PyOSSE: A python package for Observation System Simulation Experiments

Liang Feng and Paul Palmer

University of Edinburgh
June, 2013

I. Introduction

Developing a new space-based observation system represents a substantial financial investment. Observation System Simulation Experiments (**OSSE**) are a cost-effective numerical approach to realistically describe space-borne measurements and to evaluate their impact on current knowledge as part of preparing a science case for a particular space-borne mission. For example, in the included reference experiment we quantify the impact of atmospheric measurements of a trace gas on improving our current prior understanding of surface fluxes (emission minus uptake) of that gas. The example OSSE, relevant to atmospheric composition, includes methods to: 1) simulate the 4-D distribution of atmospheric constituents and sample it as it is observed by the space-borne sensor; and 2) estimate the magnitude and distribution of surface fluxes by fitting prior model surface fluxes, which describe physics, chemistry and biology underpinning the surface flux processes, to the pseudo observations using a data assimilation algorithm, accounting for model and observation errors. Despite some limitations of OSSEs (Lahoz et al., 2006), self-consistent numerical experiments have been useful to evaluate newly proposed space-based instruments. For example, the impact of data from the NASA Orbiting Carbon Observatory (**OCO**) was examined by several OSSE systems based on different transport models and different data assimilation techniques (e.g., Baker et al., 2006; Chevelliar et al, 2007; Feng et al., 2009).

It is time-consuming to develop a robust, realistic OSSE for any individual space-borne sensor, reflecting the challenges associated with combining an atmosphere transport model, a surface (process or empirical) model, and an instrument observation model with modern data assimilation techniques. Such a tool should also be flexible, considering all possible configurations of a proposed instrument and its potential applications, which can evolve during the design phase. We have developed the software package **PyOSSE** to help researchers to build their own OSSE systems with minimum overhead. PyOSSE consists of modules written in FORTRAN and Python, with emphasis on flexibility and simplicity, so that the user can easily understand, and apply these modules to their research. We assume that the user will have at least some rudimentary knowledge of Python and FORTRAN computer languages.

1.1 Overview of PyOSSE tool

Figure 1 shows these modules are organized around the two major tasks of one typical OSSE system:

1. Generating synthetic observations by sampling and screening model atmosphere.
2. Using an Ensemble Kalman Filter (EnKF) (Feng et al., 2009) approach to assimilate simulated observations to evaluate their impacts.

Left column of Figure 1 shows the main steps required to simulate satellite observations, including: 1) generating 4D-distributions of targeted atmospheric components by using a Chemistry Transport Model (**CTM**); 2) sampling model profiles along satellite tracks; 3) screening possible cloud and aerosol contaminations; and 4) convolving clear-sky profiles with scene-specific sensor averaging kernels, which reflect different instrument sensitivities at different vertical levels.

PyOSSE provides several classes as generic containers for observation specifications including satellite orbit (*orbit_m.py*) and instrument averaging kernels (*avk_m.py*) etc. These

classes read in multi-dimensional data tables by using file access modules (i.e., *orbit_file_m.py* and *avk_file_m.py*), which can be reconfigured or modified for different file formats. These classes also contain facilities to interpolate these pre-calculated tables to different observation scenes to reflect changes of instrument sensitivities. Besides the targeted atmospheric components, PyOSSE has also classes to represent other relevant atmospheric conditions that may affect observations from space. For example, classes *cloud_cl* (*cloud_m.py*), and *aod_cl* (*aod_m.py)* collect climatology on cloud and aerosol probabilities from different seasons. Randomly sampling these probability distribution functions help to realistically describe (and remove) cloudy and aerosol contaminated scenes. These modules can readily be exchanged with files provided by the user.

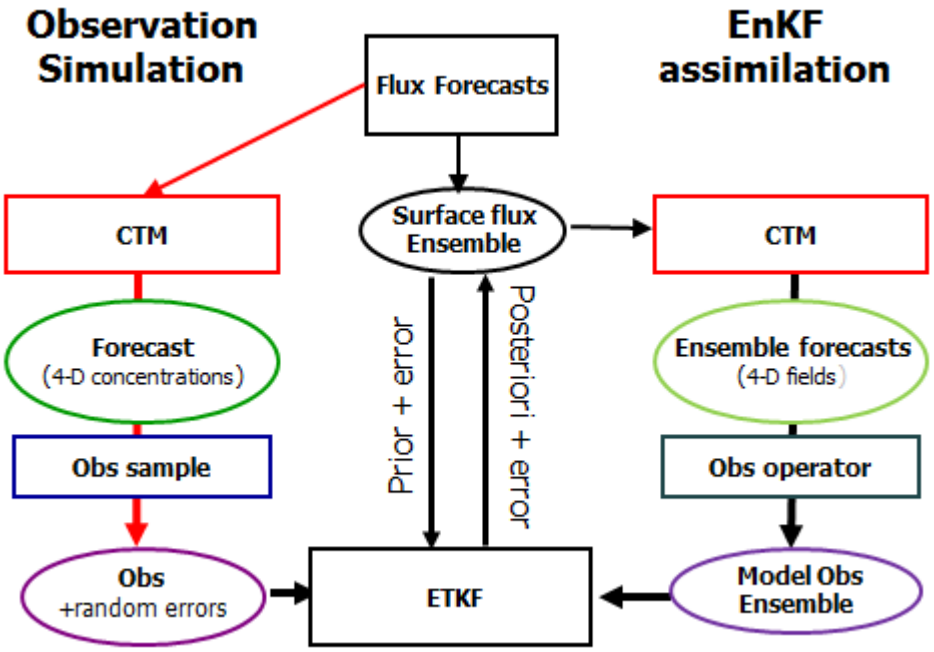

Figure 1: A schematic describing the PyOSSE Observation System Simulation Experiment infrastructure, which in this example uses output from the GEOS-Chem chemistry transport model.

