Local scale - postprocessing and advanced methods.

Polyphemus Training Session

April 20, 2009

About

Purpose: Local scale simulations: postprocessing and plume-in-grid simulations.

Author: Irène Korsakissok, `Irene.korsakissok@cerea.enpc.fr`

Polyphemus version: 1.5

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

Contents

Installing the Session

You are advised to extract the archive **gaussian_advanced.tar.bz2** in your home directory.

```
$ tar -xjvf gaussian-advanced.tar.bz2
```

This creates a directory `gaussian_advanced/` in `polyphemus-sessions/`, which has also been created if necessary.

Place the source `Polyphemus` in `polyphemus-sessions/` also so that the paths given in configuration files are correct.

In what follows some command lines might be divided by \, they must be typed as a single line and are only divided for clarity's sake.

Introduction

The aim of this training session is to use advanced methods related to local scale. The first part consists in comparing results obtained with the Gaussian plume model to experiments, using the Prairie Grass data. It allows to compare different parameterizations to compute standard deviations, and to have a glimpse on the method used to do the full comparison. In the second part, a simulation is run with a plume-in-grid model, that is, a Gaussian puff model embedded into an Eulerian model. So this part will also allow the user to see how to run simulations with an Eulerian model. Finally, the part about liquid water content diagnosis consists in running simulations at local scale, both with an Eulerian model for local scale and with a Gaussian plume model, and to use postprocessing program to diagnose the liquid water content in the resulting concentration field.
All three parts are completely independant, so you may choose what is interesting for you, depending on what your use of Polyphemus will be after the training session.

1 Comparison to experiments: Prairie Grass

1.1 Description

In this part, comparisons to the Prairie Grass experiments will be made. The Prairie Grass experiment has become a standard database on which parameterizations have been fitted and that has been used for many models evaluation. The experiment took place in O'Neil, Nebraska, during summer 1956. The site was a flat terrain of short cut grass. A continuous plume of SO2 was released, without plume rise, near the ground (at 0.46m). Measurements where taken on five arcs at 50, 100, 200, 400 and 800m from the source (see figure **??**). There were nearly 70 trials. 43 of them will be tested here.

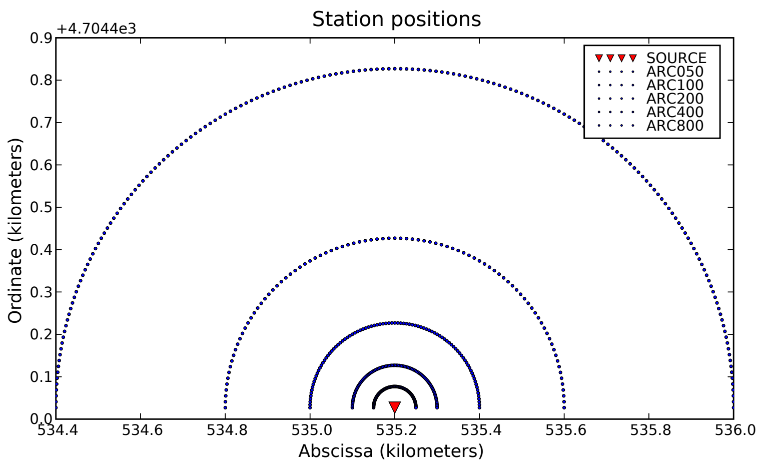


Figure 1: Position of measurement stations in the prairie grass field.

The aim of this part is, first, to have a glimpse of the different parameterizations for standard deviations and of their performances, and also to use some post-processing methods for comparison to observations. It will also permit to see how to save concentrations at given points instead of a whole domain, to make accurate comparisons.

The directory `~/polyphemus-sessions/gaussian_advanced/prairie_grass/` contains:

- A subdirectory `config/` which holds all the configuration files used for the simulation.
- A subdirectory `raw_data/` which holds files containing all necessary information about Prairie Grass experiment setup and observed concentrations.
- A subdirectory `results/` to hold the simulation results. There are already the results for all simulations and experiments:

  `results-briggs` contains the simulation results with Briggs parameterization.

  `results-doury` contains the simulation results with Doury parameterization.

`results-similarity` contains the simulation results with similarity theory parameterization.

`results-exp` contains the experimental results.

Note that each simulation results file is named `sim-results-pg**.bin`, where ** stands for the experiment ID, which ranges between 1 and 70. As only 43 trials have been tested here, not all IDs can be seen. The experiment ID is important because it can be used to check the corresponding data in the data files.

1.2 Data files

All necessary information are written in three files. They are located in:
`polyphemus-sessions/gaussian_advanced/prairie_grass/data`.

