Data assimilation

Polyphemus Training Session

May 29, 2008

About

Purpose: introduction to optimal interpolation, Kalman filtering and variational data assimilation with Polair3D; application to photochemistry

Authors: Vivien Mallet, `Vivien.Mallet@cerea.enpc.fr`; Lin Wu, `Lin.Wu@inria.fr`

Polyphemus version: 1.3

Location: `http://www.enpc.fr/cerea/polyphemus/sessions.html`

Contents

Preparing the Session

Extracting the archive creates a directory `assimilation/` in `polyphemus-sessions/`, which has also been created if necessary. Place the source `Polyphemus-1.3` in `polyphemus-sessions/` also.

Introduction

In Polyphemus, data assimilation algorithms are implemented as drivers independent from the models. Actually each driver expects a few methods in the model interface. In the base driver, used for forward simulations without assimilation, the main method (`BaseDriver::Run`, see `driver/BaseDriver.cxx`) reads

```
/*** Initializations ***/

Model.Init();
OutputSaver.Init(Model);

/*** Time loop ***/

for (int i = 0; i < Model.GetNt(); i++)
  {
    if (option_display["show_iterations"])
      cout << "Performing iteration #" << i << endl;

    if (this->option_display["show_date"])
      cout << "Current date: " <<
        this->Model.GetCurrentDate().GetDate("%y-%m-%d %h:%i") << endl;

    Model.InitStep();
    OutputSaver.InitStep(Model);

    Model.Forward();
    OutputSaver.Save(Model);
  }
```

The main calls are `Model.Init()` for initialization (reading configuration and memory allocation), `Model.InitStep()` for initialization at each timestep (reading input data) and `Model.Forward` (time integration over one timestep).

Now assume that observations are read in the driver and that a method allows to modify model concentrations. In `driver/OptimalInterpolationDriver.cxx`, the following lines are added in the main time loop:

```
        // Retrieves observations.
        this->ObsManager.SetDate(this->Model.GetCurrentDate());
```

```
            if (this->ObsManager.IsAvailable())
              {
                cout << "Performing optimal interpolation at date ["
                     << this->Model.GetCurrentDate().GetDate("%y-%m-%d %h:%i")
                     << "]...";
                cout.flush();

                this->Model.GetState(state_vector);
                Analyze(state_vector);
                this->Model.SetState(state_vector);

                this->OutputSaver.SetGroup("analysis");
                this->OutputSaver.Save(this->Model);

                cout << " done." << endl;
              }
```

The key methods are `this->Model.GetState` to retrieve the state vector in the model (concentrations, maybe emissions or whatsoever) and `this->Model.SetState` to replace the forecasted state with the analyzed state.

A few other methods in the model are required by the driver. Eulerian models Castor and Polair3D do not have these methods in their base versions. Hence derived versions of them (C++ inheritance) add the required methods. In directory include/models/, `Polair3DChemistry` is derived into `Polair3DChemistryAssimConc` in order to implement `GetState`, `SetState`, `GetState_ccl` (adjoint variable), `SetState_ccl`, `BackgroundErrorCovariance`, `ModelErrorCovariance` and a few other methods.

All drivers are in directory driver/. They implement

- the optimal interpolation algorithm (Section 2),
- Kalman filters (ensemble – Section 3 – and RRSQRT – Section 5),
- 4D-Var assimilation (Section 4).

1 Test Case

1.1 Introduction

The test case is a simulation with RACM (72 chemical species, with ozone, O_3, among targets) over Europe from 1st July 2001 at 00:00 UT to 2nd July at 23:00 UT. There is enough data to extend the simulation period up to 10 days. In order to speed up the computations, a very coarse mesh is used: the horizontal resolution is 2° with 11 cells along latitude and 16 cells along longitude, and there are 3 vertical levels (up to 1200 m). This is enough to reproduce the main physical phenomena and to test the assimilation algorithms. The domain and its discretization is shown in Figure 1.

