PYTHON FOR PLANETARY DATA ANALYSIS.

J.R. Laura, T. M. Hare, L.R. Gaddis, R.L. Fergason, Astrogeology Science Center, U.S. Geological Survey, 2255 North Gemini Drive, Flagstaff, AZ, 86001, jlaura@usgs.gov.

Introduction: Following our earlier publication on this topic [1], we continue to see increased utilization of the Python programming language by the planetary science community. A cursory search of the LPSC abstract archives shows a small, yet increasing number of abstracts explicitly making mention of access to underlying libraries via Python [e.g., 2, 3], the development of data processing capabilities within Python [e.g., 4-8], or the development of analytic solutions [e.g., 9-14]. These abstracts offer concrete examples of Python usage for processing and working with planetary data. We attribute this increase to the ease of use, readability, and portability of Python [1] as a scientific computing language. Python is commonly applied to High Performance Computing tasks and in the prototyping and development of Graphical User Interfaces, in continuing to leverage legacy code bases. This abstract reports our efforts to continue to integrate Python into our workflows and highlights additional use cases of potential benefit across the planetary science community.

High Performance Computing: Planetary data volumes are increasing rapidly due to increased data acquisition efforts associated with recent and new missions, improved spatial, temporal, and radiometric sensor resolutions, and increasingly complex process models generating ever increasing derived products [e.g., 15]. At current and future data sizes, tractable analysis requires either quantitative, repeatable methods of data reduction or the utilization of High Performance Computing (HPC) resources. Since the publication of the Atkins report [16], considerable research effort and funding has been invested in the development of Cyber Infrastructure (CI) projects. This suggests that the larger research community has avoided large-scale reduction and embraced HPC utilization. CI is the multi-tiered integration of HPC hardware embodied by distributed computing resources, "Big Data" sets, scalable processing capability, and collaborative, cross-domain research teams. Within the context of CI, Python is ideally suited to support the development of scalable high performance algorithms and the deployment of tools to reduce the complexity of HPC utilization that is within the CI middleware layer[22].

At USGS Astrogeology, we have utilized Python for the automated generation and submission of HPC jobs (e.g., Portable Batch System scripts) for the creation of Mars Odyssey Thermal Emission Imaging System (THEMIS) derived imagery [23] and the creation of rendered and animated 3D flyovers, as a full stack development environment to create RESTful services to expose underlying computational libraries through web based interfaces [18], and in utilizing HPC resources through the IPython notebook interface for proof-of-concept exploratory, big data analysis of the Kaguya Spectral Profiler data set [e.g., 19]. Scripted job submission has provided an easy-to-use interface for requesting and using HPC resources as if they are a local computer script. The development of a RESTful web interface to an analytical library provides the capability to hide the utilization of HPC resources from the end user, significantly reducing complexity. Finally, the use of IPython notebooks and a computing cluster for many-core exploratory data analysis has provided an ideal interactive environment for the development of metrics for use in larger scale automated analysis methods[1].

For the development of parallel, scalable algorithms Python offers three primary tools. First, the built-in multiprocessing module is ideal for Symmetric Multiprocessing (SMP) machines (e.g., desktop computers) where a single shared memory space is advantageous. This type of parallel computation is often used when processing large raster datasets. Second, vectorization, supported by the Numerical Python (NumPy) library, provides significant speedups for vector or matrix based computation. Image and spectral data processing are primary applications of this type of serial performance improvement technique. Finally, The Message Passing Interface (MPI) for Python (mpi4py) package offers Python native access to the MPI standard. More complex parallelization efforts, such as spatially constrained optimization, can significantly benefit from higher levels of communication across a highly distributed system.

We continue to identify use cases for high performance data storage formats, such as use of the Hierarchal Data Format (HDF5) for the storage of photogrammetric control networks and complex model output such as the multilayered thermal-diffusion model (KRC model [17]). In conjunction with Pandas, a Python library originally developed for robust big data quantitative financial analysis, there have been significant data storage reductions (due to compression) and analytical performance improvements (due to robust underlying algorithms). Future work will focus on providing concurrent access

[1] See http://tinyurl.com/q76qkod for an example

to these data structures in HPC environments for scalability testing.