1.2.1 Experiments data file

The file `exp-data.txt` contains data about experimental setup. Each line corresponds to one experiment and contains the following information:

- Experiment ID number
- Experiment date (GMT) in format YYYMMDDHHMMSS.
- Source rate (mg·s^{-1}).
- Stability class (A to F).
- Wind speed (m·s^{-1}).
- Wind angle (° from east).

This looks like:

```
Number of experiments: 44

# Experiment    Date         Source rate  Stability c Wind speed Wind angle

Experiment 7    19560710200000  89900.0          B        4.19    82.0
Experiment 8    19560710230000  91100.0          C        4.85    86.0
Experiment 9    19560711160000  92000.0          C        6.88    66.0
```

The file `exp-data-sim.txt` contains more meteorological data, needed to compute standard deviations with similarity theory.

1.2.2 Station coordinates file

The file station_coord.txt contains the coordinates of all measurement stations. A station is identified first by the arc it belongs to: "ARC050" for the arc which is 50 meters from the source, then "ARC100", "ARC200", "ARC400" and "ARC800". Then, by the station number along this arc. Each arc contains 91 stations, except the 800 m arc which contains 181 points. All station coordinates are listed in the station coordinates file, one station per line. Each line contains the station coordinates (z, then y, then x) in meters. The first 91 correspond to the stations at the 50 meters arc, and so on. This looks like:

```
1.5     4704427.0       535150.0
1.5     4704428.7       535150.0
1.5     4704430.5       535150.1
```

First line corresponds to station "ARC050 1", the second to station "ARC050 2" and so on.

1.3 Creating configuration files

The aim of this part is to run the Prairie Grass experiment of ID 24. Of course, there is a script to launch automatically all 43 simulations, but the aim here is to provide an example.

The directory gaussian_advanced/prairie_grass/config/config-base contains all configuration files, where information specific to each experiment have been replaced by a string. For example, in plume.cfg, the simulation starting date has been replace by %date. It has to be replaced by the date of the experiment.

First, copy all configuration files that are currently in config/config-base in the directory config. Then, replace all data in form %string by the corresponding value for experiment 24. You also have to change the saver file, in order to save concentrations at the list of points corresponding to the stations instead of saving all the domain.

For information about how to modify a saver configuration file to save a list of points, see Polyphemus user's Guide, part **Drivers/Output savers/SaverUnitPoint**.

1.4 Simulation and visualization

Now, run the simulation with Briggs parameterization. Use get_diff_float to compare your results file with the one contained in subdirectory results-briggs and make sure you have obtained the same results.

Note that the size of your file corresponds to $Nt \times Npoint$ where Npoint is the number of points where concentrations have been saved. This corresponds to the total number of stations, that is, 599.

Visualize concentration profiles at different arcs and to compare it to experimental results. To do that, launch IPython and import the modules needed to display the results.

As the concentration field is not a 4D field, getd does not work. You have to use the command fromfile(filename, dtype='type') which allows you to import data contained in a binary file named "filename" into an array. Data type is 'float64' for floating numbers in double precision, 'float32' for float in simple precision, and 'int' for int.

Create an array named "concentration_sim" which holds the concentrations from your simulation. Do the same to create an array "concentration_exp" with the experimental results. Check the shape of your arrays. Note that experimental data are in **double precision**, *so the data type to be used in* fromfile *is "float64".*

```
>> concentration_sim=fromfile("sim-results-pg24.bin", dtype='float32')
>> concentration_sim.shape
(599,)
```

To plot concentrations, use the command plot. To plot concentrations on each arc, you have to use only the corresponding part of the concentration array: plot(concentration_sim[i:j]) where i is the index of the first arc station and j is the index of the first station on next arc. This command will display values of concentration_sim from index i to index j-1.

Display concentrations for the first arc, for simulation and experiments. Add a legend. You should obtain figure **??**.

```
>> plot(concentration_sim[0:91], 'g-')
>> plot(concentration_exp[0:91], 'b+')
>> legend(["simulation", "experiments"], loc = 'upper right')
```

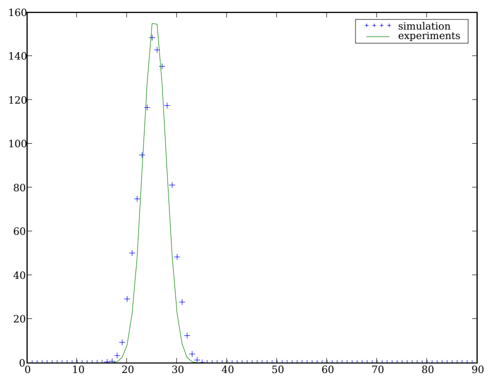


Figure 2: Concentration profile for Prairie Grass experiment 24, 50 meters from source.

