Polyphemus 1.1
User's Guide

CEREA – ENPC / EDF R&D
Meryem Ahmed de Biasi, Vivien Mallet,
Irène Korsakissok, Édouard Debry, Lin Wu

`http://www.enpc.fr/cerea/polyphemus/`
`polyphemus@cerea.enpc.fr`

Contents

Chapter 1

Introduction and Installation

1.1 Polyphemus Overview

Polyphemus [Mallet et al., 2007] is an air-quality modeling system built to manage:

- several scales: local, regional and continental scales;
- many pollutants: from non-reactive species to particulate matter;
- several chemistry-transport models;
- a bunch of advanced methods in data assimilation and ensemble forecasting;
- model integration.

Further details are available in:
Mallet, V., Quélo, D., Sportisse, B., Ahmed de Biasi, M., Debry, É., Korsakissok, I., Wu, L., Roustan, Y., Sartelet, K., Tombette, M., and Foudhil, H. (2007). Technical Note: The air quality modeling system Polyphemus. *Atmospheric Chemistry and Physics Discussions*, 7(3):6,459–6,486
This is the main reference for Polyphemus. Please cite it if you refer to Polyphemus in a publication, a talk or so.

Polyphemus is made of:

- data processing abilities (available in libraries);
- a library for physical parameterizations (library AtmoData);
- programs to compute input data to chemistry-transport models;
- chemistry-transport models;
- drivers, that is, object-oriented codes responsible for driving models in order to perform, for instance, simulations and data assimilation;
- programs to analyze and display output concentrations (primarily based on the library AtmoPy).

Its flowchart is shown in Figure 1.1, in which three steps may be identified: (1) preprocessing (interpolations, physical parameterizations), (2) model computations (possibly with data assimilation or any other method implemented in a driver), (3) postprocessing (comparisons to measurements, statistics, visualization).

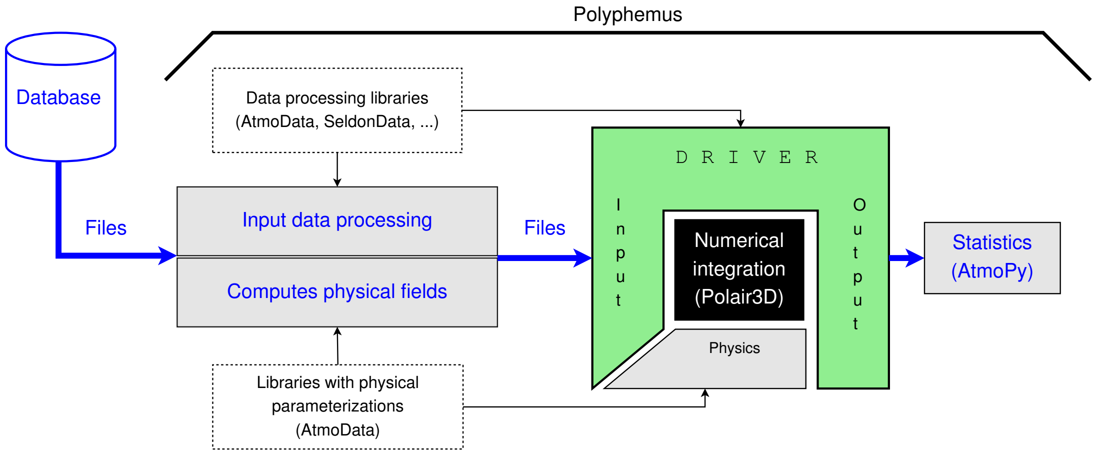


Figure 1.1: Polyphemus flowchart (preprocessing, model computations, postprocessing).

As a consequence, Polyphemus code is organized with the following directories tree:

`preprocessing`

- `bc`: boundary conditions (Mozart 2, Gocart, INCA);
- `bio`: biogenic emissions;
- `dep`: deposition velocities;
- `emissions`: pollutant emissions (EMEP);
- `ground`: ground data (land use cover, roughness);
- `ic`: initial conditions (Mozart 2);
- `meteo`: meteorological data (ECMWF and MM5, including cloud attenuation and vertical diffusion);

`driver`: forward simulations, data assimilation;

- `example`: a series of examples of configuration files for several applications;
- `observation`: observation managers for data assimilation (ground observations and simulated observations);
- `optimization`: optimization algorithms;
- `output_saver`: modules to save the results of a simulation;
- `perturbation`: management of model perturbations (Monte Carlo);

`postprocessing`: comparisons to measurements;

- `water_plume`: liquid water diagnosis in a plume;

