iceMACS

Collection of tools to calibrate and manage SWIR and VNIR data from the specMACS system, as well as retrieve ice cloud optical properties using a bispectral Nakajima-King retrieval.

Todos

- Change string logs to modern f-string syntax
- Change `open()` calls when reading files to `with open()` in order to ensure files are closed when exception occurs.
- Instead of
- Find a way to avoid * imports in **`init`** file
- Unify LUT generators, preferably into one single function.
- Restructure submodules to avoid confusion. Add classes.
- Add examples
- Complete documentation under usage
- Add a git submodules functionality
- Find better way to organize paths
- Try to replace \ line continuation with brackets
- Add documentation for SceneInterpreter

Structure

The submodules in the `iceMACS` package are organized as follows:

- The `paths` submodule defines global paths specific to you system. Adapt before usage.
- `conveniences` contains functions that are non-essential to the retrieval but are compatible with other functions and sometimes called by the `tools` submodule.
- `tools` contains functions to interpret camera data and add new variables, such as reflectivities, ice index and relative view angles. The updated `PixelInterpolator` class is defined here.
- Rest to be determined...

Usage

SWIR bad pixel interpolation

Many A(C)³ scenes are relatively dark, with a high solar zenith angle and low cirrus rediance values. Some pixels are shown to be unreliable under these conditions. The `PixelInterpolator` class finds these pixels and interpolates for the entire scene. Additionally, interpolation over invalid pixel from the bad pixel list is performed, analogous to the `runmacs BadPixelFixer`. Initiate with loaded SWIR dataset, containing the variables `radiance`and `valid` access "badness" signal with

```
from iceMACS.tools import PixelInterpolator
interp = PixelInterpolator(swir_ds, window=3)
interp.show_signals()
```

The `window` variable sets the moving average frame size. Choose a fitting cutoff value for each plotted wavelength and pass as `list`, e.g.

```
interp.add_cutoffs([4, 1.2])
```

Adjust cutoff as needed and apply filter with

```
filtered_radiance = interp.get_filtered_radiance(with_bpl=True)
```

where also interpolating pixels from bad pixel list is the default.

Data formatting

The `SceneInterpreter` class takes calibrated loaded SWIR and VNIR datasets, as view angles and solar position datasets and fascilitates computation of variables that need to be passed to the `LUTGenerator` functions. Initiate with

```
from iceMACS.tools import SceneInterpreter
scene = SceneInterpreter(swir_scene, view_angles, solar_positions)
```

and get summarized scene geometry with

```
scene.get_scene_overview()
```

Add relative view angles and reflectivity variable with

```
swir_scene['reflectivity'] = scene.get_reflectivity_variable()
swir_scene['umu'] = scene.get_umu_variable()
swir_scene['phi'] = scene.get_phi_variable()
```

Bispectral retrieval (BSR)

Habit detection

Mystic simulations at wavelengths sampling the pol camera spectral response function have to be "calibrated" in order to reporoduce the actual measured signal. The `polLutInterpolator` class provides functions to get simulated reflectivities representing polA/B signals. Initiate with LUT as xarray dataset

```
from iceMACS.tools import polLutInterpreter
interp = polLutInterpreter(polLUT)
```

and calibrate with

```
interp.calibrate(inflight_calibration_file)
```

You can check if the `interp` object is calibrated with `interp.calibrated`. The normalised spectral response, rescaled Stokes parameters and calibrated radiance are also available as object properties. Compute calibrated reflectivities, relative to kurudz E_0, with

```
interp.get_polarized_reflectivity(calibrated=True)
```

with `False` being default.

Additional functionalities