Display profiles for the other arcs.

1.5 Changing Sigma Parameterization

There are three parameterizations currently available to compute the standard deviations:

- Briggs parameterization
- Doury parameterization
- parameterization based on similarity theory

Two of them (Briggs and Doury) are based on a discrete description of the boundary layer, and they only need a few meteorological data. The third is a more advanced parameterization based on similarity theory, that is, on a description of the boundary layer with Monin-Obukhov length, friction velocity and boundary layer height. Hence, it can be more precise but it is necessary to have more meteorological input data. In Polyphemus, the file `polyphemus-sessions/Polyphemus/include/models/BriggsFormula.hxx` contains all functions to compute standard deviations in different cases. Hence, if another parameterization has to be added, it can be easily done by adding one or more functions in this file, and by modifying the model (GaussianPlume or GaussianPuff) to call the corresponding function.

Briggs parameterization is based on stability classes. There are two sets of formulae: one for rural environment and the other for urban environment. As this set of formulae has been fitted on Prairie Grass experiment, it gives very good results. Here is an example of the functions you can see in file `BriggsFormula.hxx` to compute σy with Briggs parameterization in rural cases.

```
//! Computes horizontal diffusion parameter over rural area.
/*! It uses Briggs' dispersion parameterizations for open country with
  Pasquill stability classes.
  \param distance distance downwind of the source (m).
  \param stability Pasquill stability classes in [0, 5].
  \return The horizontal plume-dispersion parameter (m).
*/
 template<class T>
T ComputeRuralPlumeHorizontalSigma(T distance, int stability)
{
  if (stability == 0)
    return 0.22 * distance / sqrt(1. + 1.e-4 * distance);
  else if (stability == 1)
    return 0.16 * distance / sqrt(1. + 1.e-4 * distance);
  else if (stability == 2)
    return 0.11 * distance / sqrt(1. + 1.e-4 * distance);
  else if (stability == 3)
    return 0.08 * distance / sqrt(1. + 1.e-4 * distance);
  else if (stability == 4)
    return 0.06 * distance / sqrt(1. + 1.e-4 * distance);
  else if (stability == 5)
    return 0.04 * distance / sqrt(1. + 1.e-4 * distance);
  else
    throw string("Stability index should be in [0, 5], but ")
+ to_str(stability) + " was provided.";
  }
```

Doury parameterization does not even need stability classes. Indeed, there are only two cases, low diffusion and normal diffusion, based on wind speed and whether it is night or day. This parameterization was implemented by CEA for radioactive species. It does not compare very well with Prairie Grass experiment. Here is the function to compute σy with Doury parameterization in cases of normal diffusion.

```
  //! Computes horizontal diffusion parameter for situations of normal diffusion.
  /*! It uses Doury's dispersion parameterizations.
    \param t transfert time from source (s).
    \return The horizontal plume-dispersion parameter (m).
  */
  template<class T>
  T ComputeDouryNormalPlumeHorizontalSigma(T t)
  {
    if (t >= 0. && t < 240.)
      return pow(0.405 * t, 0.859);
    if (t >= 240. && t < 97000.)
      return pow(0.135 * t, 1.130);
    if (t >= 97000. && t < 508000.)
      return pow(0.463 * t, 1.000);
    if (t >= 508000. && t < 1300000.)
      return pow(6.5 * t, 0.824);
    else
      return pow(2.e+05 * t, 0.5);
  }
```

The parameterization based on similarity theory needs the following parameters:

- Boundary layer height (m)
- Friction velocity ($\mathrm{m\,s^{-1}}$)
- Convective velocity ($\mathrm{m\,s^{-1}}$)
- Monin-Obukhov length (m)
- Coriolis parameter ($\mathrm{s^{-1}}$)

These information can either be provided by measurements or extracted from Eulerian meteorological fields. You can read the file `BriggsFormula.cxx` to see how standard deviations are computed.

Now, rerun the experiment with each parameterization. While using similarity theory, the meteorological data file must be `plume-meteo-sim.dat-24`*. It is contained in the directory* *gaussian_advanced/prairie_grass/config/plume-meteo-similarity. Display profiles for different parameterizations. Figure ?? provides an example of what you should obtain for the first arc.*

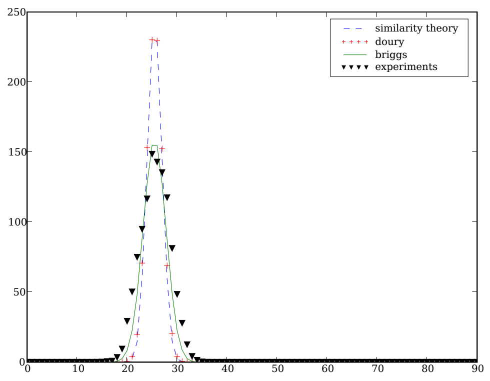


Figure 3: Concentration profile for Prairie Grass experiment 24, 50 meters from source: comparison between different sigma parameterizations.