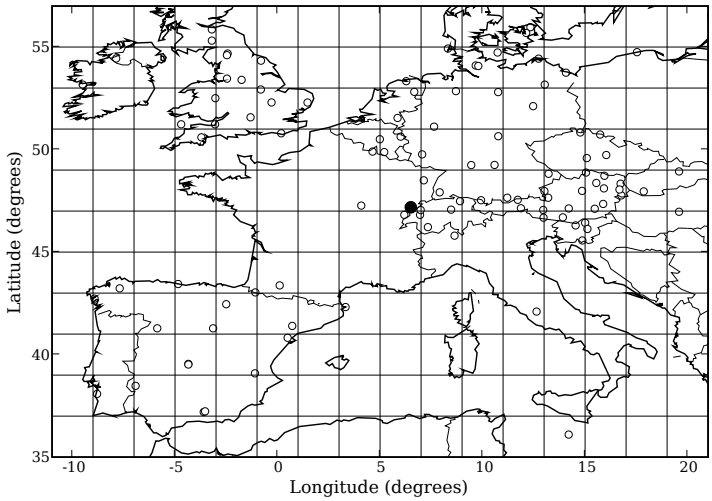

Figure 1: Horizontal discretization for the reference simulation. The circles show the locations of EMEP monitoring stations, and the disc shows the location of the monitoring station Montandon.

1.2 Main Configuration File

The configuration files are in `assimilation/config/`. Move to this directory. **All commands should be launched from this directory.** *Open* `polair3d.cfg`. It contains several sections:

- `[domain]`;
- `[data_assimilation]`: data assimilation parameters shared by all or several algorithms.
- `[EnKF]`: options for ensemble Kalman filter;
- `[RRSQRT]`: options for reduced rank square root Kalman filter;
- `[4DVar]`: options for variational assimilation;
- `[optimizer]`: options for the minimizer (in 4D-Var);
- `[display]`, `[options]` and `[data]`;
- `[output]`: note `Configuration_file` is set to `polair3d.cfg`, hence output savers configuration is put in `polair3d.cfg` too;
- `[state]`: defines the state vector in the assimilation algorithm;
- `[observation_management]`: path to observations description;
- `[perturbation_management]`: path to perturbations description;
- `[save]`: typical output saver configuration;

- sections `[@save]`: you can insert any section whose name is not expected by the code, and the code will ignore it; here, sections `[@save]` are therefore skipped – rename them into `[save]` in order to activate them;
- `[adjoint]`: options about adjoint management.

Further details are available primarily in Polyphemus user's guide and in next sections.

1.3 Simulation and Observations

Launch the reference simulation and display an ozone map on 1st July at 10:00 UT. The command is `../../Polyphemus-1.3/driver/polair3d polair3d.cfg`. You should get the map in Figure 2. You may use `getm` and `getd` with `config/disp.cfg`.

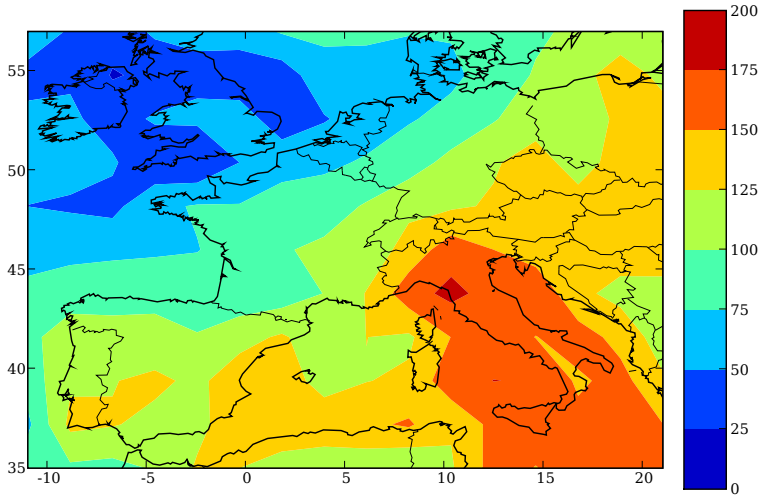

Figure 2: Ozone map of the reference simulation on 1st July at 10:00 UT.

