OpenDA Course 2023

Nils van Velzen, Martin Verlaan

July 18, 2023

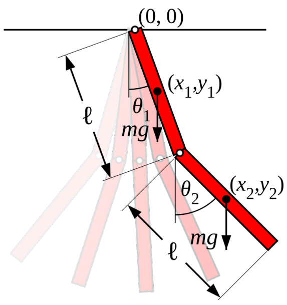
Contents

Installation of OpenDA

Before you can start with the exercises, you should first install OpenDA. For the latest instructions, you are referred to
`https://docs.openda.org/en/readthedocs/OpenDA_installation.html`.

1 Exercise: Getting Started

Directory: `exercise_double_pendulum_part1`
A pendulum is a rigid body that can swing under the influence of gravity. It is attached at the top so it can rotate freely in a two-dimensional plane (x, y). We will assume a thin rectangular shape with the mass equally distributed. A double pendulum is a pendulum connected to the end of another pendulum. Contrary to the regular movement of a pendulum, the motion of a double pendulum is very irregular when sufficient energy is put into the system.



The dynamics of a double pendulum can be described with the following equations (https://en.wikipedia.org/wiki/Double_pendulum):

$$\frac{d\theta_1}{dt} = \frac{6}{ml^2}\frac{2p_{\theta_1} - 3\cos(\theta_1-\theta_2)p_{\theta_2}}{16 - 9\cos^2(\theta_1-\theta_2)}, \tag{1}$$

$$\frac{d\theta_2}{dt} = \frac{6}{ml^2}\frac{8p_{\theta_2} - 3\cos(\theta_1-\theta_2)p_{\theta_1}}{16 - 9\cos^2(\theta_1-\theta_2)}, \tag{2}$$

$$\frac{dp_{\theta_1}}{dt} = -\frac{1}{2}ml^2\left(\frac{d\theta_1}{dt}\frac{d\theta_2}{dt}\sin(\theta_1-\theta_2) + 3\frac{g}{l}\sin(\theta_1)\right), \tag{3}$$

$$\frac{dp_{\theta_1}}{dt} = -\frac{1}{2}ml^2\left(-\frac{d\theta_1}{dt}\frac{d\theta_2}{dt}\sin(\theta_1-\theta_2) + \frac{g}{l}\sin(\theta_2)\right), \tag{4}$$

where the x, y-position of the middle of the two segments can be computed as:

$$x_1 = \frac{l}{2}\sin(\theta_1), \tag{5}$$

$$y_1 = \frac{-l}{2}\cos(\theta_1), \tag{6}$$

$$x_2 = l(\sin(\theta_1) + \frac{1}{2}\sin(\theta_2)), \tag{7}$$

$$y_2 = -l(\cos(\theta_1) + \frac{1}{2}\cos(\theta_2)). \tag{8}$$

This model, although simple, is very nonlinear and has a chaotic nature. Its solution is very sensitive to the parameters and the initial conditions: a small difference in those values can lead to a very different solution.
The purpose of this exercise is to get you started with OpenDA. You will learn to run a model in OpenDA, make modifications to the input files, and plot the results.

1.1 Input files

The input for this exercise is located in the directory `exercise_pendulum_part1`. For Linux and Mac OS X, go to this directory and start `oda_run.sh`, the main application of OpenDA. For Windows, start the main application with `oda_run_gui.bat` from the `$OPENDA/bin` directory. The main application allows you to view and edit the OpenDA configuration files, run your simulations, and visualize the results.

1.2 Simulation and postprocessing with the double-pendulum model

Try to run a simulation with the double-pendulum model. You can use the configuration file `simulation_unperturbed.oda`.
For postprocessing in Python, the results are written to the file

`simulation_unperturbed_results.py`

These results can be loaded with:

```python
import simulation_unperturbed_results as unperturbed
# import importlib; importlib.reload(unperturbed) if
    unperturbed was loaded before
```

Listing 1: Python initialize

We have added a routine `plot_movie` to create an intuitive representation of the data.

```python
import pendulum as p #needed only once
p.plot_movie(unperturbed.model_time,unperturbed.x)
```

Listing 2: Python

To create a time-series plot in Python type:

```python
plt.subplot(2,1,1)
plt.plot(unperturbed.model_time,unperturbed.x[:,0],"b")
# Python counts starting at 0
plt.ylabel(r"$\theta_1$") # use latex for label
plt.subplot(2,1,2)
plt.plot(unperturbed.model_time,unperturbed.x[:,1],"b")
plt.ylabel(r"$\theta_2$")
plt.show()
# only needed if interactive plotting is off.
# Set with plt.ioff(), plt.ion()
```

Listing 3: Python

1.3 An alternative simulation with the double-pendulum model

Then you can start an alternative simulation with the double-pendulum model that starts with a slightly different initial condition using the configuration file `simulation_perturbed.oda`. The different initial conditions can be found in

- `modelDoublePendulumStochModel.xml`, and
- `modelDoublePendulumStochModel_perturbed.xml`.