1.6 Comparison to observations: statistical results

We now want to compare simulations and observations for all 43 experiments. A way that is often used to do so is to compare centrelines, that is, maximum concentrations on each arc. It is divided by the source rate for each experiment:

$$centreline[i,j] = \max_{arcj}(concentration[i])/rate[i]$$

for experiment i and arc j; i ranges from 0 to 43 and j from 0 to 4.

First, you have to load all centrelines into an array of shape $Nexp \times Narc$ where $Nexp$ is the number of experiments, and $Narc$ is the number of arcs. Run the python script write_results.py to do so. It will load simulation results with Briggs formula and experimental results:

```
>> run write_results.py
Experiment 7
Getting simulation and experimental results...
Concentration on arc ARC050
Concentration on arc ARC100
Concentration on arc ARC200
Concentration on arc ARC400
Concentration on arc ARC800
Experiment 8
Getting simulation and experimental results...
Concentration on arc ARC050
Concentration on arc ARC100
Concentration on arc ARC200
Concentration on arc ARC400
Concentration on arc ARC800
...
```

Check that you now have two arrays named `centreline_sim` for simulation results and `centreline_exp` for experimental results:

```
>> centreline_exp.shape
(43, 5)
```

You can now use functions of the atmopy library to compare the two arrays. For example, to have the correlation between the two, use the function `stat.correlation`:

```
>> stat.correlation(centreline_exp, centreline_sim)
0.784504752893
```

Note that if you have only imported atmopy with the command `import atmopy`, you have to type the command:

```
>> atmopy.stat.correlation(centreline_exp, centreline_sim)
0.784502713668
```

You have to type `from atmopy import *` in order not to have to type "atmopy." each time you call an atmopy function.

You can see that there is a very good correlation between the two. There are other useful functions in the module stat: nmse gives the normalized mean square error between observations and experiments, fb gives the fractional bias...
You can open the file `polyphemus-sessions/Polyphemus/include/atmopy/stat/measure.py` to see all available functions and their description.

The function `loglog` is the same as the function `plot` but in logarithmic scale. Use it to draw a scatter plot in logarithmic scale. The following example draws figure **??**. *Points on the red line represent points where simulation and experimental results are equal. Points between the two green lines are within a factor of 2. the command `axis` helps defining x-axis and y-axis limits.*

```
>> loglog(centreline_exp, centreline_sim, 'v')
>> loglog([1e-03,1e04],[1e-03,1e04], 'r-')
>> loglog([1e-03,1e04],[2e-03,2e04], 'g--')
>> loglog([1e-03,1e04],[0.5e-03,0.5e04], 'g--')
>> axis([1e-03,1e03,1e-03,1e04])
```

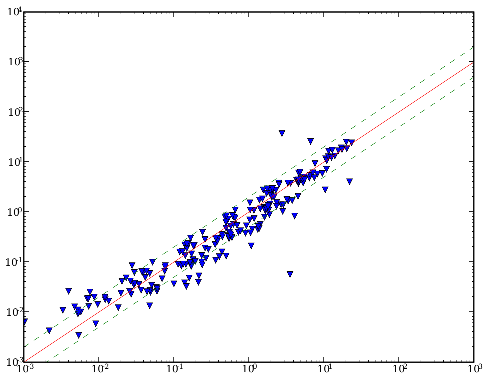


Figure 4: Scatter plot of centrelines for Briggs parameterization.

Now, compare results for other parameterizations. Use various statistical indicators to see which parameterization compares best.

2 Plume-in-Grid Model: application to Chernobyl

2.1 Description

This section provides an example of use of Gaussian models in order to perform better simulations at continental scale. This method is called plume-in-grid. It is based on the fact that point emissions in Eulerian models are assumed to be instantaneously diluted into the cell, which is a quite rude assumption, especially when using coarse grids. Hence, the plume-in-grid method consists in first treating a point emission with a Gaussian model (a puff model in our case) and, when a puff has reached the Eulerian cell size or a given time, in feeding it back to the Eulerian model. Therefore, the puff is much less diluted. In this part, we will see how this applies to the simulation of the Chernobyl case. First, the Chernobyl test case will be performed with the Eulerian model, then with the plume-in-grid model.

