**Paper I: *Open Science Decoded*** (Tony Hey & Mike C. Payne)

There is now global momentum toward “open science”. The authors find evidence for this trend from various places, including memoranda from various governmental funding agencies requiring subordinate research programs/agencies to make their results available in peer-reviewed publications and their data available in a digital format. In this way the public can access the results and the data used.

The term “open science” may be new, but it alludes to a fundamental part of the scientific process – reproducibility of research – the process of other scientists using the data and/or methods of the original research program to see if they get the same result. In modern scientific research, methods tend to be heavily dependent on computation. To complicate the issue of making science reproducible, many scientists do not publish the source code or details about the precise computing environment along with their results, so it can be very difficult to reproduce the methods of the original research.

Recognition of the problem of reproducibility of computational physics research is not new, people have been thinking about this problem since the early 1990s. More recently however, the Institute for Computational and Experimental Resarch in Mathematics at Brown University hosted a workshop on reproducibility in computational and experimental mathematics. From this workshop came the idea that both data and code should be made available and accessible so that others can have the chance to reproduce the published findings. Aside from making data and code available, there is also the problem of numerical reproducibility. Round-off error can be exacerbated when simulations are scaled up to run on large parallel systems.

The problems of providing open science and research reproducibility are amplified in the case of very large and complex science experiments, like the LHC. At LHC, reproducibility is addressed by having multiple experiments, i.e. ATLAS and CMS which produce and analyze their own data using different software. The problem of making the LHC data available to the public is challenging, although some first steps are being taken (CERN Open Data Portal).

Small scale science projects are much more common than large projects like the LHC. Open science and reproducibility on this scale is therefore just as imperative. Small research groups can make their data products and computational methods (code) available to the public and other researchers using open-source software project management infrastructures, like GitHub, which now sports 3 million users. That said, most of the scientific research software made available here is of poor quality and poorly documented/maintained. The scientific community has room for improvement in terms of open science, and also in terms of reproducibility of computationally intensive or code-reliant methods.

**Paper II: *Developing Open Source Scientific Practice*** (K. Jarrod Millman & Fernando Perez)

Modern science is inextricably linked to computational methods at every level, whether it is data collection and preprocessing, analysis, theoretical modeling, and simulation, and more. Given this fact, it is surprising that the level of attention and formal education afforded to learning about computational science is so low for new scientists. The authors of this paper argue that computational and numerical methods should play a more central role in how scientists are trained and conduct their research.

The authors point to open source software development community, which has created tools and practices tailored to testing, validating software. The authors argue that if these methods were adopted by the scientific community, we would stand to achieve enhanced reproducibility in our research. This can often be complicated by the fact that some scientific research projects use complex and sophisticated code (like the 4.5 million lines of code at LHC/ATLAS), or simulations that depend on supercomputers to run. In these cases, it is sometimes possible for a single researcher to run a scaled-down version of the code on their own computer to verify results.

The authors describe the patchwork of software used by researchers in the course of their projects, e.g. individual work using Excel, Matlab, R, etc., production scale execution work done with C++, Fortran, etc., publication with LaTeX, file sharing with Dropbox, etc. In place of this patchwork the authors suggest that researchers view the entire cycle of research as more of a continuum, by, for example, using Python on multiple phases of the project, and also adopting the open source ecosystem to make reproducible research a central aim.

The authors also explore practices and tools that should be applied to scientific research, whether on the individual level, or on the collaborative level. These are their so-called “routine practices”. First mentioned is version control. Version control can be done in an ad hoc way, i.e. using naming schemes, or emailing oneself different iterations of the same code etc., which tend to be quite cumbersome methods. Instead, the authors recommend tried and true version control software like Git/GitHub. Version control allows for ease of reproducibility, whether by other researchers or by the original researcher who finds they need to go back and make changes and then re-run the analysis. Another is “execution automation”, which follows from the idea of recording how you used the code and data to produce the results you have. The C++ **make** system helps because it automatically checks whether data/code dependencies of a script have changed since the last time they were run. Another routine practice should be testing, where the researcher frequently runs their code in order to catch bugs early in the process. Code readability is a practice that should be practiced more by scientific researchers, and it takes the form of naming variables in a meaningful way and commenting code.

In large collaborations, it is important that scientists make use of distribution version control. *Distributed* version control is when the version control system/access to it does not depend on a single central server for functioning. This means that members of a large collaboration can always access the versions, create their own branches where they can edit pieces of code without affecting the main “trunk” or the development, and also do testing with iterations of code developed by other researchers in the collaboration.

**Paper III: *Ten Simple Rules for Reproducible Computational Research*** (Geir Kjetil Sandve, Anton Nekrutenko, James Taylor, Eivind Hovig)

Although reproducibility is the cornerstone of cumulative science, new tools and technologies, massive volumes of data and interdisciplinary approaches are complicating efforts to keep science reproducible. With this perspective in mind, it has become a minimum standard for assessing the value of scientific claims to test whether the claim can be reproduced. The authors of this article point out that recently, studies have been conducted which show that often experimental methods are left out of published research. This has led to a wider discussion in the scientific community about replication, and a call to foster a culture of reproducibility in computational science.