Visualize the unperturbed and perturbed results in a single plot. Make a movie and a time-series plot of $\theta_1$ and $\theta_2$ variables. Do you see the solutions diverging like the theory predicts?
To create a movie with both results in Python type:

```python
import simulation_unperturbed_results as unperturbed
import simulation_perturbed_results as perturbed
p.plot_movie(unperturbed.model_time, unperturbed.x, perturbed
    .x)
```

Listing 4: Python initialize

To create a time-series plot with both results in Python type:

```python
plt.subplot(2,1,1)
plt.plot(unperturbed.model_time,unperturbed.x[:,0],"b")
# Python counts starting at 0
plt.plot(perturbed.model_time,perturbed.x[:,0],"g")
plt.ylabel(r"$\theta_1$") # use LaTeX for label
plt.subplot(2,1,2)
plt.plot(unperturbed.model_time,unperturbed.x[:,1],"b")
plt.plot(perturbed.model_time,perturbed.x[:,1],"g")
plt.ylabel(r"$\theta_2$")
plt.show()
```

Listing 5: Python

1.4 An ensemble of model runs

Next, we want to create an ensemble of model runs all with slightly different initial conditions. You can do this in several steps:

- First, create the input file `simulation_ensemble.oda` based on `simulation_unperturbed.oda`. Change the algorithm and the configuration of the algorithm. Hint: the algorithm is called `org.openda.algorithms.kalmanFilter.SequentialEnsembleSimulation`.
- Create a configuration file for the ensemble algorithm (e.g. named

algorithm/SequentialEnsembleSimulation.xml). It should contain the following content:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<sequentialAlgorithm>
  <analysisTimes type="fromObservationTimes"></
    analysisTimes>
  <ensembleSize>5</ensembleSize>
  <ensembleModel stochParameter="false"
                 stochForcing="false"
                 stochInit="true" />
</sequentialAlgorithm>
```

Listing 6: XML input for sequentialAlgorithm

Hint: do not forget to refer to

algorithm/SequentialEnsembleSimulation.xml in

simulation_ensemble.oda and do not forget to give a different name to the output files.

- Run the new configuration with OpenDA.
- Make a plot of the first and second variables of the five ensemble members in a single time-series plot

```python
import ensemble
import simulation_ensemble_results as res
(t,ens)=ensemble.reshape_ensemble(res)
ens1=ens[:,0,:] #note we start counting at 0
ens2=ens[:,1,:]
plt.subplot(2,1,1)
plt.plot(t[1:],ens1,"b")
plt.ylabel(r"$\theta_1$")
plt.subplot(2,1,2)
plt.plot(t[1:],ens2,"b")
plt.ylabel(r"$\theta_2$")
plt.show()
```

Listing 7: Python

- Observations of θ_1 and θ_2 are available as well. Make a plot of the observations together with the simulation results.

```python
import simulation_unperturbed_results as unperturbed
plt.subplot(2,1,1)
plt.plot(unperturbed.model_time,unperturbed.x[:,0],"b")
plt.plot(unperturbed.analysis_time,unperturbed.obs[:,0],
    "k+")
```

```python
plt.ylabel(r"$\theta_1$")
plt.subplot(2,1,2)
plt.plot(unperturbed.model_time,unperturbed.x[:,1],"b")
plt.plot(unperturbed.analysis_time,unperturbed.obs[:,1],
    "k+")
plt.ylabel(r"$\theta_2$")
plt.show()
```

Listing 8: Python

We can see that although our simulation starts on the right track, it quickly diverges from the observations. The aim of the Ensemble Kalman filter or data assimilation in general, is to keep the model on track.

2 Exercise: Some basic properties of the EnKF

Directory: `exercise_double_pendulum_part2`
In this exercise, you will learn how to set up and run the EnKF method in OpenDA.

- Prepare the input files for a run with the EnKF method. Use the input files from exercise 1 as a template. Hint: the Ensemble Kalman filter is called `org.openda.algorithms.kalmanFilter.EnKF`. The algorithm configuration file has the following content