2.2 Chernobyl with Eulerian model

2.2.1 Domain

Here is the description of the domain (it is the same domain for the preprocessing and the simulation):

- Along x:
  - minimal value: -10 °
  - step: 1.5 °
  - Number of steps: 49
- Along y:

  - minimal value: 35 °
  - step: 1.5 °
  - Number of steps: 26

- Along z:
  - 9 vertical levels
  - Interfaces of the levels: 0, 64, 236, 484, 796, 1184, 1616, 1984, 2616, 3184.

and a represention of the domain and how it is meshed:

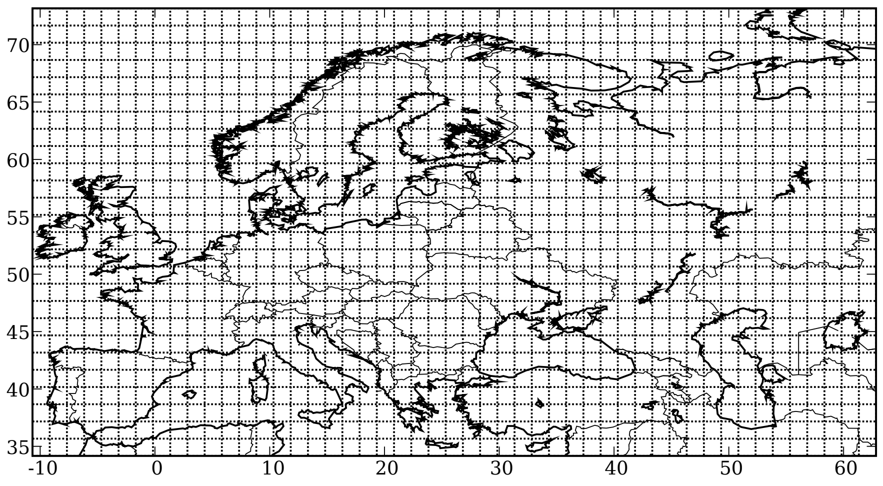


Figure 5: Domain and mesh considered for the simulation.

2.2.2 Preprocessing

To run this simulation, you need meteorological data. The binary files files contained in `~/polyphemus-sessions/gaussian_advanced/plume_in_grid/data/meteo` contain all meteorological data you will need. They have been created during the preprocessing step described in the training session about radionuclides, so you have to refer to this session if you want to know how to create such files. For this session, you do not need to modify them.

There are no initial or boundary conditions, since Caesium 137 is not a species that is naturally present in the atmosphere. There are no surface or volumic emissions either, just a point source representing the Chernobyl emission.

2.2.3 Simulation

Configuration files are located in directory:
~/polyphemus-sessions/gaussian_advanced/plume_in_grid/config.

- chernobyl.cfg, which gives all options for the simulation and the description of the domain;
- chernobyl-data.cfg, which gives the input data;
- chernobyl-saver.cfg, which gives the options to save the results;
- source.dat, which describes the point source of Cs137;
- species.dat, which gives information on the species involved.

For more information about all those files, see Polyphemus user's Guide.

Go in the directory where the configuration files are placed. Run the simulation with polair3d and the main configuration file chernobyl.cfg. Check your results with get_info_float.

To visualize results at continental scale with atmopy, you can use the command getmd in order to display a background map. This command works with a configuration file containing the size of data to be vizualised and the results file name. For more information about it, see Polyphemus guide, part **Postprocessing/Visualizing results/Configuration file disp.cfg**.

Modify the file results/disp.cfg to make it work with your simulation domain. Especially the number of time steps must be equal to the number of time steps when data are saved and not to the total number of time steps of the simulation, and the same is true for vertical levels. Then, use the command dispcf to display concentration maps.

```
>> m,d = getmd("disp.cfg")
>> dispcf(m, d[0,0])
```

This works like getd and contourf: the array d is of size $Nt \times Nz \times Ny \times Nx$. In order to have a colorbar that goes the whole range of concentrations, and not to have white spaces where concentrations are equal to 0., use the option extend=‘‘both’’:

```
>> dispcf(m, d[0,0], extend='both')
```

Visualize several time steps. Figure **??** *gives an example of what you can obtain at time step 220.*

2.3 Chernobyl with plume-in-grid model

The program files for plume-in-grid model are in polyphemus-sessions/Polyphemus/include/models/. They are named PlumeInGrid.cxx and PlumeInGrid.hxx.

