Developer Handover Notes - February 2016

Monday, 1 February 2016

4:14 PM

Good Documents to Read First

|  |  |
| --- | --- |
| **Document** | **Comments** |
| sambuca/reference/docs/Installation on Windows.pdf  sambuca/reference/docs/Sambuca and Bioopti on Windows - WinPython.docx  sambuca/reference/docs/Sambuca and Bioopti on Windows.docx  sambuca/reference/docs/sambuca\_install.mp4 | Installation on Windows, with a lot of handholding. Intended for the SAMBUCA team who primarily develop on Windows machines. Installation on Linux is much more straightforward, and is documented in the Sphinx docs (build the html target and read that). |
| sambuca/reference/notebooks/New Sambuca.ipynb | My final work on the project. Contains a lot of documented code showing how to use the sambuca and sambuca-core code to process a raster.  Note that this is all serial code, but I have put some ideas down about how the parallel processing could be approached.    The other notebooks may be interesting, but I don't know if they still run. I have not kept them up to date with the code changes. |
| bioopti\_data | The canonical reference for the data formats supported by the sambuca and bioopti code. Note that the unit tests do not reference these files directly, so if a format changes, the relevant unit test data files will also need to be updated. |
| Sphinx docs for sambuca, sambuca\_core, and bioopti | Run the "make html" target on the utility makefile |
| My presentation on Python packaging | All these projects follow the guidelines in that presentation. |
| The code and unit tests |  |

Major Items on the To Do List

1. Consider creating JIRA tickets somewhere.
   1. While it was just me developing the code, I used some GTD software running locally to manage my tasks. Fine for me, terrible for collaboration. Setting up some JIRA tickets somewhere will be handy if you have more people working on this (yourself, other SC staff, Steve, Janet etc). Most of these bullet points could translate into tasks or epics.
2. Parallel processing
   1. Task is embarrassingly parallel, which helps greatly. Optimisation for each pixel is independent.
      1. Tiling a large raster, having parallel processes compute a tile, and then reassemble the results is the main approach I planned
   2. I was planning on using IPython parallel for parallel processing, but there are other options that might work. I ramble on about an experimental framework built on rasterio in New Sambuca.ipynb
3. Running on NCI
4. AGDC integration
   1. Is this in or out of scope for your work? Ask Janet.
   2. Note that the AGDC has inbuilt support for parallel processing on a cluster.
      1. However, there is a requirement that Sambuca be fully functional in a stand-alone mode (no AGDC), so whether you use the AGDC parallel model or not, Sambuca will still require it's own parallel solution.
5. Decide whether the spatial/parallel sambuca runs on NCI will be orchestrated at the highest level from a Jupyter notebook or from a batch script. This will impact:
   1. how the code needs to be installed
   2. how the runs are configured (see below point)
   3. how integration to AGDC is achieved
      1. may also be driven in part by whether you use the AGDC parallel processing or not for AGDC runs.
   4. My inclination was towards pure CLI/batch mode, using ini-style configuration files for a run. It is easier to automate. But they are not mutually exclusive options.
6. Logging. Sambuca and sambuca\_core do not have logging, but they should. Be sure to follow the guidelines on setting up logging inside a Python library. The key point is that the library should respect the logging settings of the calling code, and also log to a separate namespace so that logs can be filtered.
   1. The bioopti code has extensive logging which could be used as a guide.
7. Finish implementation of reference data formats in the bioopti\_data repo.
   1. It really depends on what the SAMBUCA team require
8. Capture of sambuca raster outputs into appropriate file structures.
   1. the current pixel result handler class simply captures a bunch of hardcoded values into memory. This should become:
      1. data driven
      2. saveable to file (perhaps in a separate step once the run has finished)
9. Spectral input data interpolation/resampling
   1. The sambuca model operates on discrete 1nm bands (specified by the num\_bands and wavelengths) inputs. For the model results to make sense, all spectral inputs must have the same coverage of exact 1nm bands. Much of the input data (substrates, SIOPS etc) is already sampled to the required 1nm bands, but much of it is messier. Some input data has less resolution, while some is higher than 1nm. Additionally, much of the input data has unequal input band spacing.
   2. Current functionality:
      1. data loading functions in sambuca\_core contain validation code that checks for
         1. exact 1nm wavelengths
         2. strictly increasing wavelengths
      2. If the validation fails, then an exception is raised
   3. Required functionality:
      1. sensible interpolation of uneven input data to 1nm bands
      2. Needs to handle both under and oversampled data (relative to the 1nm bands used by the model)
      3. Linear interpolation probably makes sense, but check with the team.
10. Sensor filter handling:
    1. The sambuca model uses a number of spectral inputs. For the model outputs to make sense, all spectral inputs must cover the same number and range of 1nm bands.
       1. For example, if one input has values from 350 to 700nm, and another covers only 400nm to 700nm, then the model should be restricted to the 400 - 700nm range.
       2. There is code in sambuca\_core that accepts an arbitrarily long set of spectral inputs (wavelength, data) numpy array tuples and returns the lowest common set of wavelengths.
       3. There is also code that will use a wavelength range to mask the inputs, effectively preparing them for the model.
    2. Point a is already implemented. However, sensor filters require special handling. It is not considered appropriate to simply truncate the wavelength range of each sensor band to the lowest common subset of wavelengths.
       1. One suggestion is that any sensor band that contains non-zero values at wavelengths outside the common subset should be removed entirely from the analysis. This will mean dropping the entire band from the sensor matrix used to transform the modelled spectra. Of course less bands makes it harder to converge on a parameter set.
11. Command line configuration for sambuca.
    1. bioopti has command line configuration using configargparse. This allows a single set of command line argument definitions to be specified through a hierarchy of environment variables, ini-style config files, or command line arguments.
    2. If sambuca is going to be run from batch jobs on a cluster, then configuration options will probably be required, and the configargparse approach used in bioopti would work.
    3. Alternative approaches:
       1. GUI configuration. The IDL sambuca had a GUI that let the user configure a run. If this is still seen as essential, I would suggest using configargparse and gooey to generate a GUI wrapper for customising a run.
       2. Running sambuca from a notebook.
          1. If it is decided to do production runs of sambuca from a Jupyter notebook (rather than a CLI batch script), then configargparse may be of less value. In a notebook, I find it easier to simply have one or more code cells containing the configurable values.
12. Python package metadata
    1. sambuca and sambuca\_core setup.py files do not contain very good metadata. This needs to be filled in based on information from the sambuca science team.
13. Consider the relationship between the existing bioopti\_data reference data, and the requirements for a Sambuca reference dataset.
    1. They should be the same file formats, as both projects use the IO functions in sambuca\_core
    2. However, the scope of included data will likely be difference for bioopti and sambuca.
       1. bioopti is planned to be released as an open source project. To do so requires a simple reference dataset. This is the intended use of bioopti\_data.
       2. Sambuca will require additional datasets that should not be released with bioopti.
    3. To reconcile these positions, you may need to create a sambuca\_data repository that can contain all sambuca input data (apart from the remotely sensed reflectance rasters that would be sourced from AGDC or elsewhere). It probably doesn't matter if there are some duplicated files between sambuca\_data and bioopti\_data so long as they are kept up to date.
14. Separate Windows installation documents don't make sense. It would be better if some form of the Windows-specific installation instructions found their way into the Sphinx generated documentation for Sambuca. That is intended to be the definitive project documentation.