Right column of Figure 1 demonstrates top-down flux estimation from simulated satellite observations by using an Ensemble Kalman Filter (EnKF) to: 1) generate an ensemble of flux perturbations to represent prior uncertainty; 2) use a CTM to project the resulting flux ensemble to an ensemble of model distributions of the targeted component; 3) use an observation operator to sample the resulting model concentrations, and convolve model profiles with instrument averaging kernels; 4) optimally determine posterior estimates by comparing model observation ensemble with simulated observations (as well as with their uncertainties).

1.2 EnKF approach

PyOSSE uses an EnKF approach (described by Feng et al, 2009) to estimate surface fluxes from simulated observations. The EnKF approach does not require any adjoint model specified for certain atmospheric transport model and observation operator so it can easily be applied to various experiments for top-down flux estimates. The EnKF approach also

provides a direct estimate of posterior error covariance, which is critical for properly quantifying impacts of a new instrument.

For this example, the state vector, describing the subject of interest, is estimated by the EnKF as scaling factor coefficients for a set of user-defined basis functions, representing a unique (temporal and) spatial pattern of surface fluxes at a given time period. By choosing basis function sets, the user can test the ability of the new instrument to determine the magnitude and distribution of fluxes. PyOSSE provides modules that help to define basis functions either by dividing continental-scale geographical regions (*divide_region.py*) or by decomposing error covariance for prior flux estimates to a set of major eigen-states (*svd_region_flux.py*).

Module *gen_ensemble_flux.py* uses sampling routines in *sample_reg_flux.py* to construct an ensemble for a full (*sample_by_reg*) or partial (*random_sample_reg)* representation of prior error covariance of the state vector (i.e., coefficients for the basis function set). It generates an ensemble of flux perturbations, which are subsequently used as surface boundary conditions for CTM simulations for projecting perturbation ensemble for model observations. The user can change the menus in configuration file, *flux_ensemble_def.cfg*, to read their own basis functions and sampling routines for flux perturbation ensembles.

The user is responsible for converting the netcdf file generated for the flux perturbation ensemble from *gen_ensemble_flux.py* to a format that is readable by their CTM. PyOSSE provides example code, *convert_netcdf_flux_bpch.py*, to convert the ensemble of perturbed fluxes to a binary format used by the GEOS-Chem CTM (see **wiki**.seas.harvard.edu/**geos-chem for more details**). PyOSSE also includes a prototype module called *ensemble_run_drive.py* to schedule individual ensemble model runs and to generate a text file *run_desc.dat (or user-chosen name)* for descriptions on emission inputs and model outputs etc, which will be used by *construct_state_vector.py* to set up state vector and configure the projection from coefficients to surface fluxes.

PyOSSE uses functions in *satellite_operator.py* to sample model profiles at the time and location of observed scenes and convolve them with the sensor scene-dependent averaging kernels that describe the vertical sensitivity of the sensor. Its default file IO functions can be re-configured or be replaced to access model fields from a user-defined CTM. PyOSSE then uses module *etkf_half.py* or *etkf_cor.py* as a numeric solver for the Ensemble Transform Kalman Filter (**ETKF**) algorithm (See Appenix A) to calculate the corresponding analysis increment and to estimate transform matrices by comparing model observation ensemble with (simulated) satellite observations and the associated error covariance. Module *etkf_half.py* is based on singular value decomposition (SVD), and is most efficient when no observation error correlations (due to correlated model transport errors etc) are taken into account (Feng et al., 2009). *etkf_cor.py* uses an LU-solver that can handle observation error correlations (Palmer et al., 2011).

1.3 OSSE example system

We have also provided a complete example OSSE system for an instrument based loosely on the specification of the NASA Orbiting Carbon Observatory that measured column-integrated dry-air CO_2 mole fraction $\mathbf{X_{CO2}}$. This example system can be extended or modified to describe other similar instruments. We have used the GEOS-Chem CTM as the default CTM to simulate the temporal and spatial distributions of atmospheric constituents,

but the user is able to replace it with their own CTM without extensive changes to the existing python modules that describe the simulated observations simulation or the data assimilation method. In our example OSSE, we first generate virtual observations by sampling a dummy atmosphere distribution with fixed vertical profiles (*gen_dummy_obs.py*). The user can insert their own instrument specification, and observation screening data and algorithm for cloud and aerosol contaminations by changing definitions in menu file *vob_def.cfg*. The virtual observation files provide information which is not directly linked to component distributions simulated by CTM, such as time and locations of clear scenes and sensor averaging kernel and uncertainties for these clear scenes. By default, *gen_sat_obs.py* converts dummy observations to model observation by convolving model CO_2 profiles sampled from actual GEOS-Chem model outputs to dry-air columns. This step can be readily reconfigured to use outputs from other CTMs by changing menu defined in *sob_def.cfg.*

To digest simulated OCO observations, we have defined 144 basis functions for monthly fluxes over 144 global regions using *split_t3_region.py*, and then constructed a full representation of prior error covariance using *gen_ensemble_flux.py*. We are using a sequential approach to assimilate observations, and have introduced a lag window of 5 months to reduce computational costs (Appendix A). The resulting flux perturbations are converted from netcdf format to a binary format using *convert_netcdf_flux_bpch.py* to drive tagged GEOS-Chem simulations. The EnKF algorithm is then used to update flux estimates by digesting simulated OCO observations step by step.

