Tutorial I

Reading in data

This first tutorial explains how LIS output can be read in.

Table of contents

Always start your script or notebook by enabling `pylis`. Replace the path in the following block by the foler in which your version of `pylis` is located.

In [ ]:

```
import sys
sys.path.append("/dodrio/scratch/projects/2022_200/project_output/rsda/vsc34049")
```

1. LIS model output

The `pylis.readers` module contains routines that enable reading in LIS output. The `lis_cube` function reads in a data cube of a model variable (3- or 4-dimensional). The inputs are:

- `lis_dir` : where is the LIS output stored? This is the path provided in the `lis.config` file, i.e., don't include `"SURFACEMODEL"` ;
- `lis_input_file` : the LDT output with the LIS domain. We need this for the latitude and longitude (these variables are masked over water in the LIS output);
- `var` : the variable for which to read in the data cube. Make sure to use the correct suffix (`_inst` or `_tavg`);
- `start` : first date of the data cube;
- `end` : last date of the data cube;
- `subfolder` : only change if your data are not stored in the `"SURFACEMODEL"` folder (default);
- `d` : only change if your are using nested domains (default `"01"`);
- `freq` : only change if you don't have daily outputs (default `"1D"`).

As an example, we can read in a data cube of soil moisture.

In [ ]:

```
from pylis import readers

# we will need the lis_input file to obtain the latitude and longitude
lis_input_file = "/dodrio/scratch/projects/2022_200/project_output/rsda/vsc34049/awu/lis_inp

# read in soil moisture data cube over US (hourly data)
dc_sm = readers.lis_cube(
    lis_dir = "/dodrio/scratch/projects/2022_200/project_output/rsda/vsc34049/awu/notile_noi
    lis_input_file = lis_input_file,
    var = "SoilMoist_tavg",
    start = "01/01/2020",
    end = "31/12/2020",
    freq = "1H"
)
```

```
100%|██████████████████████████████████████████████████| 8761/8761 [05:13<00:00, 27.93it/s]
```

The resulting object is an `xr.DataArray` with in this case 4 dimensions: time, soil layer, and x- and y-direction of the grid. For a regular latitude-longitude grid such as this one, `x` and `y` will have a one-to-one correspondence with the longitude and latitude coordinates respectively. For non-linear grids (e.g., Lambert conformal) this will not be the case, the `lat` and `lon` coordinates themselves are then two-dimensional on the `xy` -grid themselves.

In [ ]:

```
dc_sm
```

Out[ ]:

```
xarray.DataArray    (time: 8761, layer: 4, x: 99, y: 233)

nan nan nan nan nan nan nan ... 0.266 0.2732 0.2753 0.274 nan nan

▼ Coordinates:
    lon          (x, y)        float32   -124.9 -124.6 ... -67.12 -66.88
    lat          (x, y)        float32   24.88 24.88 24.88 ... 49.38 49.38
    layer        (layer)         int64   1 2 3 4
    time         (time)  datetime64[ns]  2020-01-01 ... 2020-12-31
► Indexes:   (2)

▼ Attributes:
    description :   LIS model output
    variable :      SoilMoist_tavg
```

A handy function to compute the root-zone soil moisture is available in the `pylis.help` module:

In [ ]:

```python
from pylis.help import root_zone

dc_rzsm = root_zone(dc_sm)
```

Default depths of the layers are for the Noah-MP model, but any list can be provided through the optional `weights` argument:

In [ ]:

```
?root_zone
```

```
Signature: root_zone(dc, weights=[0.1, 0.3, 0.6])
Docstring:
Compute the root-zone (soil moisture/temperature) based on a weighted average of the layers.
Default weights work for Noah-MP output.
File:      /dodrio/scratch/projects/2022_200/project_output/rsda/vsc34049/pylis/help.py
Type:      function
```

Next time, it is much quicker to read in the data if you store this `xr.DataArray` as a netcdf file. First give the variable a name since the `to_netcdf` function works on `xr.DataSets` rather than `xr.DataArrays` .