Original IDL Code

There is a VM (idl-01 on nexus) that contains the original IDL code for Sambuca and Bioopti should you need to refer to this. I think I have finished reimplementation of all the IDL functionality, but you may need to refer to it at some point. Also, I have created a self-contained Windows installation of the code there for reference. See e:\sambuca\_project once you are logged in.

ArcGIS and ENVI

The IDL-01 VM has full versions of ArcGIS, ENVI, and IDL. This can be handy when working with some of the input data formats, or \*\*shudder\*\*, the original IDL code.

Running the Jupyter notebooks on SC systems

Note that I run all the code in a Python virtual environment. While this is fine on SC systems, running a Jupyter notebook with the kernel executing in a virtual environment can require a bit more work to set up the correct kernel spec. I have some that I set up last year, and Sam is familiar with the process (as well as having plans for a better approach).

Notes on Development

All code should be written as Python 3 code. Each file contains imports from the python-future package to maintain backwards compatibility with Python 2. Although I developed on Python 3, the unit tests (at a minimum) should be executed on Python 2.7 from time to time. I don't see any point in testing on Python <= 2.6 unless constrained to do so by NCI.

Development is far easier on Linux than Windows.

If you want to use an IDE (rather than text editor + unit tests in a terminal), I found PyCharm Community edition to work the best. You don't need any of the features from the paid PyCharm version.

The pydoc strings use the Google Style. I find it easier to type and more readable than the default Python style. I suggest sticking to this style for the project, so that code is consistent.

Note that in sambuca, you will see both a "reference" and a "docs" directory. The docs directory is for Sphinx documentation templates only. All other supporting documentation should go into reference, often in "reference/docs".

Windows Development

Although it is sometimes a battle, it is important to make sure that all the code and dev environment (PyCharm is recommended) continue to work on Windows. Additionally, for Bioopti (which has an open source release planned), we should not constrain the end users.

Although Anaconda should work, I recently encountered issues with specific Anaconda packages that led me to move to WinPython. Thus some of the older documentation refers to Anaconda on Windows, while the newer stuff uses WinPython.

The specific Anaconda issue that I remember (there may have been others!) is that their matplotlib packages are not compiled with functional tkagg backend support. As this is required for embedding the matplotlib canvas into the bioopti TK GUI, using Anaconda requires manually replacing the Anaconda matplotlib library with an appropriate one from [here](http://www.lfd.uci.edu/~gohlke/pythonlibs/). I moved to the portable WinPython distribution hoping to avoid manual package installations on Windows, but the rasterio and GDAL packages required by the New Sambuca.ipynb are not provided by WinPython. So either way there is the requirement for manual package handling on Windows.

I did have some issues on Python 3.5 and WinPython, but I don't recall what they were. There is no intrinsic reason for the code to fail with Python 3.5, as it works fine with Python 3.5.1 on my Arch Linux box.

pytest sugar plugin

Note that the setup.py files list pytest-sugar as a dependency for dev mode. If your terminal supports it, this gives pretty unit test output:

Machine generated alternative text:
(sambuca) [dc@nibblet sambuca core] $ make test 
Test session starts (platform: Linux, Python 3.5. 1, pytest 2.8. 3, pytest-sugar 0.5. 1 
cachedir: 
rootdir: 
plugins: 
sambuca 
sambuca 
sambuca 
sambuca 
sambuca 
sambuca 
. cache 
/home/dc/code/sambuca-proj ect/sambuca_core, inifile: setup .cfg 
sugar-@.5.1 
core/ tests/ test 
core/ tests/ test 
core/ tests/ test 
core/ tests/ test 
core/ tests/ test 
core/ tests/ test 
forward model . py 
sensor filter. py 
sensor filter loading. py 
spectra loading. py 
spectra operations . py 
utility_numpy . py 
Results (3.24s) : 
68 passed 

However, on Windows and on some Linux terminals, the output will be a garbled mess. I haven't looked into whether it is an issue with missing fonts, unicode support, curses, or basic terminal capabilities. If the output from your unit tests doesn't look like the screenshot, just uninstall the pytest-sugar package from your virtual environment.