iceMACS

Collection of tools to calibrate and manage SWIR, VNIR and pol cam data from specMACS, as well as retrieve ice cloud optical properties and habit estimates using a bispectral Nakajima-King retrieval and angular 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.
- Find a way to avoid * imports in `__init__` file
- Unify LUT generators, preferably into one single function.
- 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
- Update documentation for filter
- Add type hints
- Include functions for polarized retrieval
- Fill the examples directory
- Move repo to gitlab

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. For example, loading data from the A(C)³ archive directory, plotting, reading and writing NetCDF files etc.
- `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 (AC)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.filtered_radiance(with_bpl=True)
```

or

```
filtered_radiance = interp.interpolated_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.overview()
```

Add relative view angles and reflectivity variable with

```
swir_scene['reflectivity'] = scene.reflectivity()
swir_scene['umu'] = scene.umu()
```

```
swir_scene['phi'] = scene.phi()
```

or get summarized scene information with

```
scene.merged_data()
```

LUT generation

LUT handling and inversion

The LUT dataset containing the simulation results can be passed to the `BSRLookupTable` class. To initiate call

```
from iceMACS.tools import BSRLookupTable
LUT = BSRLookupTable(LUT_ds)
```

or from path

```
LUT = BSRLookupTable.from_path('LUT_ds_path')
```

The dataset has to contain the two wavelengths intended to be used in the retrieval. Original data is saved in 'LUT.dataset' You can visualize the splitting of reflectivities with

```
LUT.display_nadir()
```

The `BSRLookupTable` class provides an automated lookup table inversion based on Paul's `luti` package. Call

```
invertedLUT = LUT.inverted(num=200, alpha=4)
```

where `num` is the sample number within the relevant reflectivity range and `alpha` is a parameter used to define the convex hull. `alpha=4.` has been found to work well for bispectral cloud lookup tables.

BSR retrieval

The inverted lookup table contains reflectivities as coordinates and the cloud parameters in the variable `input_params`. Without any further formatting, you can pass the inverted dataset to the `SceneInterpreter` instance you want to retrieve by calling

```
scene.cloud_properties_fast_BSR(invertedLUT, LUT.wvl1, LUT.wvl2,
                                LUT.Rone_name, LUT.Rtwo_name,
                                umu_bins=20, phi_bins=50, interpolate=True)
```

Here, `LUT` is the `BSRLookupTable` instance that produced the inverted dataset. `interpolate` chooses the method by which the simulations are cut to pixel geometries. `True` interpolates between simulated viewing geometries while `False` chooses the closest existing coordinate.

Angular habit retrieval

Additional functionalities