In a practical sense, publication pressure and deadlines can affect the extent to which the researcher is focused on reproducibility of their research. The pressure to get research published while it is still relevant is often balanced (sometimes unwittingly) against the need to make sure the work is reproducible. The authors suggest that as a minimum requirement, the research should at least be able to reproduce the results themselves, and a simple extension that scientific peers would also have a practical possibility of reproducing the results. The authors present ten simple rules for reproducibility of computational research.

The first rule is that *for every result keep track of how it was produced.* For example, if you’ve created a plot, as a minimum write down the settings in the code you’ve used to produce it, or save the data and code made to use that plot. The second rule is to *avoid manual data manipulation steps,* where for example you go into the data file in a word processor or text editor and edit something manually. Often this kind of thing is hard to keep track of and is easily forgotten. The third rule is to *archive the exact versions of all external programs used*. Rule four is *version control all custom scripts*, where if you write the script/code yourself, keep versions of all iterations, or save the differences between iterations, using version control. Rule five is to *record all intermediate results, when possible in standardized formats,* which can help because quickly browsing through intermediate results can reveal discrepancies toward what is assumed and can in this way uncover bugs etc. Rule six is *for analyses that include randomness, note underlying seeds*, because for some random number generators, using the same seed can give identical results every time the generator is run, possibly invalidating research that relies on randomness of samples. Rule seven is *always store raw data behind plots*. Rule 8 is to *generate hierarchical analysis output, allowing layers of increasing detail to be inspected.* The final results that are presented in an article often represent highly summarized data. In order to fully validate and understand the result, it’s often useful to inspect the detailed values underlying the summaries. Rule 9 is to *connect textual statements to underlying results*, whereby the textual interpretations of results (which usually live in text documents or emails to collaborators) should be more closely connected (e.g. in the same folder) to the results of analysis (which usually live separately on, say, a server). The final rule 10 is to *provide public access to scripts, runs, and results,* the authors mention that most journals allow articles to be supplemented with online material and some journals have initiated further efforts for making data and code more integrated with publications.

**Paper IV: *Software for Analyzing Supernova Light Curve Data for Cosmology*** (Kyle Barbary)

This is a case study presented by Justin Kitzes, Daniel Turek, and Fatma Deniz in *The Practice of Reproducible Research*. The author of this case study is a postdoc at the physics department at UC Berkeley. His work is observational cosmology. The researcher is asked a few questions and his answers are summarized here:

*What does “reproducibility” mean to you?* To Barbay, reproducibility has two main facets: the availability of usable (preferably open-source) software and the availability of data (preferably in both raw and reduced forms). With these, a scientist should be able to reproduce the results of study from start to finish. To answer the question (*why do you think that reproducibility in your domain is important?),* Barbary emphasizes efficiency. Reproducibility makes cosmology research more efficient by allowing researchers to save time by re-using code instead of rewriting it themselves, it allows for better understanding of algorithms since the researcher can use them him/herself, instead of relying on coarse descriptions found in publications, and finally because conflicting results can be readily be resolved by allowing different groups to reproduce each other’s results by using the same data and code. Barbary learned about reproducibility through working on the AstroPy project on GitHub, which also allowed Barbary to learn about best practices in programming like unit testing, automated documentation builds, etc. *What do you see as the major challenges to doing reproducible research in your domain, and do you have any suggestions?* In astronomy, observers have a tendency to guard their hard-won data until they have eeked out every possible analysis. On the other hand, researchers can also be reluctant to release software, and Barbary is less sympathetic to this reluctance. Some reasons why code may not be shared are that the authors feel the code lacks in quality, they think there’s too much overhead involved in cleaning it up/maintain/support it, or a perceived competitive advantage to not sharing the software.

**Paper V: *Lessons Learned*** (Kathryn Huff)

Section from *The Practice of Reproducible Research* (Kitzes, Turek, and Deniz). This piece reflects upon the collection of case studies collected for the main article. Similar struggles appear in differing scientific fields, and nearly irrespective of preferred programming language. This piece summarizes some of the common themes among the case studies.

In terms of computational reproducibility, there was general agreement among all the respondents: that when provided with identical source code, input data, software, and computing environment configurations, that an independent party can exactly reproduce the results of the original work. Many “pain points”, or hurdles to the goal of reproducibility in research are shared in common between respondents from different scientific fields. One is that the blinding pace of innovation often can make manuscript preparation software, database formats, analysis software, etc. used by different researchers incompatible, this fact is also made manifest through the familiarity/competency levels of different researchers with respect to different software and/or formats.

Build systems and dependencies are also a sticking point for scientists across disciplines; the battle to even sometimes get a software pipeline just running on a different computing environment can sometimes cause the goal of reproducibility to take a back seat to the practical importance of getting the results out while they are still relevant.