In [ ]:

```python
dc_sm.to_dataset(name = "SoilMoisture").to_netcdf("/path/to/folder/dc_sm.nc")
```

In [ ]:

```python
import xarray as xr

# faster next time we need to use the data
dc_sm = xr.open_dataset("/path/to/folder/dc_sm.nc").SoilMoisture
```

2. Innovations, increments and spread

If you ran a DA experiment, you will also have an `EnKF` folder (containing innovations, increments, spread) alsongside the `SURFACEMODEL` folder. You can use the `innov_cube` and `incr_cube` functions to read in innovations and increments. For `innov_cube` , the inputs are:

- `lis_dir` : where is the LIS output stored? This is the path provided in the `lis.config` file, i.e., don't include `"EnKF"` ;
- `lis_input_file` : the LDT output with the LIS domain. We need this for the latitude and longitude (these variables are masked over water in the LIS output);
- `start` : first date of the data cube;

- `end` : last date of the data cube;
- `var` : the variable for which to read in the data cube: `innov` for innovations (default), `ninnov` for normalized innovations;
- `subfolder` : only change if your data are not stored in the `"EnKF"` folder (default);
- `a` : only change if you perform a multi-sensor DA (default `"01"`);
- `d` : only change if your are using nested domains (default `"01"`);
- `freq` : the temporal frequency to which innovations should be resampled by averaging (default `None` , i.e., don't resample).

Example reading the normalized innovations:

In [ ]:

```python
from pylis import readers

# we will need the lis_input file to obtain the latitude and longitude
lis_input_file = "/dodrio/scratch/projects/2022_200/project_output/rsda/vsc34049/nu-wrf/OL_1

sm_ninnov = readers.innov_cube(
    lis_dir = "/dodrio/scratch/projects/2022_200/project_output/rsda/vsc34049/nu-wrf/smap_da
    lis_input_file = lis_input_file,
    var = "ninnov",
    start = "01/01/2020",
    end = "31/01/2020",
)
```

```
Constructing innovation cube ...
100%|██████████████████████████████████████| 15602/15602 [06:15<00:00, 41.59it/s]
```

In [ ]:

```python
dc_ninnov
```

Out[ ]:

```
xarray.DataArray    (time: 142, x: 159, y: 261)

nan nan nan nan nan nan nan nan ... nan nan nan nan nan nan nan nan
▼Coordinates:
    lon     (x, y)        float32  -120.8 -120.6 ... -67.69 -67.46
    lat     (x, y)        float32  22.8 22.84 22.87 ... 47.3 47.25
    time    (time)  datetime64[ns]  2020-01-01T11:15:00 ... 2020-01-...
▶Indexes:  (1)
▼Attributes:
    description :   LIS innovations
    variable :      ninnov_01
```

Behavior is similar for `incr_cube` . Here, the `var` argument relates to the model state (e.g., `"Soil Moisture"` , `"LAI"` , ...). An additional argument `layers` expects `None` for a variable without layers such as LAI, and a list (e.g., `[1]` or `[1,2,3,4]`) for a variable with layer such as soil moisture. Increments that are exactly zero are assigned a missing value: they correspond to times and locations without available observations.

Ensemble spread can be read in using the `spread_cube` function. Inputs are the same as for `incr_cube` . Note that the `freq` argument in this case refers to the model output frequency (as in `lis_cube`), not the desired temporal frequency after resampling. It defaults to `"1D"` .

