White paper in support of

NASA'S PROPOSED OPEN CODE POLICY

James Paul Mason

301-286-4474
NASA Goddard Space Flight Center
james.p.mason@nasa.gov

This paper addresses two main topics related to the proposed open code policy – data processing pipelines and science data analysis – and makes some smaller additional points near the end. In short, I am in favor of the policy, but note that there will be some issues with implementation, and believe that positive incentives and easing of barriers should supplement the policy to reach further toward the end goal of making public code the norm among NASA-funded efforts.

NASA projects generate a lot of data. In order to be useful for science, raw data must be processed and visualized. Visualization generally falls under the purview of science data analysis and I will return to that later. Here the focus is on data generated by missions/instruments. Data processing often involves corrections, interpolations, propagation of uncertainties, folding in of ancillary data, discarding or dampening of noise; numerous processes that involve choices and assumptions. The code to do this processing is often developed, naturally, by the team involved in the creation of the instrument. That code should be open source so that other people can, if they wish, 1) see what was done to the raw data and make different choices and assumptions, 2) contribute directly to the "default" processing pipeline, thus easing the burden on the instrument team, and 3) avoid time spent reinventing the wheel for other instruments. Those three points are the driving ideas behind solarsoft. This suite of code written in IDL has incorporated most of the world's space-based solar instruments. It has been a boon to the solar science community. The lessons learned from that effort can be applied to any future broader efforts.

The third point, reuse, will be a particular advantage in the coming years. As access to space becomes cheaper and as technology miniaturizes, smaller teams are becoming common. CubeSat projects, like the ones that the I have been heavily involved in, are an excellent example of this. With few people on the team, each person has a diverse set of responsibilities. Using freely available code rather than developing it from scratch is a massive savings in scarce time that can then be applied to ensuring missions success. Moreover, code that has already been used elsewhere will tend to be more robust because it will have presumably already gone through many cycles of debugging, testing, and perhaps even have flight heritage. This idea also fits well with the principles of the CubeSat movement, which called for standardization to drive down costs. As such, we plan to make the data processing pipeline for the NASA-funded Miniature X-ray Solar Spectrometer (MinXSS) CubeSat (Mason, 2016) open source via solarsoft and GitHub, regardless of the outcome of the NASA policy decision.

I also made an open-source beacon decoder so that ham radio operators around the world can view a subset of real-time telemetry from the spacecraft, and it automatically forwards all of the data they receive to us (link to GitHub repository). In the development of this application, I was provided immediate voluntary feedback from the ham radio community about bugs, feature requests, and platform-specific issues. This feedback was facilitated directly on GitHub with a very simple interface for both the developer and the user. The decoder software has been used by ham operators around the world and has provided the mission with more data than it would have had otherwise. Recall that I only made a subset of the real-time telemetry visible to the ham operators. This was a direct concession to ITAR. At the time, we believed that the attitude determination and control telemetry fell in the grey area of ITAR, so I excluded it from being decoded and displayed. We've since been informed by a lawyer that this should be okay. Nevertheless, this raises the concern of where open code and ITAR may overlap. Data processing pipelines may face the same issue. I believe that what this calls for is a very clear

delineation between what is okay and what isn't. As a developer, I *want* my code to be freely accessible but I also do not want to get in trouble with the law. As a result, if it seems like a grey area I will just err on the side of caution. It requires nontrivial effort to seek out the definitive answer of whether some particular bit of code can be published or not. Trivializing that effort is something that I would hope to see come as a result of this NASA policy. There should be a very brief checklist online to determine whether code falls under ITAR or not. Many people choose simply to publish nothing rather than take the risk, so eliminating the risk would go a long way to getting more open code.

Once data has been decoded and processed, it must be visualized in order for humans, be they scientists or the public, to understand. Often, additional processing is done to the data products provided by models or instruments by the end-user scientist. The arguments above about assumptions and choices are equally applicable to the processing code here. Even the smallest choices in that final step, how to visualize the processed data, can have an important impact. For example, the limits on the axes of a plot can have a dramatic impact on how the data are perceived. The limits can even be set to exclude some data. The choice of color palette and transparency can have similar effects. It is simply not practical to include every one of these granular choices directly in paper publications. Nevertheless, those choices should be transparent and the results reproducible. Providing a link within the paper to the code is a simple resolution.

The ethos of science and open code have a great deal of overlap. Transparency is core to both. They are both perpetually improving revisions. Reproducibility is a cornerstone of science that can be immeasurably bolstered by open code. Thus, it is entirely natural that the two should go hand in hand. In homework and in tests, we expect that students show their work. Simply writing an answer is not enough. Why is the same not true once those students become professional researchers? A paper's method section contains some information, but by pure necessity of the format, can only provide highlights. Again, simply providing a link to the code that was used to produce the results in the paper, while still describing the highlights, allows us to "show our work" without impacting the readability of a paper or the effort in producing a paper.