`include`

- `Talos`: C++ library to manage configuration files (used everywhere in Polyphemus), dates and string processing;
- `SeldonData`: C++ library to perform data processing (interpolations, input/output operations);

- `AtmoData`: C++ and Fortran library for physical parameterizations and atmospheric data processing;
- `atmopy`: AtmoPy is a Python library for statistical analysis and visualization;
- `common`: functions used to parse and manage the arguments of preprocessing programs;
- `models`: chemistry-transport models to be used by the drivers;
- `modules`
  - `common`: a base module from which transport and chemistry modules derive;
  - `transport`: numerical schemes for advection and diffusion;
  - `chemistry`: chemical mechanisms;
  - `aerosol`: chemical mechanisms for aerosol species;

`utils`: useful tools, mostly to get information on binary files.

Polyphemus is an open source software distributed under the GNU General Public License. It is available at `http://www.enpc.fr/cerea/polyphemus/`. Polyphemus development and support team can be contacted at `polyphemus@cerea.enpc.fr`.

1.2 Requirements

Polyphemus is designed to run under Unix or Linux-based systems. It should be able to run under Windows. AtmoPy has been tested under Windows and the Eulerian model Polair3D has been compiled with Microsoft Visual Studio.NET 2003. There is no obvious reason why other parts of Polyphemus should not work under Windows.

Polyphemus is based on three computer languages: C++, Fortran 77 and Python. There are also a very few lines of C.

Supported C++ compilers are GNU GCC (G++) 3.2, 3.3, 3.4, 4.0 and 4.1. GNU GCC 2.x series is too old to compile Polyphemus. Intel C++ compiler (ICC, versions 7.1 and 8.0) should work. Any other decent C++ compiler (compliant with the standard) should compile Polyphemus. If not, please report to `polyphemus@cerea.enpc.fr`.

Corresponding Fortran compilers are acceptable: GNU G77 3.2, 3.3 and 3.4, GNU GFortran 4.0 and 4.1 (take care: GFortran is rather slow according to our tests), and Intel Fortran compilers IFC 7.1 and IFORT 8.0.

Note for GCC users: Please note that the Fortran compiler has changed between versions 3.x and versions 4.x of GCC. If you use G++ 4.x, please use GFortran to compile Polyphemus Fortran programs, unless you know what you are doing. Similarly, if you use G++ 3.x, please use G77.

Python version 2.3 and higher are supported.

With regard to software requirements, below is a list of possible requirements (depending on the programs to be run):

- the C++ library Blitz++ (`http://www.oonumerics.org/blitz/`): versions 0.6, 0.7, 0.8 and 0.9 are supported. Note that your compiler may exclude a few versions.
- Blas/Lapack (compiled libraries): any recent version.
- NewRan: C++ library for generation of random numbers, from version 2.0.

- NetCDF (compiled libraries and headers): C++ library, any version from series 3.x should work.
- NumPy: any recent version. Make sure that your versions of NumPy and Matplotlib (see below) are compatible.
- Matplotlib: any recent version and corresponding pylab version (usually, pylab is included in Matplotlib package). It is recommended to install the corresponding version of Basemap in order to benefit from AtmoPy map-visualizations. Basemap is a toolkit available on Matplotlib website (`http://matplotlib.sourceforge.net/`), but usually not included in Matplotlib package.
- SciPy: any recent version.

All of them are open source software. Requirements are shown in Table 1.1.

NewRan is not included in Table 1.1 because it is only needed if one performs data assimilation.

Table 1.1: Polyphemus requirements.

	Blitz++	Blas/Lapack	NetCDF	NumPy	Matplotlib	SciPy
`driver`	X	X				
`include`						
`/atmopy`				X	X	X
`preprocessing`						
`/bc`	X		X			
`/bio`	X					
`/dep`	X					
`/emissions`	X					
`/ground`	X					
`/ic`	X		X			
`/meteo`	X					
`postprocessing`				X	X	X
`/water_plume`	X					

1.3 Installation

1.3.1 Main instructions

As soon as libraries and compilers are available, Polyphemus is almost installed. First, extract Polyphemus sources to a given directory. Polyphemus is usually distributed in a `.tar`, `.tgz`, `.tar.gz` or `.tar.bz2` file. These files are extracted with one of these commands:

```
tar -xvf Polyphemus.tar
tar -zxvf Polyphemus.tgz
tar -zxvf Polyphemus.tar.gz
tar -jxvf Polyphemus.tar.bz2
```

Polyphemus programs must be compiled by the user when needed. Makefiles (called `makefile`) are provided so that program compilation should be easy. Depending on your compilers and maybe the paths to the libraries, you might need to slightly modify the makefiles.

