Department of Meteorology

National Centre for Earth Observation
NATURAL ENVIRONMENT RESEARCH COUNCIL

University of Reading

pyenda - The Python Ensemble Data Assimilaton framework

Gernot Geppert | Claire Merker (now at MeteoSwiss)

cen

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

https://gitlab.dkrz.de/pyenda/pyenda

Motivation

Data assimilation (DA) experiments typically employ one of two types of DA implementation:

- **Large DA frameworks** usually offer well-tested implementations with high performance but with limited flexibility. They often have a steep learning curve and may require time-consuming coupling exercises.
- Individual, **purpose-built DA tools** are easy to use and can be modified quickly but they are difficult to re-use for other projects and often require complete re-writes or new implementations.

pyenda, the Python Ensemble Data Assimilation Framework, is a DA framework that facilitates the quick setup of DA experiments with models of any level of complexity. pyenda also makes it easy to modify existing and introduce new DA methods which can make use of the whole Python ecosystem. pyenda furthers offers support for different types of locations and has extensive input and output capabilities.

Why Python?

Python favours ease-of-use and productivity over run-time performance.

- Python is easy to learn and easy to use.
- Python code is easy to read and easy to maintain.
- Python facilitates quick development of new code.
- Python offers a wealth of modules for input/output, scientific computing, statistics, visualization, and many more.

```python
def analysis(X, y, R, HX, model_domain=slice(None), obs_domain=slice(None)):
    HX_mean, HX = HX.get_mean_perturbs(obs_domain)
    Zy = 1./np.sqrt(ensmbl_size - 1) * HX

    try:
        Zy_T_Rinv = np.dot(Zy.T, np.linalg.inv(R[obs_domain][:, obs_domain]))
    except:
        raise Exception("Could not invert R.")
    Zy_T_Rinv_Zy = np.dot(Zy_T_Rinv, Zy)

    G, T = np.linalg.eigh(Zy_T_Rinv_Zy)
    G = np.maximum(G, 0)
    G = np.diagflat(1.0 / np.sqrt(1.0 + G))
    T = np.dot(T, np.dot(G, T.T))
    X_mean, X = X.get_mean_perturbs(model_domain)
    Za = np.dot(1./np.sqrt(ensmbl_size - 1) * X, T)
    K = np.dot(Za, np.dot(T.T, Zy_T_Rinv))
    y = X_mean + np.dot(K, y[obs_domain] - HX_mean)
    return (y + (np.sqrt(ensmbl_size - 1) * Za).T).T
```

Figure 1: ETKF analysis code in pyenda (inline documentation and comments have been removed).

Location handling

pyenda supports

- Cartesian (1D, 2D, 3D),
- polar/spherical,
- geographical (Lat/Lon, Lat/Lon/z),
- and "index" locations.

Each location class includes distance calculations for localization and a function to find the closest location for each location in a given set of locations.

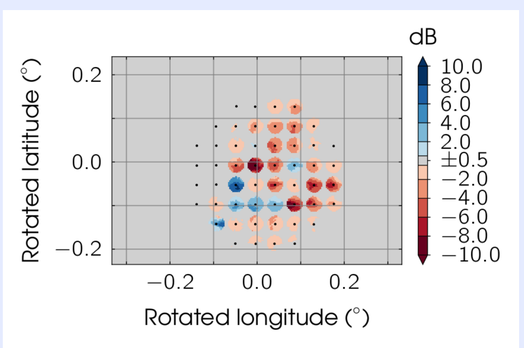

Figure 2: Analysis increments in a precipitation nowcasting scheme (APEX). Black dots indicate observations locations.

