Swampy Main Call

Note that I have 21hr flight to improve on this acronymn. Don't get too attached....

- Utility functions used to populate dictionaries and arrays required to be passed to core SAMBUCA inversion algorithm
- Combination of hardcoded and local path calls are currently required, with the 'end goal' of defining XMLs to pass straight to these dictoniaries, dervied by any API/method (manual, SNAP GUI, AGDC API etc)
- Processing and Output options are hardcoded in corresponding modules as noted.

Developments to Date that extend the original Brando et. al (2009) SAMBUCA implementation:

- Inclusion of 3 substrates in the inversion
- The option to relax the sum-to-one constraint on the substrate composition (see Petit et. al. (2017))
- Option of selecting bands with which to use in the inversion

NOTE: Sambuca modules as well as Sen2Coral utility scripts have been amended from the SAMBUCA implementation due to a mismatch between how the array dimension is represented. The image data is read to an array (bands, rows, columns), however the original sambuca code treats this as (bands, x, y) or (bands, width, height). Original testing on a square image did not highlight this bug. The original sambuca repository needs to be fixed still.

Set up the environment

In [ ]:

```
import numpy as np
import sys

#import Marcos script modules and set up path to these

sys.path.append('C:\\Users\\PCUSER\\sambuca_project\\Swampy\\sen2coral')

import sambuca_input_rrs
import sambuca_input_parameters
import sambuca_preparation
import sambuca_calculations
import sambuca_outputs
import define_outputs


# set some controls on numpy formatting
# 5 decimal places, suppress scientific notation
np.set_printoptions(precision=5, suppress=True)
```

Now import the sambuca and sambuca-core packages:

In [ ]:

```
import sambuca as sb
import sambuca_core as sbc
```

In [ ]:

```
# Main statement needed if exporting to .py script.
#if __name__=='__main__':
```

Define a base project input path

In this path the code expects the following folders

- image
- siop
- nedr
- substrates
- sensor_filters

This path can used to point to XML files to ingest when ready for that. At the moment, files specified in the input folders are hardcoded in sambuca_input_rrs and sambuca_input_parameters modules.

In [ ]:

```
base_path = 'C:\\Users\\PCUSER\\sambuca_project\\Swampy\\input_data\\'
```

Load the Input Image Data

sambuca_input_rrs.py

- The code requires below-surface remote sensing reflectance (rrs)
- The input data is loaded into an observed_rrs array of shape (bands, rows, columns) using rasterio.
- Other image metadata is loaded into a tagged image_info dictionary based on rasterio src class. (e.g crs, affine, width, height)
- Full sensor filter dictonary is loaded, and specific sensor filter extracted based on sensor id key
- If Rrs is set to True, the user is supplying above-surface remote sensing reflectance. This is coverted to below-surface rrs if True is set. Default value if not set is False
- There is a hardcoded option to select a sub-set of bands to run the inversion on. The user must ensure that the sensor filter is then of the same size. For example: S25 for 5 bands of Sentinel-2. The user must also make a sensible choice based on the minimum common wavelengths of the other input spectra (substrates, a_ph_star, awater) (see the Prepare the Data call)
- Noise Equivilant (nedr) is also loaded here, with the path specified. Similar to the sensor filter, the user must also make sure the number of bands in the nedr selected matches that of the selected bands in the input data.

File names and sensor id key are currently hardcoded.

In [ ]:

```
[observed_rrs, image_info]=sambuca_input_rrs.sam_obs(base_path, Rrs = True)
```

Load the Parameters

sambuca_input_parameters.py

Builds Dictionaries to pass to the data prep and inversion

- SIOPs: spectral, values, substrates, free parameter bounds
- Enviromental data: sun angle. off nadir, q factor

Currently hardcoded and read in from input data folder, aim to build from XML

Bounds for the three substrates here should reflect if the user is using a sum to one constraint, or the relaxed constraint (RASC) (specified in the Run the Inversion step).

- If Sum-to-One, lower bounds = 0 and upper bounds = 1 for each substrate
- For RASC, lower bounds = 0 and upper bounds = upper limit of RASC.

In [ ]:

```
[siop, envmeta]=sambuca_input_parameters.sam_par(base_path)
```

Prepare the Data

sambuca_preparation.py

- Prepares the spectral inputs (currently truncates to a common wavelength range).
- Builds a fixed parameter set out of truncated inputs, siop and envdata dictionaries.
- Prepares a result recorder for storing array based results.
- Defines the objective function (Hardcoded call currently made to specific error function here)

In [ ]:

```
[wavelengths, siop, image_info, fixed_parameters, result_recorder, objective]=sambuca_p
reparation.sam_prep(siop, envmeta,

                    image_info)
```

Run the Inversion

sambuca_calculations.py

Sets up and runs the inversion:

- Type of SciPy optimisation used
  - For RASC application, perhaps only SLSQP is appropriate
- Definition of an image subset to be processed if required (primarily for testing; as full result recorder array extent of original image is still used, and crs/affine may not translate when written to geotiff in outputs)
- Parameter starting points (currently mid point of range)
- Option to select and set relaxed constraints on substrate abundances (RASC), or to maintain sum-to-one
  - Suitable substrate parameter bounds should also be set (see Load the Parameters)
- Option for SAMBUCA option of 'go shallow', when set to true, retrieves as shallow depth as possible for optically deep waters whilst maintaining an SDI value of < 1.

Returns the results, spatial subset range processed, and number of pixels processed

In [ ]:

```
[result_recorder, coordinates, num_pixels]=sambuca_calculations.sam_com(observed_rrs, o
bjective, siop,

result_recorder, image_info, shallow = True)
```

Define and Write some Outputs

define_outputs.py

- Outputs types wanted are defined in define_outputs.py and call to a writeout function within sambuca_outputs.py
  - Examples can be extended. Some variables are calculated on the fly in here, for example, the total abundance of the three RASC substrates proportions and the normalised RASC fractions.
- Written out as geotiffs with crs info inherited from the input rrs data
- Examples options of writing to png plots generated by MatPlotlib
- If a subsection of image to be processed was selected in sambuca_calculations, coordinates ensure only this section is written (validation needed to see if crs and affine info translates correctly for a subset write)

In [ ]:

```
define_outputs.output_suite(result_recorder, image_info, coordinates)
```