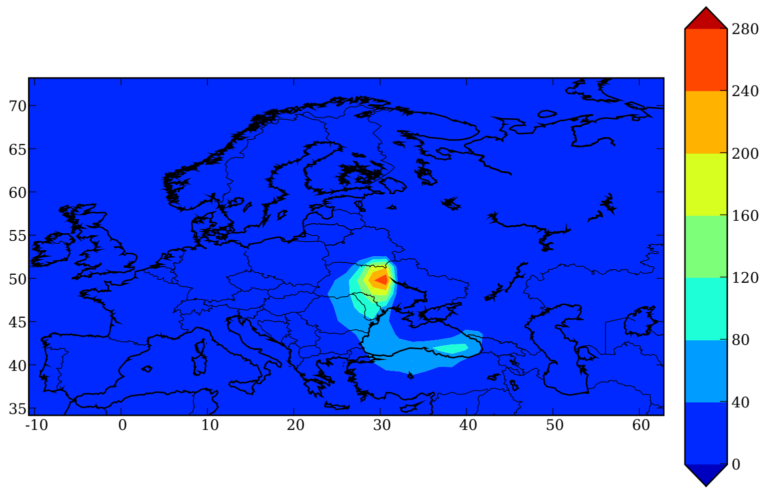


Figure 6: Ground concentration over Europe fo Chernobyl case without plume-in-grid.

2.3.1 Configuration files

To run the program `plume-in-grid` with the Chernobyl case, you basically need the same configuration files as before. The only modifications are:

- in the main configuration file, the option `With_point_emission` has to be set to "no". There also is a new section named `[gaussian]` where the name of a configuration file for the puff model has to be provided (field `file_gaussian`). If you forget to set the option `With_point_emission` to "no", the source file will be read and treated twice: once by the Eulerian model and once by the plume-in-grid model.

- in the data configuration file, there is a new section `[gaussian_meteo]` that provides more meteorological fields, needed by the Gaussian model. It consists in cloudiness and solar radiation data that are used to compute Pasquill stability class. In case those fields are not provided (like here), put whatever value you want for those fields and use the Doury parameterization, so that stability class will not be used. Since the values are not known, only Doury parameterization can be used to compute the standard deviations in the present case.

- The section `[plume-in-grid]` provides the file containing the source to be treated with plume-in-grid model (field `[file_source]`) and additional information about the method to inject the puff in the Eulerian model. For a description of the different options, see Polyphemus user's Guide, part **Models/PlumeInGrid**.

In addition, the Gaussian puff model needs the usual configuration file `puff.cfg`. However, few of their information are actually used, since most information are directly provided by the plume-in-grid model. The time step for the Gaussian model `Delta_t` is still read. If it is larger than the time step for the Eulerian model, it is considered equal to it. Otherwise, a number N of iterations for the Gaussian model are performed at each iteration of the Eulerian model, where:

$$N = int\left[\frac{\Delta t_{Eulerian}}{\Delta t_{Gaussian}}\right] \quad (1)$$

The Gaussian options and parameterizations are also read. However, for now, scavenging and deposition cannot be used with plume-in-grid, so the corresponding options have to be set to "no". Radioactive decay can be used. Also note that, for plume-in-grid model, "With_increasing_sigma" is advised to be set to "yes". It ensures that the puff can only grow and never decrease in size, even when the meteorological situation becomes more stable (during nighttime for example). This option is not necessary when using the Gaussian puff model alone since the meteorological situation is supposed to be homogeneous and constant.

2.3.2 Simulation and visualization

Run the simulation again with the plume-in-grid model (program `plume-in-grid`*). The main configuration file is* `chernobyl-ping.cfg`*. Be careful not to overwrite your previous results.*

Visualize your results and compare them with the previous ones. If you want to display both at the same time, you can create two figures. Here is an example of commands to achieve that, supposing you have a configuration file named `disp-ping.cfg` *for plume-in-grid model results.*

```
>> figure(1)
>> m = getm("disp.cfg")
>> concentration = getd("disp.cfg")
>> figure(2)
>> concentration_ping = getd("disp-ping.cfg")
```

Then, to visualize results at time step i, issue the command:

```
>> figure(1)
>> dispcf(m, concentration[i, 0], extend='both')
>> figure(2)
>> dispcf(m, concentration_ping[i, 0], extend='both')
```

To set the scale of your figure, you have first to create an array with the values of the contours to be displayed.

```
>> V=arange(0, 240, 20)
>> V
array([  0,  20,  40,  60,  80, 100, 120, 140, 160, 180, 200, 220])
```

Now that you have created V, display your map with it. Try to change the values of V several times.

