OpenDA course and exercises

Nils van Velzen, Martin Verlaan, Stef Hummel

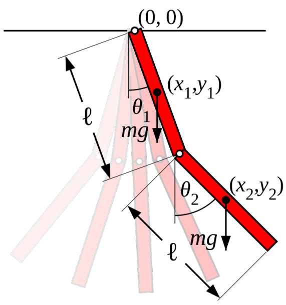
July 5, 2017

Installation of OpenDA

Before you can start with the exercises you must first install OpenDA. For the latest instructions, you are referred to `$OPENDA/doc/OpenDA_domunentation.pdf`, section "Installation" or the same document on our website `www.openda.org`.

1 Exercise 1 part1: Getting started

A pendulum is a rigid body that can swing under the influence of gravity. It is attached at the top so it can roatate 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 descibed with the following equations (This example was copied from https://en.wikipedia.org/wiki/Double_pendulum)

variables $\theta_1, \theta_2, p_{\theta_1}, p_{\theta_2}$:

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

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

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

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

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.

- The input for this exercise is located in 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.

- Try to run a simulation with the Lorenz model. You can use the configuration file `simulation_unperturbed.oda`.

  For postprocessing in Matlab the results are written to `simulation_unperturbed_results.m`. Next start Matlab and load the results. We have added a routine `plot_movie` to create an intuitive representation of the data. Please type (or copy-paste):

```
[t,unperturbed,tobs,obs]= ...
load_results('simulation_unperturbed_results');
plot_movie(t,unperturbed)
```

  Listing 1: Matlab

  For postprocessing in Python the results are written to `simulation_unperturbed_results.py`.

```
import numpy as np
import matplotlib.pyplot as plt
```

  Listing 2: Python initialize

  Next load the results. We have added a routine `plot_movie` to create an intuitive representation of the data. Please type (or copy-paste):

```
#load data
import simulation_unperturbed_results as sim
```

```
# make 3d line plot
from mpl_toolkits.mplot3d import Axes3D
fig1 = plt.figure()
```

Listing 3: Python

To create a time-series plot in Matlab type:

```
subplot(2,1,1);
plot(t,states(1,:),'b-');
ylabel('\theta_1');
subplot(2,1,2);
plot(t,states(2,:),'b-');
ylabel('\theta_2');
xlabel('time');
```

Listing 4: Matlab

To create a time-series plot in Python type:

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

Listing 5: Python

- Observations of the first variable are available as well. Make a plot of the observations together with the simulation results.

```
[t,xyz,tobs,obs]=load_results('
simulation_unperturbed_results');
plot(t,xyz(1,:),'b')
hold on
plot(tobs,obs,'r*');
hold off
```

Listing 6: Matlab

```
import simulation_unperturbed_results as sim
plt.plot(sim.model_time,sim.x[:,0])
plt.plot(sim.analysis_time,sim.obs,'r*')
```

Listing 7: Python

- Then you can start an alternative simulation with the lorenz model that starts with a slightly different initial condition using the configuration file `simulation_perturbed.oda` that starts with slightly different initial conditions.
- Visualize the unperturbed and perturbed results in a single plot. Make a 3d trajectory plot and a 2d plot in time of first variable. Do you see the solutions diverging like the theory predicts?

```
   [t1,xyz1,tobs1,obs1]=load_results('
simulation_unperturbed_results');
   [t2,xyz2,tobs2,obs2]=load_results('
simulation_perturbed_results');
   figure(1)
   plot3(xyz1(1,:),xyz1(2,:),xyz1(3,:),'b');
   hold on
   plot3(xyz2(1,:),xyz2(2,:),xyz2(3,:),'r');
   hold off
   legend('unperturbed','perturbed')

   figure(2)
   plot(t1,xyz1(1,:),'b')
   hold on
   plot(t2,xyz2(1,:),'r')
   hold off
   legend('unperturbed','perturbed')
```

Listing 8: Matlab

```
  #load unperturbed and perturbed results
  import simulation_unperturbed_results as sim
  import simulation_perturbed_results as simp
  fig3 = plt.figure()
  ax = fig3.add_subplot(111, projection='3d')
  Axes3D.plot(ax,sim.x[:,0],sim.x[:,1],sim.x[:,2],'b')
  Axes3D.plot(ax,simp.x[:,0],simp.x[:,1],simp.x[:,2],'r'
)

  fig4 = plt.figure()
  plt.plot(sim.model_time,sim.x[:,0],'b')
  plt.plot(simp.model_time,simp.x[:,0],'r')
```

Listing 9: Python

- Create a modified example that uses an ensemble forecast with perturbed initial conditions. You can do this in a number of steps:

- Create the input file `simulation_Ens.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.
- Write the configuration file of the Ensemble algorithm (e.g. named `algorithm/EnsSimulation.xml`) with the following content:

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

Listing 10: XML-input for sequentialAlgorithm

Hint: do not forget to reference `algorithm/EnsSimulation.xml` in `simulation_Ens.oda`.