Comparison with observations is performed with AtmoPy. The file `config/score.py` shows an example. *Launch `run -i score` in a IPython shell (launched in `assimilation/config/`.*

Browse `score.py`. In this program, the main variables are

- `sim3d`: concentration field (same as `sim3d = getd("disp.cfg")`);
- `sim`: list (indexed by 87 monitoring stations) of hourly *simulated* concentrations at monitoring stations – `sim[i]` is therefore a vector of simulated concentrations at station #`i`;
- `obs`: list (indexed by 87 monitoring stations) of hourly *measured* concentrations at monitoring stations – `obs` has the same structure as `sim`;
- `rmse`: root mean square error as function of time (hours).

Compute the root mean square error (RMSE) over the whole simulation period (two days) using `ensemble.collect`. Help about `ensemble.collect` may be found in AtmoPy documentation (`http://cerea.enpc.fr/polyphemus/doc/atmopy/index.html`, module `ensemble.combine`) or in IPython with command `help ensemble.collect`.

Display observations and simulated concentrations at Montandon, France. Use `sim` and `obs` in which Montandon is the first station and command `atmopy.display.plot`. You should get Figure 3. Note that Montandon is in cell ($j = 6$, $i = 8$). We chose not to interpolate concentrations at the station; hence concentrations at Montandon are the same as concentrations in cell (6, 8).

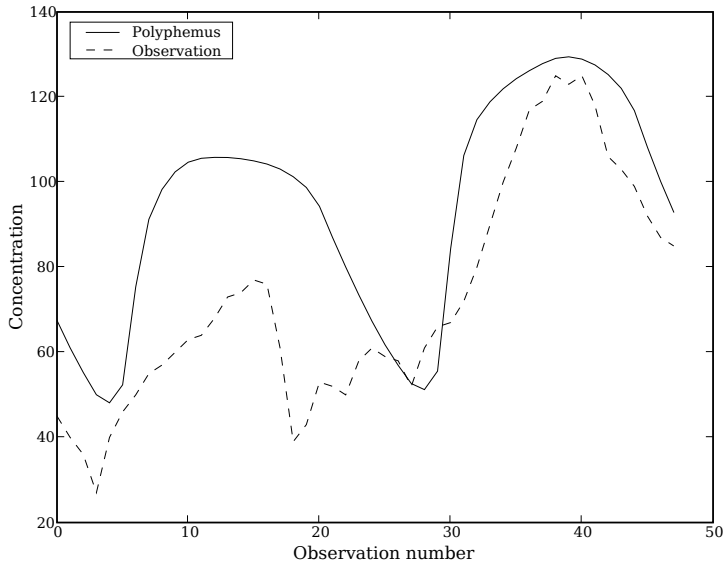

Figure 3: Ozone concentrations at Montandon (EMEP station in France).

2 Optimal Interpolation

2.1 Introduction

Optimal interpolation (OI) improves the state vector c each time a new vector o of observations is available:

$$c = c^b + PH^T(HPH^T + R)^{-1}(o - Hc^b) \tag{1}$$

where c^b is the background value of the state vector, H is the observation operator that maps the state vector to the observation space, R is the covariance matrix of observation errors and P is the covariance matrix of background error.

In this session, hourly ozone measurements from the EMEP network are assimilated.

The implementation of the algorithm may be found in `../../Polyphemus-1.3/driver/OptimalInterpolationDriver.cxx`.

From a technical point of view, data assimilation involves (1) a model with the suitable interface, (2) an observation manager, (3) a driver that gathers both and implements the assimilation algorithm. The model here derives from `Polair3DChemistry` and is called `Polair3DChemistryAssimConc`. This derived model implements a few additional methods needed for data assimilation with concentrations as state variable. The observation manager may be `GroundObservationManager` or `SimObservationManager`. In this session, only `GroundObservationManager` is used.