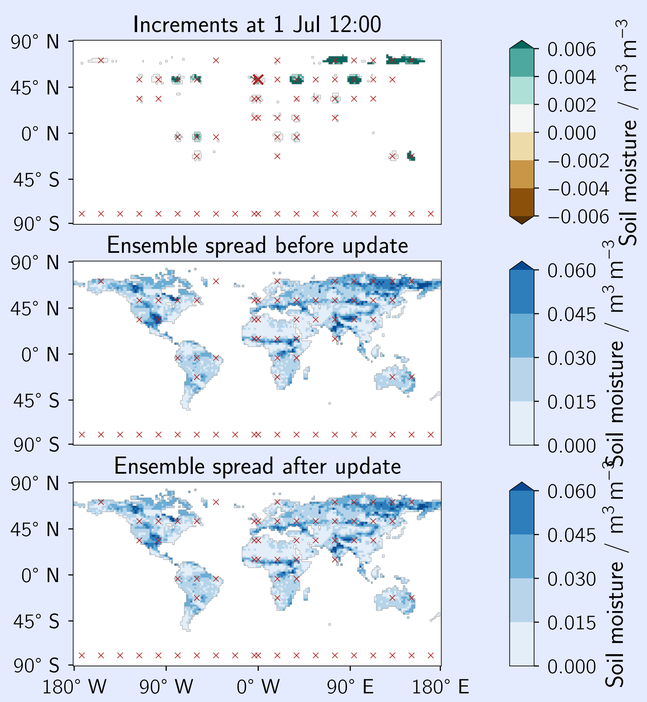

Figure 3: Analysis increments and pre-/post-analysis ensemble spread for soil moisture in a global dynamic vegetation model (JSBACH). Red crosses indicate observations locations.

Input and output

pyenda reads and writes netCDF files. The output of state variables used in the assimilation contains ensemble and time dimensions plus the dimensions from the locations (1D, 2D, or 3D).
pyenda reads observations directly from netCDF files without the need of pre-processing. Observation locations and times are specified via YAML-based configuration files (Fig. 7) and the respective data is read when required.
Observation errors can be read from the observation file, a seperate netCDF file, or the configuration files.

Coupling a model

pyenda uses an abstract base class `ForecastModel` to interface with models and for every model coupled to pyenda, a child of `ForecastModel` has to be implemented. This child class can

- use pure Python,
- use Cython and call C libraries,
- use f2py and call Fortran libraries,
- or use MPI and communicate with a separate executable.

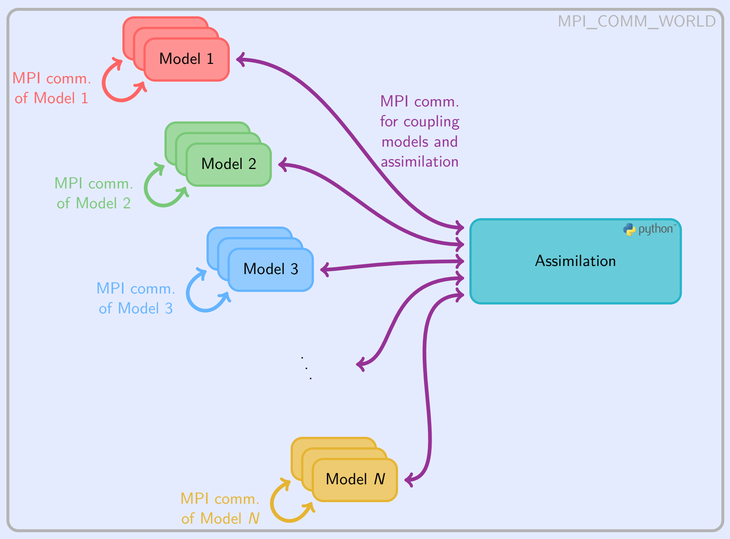

Figure 4: Multiple model instances run in parallel and communicate with pyenda via MPI.

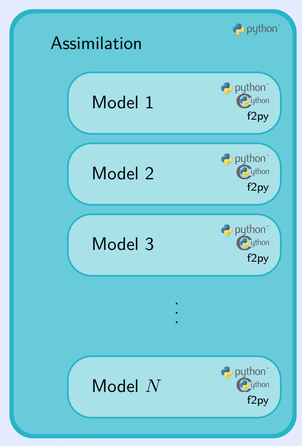

Figure 5: Multiple instances of the derived model class are generated and contained within the pyenda main script.

Object-oriented implementation

pyenda is completely object-oriented and makes use of abstract base classes to enforce that derived model and filter classes provide the necessary functionality. Inheritance is applied to re-use existing code and to simplify maintenance.