PyOSSE is developed by the University of Edinburgh as part of the ESA Data Assimilation Project (www.esa-da.org), contract no ESRIN/RFQ/3-13354/11/I-LG. It is free for academic research. This document describes structure of the package. Detailed interfaces of individual modules can be found in its documentation sub-directory *otool/ESA/doc/* or from xweb.geos.ed.ac.uk/~lfeng/reference.

Section II describes procedures for installing and testing PyOSSE. Section III is an overview of the structure and concepts of the package. It also discusses the most important modules and data files in each subdirectory. Appendix A provides a detailed description of the EnKF algorithm. Appendix B details the complete OSSE system for OCO-like instruments.

II. Installation

2.1 Download

The package is freely available from website xweb.geos.ed.ac.uk/~lfeng as a tar.gz archive file.

2.2 Prerequisites

This package is developed for a Linux operation system. As it involves running a 3-D chemistry transport model with multiple tracers, we assume that the user has access to a workstation with sufficient internal memory (>4 Gb) and free disk space (>300 Gb).

Many modules in this package have been built over other freely distributed packages and libraries. Before installing this package, following software packages are needed:

- Python interpreter (>version 2.4), freely available at www.python.org

- Numeric extensions to the Python (numpy), freely available at: http://sourceforge.net/projects/numpy.
- Scientific Python, available at: http://dirac.cnrsorleans.fr/plone/software/scientificpython
- Python plotting library, available at http://matplotlib.sourceforge.net/
- Intel fortran compiler and Intel kernel math library, which academic users can freely downloaded via http://software.intel.com/en-us/articles/intel-education-offerings/.
- NetCDF library, freely available at: http://www.unidata.ucar.edu
- HDF library, freely available at: http://www.hdfgroup.org.

To avoid installing the individual python packages, we recommend to download and install the pre-compiled Enthought python installation of EPD 7.0 (or higher) from www.enthough.com. This Python distribution is free for academic users, and has already included the common python extensions needed by PyOSSE.

PyOSSE contains a complete example OSSE system for OCO-like instruments. The example experiment is based on the GEOS-Chem community CTM (v8.02) for which more information is available at **wiki**.seas.harvard.edu/**geos-chem.** For PyOSSE, we have provided GEOS-Chem outputs to demonstrate the example. If you want to use GEOS-Chem directly please consult the wiki for further details on how to download and install the model. If you choose to download the model the relevant modifications are listed in *otool/ESA/example/enkf_oco/geos_chem*.

2.3 Unpacking and compiling

2.3.1 Unpacking

After downloading the file archive otool.tar.gz into your workstation, you can extract the files by using Linux command:

tar -zxvf otool.tar.gz

There will be 8 sub-directories built under otool/ESA/:

- atmosphere
- doc
- enkf
- example
- instrument
- observation
- surface
- util

The contents and purpose of these sub-directories are explained in Section III.

In order to let the user easily read, apply or even change existing modules, we have not developed PyOSSE with any particular version of Python. The user is expected to change python searching path to load this package manually and then build the required shared libraries, following instructions in 2.3.2 and 2.3.3

2.3.2 Setting python searching path

You can add PyOSSE to python searching path by inserting the following line into your .bashrc file

export PYTHONPATH=$PYTHONPATH:/*your_PyOSSE_directory*/

For some systems, it is also necessary to include runtime libraries for python installation into LD_RUN_PATH by adding:

export LD_RUN_PATH=$LD_RUN_PATH:/*your_python_root/lib*

2.3.3 Build shared libraries

First, go to subdirectory *otool/ESA/util*, and change one shell script called '*f2py_ifort*', by inserting the locations of your python installation and FORTRAN compilers:

F90=<your_f90_compiler>
FPY=<f2py of your python installation>
LIBPATH=< path for fortran lib>
LIBS=<your fortran libraries>

Second, build the FORTRAN shared libraries by running the shell script '*build_share_lib.sh*'. If compilations are successful, there will be 11 shared libraries under sub-directory *PyOSSE/ESA/util/*:

bpch2_rw_py.so
flux_regrid.so
flib.so
interpolate_f.so
sigma_pres_mod.so
process_nf_array.so
sampl_model_field.so
vertical_column.so
vertical_profile.so
read_data.so
great_circle_distance.so

Contents of these libraries can be found in Section 3 for important modules. You can test these libraries by running python scripts *test_interpolate.py* and *test_sigma_pres.py.*

2.4 Generate HTML files for module interfaces

Use *build_doc.sh* in otool/ESA/ to generate one HTML file for each module that describes its contents and usage. These html files are stored in otool/ESA/doc for future reference.

2.5 Data set

You can download *otool_data.tar.gz* from (www.esa-da.org/data/otool_data.tar.gz). They are the default data inputs, mainly for supporting complete OSSE examples. After unzipping the data archive, you have 7 sub-directories under *otool_data* :

1. clim_data: climatology for CO_2, CH_4 etc profiles
2. ecmwf: ecmwf cloud analysis.
3. enkf_output: GEOS-Chem outputs from tagged simulations
4. enkf_rerun: GEOS-Chem outputs from single tracer run.
5. LAI: surface vegetation type etc.
6. oco: satellite orbit and data tables for OCO X_{CO2} averaging kernel and error.
7. gc_std: GEOS-Chem outputs from single tracer run forced by standard (climatological) emission inventories.