Consequently, in ../../Polyphemus-1.3/driver/polair3d-oi.cpp, the driver is declared this way:

```
OptimalInterpolationDriver<real, ClassModel,
  BaseOutputSaver<real, ClassModel>,
  GroundObservationManager<real> >
  Driver(argv[1]);
```

2.2 Configuration

The covariance matrix of background error is either diagonal or in Balgovind form. In the latter option, the error covariance between two points depends on the distance r between these two points:

$$f(r) = \left(1 + \frac{r}{L}\right) \exp\left(-\frac{r}{L}\right) v \tag{2}$$

where L is a characteristic length (`Balgovind_scale_background`) and v is a variance (`Background_error_variance`). In the configuration, section `[data_assimilation]` describes error statistics. Refer to the user's guide about `Polair3DChemistryAssimConc` for further details.

The configuration of `Polair3DChemistryAssimConc` includes the section `[state]` which describes the state variables for the assimilation procedure.

The observation manager `GroundObservationManager` is configured in observation.cfg.

2.3 Experiment

It is advocated to save the results of the reference simulation, otherwise they will be replaced with the new results (in directory results/). For instance, execute cp ../results/O3.bin ../results/save/O3-ref.bin. Later, you may reload the results with run score ../results/save/O3-ref.bin.

Launch the assimilation experiment. The command is ../../Polyphemus-1.3/driver/polair3d-oi polair3d.cfg. *Estimate the performances of the new simulation.* In IPython, launch the command run score.

In polair3d.cfg, rename the first two sections `[@save]` into `[save]`. This will add the saver unit of type `domain_assimilation` and `domain_prediction` (entry `Type`) which saves analyzed and predicted concentrations and their dates. *Display the analyzed and predicted concentrations.*

In order to check the algorithm behavior, assimilate only observations from Montandon. Since Montandon is the first station, we select it if you set `Nstation` to 1 in observation.cfg. *Compare two simulations with Montandon observations solely assimilated and with different Balgovind parameters.*

3 Ensemble Kalman Filter

3.1 Introduction

Ensemble Kalman filter (EnKF) differs with optimal interpolation in that the error covariance matrix is flow-dependent and approximated by ensemble statistics. The model applies to the $r-$member ensemble $\{c_i, i = 1, \ldots, r\}$, and produces forecast $\{c_i^f, i = 1, \ldots, r\}$. Whenever

observations are available, each ensemble member c_i^f is updated according to the OI formula (1). The statistics of the updated ensemble $\{c_i^a, i = 1, \dots, r\}$ gives the estimation of model status conditioned to observations. The forecast error covariance matrix P^f is approximated by

$$P^f = \frac{1}{r-1} \sum_{i=1}^{r} \left(c_i^f - \bar{c}^f\right) \left(c_i^f - \bar{c}^f\right)^T \tag{3}$$

where $\bar{c}^f$ is the ensemble mean. After the update, the estimated error covariance matrix P^a is approximated by

$$P^a = \frac{1}{r-1} \sum_{i=1}^{r} \left(c_i^a - \bar{c}^a\right) \left(c_i^a - \bar{c}^a\right)^T \tag{4}$$

The implementation of the algorithm may be found in `../../Polyphemus-1.3/driver/EnKFDriver.cxx`.

3.2 Configuration

The configurations for `Polair3DChemistryAssimConc` and `GroundObservationManager` are similar to those of OI. In `polair3d.cfg`, the ensemble number r can be modified in section `[EnKF]`. The observations in the update formula can be perturbed to preserve consistent statistics.

The ensemble is generated by perturbation of fields defined in `perturbation.cfg`.

3.3 Experiment

The experiment settings are similar to those of OI. *Launch the assimilation experiment.* The command is `../../Polyphemus-1.3/driver/polair3d-enkf polair3d.cfg`. In IPython, launch the command `run score` to *estimate the performances of the new simulation.*

In `polair3d.cfg`, rename the third section `[@save]` into `[save]`. This will add the saver unit of type `domain_ensemble` which saves ensemble forecast during assimilation. Program `load_ensemble.py` is provided to load ensemble date. *Browse the program.* The main variable is `ens`; it stores concentrations of all ensemble members. *Display concentrations of all members at Montandon. Analyze the spatial distribution of uncertainties.* You may use `atmopy.stat.spatial_distribution` (refer to user's guide).

Try with other fields perturbations. Evaluate the impact of the random-generator seed.