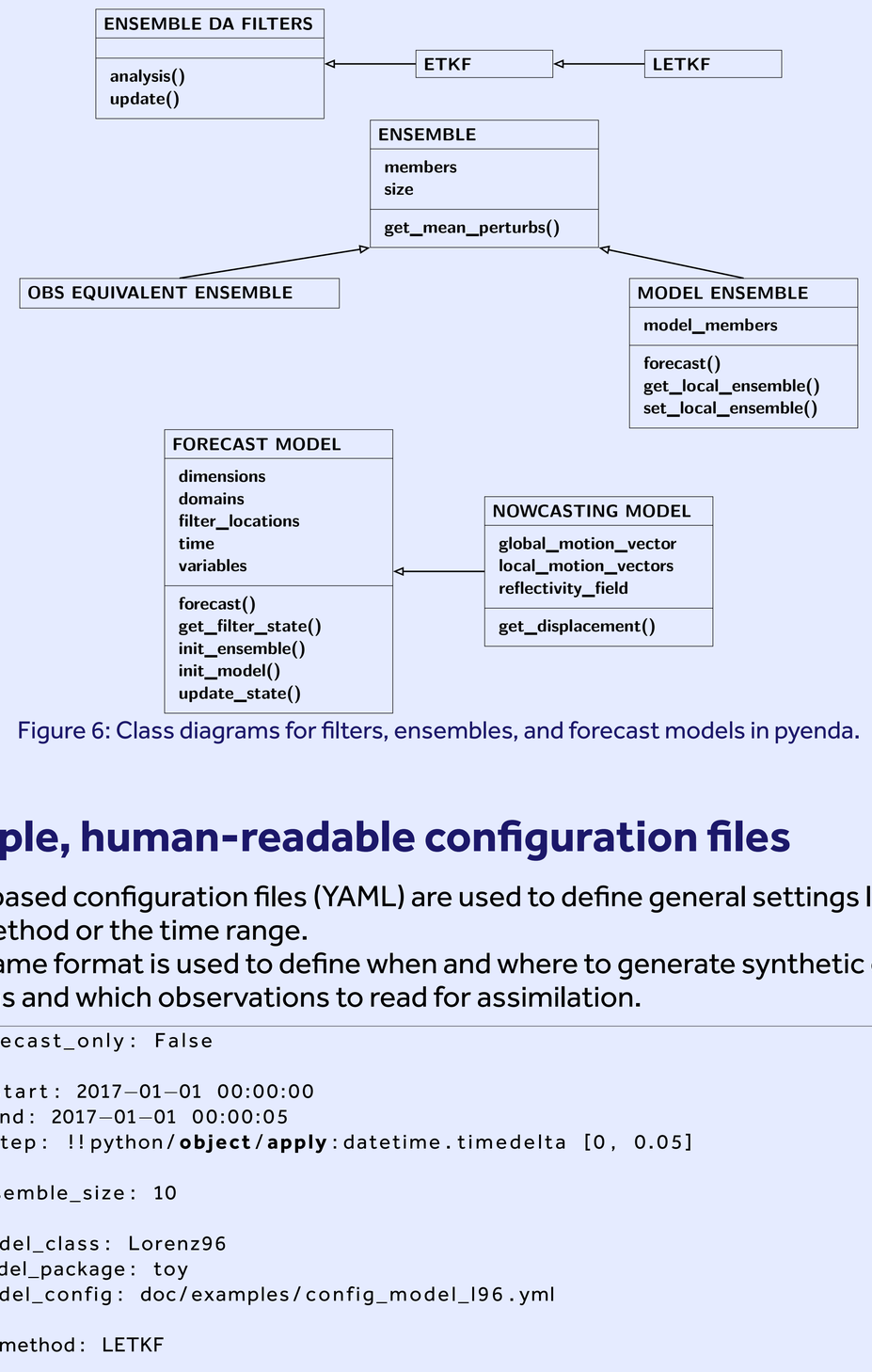

Figure 6: Class diagrams for filters, ensembles, and forecast models in pyenda.

Simple, human-readable configuration files

Text-based configuration files (YAML) are used to define general settings like the DA method or the time range.
The same format is used to define when and where to generate synthetic observations and which observations to read for assimilation.

```yaml
forecast_only: False

t_start: 2017-01-01 00:00:00
t_end: 2017-01-01 00:00:05
t_step: !!python/object/apply:datetime.timedelta [0, 0.05]

ensemble_size: 10

model_class: Lorenz96
model_package: toy
model_config: doc/examples/config_model_l96.yml

da_method: LETKF

observations: doc/examples/obslist_assimilate_l96.yml

write_obs_ens: True
da_output: doc/examples/l96_results.nc
aux_da_output:
```

Figure 7: YAML configuration file for assimilation with LETKF and Lorenz '96 model (inline documentation and comments have been removed).