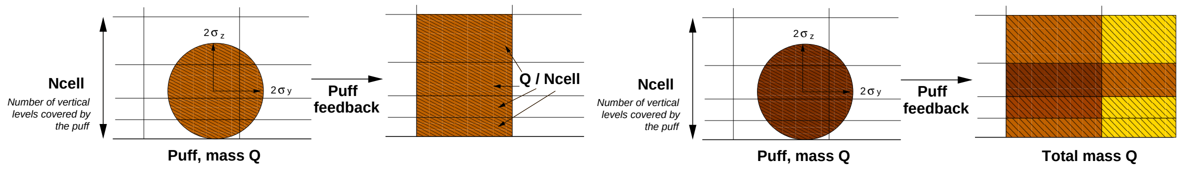
```
>>dispcf(m, d[220, 0], V=V, extend="both")
```

Try to visualize your results with the same scale, in order to see the difference between the two models. To achieve that, you have to create an array of values that goes from the minimum concentration for both models to the maximum concentration for both model, and use it for the values of the contours to be displayed.

2.3.3 Changing injection parameters

The default configuration file `polair3d-data-ping.cfg`, in section `[plume-in-grid]`, specifies that the puffs must be injected after a given time ("With_reinjection_time" is set to "yes"), which is one hour after it was emitted ("Reinjection_time" is equal to 3600 seconds). When "With_reinjection_time" is set to "no", the puff is injected only when it has reached the cell size. However, the mesh used here is very coarse in order to ensure a small computational time with the Eulerian model. Hence, this option is not advised to be used in this case.

The injection method is "column", which injects the puff on one vertical column. Another method is "integrated", which means that the puff is injected in several neighbouring cells around its center location, proportionally to the puff quantity in the cells. Figure **??** illustrates the two methods.



(a) Column (b) Integrated

Figure 7: Injection methods for the puff feedback in the Eulerian model.

Try to change the injection time from one to three hours. You should see that the concentrations are much higher in that case around the source location. Then, try again with an injection time of one hour and the method "integrated". Compare your results to the reference simulation. You can also check the advanced formation session about radionuclides in order to see how you compare with observations in the different cases.

3 Water liquid content diagnosis: Noroxo case

3.1 Description

The aim of this section is to use the water liquid content diagnosis on the Noroxo case. For that purpose, we will use the Eulerian model for local scale (program `polair3d-local.cpp`). This consists in three main steps:

1. Preprocessing. The Eulerian preprocessing consists in generating meteorological and ground data used in the main simulation. The preprocessing specific to water liquid content diagnosis consists in generating volumic emissions of liquid and vapor water.
2. Running simulation. This generates a file containing total water concentration.
3. Post-processing. The program `water_plume.cpp` diagnoses the liquid water content using the simulation results.

The directory `~/polyphemus-sessions/gaussian_advanced/water_plume` contains:

- A subdirectory `config/` which holds all the configuration files used for the preprocessing, simulation and post-processing.
- A subdirectory `data/` which holds files containing data files generated during preprocessing:

  `meteo` contains the meteorological data (the files have already been generated).
  `ground` contains the ground data.
  `emission` contains the water emissions (which have also been already generated).

- A subdirectory `results/` to hold the simulation results.

For the diagnosis you will work in `water_plume`.

3.2 Preprocessing

3.2.1 Ground data

The ground data preprocessing for Eulerian models consists in generating a file containing the land use cover category of the simulation domain. For more details about it, see Polyphemus Guide, part **Preprocessing/Ground data**. As in our case the simulation domain is very small, it corresponds to only one category, and the usual preprocessing program does not work. Use the python program `~/polyphemus-sessions/gaussian_advanced/water_plume/data/ground/luc.py` to create a binary file that corresponds to your simulation domain with one chosen usgs land category.

The command is `python luc.py filename Nluc index` where filename is the name of the file containing the simulation domain, Nluc is the total number of land categories that exist (24 for usgs), and index is the index of land category of you simulation domain. For example, for forest land, it will be 4.

Go in the ground data directory. Generate ground data for your simulation domain. This will generate a file named `LUC.bin`.

```
python luc.py ../../config/general.cfg 24 4
```

3.2.2 Emissions

The preprocessing program to generate emissions is not available in the current version, so the file containing water emissions has been directly provided for this session. This file is named `WaterContentVolumicSource.bin` and placed in subdirectory `data/emission`. It contains volumic emissions of water (liquid and vapor). It must be of size $Nt \times Nz \times Ny \times Nx \times 4.$, where:

- Nt is the number of emission time steps. It corresponds to the total emission duration divided by the emission time step `Delta_t_s` given in `local-emission.cfg`.
- Nz is the number of emission levels `Nz_out` given in `local-emission.cfg`.
- Ny and Nx correspond to the size of the cartesian simulation domain. They are given in section `[domain_cartesian]` of file `local-emission.cfg` and also in section `[domain]` of the main configuration file for simulation.