3. Satellite observations

After completing a DA run in LIS, satellite observations are stored in binary files in the `DAOBS` subfolder. You can read them in via `pylis` using the `obs_cube` function. The inputs are:

- `lis_dir` : where is the LIS output stored? This is the path provided in the `lis.config` file, i.e., don't include `"DAOBS"` ;

- `lis_input_file` : the LDT output with the LIS domain. We need this for the latitude and longitude (these variables are masked over water in the LIS output);
- `start` : first date of the data cube;
- `end` : last date of the data cube;
- `rescaled` : if a rescaling is performed (e.g., CDF matching), you can choose to read in the original or rescaled observations (default `False`);
- `a` : only change if you perform a multi-sensor DA (default `"01"`);
- `d` : only change if your are using nested domains (default `"01"`);
- `freq` : the temporal frequency to which observations should be resampled by averaging (default `None` , i.e., don't resample).

In [ ]:

```python
from pylis import readers

# we will need the lis_input file to obtain the latitude and longitude
lis_input_file = "/dodrio/scratch/projects/2022_200/project_output/rsda/vsc34049/nu-wrf/OL_1

dc_obs = readers.obs_cube(
    lis_dir = "/dodrio/scratch/projects/2022_200/project_output/rsda/vsc34049/nu-wrf/smap_da
    lis_input_file = lis_input_file,
    start = "01/01/2020",
    end = "31/01/2020",
    rescaled = True
)
```

```
Counting the number of observations ...
Constructing observation cube ...
100%|████████████████████████████████████| 15602/15602 [07:08<00:00, 36.45it/s]
```

In [ ]:

```python
dc_obs
```

Out[ ]:

```
xarray.DataArray    (time: 142, x: 159, y: 261)

nan nan nan nan nan nan nan nan ... nan nan nan nan nan nan nan nan
▼Coordinates:
    lon      (x, y)          float32  -120.8 -120.6 ... -67.69 -67.46
    lat      (x, y)          float32  22.8 22.84 22.87 ... 47.3 47.25
    time     (time)    datetime64[ns]  2020-01-01T11:15:00 ... 2020-01-...
▶Indexes:  (1)
▼Attributes:
    description :    Observations obtained form binaray DAOBS files
```

You can easily count the total number of observations in time and space:

In [ ]:

```python
import numpy as np

np.isfinite(dc_obs).sum(dim = ("time", "x", "y")).values
```

Out[ ]:

```
array(148458)
```

4. Ancillary data

There are some more functions in `pylis.readers` to read out the landmask and landcover from the LIS input file.

- `landflag` takes `lis_input_file` as input and returns a binary `xr.DataArray` ;
- `landcover` takes `lis_input_file` as input. It either returns a 2D array of majority landcover classes (if `majority = True`) or a 3D array with fraction of each landcover class (if `majority = False`).

In [ ]:

```
from pylis import readers

# we will need the lis_input file to obtain the latitude and longitude
lis_input_file = "/dodrio/scratch/projects/2022_200/project_output/rsda/vsc34049/awu/lis_inp

lc = readers.landcover(lis_input_file, majority = True)
lc
```

Out[ ]:

```
xarray.DataArray    (x: 99, y: 233)

array([['Water', 'Water', 'Water', ..., 'Water', 'Water', 'Water'],
       ['Water', 'Water', 'Water', ..., 'Water', 'Water', 'Water'],
       ['Water', 'Water', 'Water', ..., 'Water', 'Water', 'Water'],
       ...,
       ['Evergreen Needleleaf Forest', 'Evergreen Needleleaf Forest',
        'Evergreen Needleleaf Forest', ..., 'Water', 'Mixed Forests',
        'Mixed Forests'],
       ['Evergreen Needleleaf Forest', 'Evergreen Needleleaf Forest',
        'Evergreen Needleleaf Forest', ..., 'Water', 'Water', 'Water'],
       ['Evergreen Needleleaf Forest', 'Evergreen Needleleaf Forest',
        'Water', ..., 'Mixed Forests', 'Water', 'Water']], dtype='<U100')

▼ Coordinates:
    lon        (x, y)  float32  -124.9 -124.6 ... -67.12 -66.88
    lat        (x, y)  float32  24.88 24.88 24.88 ... 49.38 49.38
► Indexes:  (0)
► Attributes:  (0)
```