```xml
      <?xml version="1.0" encoding="UTF-8"?>
      <EnkfConfig>
         <ensembleSize>10</ensembleSize>
         <ensembleModel stochParameter="false"
                        stochForcing="false"
                        stochInit="true" />
      </EnkfConfig>

```

Listing 9: XML input for EnKF algorithm

  Note: in this exercise, we are only considering the uncertainty in the initial conditions. In general, also the uncertainty of the parameters or the model forcing, such as boundary conditions can be considered.

- Plot the ensemble mean of the first model variable and the observations. With some luck, the solution should track the observations.
  For comparison, we have also added the configurations for the 'truth' and an oda_run without data assimilation called 'initial'.

```python
            import simulation_truth_results as truth
            import simulation_initial_results as initial
            import simulation_enkf_results as enkf
            plt.subplot(2,1,1)
```

```
    plt.plot(initial.model_time,initial.x[:,0],"g")
    plt.plot(truth.model_time,truth.x[:,0],"k")
    plt.plot(enkf.analysis_time,enkf.x_f_central[:,0],"b");
    plt.legend(("initial","truth","EnKF"))
    plt.ylabel(r"$\theta_1$")
    plt.subplot(2,1,2)
    plt.plot(initial.model_time,initial.x[:,1],"g")
    plt.plot(truth.model_time,truth.x[:,1],"k")
    plt.plot(enkf.analysis_time,enkf.x_f_central[:,1],"b");
    plt.ylabel(r"$\theta_2$")
    plt.xlabel(r"$t$")
    plt.show()
```

Listing 10: Python initialize

- The Ensemble Kalman filter uses a random number generator. In OpenDA we can control the initial value of the generator by adding a line like: `<initialSeed type="specify" seedValue="21" />` near the end of the main configuration file. Do you get the same results if you rerun with the same value for the initial seed? And what if you use a different value?
- Look at the observation input file of the StochObserver. The StochObserver does not only describe the observations but the accuracy as well. Can you make a new observation input file with similar observed values but with a 10 times larger standard deviation for the observation error? Tip: you can edit the file in OpenOffice or MS Excel or use the find-and-replace function of an advanced text editor. Repeat the run with EnKF but now for the new observations and plot the first variable of the ensemble means and the observations. What do you see, and what is the reason for this behavior of the algorithm?
- The number of ensemble members controls the accuracy of the ensemble approximation. What happens if you decrease it to 10 or 6?

3 Exercise: Localization

Directory: `exercise_localization`
In this exercise, you will learn about localization techniques and how to use them in OpenDA. This exercise is inspired by the example model and experiments from "Impacts of localisation in the EnKF and EnOI: experiments with a small model", Peter R. Oke, Pavel Sakov and Stuart P. Corney, Ocean Dynamics (2007) 57: 32-45.
The model we use is a simple circular advection model

$$\frac{\partial a}{\partial t} + u\frac{\partial a}{\partial x} = 0, \tag{9}$$

where $u = 1$ is the speed of advection, a is a model variable, t is time and x is a space ranging from 1 to 1000 with grid spacings of 1. The computational domain is periodic in x.

In this model, there are two related variables a and b, where b is initialized with a balance relationship:

$$b = 0.5 + 10\frac{da}{dx} \tag{10}$$

and propagated with an advection model similar to the one for a, i.e.:

$$\frac{\partial b}{\partial t} + u\frac{\partial b}{\partial x} = 0. \tag{11}$$

Since a and b are propagated with the same flow, the balance relationship will remain valid also for $t > 0$. The relationship between a and b is motivated by the geostrophic balance relationship between pressure (a) and velocity (b) in oceanographic and atmospheric applications.
In this experiment, we will only observe and assimilate a and investigate how both a and b are updated. The ensemble is carefully constructed in order to have the correct statistics. The initial ensembles are generated offline and they will be read when the model is initialized in OpenDA.

- Investigate the script `generate_ensemble.py` and figure out how the ensembles are generated.
- Run Python script `generate_ensemble.py` to generate ensembles, observations, and true state for a 25, 50, and 100 ensemble experiment.
- Run the experiment for 50 ensemble members (`enkf_50.oda`).
- The variables a and b can be compared to the true state using the Python script `plot_results.py`.
- Run the experiment for 25 ensembles, copy the script `plot_results.py` to e.g. `plot_results_25.py` and adjust it in order to read the results from `enkf25_results.py` (change 2nd line of the `plot_results.py` script). You will see that the 25 ensemble run is not able to improve the model.
- Create input to run a 100-ensemble experiment. Note: do not forget to change the name of the output file (section `resultWriter`) to avoid your previously-generated results being overwritten.