You can now test these data sets, and first generate virtual ('dummy') observations for OCO satellite, where atmospheric CO_2 concentrations are fixed to a CO_2 climatology profile, and then generate satellite observations by sampling GEOS-Chem outputs by:

1) changing DATAPATH in *vob_def.cfg* under otool/ESA/example/obs_simulation/
2) running *gen_dummy_obs.py* there.
3) changing *sob_def.cfg* and running *gen_sat_obs.py*

If you want to run GEOS-Chem simulation by yourself, please consult the wiki for further details on how to download and install the model **wiki**.seas.harvard.edu/**geos-chem,** and download GEOS-5 meteorological data at a horizontal resolution of 4º (latitude)×5º (longitude) for 2009. Once the model infrastructure is established, following the guidelines associated with GEOS-Chem, you can run the script '*rungeos.sh*' in /ESA/example/enkf_oco/std_run for a control calculation. As a result, a GEOS-Chem forward simulation will be launched to produce daily 3-D CO2 fields, which can be sampled to simulate satellite observations as discussed above.

III. Directory and module

3.0 Structure

Figure 2 shows the modules and shared libraries that are stored in 8 sub-directories. Table 1 summarises the file types and purposes of these sub-directories. Further more details for each sub-directory are discussed in this chapter. If you want to use any module in your own python codes, you can import it by using sentences like:

import ESA.<SUBDIR>.<MODULE>,

where <MODULE> is the name of the module you want, and <SUBDIR> is its location.

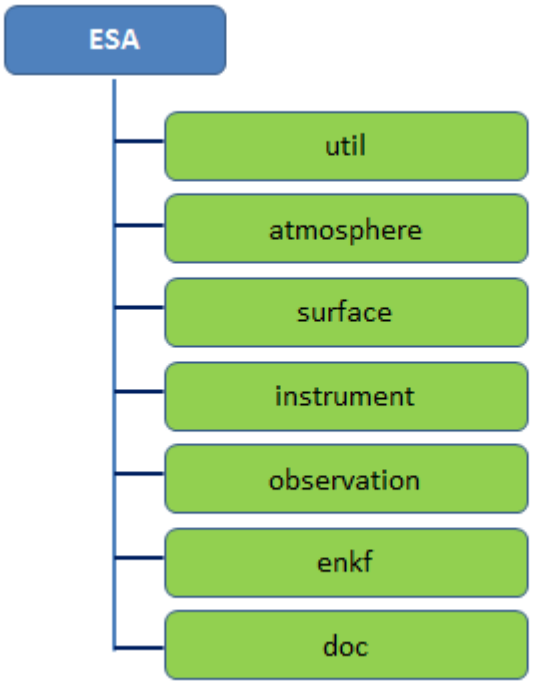

Figure 2: Directory tree of PyOSSE package.

Directory	File types	Contents
ESA (main)	*.py *.sh	1) Shell files for installing and testing the package. 2) Python files for package initialization.
util	*.py *.so *.f90	Python modules, FORTRAN modules and shared libraries for: 1. object managing 2. file IO accessing 3. data processing (gridding or interpolation)
enkf	*.py	Classes and modules for 1. constructing / maintaining state vector 2. solving ETKF equations for posterior increment and transform matrices by assimilating observations.
example/enkf_oco	*.py *.sh *.cfg *.bpch	EnKF data assimilation system based on GEOS-Chem CTM outputs
instrument	*.py	Classes and modules to 1) 'model' satellite instruments (orbit; averaging kernel etc). 2) collect climatology or model outputs. 3) sample cloud and aod PDF. 4) screen observation scenes.
atmosphere	*.py	Classes to 1. access and store 3D or 2D model outputs. 2. sample model outputs at observation locations. 3. interpolate/integrate model profiles.
example/obs_simulation	*.py *.cfg	Modules to 1) generate virtual observations for OCO-like satellite by sampling fixed atmosphere CO_2 profile along satellite orbits. 2) convert virtual observations to model observations by sampling GEOS-Chem model outputs. .
surface	*.py *.nc *.cfg *.dat	Modules to 1. set up basis function sets; 2. sample basis functions to construct flux perturbation ensemble.

Table 1: contents in package directories

3.1 Classes

PyOSSE uses classes to represent the objects included by a typical OSSE system. They are containers for both the data and functions of these objects. Table 2 lists the major classes defined in PyOSSE. These classes can be used as parents to derive users' classes. Also, their member functions can be changed through inputs during run-time or be overridden within derived classes. In particular, access to pre-defined data or model outputs are through so-called configurable file IO modules (see 3.2)

Location	Module	Class	Object
surface	bf_m.py	bf_cl	Basis (or ensemble) functions for surface fluxes
instrument	landcover_m.py	landcover_cl	Land cover (surface type)
instrument	avk_m.py	avk_cl	Predefined averaging kernel
instrument	err_m.py	err_cl	Predefined Observation error
instrument	cloud_m.py	cloud_cl	Cloud PDF
Instrument	aod_m.py	aod_cl	AOD PDF
instrument	orbit_m.py	orbit_cl	Predefined satellite orbit
atmosphere	ctm_grid_m.py	ctm_grid_cl	Generic grid used in CTM
atmosphere	ctm_grid_2d.py	ctm_grid_2d	Grid for 2-D data
atmosphere	ctm_grid_3d.py	ctm_grid_3d	Grid for 3-D data
atmosphere	ctm_slice_m.py	ctm_slice_cl	Slice of atmosphere
atmosphere	ctm_profile_m.py	ctm_profile_cl	Vertical profiles
atmosphere	ctm_world_m.py	ctm_world_cl	Collection of CTM fields.
observation	satellite_obs.py	satellite_obs_cl	Satellite observation
observation	satellite_operator.py	satellite_xgp_cl	Model satellite dry-air column observation
observation	sobs_def_m.py	sobs_def_cl	Configuration for observation simulation
enkf	state_vector.py	stv_cl	State vector used in EnKF.
enkf	etkf_half.py	etkf_cl	SVD solver for ETKF equations
enkf	etkf_cor.py	etkf_cor_cl	LU solver for ETKF equations
enkf	run_desc_m.py	run_desc_cl	Ensemble run configuration
enkf	assim_def_m.py	assim_def_cl	Data assimilation configuration
enkf	x2flux_m.py	x2flux_cl	Projections from