Legacy Code Bases: The redevelopment of an existing code base in a new language can be a costly, ill-advised endeavor due to the aggregate time already invested in the original development and the difficulty in regression testing between implementations. To that end, f2py and the Python native CTypes libraries provide two invaluable tools for wrapping legacy Fortran and C code, respectively. While the complexity of the wrapping scales with the complexity of the underlying code, we note that most Fortran subroutines are immediately wrappable with simply the definition of a few variable types. Likewise, wrapping of a C (or C++) library requires minimal additional development. Assuming that a complex legacy system can be split into smaller components, code portability can be readily realized. The additional development can be focused external to the algorithm logic, helping to reduce the potential to introduce bugs.

While f2py and CTypes frequently find application working with legacy systems, significant benefit can be realized with actively developed code bases. In the context of an HPC system, the ability to write and wrap small algorithm components in low level, high performance languages, while still maintaining rapid development via a higher level language is essential. This is primary reason why Fortran, C, and Python are considered dominant HPC languages. In practice, we most frequently apply this approach when performing a sequential operation for which vectorization is unsuited.

IPython / Jupyter: The IPython project [20], recently renamed to Jupyter, is composed of a local, lightweight web server and browser-based interface which allows for development, inline images, and LaTeX or MarkDown structured mathematics. In addition to Python, IPython also supports other environments and languages, for example Julia, Haskell, Cython, R, Octave (a MatLab alternative), Bash, Perl, and Ruby. We find extensive application of IPython notebooks for exploratory data analysis in the context of model development and validation, local and remote data access testing, for example in reading complex binary data structures, GUI development where an interactive window is spawned from within a web browser, interfacing with our HPC resources, and finally portability of analytical methods and results to collaborators. For this final use case, shipment of a single, Javascript Object Notation (JSON) file and any supplemental data files, e.g. Planetary Data System (PDS) image file, is all that is required for complete reproducibility. Each instance of an IPython notebook is run local to a single desktop computer and the new Jupyter project offers the ability to run a single access server to a distributed set of users.

Graphical User Interface Development: Python provides an ideal platform for the development of high end Graphic User Interfaces (GUIs), as well as stand-alone visualizations. Libraries such as PyQt, PySide, WxPython, and Tkinter offer access to robust GUI development libraries. At USGS Astrogeology, we have developed multiple cross-platform, stand-alone GUI interfaces in pure Python using PySide to call the Qt4 library. These tools are rapid to develop, robust to maintain, and relatively straight-forward to deploy.

Conclusion: Use of Python for scientific computing and data processing in planetary science is well underway. While research projects at USGS are now using Python tools, the tools generally are not made public for more general use. We are currently exploring ways to integrate both existing and new Python software into the USGS Astrogeology ISIS software [e.g., 21 and references therein] so that more general planetary applications can be realized.

References: [1] Laura et al., 2014, LPSC XLV Abs. #2226. [2] Leone et al., 2014, LPSC XLV Abs. #2058. [3] Sylvest et al. 2014, LPSC XLV, Abs. #2309 [4] Neakrase, et al., 2013,LPSC XLIII, Abs. #2557. [5] Cikota et al., 2013, LPSC XLIV, Abs. # 1520. [6] Hare et al., 2014, LPSC LXV, Abs. #2474. [7] Lust and Britt, 2014, LPSC LXV, Abs. #2571. [8] Watters and Radford, 2014, LPSC XLV, Abs. #2836. [9] Laura et al., 2012, LPSC XLIII, Abs. #2371. [10] Levengood and Shepard, 2012, LPSC XLIII, Abs. #1230. [11] Gaddis et al., 2013, LPSC XLIV, Abs. #2587 [12] Oosthoek et al., 2013, LPSC XLIV, Abs. #2523 [13] Calzada-Diaz et al., 2014, LPSC XLV, Abs. #1424. [14] Narlesky and Gulick, 2014, LPSC XLV, Abs. #2870. [15] Gaddis et al., USGS Open-File Report 2014-1056. [16] Atkins et al. (2003), Revolutionizing Science and Engineering Through Cyberinfrastructure. [17] Fergason et al., this meeting. [18] Laura, J. et al., 2014, Development of a restful api for the python spatial analysis library(2014)., 61st Annual NARSC. [19] Gaddis et al., this meeting. [20] Pérez, F and Granger, B.E., 2007, Comp. Sci. and Engin., 9:21–29, doi:10.1109/MCSE.2007.53. [21] Keszthelyi et al., 2013, LPSC XLIV, abs. #2546. [22] Wang et al., 2013, CyberGIS: Blueprint for integrated and scalable geospatial software ecosystems. IJGIS, [23] Fergason et al. LPSC LXIV Abs. # 2822.