- Run an experiment with 25 ensembles with localization (see the script `enkf_25_loc.oda`) and generate the plots.
- The results (for 25 ensembles) with localization should look better than the experiment without localization.
- Investigate whether the relation between a and b is violated by the various experiments. You can use the script `check_balance.py`.
- Try changing the localization radius (initial value is 50) and see how the performance of the algorithms changes (both for results as balance between a and b). You can plot the localization weight functions for each observation location (`rho_0`, `rho_1`, `rho_2`, and `rho_3`) as well.

4 Exercise: A black-box model — Calibration

A simple way to connect a model to OpenDA is by letting OpenDA access the input and output files of the model. OpenDA cannot directly understand the input and output files of an arbitrary model. Some code has to be written such that the black-box model implementation of OpenDA can read and write these files. In this exercise, you will learn how to connect an existing model to OpenDA assuming that all the input and output files of the model can indeed be accessed by OpenDA. The exercise focuses on the configuration of the black-box wrapper in OpenDA.
The model describes the advection of two chemical substances. The first substance c_1 is emitted as a pollutant by a number of sources. However, in this case this substance reacts with the oxygen in the air to form a more toxic substance c_2. The model implements the following equations:

$$\frac{\partial c_1}{\partial t} + u\frac{\partial c_1}{\partial x} = -1/Tc_1, \tag{12}$$

$$\frac{\partial c_2}{\partial t} + u\frac{\partial c_2}{\partial x} = 1/Tc_1. \tag{13}$$

In the directory
`exercise_black_box_calibration_polution/original_model/`
you will find:

1. the model executable: `reactive_pollution_model.py` (Linux and Mac) and `reactive_pollution_model.exe` (Windows);
2. the model configuration file: `config.yaml`;
3. the forcings of the model (injection of pollutant): `forcings`;
4. the initial model state: `input`.

- Run the model, in the `original_model` directory from the command line, not using OpenDA.

The model generates the output files
`c1_locA`,`c1_locB`, `c1_locC`, `c1_locA`, `c2_locB`, and `c2_locC`,
with time series of substance c_1 and c_2 at three predefined locations in the model. The folder `maps` contains output files with the concentration of c_1 and c_2 on each grid point at specified times. The folder `restart` contains files that allow the model to restart; continue the computations from the point where a restart file has been written.

- Investigate the input and output files of the model.
- Generate a movie by running the script `plot_movie_orig.py` script from the `exercise_black_box_calibration_polution` (!) directory. This allows you to study the behavior of the model.

4.1 Wrapper configuration files

The input and output files of this model are all easy-to-interpret ASCII files. Therefore, we do not need model-specific routines to couple this model to OpenDA.
When you couple an arbitrary model to OpenDA and you want to use the black-box coupler, there are two approaches:

- write a pre- and post-processing script that translates the (relevant) model files into a more generic format that is already supported (e.g. ASCII or NetCDF).
- write your own adapter in Java (data object) to read and write the model input and output files.

A black-box wrapper configuration usually consists of three XML files. For our pollution model, these files are:

1. `polluteWrapper.xml`: This file specifies how OpenDA can run the model, which input and output files are involved, and which data objects are used to interpret the model files.
   This file consists of the parts:
   - `aliasDefinitions:` This is a list of strings that can be aliased in the other XML files. This helps to make the wrapper XML file more generic. E.g. the alias definition `%outputFile%` can be used to refer to the output file of the model, without having to know the actual name of that output file.
     Note the special alias definition `%instanceNumber%`. This will be replaced internally at runtime with the member number of each created model instance.
   - `run:` the specification of what commands need to be executed when the model is run.
   - `inputOutput:` the list of 'input/output objects', usually files, that are used to access the model, i.e. to adjust the model's input, and to retrieve the model's results. For each 'dataObject', one must specify:
     - the Java class that handles the reading from and/or writing to the file
     - the identifier of the dataObject, so that the model configuration file can refer to it when specifying the model variables that can be accessed by OpenDA, the so-called 'exchange items' (see below)
     - optionally, the arguments that are needed to initialize the data object, i.e. to open the file.
2. `polluteModel.xml`: This is the main specification of the (deterministic) model. It contains the following elements:
   - `wrapperConfig`: A reference to the wrapper config file mentioned above.