4 4D-Var

4.1 Introduction

In 4DVar, the optimal initial condition is obtained by minimizing a cost function,

$$J(c) = \underbrace{\frac{1}{2}(c - c^b)^T B^{-1} (c - c^b)}_{J_b} + \underbrace{\frac{1}{2} \sum_{t=0}^{N} \overbrace{\left(o_t - H_t(c_t)\right)^T R_t^{-1} \left(o_t - H_t(c_t)\right)}^{J_{ot}}}_{J_o} \tag{5}$$

under the constraint $c_t = M_{0 \to t}(c) = M_{t-1 \to t} \dots M_{0 \to 1} c$ – where $M_{t-1 \to t}$ denotes integrations of the underlying chemistry-transport model. The gradient is calculated with the adjoint model. Based on the optimal initial condition, the assimilated concentrations are simulated from time step 0 to N. Further integrations (after time step N) provide a subsequent forecast. Note that

in this version, only diagonal B is supported. The implementation of the algorithm may be found in `FourDimVarDriver.cxx`.

4.2 Configuration and Experiments

The configuration and experiment setting are similar to those of EnKF. The tuning parameters for 4DVar can be adjusted in `polair3d.cfg`. *Launch the assimilation experiment.* The command is `../../Polyphemus-1.3/driver/polair3d-4dvar polair3d.cfg`. In IPython, launch the command `run score` to *estimate the performances of the new simulation. Display concentrations at Montandon.*

Add the second model layer in the state vector and compare with previous results.

5 Reduced-Rank Kalman Filter

5.1 Introduction

Reduced-rank square root Kalman filter (RRSQRT) uses a low-rank representation LL^T of error covariances matrix P. $L = [l^1, \ldots, l^q]$ is the mode matrix whose columns (modes) are the dominant directions of the forecast error. The modes can also be interpreted as bases for how the true states might differ from mean state. The evolution of a mode can thus be approximated by the differences of the forecasts based on the mean state and its perturbation by this mode, that is,

$$l_t^i = M_{t-1\to t}(\bar{c}_{t-1} + l_{t-1}^i) - M_{t-1\to t}(\bar{c}_{t-1}) \tag{6}$$

where $t-1, t$ are consecutive time index. Note that in extended Kalman filter, the forecast error covariance matrix is calculated by

$$L_t L_t^T = P_t = \mathbf{M} P_{t-1} \mathbf{M}^T + Q_{t-1} = \begin{bmatrix} \mathbf{M}L_{t-1} & Q_{t-1}^{\frac{1}{2}} \end{bmatrix} \begin{bmatrix} \mathbf{M}L_{t-1} & Q_{t-1}^{\frac{1}{2}} \end{bmatrix}^T \tag{7}$$

where $\mathbf{M}$ is the tangent linear model of $M_{t-1\to t}$, $Q_{t-1}^{\frac{1}{2}}$ is the square root of model error covariance matrix. We do not use tangent linear model, but employ formula (6) to simulate $\mathbf{M}L_{t-1}$. $Q_{t-1}^{\frac{1}{2}}$ is obtained by perturbing fields as in the EnKF implementation. The update formula for the mode matrix can be sketched as $L_t^a = L_t \Pi$. where Π is the square root of $\mathcal{I} - \mathbf{KH}$, $\mathcal{I}$ is the identity matrix, $\mathbf{K}$ is the Kalman gain calculated from L_t, and $\mathbf{H}$ is the linearized observation operator. Implementation details can be found in `../../Polyphemus-1.3/RRSQRTDriver.cxx`.

5.2 Configuration and Experiment

The configuration and experiment setting are similar to those of EnKF. The tuning parameters for RRSQRT can be adjusted in `polair3d.cfg` (section `[RRSQRT]`. *Launch the assimilation experiment.* The command is `../../Polyphemus-1.3/driver/polair3d-rrsqrt polair3d.cfg`. In IPython, launch the command `run score` to *estimate the performances of the new simulation.*

6 Optional Questions

Try to use the drivers `AdjointDriver` and `GradientDriver`.

In the previous questions, the assimilation is performed over the first day and prediction occurs the second day. *Assimilate over the first two days and predict for the third day. Compare with results where assimilation is performed only over the second day.*