			coefficients of basis functions to surface fluxes.
util	otool_menu_m.py	menu_cl	Menu (i,e., configurations)
util	gp_axis_m.py	gp_axis_cl	Axis
util	otool_ncfile_io.py	ncfile_desc_cl	NetCDF file access
util	otool_txtfile_io.py	file_desc_cl	Text file access
util	otool_gcfile_io.py	diag_info_cl	GEOS-Chem diaginfo
util	otool_gcfile_io.py	tracer_info_cl	GEOS-Chem tracer info
util	otool_gcfile_io.py	gcfile_desc_cl	GEOS-Chem BPCH file access
util	horizontal_interp_m.py	hinterp_cl	Horizontal interpolation
util	vertical_interp_m.py	vintpl_cl	Vertical interpolation
util	gp_grid_m.py	gp_grid_cl	Generic Grid
util	otool_grdfile_io.py	grdfile_desc_cl	File access for gridded text table
util	gp_data_m.py	gp_data_cl	Gridded data

Table 2: major classes defined in PyOSSE

3.2 Configurable IO modules

OSSE systems rely on pre-defined data on instrument configurations, atmospheric and surface condition, etc. They also need to include a method of transferring data to and from a CTM. These data can be given in different format with different variable names. To simplify the main PyOSSE classes used to manipulate data, we have developed configurable IO modules. Typically, one IO module (for example *avk_file_m*.py) defines functions to open a file and read data to a dictionary or table (recarray) member of a file access class called *fdesc*. Its host (such as class *avk_cl* in *avk_m.py*) use *fdesc* to perform basic disk file IO as well as other data retrieval functions such as sampling and interpolation etc.

In a typical IO module, we define its file structure and one default *varname_lst* to specify the names of variables to be invoked by the host class. We have also defined a default dictionary *varname_dict* which translates variable names used in the host class to the names saved in the disk file. The default file structure, *varname_dict* and to some extent *varname_lst* can be changed in the run time by inputs so that the host class can access similar data files by adjusting their inputs. When it is necessary, the user can develop a similar IO module to replace the existing one to provide file IO services required by the host classes. Currently we have defined 8 such IO modules in PyOSSE:

Location	Module	Host class	Data format
surface	bf_file_m.py	bf_cl	netcdf
util	gc_ts_file_m.py	satellite_xgp_cl	bpch2
instrument	landcover_file_m.py	landcover_cl	netcdf
instrument	avk_file_m.py	avk_cl	Gridded text table

instrument	err_file_m.py	err_cl	Gridded text table
instrument	cloud_file_m.py	cloud_cl	Gridded text table
Instrument	aod_file_m.py	aod_cl	Gridded text table
instrument	orbit_file_m.py	orbit_cl	Text table
enkf	run_desc_file_m.py	run_desc_cl	Text table

Table 3: configurable IO modules defined in PyOSSE.

3.3 Configuration files

PyOSSE includes two types of text files for the user to control processes. The first type is the description file for information on process output (for example, the size of flux ensemble generated by *gen_ensemble_flux.py*), which can be used by other module as inputs. The second type is the menu file. A typical menu file contains several menus. For example, in *vob_def.cfg*, we have 6 menus, one of which is called 'orbit'.

```
#MENU (orbit)
#=========================================
name|orbit
path|$DATAPATH$/oco/aqua_0.25x0.25/
flnm|XVIEWTYPEX_num_XVIEWMODEX_XDOYX.dat
dict|__load:$MDPATH$.orbit_file_m:orb_varname_dict
fopen|__load:$MDPATH$.orbit_file_m:open_orbit_file
fread|__load:$MDPATH$.orbit_file_m:read_orbit_file
fclose|__load:$MDPATH$.orbit_file_m:close_orbit_file
fget|__load:$MDPATH$.orbit_file_m:get_orbit_data
keywords|__dict:Nil
fclass|__load:$MDPATH$.orbit_m:orbit_cl
#MEND
```

The menu 'orbit' not only provides parameters for orbit file name and locations etc, but also tells *gen_dummy_obs.py* to use class *orbit_cl* defined by *orbit_m.py* as host container and functions defined in module *orbit_file_m*.py as IO functions to access these orbit files.

Currently we have defined 5 menu file prototypes in PyOSSE.

Location	Name	Main module	Comment
surface	flux_ensemble_def.cfg	gen_ensemble_flux.py	Generate ensemble fluxes
enKF	assim_def.cfg	assim_daily_obs.py	Assimilate daily observations
example/obs_simulation	vob_def.cfg	gen_dummy_obs.py	Generate virtual observations
example/obs_simulation	sob_def.cfg	gen_sat_obs.py	Generate model observations

example/enkf_oco	geos_chem_def.cfg	run_job.py	Assimilate OCO observation

Table 4: Menu files in PyOSSE.