   - `aliasValues`: The actual values to be used for the aliases defined in the wrapper config file. For instance, the `%configFile%` alias is set to the value `config.yaml`.

    - `timeInfoExchangeItems`: The name of the model variables (the 'exchange items') that can be accessed to modify the start and end time of the period that the model should compute to propagate itself to the next analysis time.
    - `exchangeItems`: The model variables that are allowed to be accessed by OpenDA, for instance, parameters, boundary conditions, and computed values at certain locations. Each variable exchange item consists of its id, the dataObject that contains the item, and the 'element name', the name of the exchange item in the dataObject.

3. `polluteStochModel.xml`: the specification of the stochastic model. It consists of two parts:
    - `modelConfig`: A reference to the deterministic model configuration file mentioned above `polluteModel.xml`.
    - `vectorSpecification`: The specification of the vectors that will be accessed by the OpenDA algorithm. These vectors are grouped into two parts:
        - The state that is manipulated by an OpenDA filtering algorithm, i.e. the state of the model combined with the noise model(s).
        - The so-called predictions, i.e. the values on observation locations as computed by the model.

Start with a single OpenDA run to understand where the model results appear for this configuration:

- Have a look at the files `polluteWrapper.xml`, `polluteModel.xml` and `polluteStochModel.xml`, and recognize the various items mentioned above. Start the OpenDA GUI from the `public/bin` directory and run the model using the `Simulation.oda` configuration. Note that the actual model results are available in the directory where the black-box wrapper has let the model perform its computation: `work/work0`.

Directory: `exercise_black_box_calibration_polution`
In this exercise, we will calibrate the value of the reaction-rate constant. The algorithm used in this example is the Dud (which stands for Doesn't Use Derivative).

- Have a look at the `Dud.oda` and the configuration files it refers to. Run it from the OpenDA GUI and have a look at the results. What could you do to improve the results?
- Figure out where to change the control parameters for the calibration procedure and play around with the settings to improve your results.

Calibration runs normally take longer than a few minutes. In that case, it becomes convenient to be able to restart from a previous run.

- Adapt the configuration in such a way that you are able to restart the Dud.oda from the result of a previous run.

5 Exercise: A black-box model — Filtering

5.1 Sequential simulation

We will first run our pollution model from OpenDA using the SequentialSimulation algorithm. This run is exactly the same as running the model outside OpenDA. However, the difference is that we provide a set of observations and run the model and restart the model between the observation times. The output will be available at the end in the generic OpenDA format that allows us to compare the model results with the available observations of the system.

- Run the model within OpenDA by using the `SequentialSimulation.oda` configuration. This will create the result file `sequentialSimulation_results.py`. Use the script `plot_movie_seq.py` to visualize the simulation results. The script `plot_obs_seq.py` shows the difference in time between the model results (prediction) and observed values of the system.

5.2 Sequential ensemble simulation

The next step is running an ensemble of simulations. In this case, we consider the injection of pollutant c1 in the model as our main source of uncertainty. Similar to the sequential simulation we do not assimilate any data (yet).

- Run an ensemble forecast model by using the `SequentialEnsembleSimulation.oda` configuration. Look at the configuration file of the model (`stochModelpolluteStochModel.xml`). To which variable does the algorithm impose stochastic forcing?
  Have a look at the `work` directory, and note that the black-box wrapper created the required ensemble members by repeatedly copying the template directory `stochModel/input` to `output/work<N>`.

- Compare the result between the mean of the ensemble and the results from `SequentialSimulation.oda`. Note the differences. You can use the script `plot_movie_enssim.py`.

5.3 Parallel computing

Running the ensembles takes a lot of time, especially starting the model takes quite some time compared to the actual computation time. Most computers have multiple cores and the reactive pollution model only uses one core, so we can use our cores to propagate multiple ensemble members forward in time simultaneously.

- Compare the configurations `SequentialEnsembleSimulation.oda` and `enkf.oda` which uses parallel propagation of ensemble members. Set the number of simultaneous models equal to the number of cores on your computer (maxThreads).

5.4 Ensemble Kalman filter

