubs/icm06`.

[21] R. J. LeVeque. `mathcode2html` software. `http://www.amath.washington.edu/~rjl/mathcode2html/`, 2007.

[22] R. J. LeVeque. Python tools for reproducible research on hyperbolic problems, webpage to accompany this paper. `http://www.amath.washington.edu/~rjl/pubs/cise08/`, 2008.

[23] R. J. LeVeque et al. CLAWPACK software. `www.clawpack.org`.

[24] Madagascar software. `http://rsf.sourceforge.net/Main_Page`, 2008.

[25] National Institutes of Health. Revised policy on enhancing public access to archived publications resulting from nih-funded research. `http://grants.nih.gov/grants/guide/notice-files/NOT-OD-08-033.html`, 2008.

[26] J. J. Quirk. AMRITA. `http://www.amrita-cfd.org/`, 2007.

[27] N. Ramsey. Noweb — a simple, extensible tool for literate programming. `http://www.eecs.harvard.edu/nr/noweb/`.

[28] N. Ramsey. Literate programming simplified. *IEEE Software*, 11:97–105, 1994.

[29] C. J. Roy. Review of code and solution verification procedures for computational simulation. *J. Comput. Phys.*, 205:131–156, 2005.

[30] M. Schwab, N. Karrenbach, and J. Claerbout. Making scientific computations reproducible. *Comput. in Sci. & Eng.*, 2:61–67, 2000.

[31] SciPy and NumPy software. `http://www.scipy.org/`.

[32] Subversion version control system. `http://subversion.tigris.org/`, 2008.

[33] L. N. Trefethen. *Spectral Methods in Matlab.* SIAM, Philadelphia, 2000.

[34] L. N. Trefethen. Ten digit algorithms. `http://www.comlab.ox.ac.uk/nick.trefethen/ten_digit_algs.htm`, 2005.

[35] VisIt software. Lawrence Livermore National Laboratory, `http://www.llnl.gov/visit/`.

Figure 3: The Python script `clawtestsubset.py` for running 16 test problems. The full module `clawtest.py` for all 24 test cases, which also includes other functions used below, can be found on the web page [22].

```
import clawtools
from clawtest import *

# special data structure for Clawpack parameters:
data = clawtools.ClawData()
data.tfinal = 0.75              # final time
data.nout = 1                   # output solution only at final time

# List parameters for tests to be performed:
grids = [1,3]                   # set of grids to test
limiters = [0,3]                # set of limiters (mthlim values) to test
mxvals = array([30,60,120])     # mx values
myvals = array([30,60,120])     # my values
area = pi                       # area of circle, for L1 norm

table = {}  # dictionary of data and results for each test

for mthlim in limiters:
    data.mthlim = mthlim
    for igrid in grids:
        # Write the value igrid into data file setprob.data:
        data.igrid = igrid;
        data.write('setprob.data')

        # create a dictionary to hold the data and results for this test:
        table[(mthlim,igrid)] = {}

        this_table = table[(mthlim,igrid)]  # short name
        this_table['mxvals'] = mxvals       # grid resolutions to test
        this_table['myvals'] = myvals
        this_table['ave_cell_area'] = area / (mxvals*myvals)
        this_table['errors'] = empty(len(mxvals))  # filled with results below

        for itest in range(len(mxvals)):
            data.mx = mxvals[itest];   mx = data.mx   # short form
            data.my = myvals[itest];   my = data.my   # short form
            data.write('claw2ez.data')                # write mx,my,tfinal,nout

            # run Fortran code:
            clawtools.runclaw()

            # compute errors:
            errors = compute_errors(frame=data.nout)
            # approx 1-norm of error:
            errorsum = abs(errors).sum()
            error1 = errorsum * this_table['ave_cell_area'][itest]
            this_table['errors'][itest] = error1

# Create Tables 1--3 and Figure 2 of this paper:
make_latex_table(table, limiters, grids, fname='errortables.tex')
make_error_plots(table, limiters, grids, fname='errors.png')
```