3.4 Subdirectory

3.4.1 util

Modules and shared libraries in *util* are used as library by other PyOSSE module. They can be categorized as 6 groups:

1. Object (class) management
   - message_m.py
   - ot_constant.py
   - otool_obj.py

2. File access
   - line_process_m.py
   - otool_gcfile_io.py
   - otool_grdfile_io.py
   - otool_ncfile_io.py
   - otool_txtfile_io.py
   - otool_var_io.py
   - bpch2_rw_smp.py
   - gc_ts_file_m.py
   - bpch2_rw_py.so
   - read_data.so

3. Grid and data container
   - gp_axis_m.py
   - gp_data_m.py
   - gp_grid_m.py
   - flux_regrid.so
   - pres_m.py
   - sigma_pres_mod.so

4. Data processing
   - horizontal_interp_m.py
   - vertical_interp_m.py
   - sample_model_field.so
   - flib.so
   - interpolate_f.so
   - process_nf_array.so
   - vertical_column.so
   - vertical_profile.so

5. Menu and configuration file

- otool_menu_m.py
- otool_descfile_io.py

6. Time and visualization
   - gen_plots.py
   - time_module.py

3.4.2 atmosphere

Modules in this directory include containers for 2D/3D/4D model outputs. For example *ctm_slice_m.py* defines a class *ctm_slice_cl.py* which can be used to store model profile ensemble sampled at observation locations. Except for GEOS-Chem (*otool_gcdesc_m.py*), we have not explicitly defined IO modules for these container. The user is responsible for developing IO modules for their CTM outputs, which are usually easy to be implemented.

3.4.3 surface

Modules in this directory are mainly designed to: 1) define basis function set (*split_t3_region.py* or *svd_t3_region.py*); and 2) generate ensemble fluxes (*gen_ensemble_flux.py*). The resulting flux perturbation ensembles will be used to force CTM ensemble simulations, and construct state vector for the following data assimilation.

3.4.4 instrument

Classes defined in this directory by modules *such as orbit_m.py, avk_m.py, cloud_m.py, aod_m.py, landcover_m.py* represent objects needed for observation simulations, such as satellite orbit, averaging kernel, cloud coverage, aod climatology, and land cover etc. Their accesses to pre-defined data sets are through its own IO modules such as *orbit_file_m.py, avk_file_m.py, cloud_file_m.py, aod_file_m.py, landcover_file_m.py*

3.4.5 observation

Modules in this subdirectory are used to: 1) read satellite observations (*satellite_obs.py*); and 2) sample model profiles and convolve them into dry air columns (*satellite_operator.py* and *xgp_jacobian_m.py*)

3.4.6 enkf

Modules in this subdirectory are used to: 1) construct and manage state vector (*construct_state_vector.py* and *state_vector.py*); 2) assimilate observations using ETKF algorithm (*assim_daily_obs.py*, *etkf_half.py* and *etkf_cor.py*)

IV. Appendix

A. Ensemble Kalman Filter and numerical implementation

In the PyOSSE package, we use an Ensemble Kalman Filter (EnKF) to estimate surface fluxes by fitting a model to observations of atmospheric constituents. Generally, we assume the surface fluxes *F*(x, y, t) is the form of:

$$F(x,y,t) = F_0(x,y,t) + \sum_i^N \lambda_i \, BF_i(x,y,t), \quad \text{(A1)}$$

where $F_0(x,y,t)$ is flux estimates from either process-based model or simply climatology, and $BF_i(x,y,t)$ is the pulse-like flux perturbations, and λ_i is the coefficients to be estimated. So, the state vector **x** is chosen to be

$$\mathbf{x} = [\lambda_1, \lambda_2, \dots, \lambda_N], \quad \text{(A2)}$$

As an ensemble approach, we represent uncertainties of the estimates P by using an ensemble of perturbations

$$\Delta\mathbf{X} = [\delta\mathbf{x}_1, \delta\mathbf{x}_2, \dots, \delta\mathbf{x}_L], \text{ (A3)}$$

so that

$$\mathrm{P} = \Delta\mathbf{X}(\Delta\mathbf{X})^{\boldsymbol{T}}. \text{ (A4)}$$

In the above equations, *L* is the size of the ensemble, which can be smaller than the size of state vector as an approximation of prior error covariance. Using a proper partial representation of the error covariance, with an ensemble size *L* chosen to be much smaller than the size of state vector *N*, can significantly reduce computational costs (See for example, Houtekamer et al., 1998; Peters et al., 2005). At the meantime, as an approximation, partial representation may also introduce artificial correlations between control variables. To suppress the effects of those artificial correlations on posterior flux estimates, localization of impacts from assimilated observations may be necessary. We have included a function called *set_localization_wgt* in *x2flux_m.py* to limit the spatial range of the flux analysis increments from assimilated satellite observations. However, the user should quantify/estimate appropriate parameters or even develop new approaches for localization by experiments according to their assimilation setup and observation characteristics etc.

After assimilating observation y_{obs} with error covariance **R,** the posterior state vector $\mathbf{x}^{\mathrm{a}}$ is calculated by updating a~prior $\mathbf{x}^{\mathrm{f}}$ via Kalman gain matrx **K**:

$$\mathbf{x}^a = \mathbf{x}^f + \mathbf{K}[y_{obs} - H(\mathbf{x}^f)], \text{ (A5)}$$

where *H* is the system observation operator, which projects the state vector (i.e., coefficients of the basis function set) to observation space (for example, the dry-air column X_{CO2} measured by OCO-like instrument). It includes transporting atmospheric components, sampling model distributions at observation locations, and convolving model profiles using averaging kernels. The gain matrix **K** is calculated by

$$\mathbf{K} = \Delta\mathbf{X}^f \, (\Delta\mathbf{Y})^T)[(\Delta\mathbf{Y}) \quad (\Delta\mathbf{Y})^T + \mathbf{R}]^{-1}, \text{ (A6)}$$

where

$$\Delta\mathbf{Y} = H(\mathbf{x}^f + \Delta\mathbf{X}^f) - H(\mathbf{x}^f), \text{ (A7)}$$