3.2.3 Meteorological Data

It is already available in `data/meteo/` and does not need to be generated again.

3.3 Simulation

The simulation is an Eulerian simulation for local scale, that is, it uses the model `StationaryModel`. This consists in running the number of time steps given in section `[stationary]` of the configuration file, but with an inner-loop with smaller time steps, given in section `[domain]`, that ensures to find the stationary solution. The species is Water, and the volumic emissions are given by the file `WaterContentVolumicSource.bin`. The program to be used is `../../Polyphemus/driver/polair3d-local`.

Launch the simulation for three hours. The main configuration file to be used is `config/polair3d.cfg`.

The convergence to the stationary solution is insured with the parameters given in `config/polair3d.cfg`:

	`[domain]`
`Nt`	Maximum number of iterations of the underlying model. **This is different from what is normally defined here.**
`Delta_t`	Time-step of the internal loop ("internal time step"). **This is different from what is normally defined here.**
	`[stationary]`
`Nt`	Number of time steps for the simulation.
`Delta_t`	Time-step of the simulation.
	`[convergence]`
`Norm`	Norm used to check convergence: `one`, `two` or `infinity`.
`Method`	Method used to normalize the norm: `mean` or `max`.
`Epsilon`	Convergence criterion (for instance `1.e-4`).

3.4 Liquid water content diagnosis.

The post-processing program water_plume.cpp is located in cmd../../Polyphemus/postprocessing/water_plume
It uses meteorological data and a concentration field of water (liquid and vapor) and diagnoses the proportion of liquid water. It uses a configuration file containing the following information:

- [simulation] contains the simulation data (path to the results, first simulated day and time discretization over each day). The number of time steps that are given correspond to the number of times concentrations have been saved.
- [meteo] contains the meteorological files to be read, and to the file containing water concentrations (the simulation result file).
- [parameters] gives source parameters that are needed in order to do a similarity assumption to compute liquid potential temperature.
- [output] gives the output file name where liquid water content will be written and output options.

For more information about the configuration file, see Polyphemus Guide, part **Postprocessing/Liquid water content diagnosis**.

The values that are given in section [parameters] are:

- source_temperature is the the liquid water potential temperature at the source (in K).
- source_water_content is the total water content (i.e. mass fraction) at the source.

Those two parameters have been computed during preprocessing. Their values at each emission time step have been written in the file results/Plume_BC_values.dat. In this file, each line corresponds to one emission time step. The first value is the mass water fraction, the second is the liquid potential temperature. This looks like:

```
1.442387e-02 2.913831e+02
1.493254e-02 2.942439e+02
1.579670e-02 2.989448e+02
1.732877e-02 3.068351e+02
1.451796e-02 2.919177e+02
1.509825e-02 2.951605e+02
1.609392e-02 3.005181e+02
1.788358e-02 3.095661e+02
```

Choose values corresponding to the last emission time step and provide it in the configuration file.

The water content diagnosis is done at each simulation time step for the whole domain. The domain description is contained in general.cfg. The program is launched with two configuration files, water_plume.cfg and general.cfg, the simulation date and the simulation length. Note that you may have to change the number of vertical levels in general.cfg and to put the number of levels you saved during the simulation.

Launch the liquid water content diagnosis program:

```
../../Polyphemus/postprocessing/water_plume/water_plume
config/general.cfg \
config/water_plume.cfg 20031106 3h
```

The output on screen will be:

```
Reading configuration files... done.
Memory allocation for data fields... done.

Extracting data... done.
Performing diagnoses... done.
Writing data... done.
```

The resulting file, `PlumeLiquidWaterContent.bin`, must be of the same size as the file `Water.bin`. You must erase it if you want to rerun the postprocessing, otherwise your results will be append to the previous file.

Use ipython to visualize the map of total water concentration and the map of liquid water.

3.5 Simulation with Gaussian model.

The aim of this section is to perform the same simulation with Gaussian model. It takes into account a continuous point emission emitting a total water flux. The simulation domain is the same as before. The file `config/plume-meteo.dat` provides all meteorological information that have been extracted at the source location for the Gaussian model.

Copy configuration files for Gaussian plume model in the subdirectory `config`. Change the simulation domain and all data to test the Noroxo case. Note that source data (temperature, speed, surface) can be found in `config/local-emission.cfg` and the option `With_plume_rise` has to be set to no. Run the simulation with program `plume`.