Perhaps the biggest barrier to the day-to-day implementation of the proposed NASA policy is resistance to the burden it will impose on individual researchers. So, it should be made easy. Leveraging existing tools can help. Providing examples, tutorials, and experienced volunteers can help. Putting all of those resources on an attractive website can help. Such a website should have a modern, well-designed, easy user experience. It should include a few basic, brief how-to videos. There should also be contact information for experienced volunteers willing to help others with the process of getting code posted. It should also be easy to opt in and out of that volunteer system. I am sure that there would be many such volunteers. I would be one. All of my code is already on GitHub and I have a workflow that makes it extremely easy and quick (< 30 seconds) to post code. The site should also include brief checklists for determining things like whether your code falls under ITAR or other legal bindings; we can't be expected to read lengthy legal documents. These measures would contribute to easing the burden imposed by the policy and hopefully embody an upbeat attitude that would inspire people to see this as exciting rather than as drudgery.

A few other concerns and ideas occur to me. The first is finding the right threshold for how trivial code can be before it is required to be open sourced. In day-to-day work, we often write quick ~5-line scripts to do something small – a recent example from my own life, to create a plot of altitude versus time for two CubeSats released from the International Space Station a year apart to estimate how much longer the more recent one may remain on orbit. Surely, this need not be made open source. I think the threshold should be higher than this but no higher than the publication of results. A sensible threshold may fall somewhere around code used to produce the results in a poster publication.

A second list of ideas falls under the category of non-policy approaches to incentive open source licenses. The biggest of them is funding for open source community projects. The ultimate indicator of priorities in the agency is where money goes. There are many examples of open source community projects that have resulted in a widely adopted product, despite the trouble the developers had finding funding to develop it, e.g., matplotlib, numpy, and astropy (Muna, 2016). These particular examples remain in active development. As a scientist and engineer deciding what to do with my time, a critical factor in that decision is what I can get funded to do. If I could propose to add new, useful modules to an open source library, I would do so. It could cover, say, 20% of an FTE. Another possible incentive would be through journals. If editors and peer reviewers asked for the code used to process the data and produce the plots in a paper, that would apply some pressure on authors. The paper could then include a link to that code. Positive incentive could come from that link, or an associated badge, being featured prominently both within the paper and on websites used to access the paper, e.g., the publisher website, the Astrophysical Data System (ADS).

Thirdly, I want to reemphasize how important ease of implementation is. Making it easy to post, copy, edit, version control, and refer to code is critical. If it takes less than 30 seconds to post some new code, then why not? If it's a big burden, then it's less likely to happen. Part of that calculus is the choice of where to post the code. Thus, a potential barrier is fracturing. If there are too many repository host options, it can result in paralysis and the failure to post code. Effort will have to be expended figuring out which is the right place to post some particular code. A single author's code may then be scattered across many platforms. Instead of spending time developing code, they would be spending it managing competing systems, each with different interfaces and tools. For papers, we all know to go to ADS and arXiv. Right now for code, the community's most popular choice is GitHub, with over 8 million users. I recommend that any policy or incentives encourage the use of this platform. It makes the entire process easy, including referencing a frozen version of code with a DOI. They have even built a text editor that is itself open source and extremely well designed: Atom. If NASA were to develop, fund the development of, or select a competing tool, then it should be at least as good, in every respect, as GitHub.

Finally, there will be a challenge in long-term reproducibility as code becomes obsolescent and incompatible. As code repositories become inactive, it's unlikely they will remain useable. Similarly, the Library of Congress is very concerned with digital format "rot". The only tenable solution they have found is to periodically convert file formats to the latest standard (Pogue, 2017). Updating code, however, often requires much more manual labor than updating a file format. Thus, alternative solutions must be considered. Some programming languages (e.g., python) allow easy management of multiple environments which can independently run different versions of software. It has been over 9 years since python 3.0 was released, but many

modules have still not updated to support anything beyond 2.7. Python environments, particularly through the anaconda distribution, make it easy to house multiple versions of python itself and any modules on a single computer, and to switch between them. This idea might be scaled up to something like an on-call virtual machine that may even emulate aging hardware architectures so that old software can still be run. There are surely other and better ideas for how to address this problem, but it is one that merits thought.

I support NASA's proposed policy to make the community's code open source. The funding is public and the product of that support should be as well. The ethos of science and open source are highly compatible. While pressure from NASA may not get every bit of code open sourced, it will certainly get more code out there than if the status quo continues.

References

Mason, J. P., Woods, T. N., Caspi, A., et al. 2016, "Miniature X-Ray Solar Spectrometer: A Science-Oriented, University 3U CubeSat", JSpRo, 53, 328

Muna, D., Alexander, M., Allen, A., et al. 2016, "The Astropy Problem", eprint arXiv:1610.03159

Pogue, D., 2017, "How to Fight Format Rot", Scientific American