By using the Ensemble Transform Kalman Filter (ETKF) algorithm (see for example, Livings et al., 2008), the posterior uncertainty is determined by

$$\mathbf{P}^{a} = \Delta\mathbf{X}^{f}\mathbf{T}\,(\Delta\mathbf{X}^{f}\mathbf{T})^{T}, \quad \text{(A8)}$$

and the transform matrix $\mathbf{T}$ is given by

$$\mathbf{T}(\mathbf{T})^{T} = \mathbf{I} - \Delta\mathbf{Y}^{T}[\Delta\mathbf{Y}\ \ \Delta\mathbf{Y}^{T} + \mathbf{R}]^{-1}\Delta\mathbf{Y}\ \ , \quad \text{(A9)}$$

Module *etkf_half.py* and *etkf_cor.py* calculates gain matrix **K** and transform matrix T by numerically solving Eqs.8 and 9 using SVD technique (Feng et al., 2009) and LU solver for sparse matrices (Palmer et al., 2011)

In the implementation, we have used a sequential data assimilation technique to digest observations step by step. Also, we have introduced a lag window to reduce computational costs. Figure A1 schematically shows the concepts of lag window for sequential data assimilation. For example, observations in step T7 have no contributions from fluxes after T7. On the other hand, estimates for fluxes long before step T7 (such as those over T1 and T2) have already been refined by observations in steps from T1 till T7, and hence further assimilation of T7 observations will not be expected to change them significantly. As a result we only consider estimates for fluxes inside the lag window as control variables in order to assimilate observations in step T7.

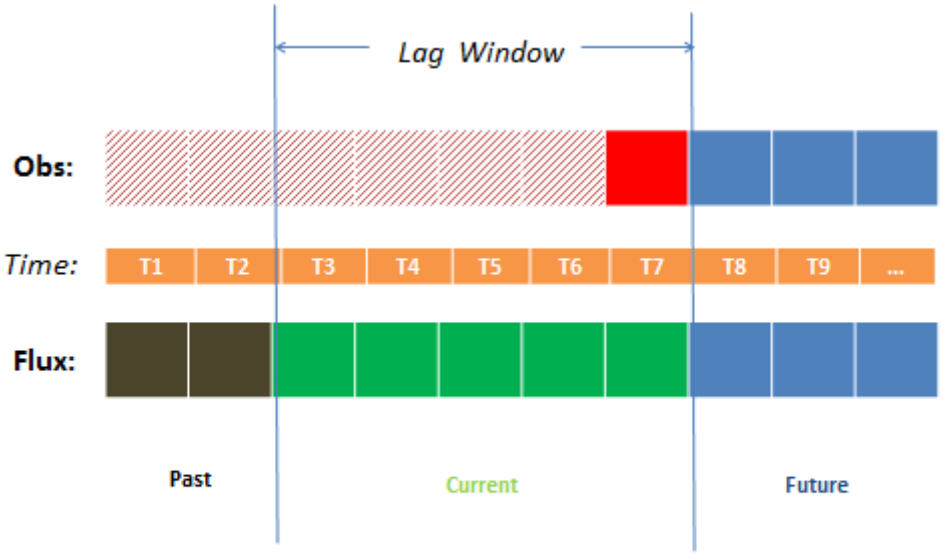

Figure A1. Schematic plot for lag window used in sequential data assimilation.

B. OSSE system for OCO-like instrument

Modules in two sub-directories */obs_simulation/* and */enkf_oco/* under *otool/ESA/example/* form a complete OSSE system based on GEOS-Chem CTM to study OCO-like observations.

B.1 Observation simulations

In *otool/ESA/example/obs_simulation/* Observation are simulated in two phases:

I. generating virtual observations by sampling dummy atmosphere (*gen_dummy_obs.py* and *vob_def.cfg*)
II. converting virtual observations to model observations by sampling GEOS-Chem outputs (*gen_sat_obs.py and sob_def.cfg*)

In phase I, *gen_vob.py* 1) reads in satellite orbits; 2) screens cloud and aerosol contaminations; 3) checks surface type; 4) gets scene-dependent averaging kernel and observation error; and 5) convolves dummy CO_2 profiles using averaging kernel to dry air columns;

In phase II, *gen_sob.py* 1) reads in virtual observations; 2) sample GEOS-Chem profiles at observation locations; 3) convolves model CO_2 profiles using averaging kernel to dry air columns; and 4) generating random errors.

Besides these two python modules, observation simulations also need pre-calculated data for:

- Satellite orbit
- Satellite Averaging Kernel
- Cloud PDF
- Aerosol PDF
- GEOS-Chem outputs

If you want to run GEOS-Chem yourself, you also need

- GEOS-5 meteorological analysis

The user can use *plot_obs_map.py* to show simulated observations, and compare them with reference results stored in *./ref_sat_obs.* Because of the random sampling etc, the results simulated by the user could be slightly different from the reference ones.