Now let us have a look at the configuration for performing OpenDA's Ensemble Kalman Filtering on our black-box model, using a twin experiment as an example. The model has been run with the 'real' (time-dependent) values for the concentrations for substance 1 as disposed of by factory 1 and factory 2. This 'truth' has been stored in the directory `truthmodel`, and the results of that run have been used to generate observation time series at the output locations. These time series (with some noise added) have been copied to the `stochObserver` directory to serve as observations for the filtering run.
The filter run takes the original unperturbed model as input, whereas the 'truth' uses a perturbed version of the original model: the concentrations for substance 1 as disposed of by the factories have been flattened out to a constant value. The filter process should modify these values in such a way that the results resemble the truth as much as possible.
To do this the filter modifies the concentration at factory 2 and uses the observations downstream of factory 2 to optimize the forecast.

- Note that the same black-box configuration is used for the sequential run, the sequential ensemble run, and for the EnKF run. Identify the part of the `polluteStochModel.xml` configuration that is used only by the EnKF run and not by the others.
- Execute the Ensemble Kalman Filtering run by using the `EnKF.oda` configuration.
  Check how good the run is performing by analyzing to what extent the filter has adjusted the predictions towards the observation.
  Note that the model output files in `stochModel/output/work0` only contain a few time steps. Can you explain why? To compare the observations with the predictions, you have to use the result file produced by the EnKF algorithm, which can be visualized using `plot_movie_enkf.py`.

Now let us extend the filtering process by incorporating also the concentration disposed of by factory 1, and by including the observation locations downstream of factory 1.

- Make a copy of the involved config files, `EnKF.oda`, `parallel.xml`, `polluteStochModel.xml`, and `timeSeriesFormatter.xml` (you could call them `EnKF2.oda`, `parallel2.xml`, etc.). Adjust the files such that all references to the files are correct.
- Now adjust `polluteStochModel2.xml`, and `timeSeriesFormatter2.xml` in such a way that the filtering process is extended as described above.
- Run the filtering process by using the `EnKF2.oda` configuration, and compare the results with the previous version of the filtering process.

6 Exercise: A black-box model — Steady-state filter

Directory: `exercise_black_box_steady_state_filter`

In this section, you will learn how to create and use a steady-state Kalman filter with OpenDA. The example continues with the black-box reaction-pollution-model. Make sure you have completed the previous exercise before you start with this one.
The steady-state Kalman filter is a special case of the Kalman filter. If the measurement stations are fixed in time and measure with a fixed time-step and the model is linear and time invariant, then the Kalman gain matrix can converge to a fixed matrix over time. Sufficient conditions for this are the stability of the model or the controllability and observability of the model. If the Kalman gain converges, then it is possible to compute, store, and re-use the stored gain matrix. This has a large impact on the computational demand of the algorithm. The steady-state filter uses only a bit more computer time than the model, which is much faster than the Ensemble Kalman filter. The steady-state filter is therefore very suitable for real-time applications when it can be applied. In a strict sense, the steady-state filter has limited applicability, but if the model is not very non-linear or well constrained by the observations, then the Kalman gain matrix can still show very little variability over time. In this case, the steady-state filter can still be used. Clearly, this will not always work, but the computational advantage is so large that it is often worth considering.

- Run the `SequentialSimulation.oda` and the `EnKf.oda` in the folder `exercise_black_box_steady_state_filter`. While running, have a look at the file `algorithmsEnKF.xml` in there and notice how the Kalman gains can be written to the disc. You can plot these Kalman gains with the script `plot_gains.py` and study how similar they are. Would you conclude that the Kalman gains are converging? If so, would you conclude that the steady-state filter is applicable? Irrespective of your answer, continue with the next step.
- Now run the `SteadyStateFilter.oda`. What do you notice about the run time? You can plot the results using the script

  `plot_movie_steady_state.py`. How do the results of the steady-state filter compare to the results of the EnKF?
- Now repeat the steady-state filter run, but first rerun the EnKf with a larger number of ensembles. What do you notice about the results of the steady-state filter?
- Finally, rerun the steady-state filter with the Kalman gain at a different time. You can modify the file

  `algorithmsSteadyStateFilter.xml` to do this. Are the results different? What is the cause of the remaining differences?