- Run this ensemble simulation and read the results in Octave or Matlab using `load_ensemble.m` and slightly different for python
  - make a plot of the first variable of the five ensemble members in a single plot

```
[t,ens]=load_ensemble('simulation_ensemble_results
');
ens1=reshape(ens(1,:,:),size(ens,2),size(ens,3));
plot(t,ens1)
```

Listing 11: Matlab

```
import ensemble
import simulation_ensemble_results as res
(t,ens)=ensemble.reshape_ensemble(res)
ens1=ens[:,0,:] #note we start counting at 0
fig5 = plt.figure()
plt.plot(t,ens1)
```

Listing 12: Python

  - make a plot of the mean of the first variable

```
plot(t,mean(ens1,2))
```

Listing 13: Matlab

```
fig6 = plt.figure()
plt.plot(t,np.mean(ens1,1))
```

Listing 14: Python

  - run the same simulation again[1] but now with an ensemble size of 10, 50, 100 and 200 and plot the mean of the first variable. What do you see, and what does this mean?

2 Exercise 1 part 2: Some basic properties of the EnKF

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 template. Hint: the Ensemble Kalman filter is called org.openda.algorithms.kalmanFilter.EnKF. The algorithm configuration file has the following content

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

Listing 15: XML-input for EnKF algorithm

- Plot the ensemble mean of the first model variable and the observations. With some luck the solution should track the observations.
  Tip: use the scripts `load_obs.m` and `load_ensemble.m` for reading the data into matlab (cf. Exercise1), or `load_ensemble.py` for python.
- 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?

[1]For large models or ensemble sizes a huge amount of output is generated. Your run will be much faster when you disable the messages in the gui, by pressing the 'Disable Logging' button. You can also run without the gui, by using the command `oda_run.sh <inputfile>` (Linux/Mac OS X) or `oda_run_batch.bat <inputfile>` (Windows)

- The number of ensemble members controls the accuracy of the ensemble approximation. What happens if you increase the number to e.g. 100, or decrease it to 5? Use (initially) observations with a standard deviation of 5.0. Experiment as well with various standard deviations of the observations.

3 Exercise 2: Localization

In this exercise you will learn about localization techniques and how to use them in OpenDA. This exercise is inspired on 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 grom 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 initialised 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 relation ship 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 as b. The ensemble is carefully constructed in order to have the right statistics. The initial ensembles are generated off line and they will be read when the model is initialised in OpenDA.

- Investigate the sript `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, b can be compared to the true state using the python script `plot_results.py`

- Run the experiment for 25 ensembles, adjust the `plot_results.py` 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 that your previous generated results are overwritten.
- Run an experiment with 25 ensembles with localization (`enkf_25_loc.oda`) and generate the plots.
- The results (for 25 ensembles) with localization should look better than the the experiment without localization.
- Investigate whether the relation between a and b is violated. You can yse 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)

4 Exercise 3: A black box model - Filtering

This exercise uses the same model as exercise 4: a model written in python that describes the advection of two chemical species. Please read the start of exercise 4 if you are not familiar with this model yet. A description of the black box wrapper configuration, usually consisting of three xml files, can also be found in exercise 4.

- Run the model from the command line, not using OpenDA, like in Exercise 4. The model generates the output files: `reactive_pollution_model.output` and `reactive_pollution_model_output.m`. Use the m-file to make plots of the output in order to study the behavior of the model. In order to check the model (plotted) results you can look at the input file as well.

We start with some single and ensemble runs to understand where for our black box wrapper configuration the model results appear:

- Have a look at the files `polluteWrapper.xml`, `polluteModel.xml` and `polluteStochModel.xml`, and look for differences compared to exercise_4. Run the model within OpenDA by using the `SequentialSimulation.oda` configuration. Use the script `plot_movie.m` (or `plot_movie.py` for python) to visualize the model results. Compare the results with those from the run you executed without using OpenDA. Note that the actual model results are available in the directory where the black box wrapper has let the model perform its computation: `stochModel/output/work0`.
- Run an ensemble forecast model by using the `SequentialEnsembleSimulation.oda` configuration. On which variable

does the algorithm impose stochastic forcing?
Have a look at the `stochModel/output` directory, and note that the black box wrapper created the required ensemble members by repeatedly copying the template directory `stochModel/input` to `stochModel/output/work<N>`.

- A special model instance is `stochModel/output/work0`. It is the so called 'main' model, and is computed with the average of the perturbations of the ensemble members. Compare the results of `stochModel/output/work0` with the results of `SequentialSimulation.oda`. Note the relatively large differences. Check if these differences are reduced by increasing the ensemble size for the sequential ensemble simulation to 20 and rerunning `SequentialEnsembleSimulation..oda` (this run may take a few minutes).

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' values (time dependent)for the concentrations for substance 1 as disposed by factory 1 and factory 2. This 'truth' 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 have been copied to the `stochObserver` directory to serve as observations for the filtering run.

The filter run takes the original model as input, which actually is a perturbed version of the 'truth' model: the concentrations for substance 1 as disposed by 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 Matlab result file in `stochModel/output/work0` only contains a few time steps. Can you explain why?
So to compare the observations with the predictions you have to use the result file produced by the EnKF algorithm.

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

- Make a copy of the involved config files, `EnKF.oda` and `polluteStochModel.xml` (you could call them `EnKF2.oda` and

polluteStochModel2.xml.
Adjust EnKF2.oda so that it refers to the right stochastic model config file and produces a matlab result file with a recognizable name, e.g. enkf_results2.m.

- Now adjust polluteStochModel2.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.