B2. EnKF data assimilation

The data flow for estimating surface fluxes using OCO-like XCO2 concentration observations is summarised as

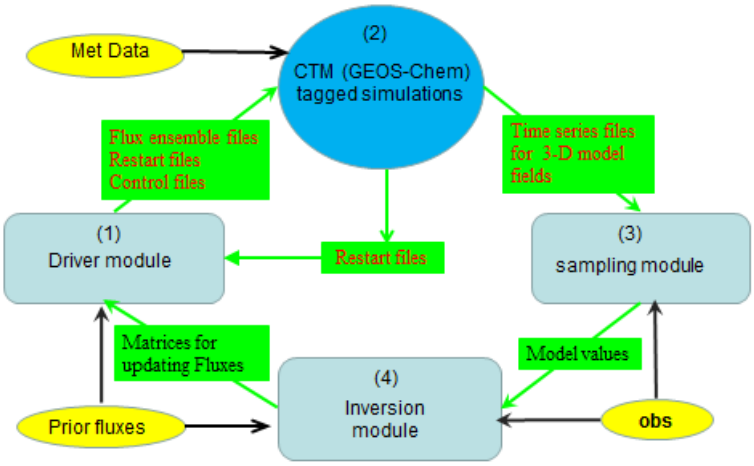

Figure 1B: Schematic plot for data flow in one EnKF data assimilation step.

However in this OSSE example, we have used a full representation for prior error covariance of monthly regional flux estimates over 144 regions. We define ensemble runs that use a pre-defined flux perturbation ensemble, and then rescale to actual prior uncertainties during the data assimilation (Feng et al., 2009) (*oco_assim_step.py)*, which simplifies the data assimilation procedure shown above.

We have saved outputs from such an ensemble run into otool_data/enkf_output. The user can generate their own forward simulations by using *ensemble_run_drive.py in ESA/example/enkf_oco/ensemble_run* to manage the jobs . Also, *bpch* files for flux perturbation ensembles by *gen_flux_ensemble.py* and *convert_flux_netcdf_bpch.py* are available in *otool_data/surface_flux.* These fluxes can be used by making a symbol link to your subdirectory surface_flux under the run directory *ESA/example/enkf_oco/ensemble_run* .

Once the ensemble run outputs are available, the user can use *run_job.py* as driver to 1) create state vector by reading in prior regional fluxes and flux perturbation ensemble; 2) sample CO_2 profiles for the prior and each of ensemble member at time and locations of simulated OCO observations; 3) compare model X_{CO2} and the perturbation ensemble with observations to generate optimal estimates of fluxes (i.e, coefficients) .

These tasks are mainly done by class members contained in *class oco_assim_step_cl*. For example member object *cl_obs* is a class for accessing (simulated) observations, and member object *cl_fc_prof* and *cl_enr_prof* for model values. Also, member *cl_ctm* provides controls

on forecast runs. Also the functions of these class members are specified by the configuration file *osse_def.cfg*, so that the user can exchange them with their own functions.

Finally, the user can compare the prior and posterior flux estimates (or model dry-air CO_2 columns) with the 'true' values by using *plot_flux.py* (or *plot_obs_cmp_ts.py* and *plot_obs_map.py* for observations). Also, the user can compare their results with the reference experiment stored in *./ref_oco_inv*

V. References

Baker, D. F., Doney, S. C., and Schimel, D. S.: Variational data assimilation for atmospheric CO2, Tellus Ser. B, 58, 359–365, 2006.

Chevallier, F., Br´eon, F.-M., and Rayner, P. J.: Contribution of the Orbiting Carbon Observatory to the estimation of CO2 sources and sinks: Theoretical study in a variational data assimilation framework, J. Geophys. Res., 112, D09307, doi:10.1029/2006JD007375, 2007a.

Cooperative Atmospheric Data Project-Carbon Dioxide, CD-ROM, NOAA GMD, Boulder, Colorado (also available via anonymous FTP to ftp.cmdl.noaa.gov, Path: ccg/co2/GLOBALVIEW).

Feng, L., Palmer, P. I., B¨osch, H., and Dance, S.: Estimating surface CO2 fluxes from space-borne CO2 dry air mole fraction observations using an ensemble Kalman Filter, Atmos. Chem. Phys., 9, 2619–2633, doi:10.5194/acp-9-2619-2009, 2009.

Houtekamer, P. L. and Mitchell, H. L.: Data assimilation using an ensemble Kalman filter technique, Mon.Weather Rev., 126, 796–811, 1998.

Lahoz, W. A., Brugge, R., Jackson, D. R., Migliorini, S., Swinbank, R., Lary, D., and Lee, A.: An observing system simulation experiment to evaluate the scientific merit of wind and ozone measurements from the future SWIFT instrument, Q. J. Roy. Meteor. Soc., 131, 503–523, doi:10.1256/qj.03.109, 2005.

Livings, D. M., Dance, S. L., and Nichols, N. K.: Unbiased ensemble square root filters, Physica D., 237/8, 1021–1028, 2008.

Palmer, P. I., Suntharalingam, P., Jones, D. B. A., Jacob, D. J., Streets, D. G., Fu, Q., Vay, S. A., and Sachse, G. W.: Using CO2:CO correlations to improve inverse analyses of carbon fluxes, J. Geophys. Res., 111, D12318, doi:10.1029/2005JD006697, 2006.

Palmer, P. I., L. Feng, and H. Boesch, "Spatial resolution of tropical terrestrial CO2 fluxes inferred using space-borne CO2 sampled in different Earth orbits: the role of spatial error correlations," Atmos. Meas. Tech., 4, 1995-2006, 2011.

Peters, W., Miller, J. B., Whitaker, J., Denning, A. S., Hirsch, A., Krol, M. C., Zupanski, D., Bruhwiler, L., and Tans, P. P.: An ensemble data assimilation system to estimate CO2 surface fluxes from atmospheric trace gas observations, J. Geophys. Res., 110, D24304, doi:10.1029/2005JD006157, 2005.