SUCCESSFUL DESIGN PATTERNS IN THE DAY-TO-DAY WORK WITH PLANETARY MISSION DATA.

K.-M. Aye, Laboratory for Atmosphere and Space Physics, University of Colorado Boulder, 1234 Innovation Drive, Boulder, CO 80303 (michael.aye@lasp.colorado.edu)

Introduction: After several years of processing and analyzing data from solar system missions and Citizen Science data I have recognized a few repeating patterns in the supporting software I created for working with these data.

Specifically, for the case of explorative data analysis, where it is often necessary to study and compare data from different processing or quality levels, or between the new derived product and the incoming data, three support structures turned out to be the most helpful patterns. These most prevalent patterns are what I call path managers data (or database) managers, and metadata searches.

For data processing, I have established a few simple tools that enable me to execute embarrassingly parallel workloads in parallel on multi-core machines without getting into parallel programming details.

Data(base)Manager: Often one has to work with different sources of data for the same analysis, either because one production set is newer and/or not published at the same source, or because one set has been produced as a derivative from the other. Other use cases include managing different versions of data dumps where for regression checking it is important that the same code can be executed easily with different sets of incoming data.

In these cases, it is very helpful to define a software tool (a Python class in my case) that helps with easy finding and access to the different data.

The characterizing features are:

1) distinguish between and provide different versions of datasets in the project storage folder; 2) node/machine-dependent storage locations (e.g. laptops usually have less capacity than desktops); 3), sub-dataset retrieval functions that are able to filter the currently active data-set for keywords and/or ranges of parameters, easily providing a subset for the ongoing work task.

Node/machine-dependent storage locations can be managed by a configuration file, that, once created, are used by the DataBaseManager for looking up the relevant paths for the current machine. This provides the user with abstraction from the local storage, while the interactive data-analysis focuses on receiving a data-object of interest. An example use looks like this:

db = DBManager() # using paths from config file

or

db = DBManager(temp_path) # using temporary paths to test a new dataset.

Finally, to receive data of interest, I simply do:

data = db.get_file_id(id)

This design abstracts me from distracting and disturbing long path management.

PathManager: Hand-in-Hand with the DataBaseManager, I employ what I call PathManagers. While sounding similar to manage database storage paths (those are managed by config files), this class is designed to help with data products during complex and long processing pipelines. Each project will have, either defined by others, or by the researcher themselves, a structure of paths where different levels of data, or metadata data for the actual data are stored.

For spacecraft missions this results in a large list of possible paths below the database path, for each and every data sub-product and it is very time-consuming to manually keep track of these while performing real-time explorative data analysis.

In recent projects I have implemented PathManagers to deal with this problem.

Basically, a (currently hard-coded) class structure is defined that has attributes for each kind of sub-data paths underneath a common observation id. Often, the actual file paths are named in strategic ways to enable alpha-numeric sortability, but those are usually not supportive for real-time data analysis, for being hard to remember or overly structured. The PathManager enables an abstraction layer between how products are stored on disk (=more structured) and how they are efficiently accessed during interactive data analysis (=more memorizable).

For example, for a specific HiRISE observation ID, there are paths to many products related to one observation ID, like COLOR, RED, and combined mosaics.

During explorative analysis, I am able to call up a PathManager object for a given OBSID and database, and an PathManager attribute called "red_mosaic" would provide the full path to that image product simply like so:

pm = PathManager(obsid, db) # receiving a DataBaseManager from above

print(pm.red_mosaic)

pm.red_mosaic now contains a long path starting from basic storage as managed by the DataBaseManager object, but also, inside that, for the given OBSID, a complex file name construct indicating the access path to the MOSAIC made from only the RED products, inside that database, under the given

OBSID folder. The beauty of it, again, is that all the distracting path management has been hidden and the user can focus on providing the path object to the next tool in the processing chain.

Metadata searches: Finding data often involves the search through metadata. To enable this without using often cumbersome and mostly un-controllable web interfaces, the Planetary Data System archive requires the delivery of a cumulative index file that summarizes a chosen set of metadata for each observation id of a data set.

I have found a way to ingest these metadata tables into a pandas DataFrame for convenient search queries. However, one current problem is that each delivery to any of the PDS nodes comes with a new cumulative index file that is stored into a new subfolder.

This creates problems for tool creators like me that want to provide the newest metadata for a given mission instrument.

It basically would create the need to parse the html code for the most recent folder that can be found which could be done at a hacking session at this conference, but recently, the Rings-Moons node has implemented my suggesting to create these static URLs that will link to the most recent delivered cumulative index file.

Using the remote database API that is provided by the PDS Rings-Moon node I created a tool that enables me to plug in an IMAGE_ID from a paper I read, retrieve the image, store it automatically at the right place using the above described DatabaseManager, and start a recalibration processing chain, if required.

Abstraction: For my efforts in creating a tool-set for planetary science in Python, similar to what `astropy` did for astrophysics, I believe it would be helpful to abstract these patterns into either a) configurable templates that should be easy adaptable for any new projects, so that it could become either part of the `planetpy` package, or b), become part of a so called "cookiecutter" project template that could be used to create a new software package for a new science project.

I will provide concrete suggestions on how this could be done and am asking for community feedback on the best way forward.

The above mentioned metadata approach using pandas `DataFrames` is ready to be implemented into the `planetpy` project, it might just be more or less cumbersome to add the most recent cumulative index file to the current project, depending on the willingness of the PDS node to provide static links to these files.

Parallel processing: An often occurring case is the application of the same function on a large set of images or other data products. As these operations all occur in the subfolder of a given data product, these operations are all independent and can be executed in a parallel fashion without creating race conditions. These workloads are defined as being "embarrassingly parallel" and it is still not the case that available analysis suites provide easy-to-use toolsets to process these automatically across all cores on a multi-core desktop PC. Using the parallelization modules of IPython [1], I have implemented a few small tools that enable me to execute a function on a list of products easily, without having to think about setting up or starting parallel engines and such. This tool also provides feedback via a graphical progress widget, as provided by the Jupyter notebook tools.

For the conference I will present and discuss these tools [2] using examples of my work with medium-sized Citizen Science databases and planetary imaging data.

References:

[1] Fernando Pérez, Brian E. Granger, IPython: A System for Interactive Scientific Computing, Computing in Science and Engineering, vol. 9, no. 3, pp. 21-29, May/June 2007, doi:10.1109/MCSE.2007.53. URL: http://ipython.org

[2] K.-Michael Aye, & jocu8995. (2016, November 11). michaelaye/pyciss: PathManager and databases (Version v0.6.0). Zenodo. http://doi.org/10.5281/zenodo.166116.