Then, one may compile the program `meteo.cpp` in this way:

```
cd Polyphemus/preprocessing/meteo
make meteo
```

Then the program `meteo` is compiled and can be run. Launch `make` in order to compile all programs in a given directory. In directory `driver/`, you may use `scons` instead of `make` if you are familiar with SCons[†1].

1.3.2 AtmoPy

A special step is required with the Python library AtmoPy. This library makes calls to a C++ program in order to parse configuration files. Follow the steps below to have AtmoPy fully installed:

```
cd Polyphemus/include/atmopy/talos
g++ -I../../Talos -o extract_configuration extract_configuration.cpp
```

You may replace g++ with any supported compiler (see Section 1.2).

1.3.3 Newran

The library Newran is required with Kalman algorithms (RRSQRT and ensemble) to generate random numbers. It should be installed in `include/newran/`.

Download Newran from `http://www.robertnz.net/download.html` (or type "Newran" in search engine). At the time these lines are written, Newran 3.0 beta is available at `http://www.robertnz.net/ftp/newran03.tar.gz`. The following commands work with Newran 3.0 beta (released 22 April 2006); there may be slight changes with other versions.

Create directory `include/newran/`, expand Newran in it:

```
mkdir Polyphemus/include/newran
cd Polyphemus/include/newran
wget http://www.robertnz.net/ftp/newran03.tar.gz
tar -zxvf newran03.tar.gz
```

If all is fine, you should have a file called `include/newran/newran.h`. Next, you have to edit `include/newran/include.h` and uncomment the line:

```
//#define use_namespace                    // define name spaces
```

That is, remove the first two slashes:

```
#define use_namespace                      // define name spaces
```

Compile the library (here, with GNU C++ compiler):

```
make -f nr_gnu.mak libnewran.a
```

This should create `include/newran/libnewran.a`. To complete the installation, you have to create a directory where the seed values are stored, for instance:

```
mkdir ~/.newran
cp fm.txt lgm.txt lgm_mix.txt mother.txt mt19937.txt multwc.txt wh.txt ~/.newran/
```

Recall the path to your seed directory since this is an entry of a configuration file (`driver/example/assimilation/perturbation.cfg`).

[†1] http://www.scons.org/

1.3.4 Fortran Subroutines

Linking a Fortran code with C++ may raise a problem when the name of a Fortran subroutine contains an underscore. In this case, the subroutine identifier (in the compiled object) may be named with two underscores at the end, instead of one (as for other Fortran subroutines). This is a strange convention that appears in G77 (but no more in GFortran which replaces G77).

In Polyphemus there are tests to deal with this. Most of the time the tests will succeed. But if you use untested compilers or if you mix compilers from different packages, you may have undefined symbols at link stage, that is, something like:

```
polair3d.o(.gnu.linkonce.t.[...]
In function ‘Polyphemus::DiffusionROS2<double>[...]
: undefined reference to ‘diff_x_’
```

In that case, Polyphemus tests have failed, but there is an easy way to fix that. You need to slightly change the compilation options. In the makefile, add to the compiler flag (`CCFLAGS`) the option: `-DPOLYPHEMUS_SINGLE_UNDERSCORE`. If this does not work, then add *instead* `-DPOLYPHEMUS_DOUBLE_UNDERSCORE`. If you change compilation options, you are advised to use `make cleanall` before compiling so that each part of the code is compiled with the same options.

If this still does not work, it is likely that your problem is elsewhere. Contact Polyphemus development team (`polyphemus@cerea.enpc.fr`) if you need help.

Chapter 2

Using Polyphemus

2.1 Remark

In configurations files, in output logs, and so on, indices start at 0 (as in C++ and Python, not at 1 as in Fortran).

2.2 Configuration Files

2.2.1 Definitions

All Polyphemus programs rely on flexible configuration files. These configuration files define simulation domains, input and output paths, options, etc.

Configurations files are text files, preferably with extension `.cfg`. They primarily contain *fields*, that is, entries associated with *values* provided by the user. In a configuration file, a line usually reads:

```
field = value
```

A practical example is a discretization definition:

```
x_min = 12.5
Delta_x = 0.5
Nx = 100
```

The *fields* `x_min`, `Delta_x` and `Nx` are associated with proper *values* specified by the user.

The characters put between a field and its value are *delimiters*. In the previous example, the delimiters are blank spaces and equal signs. *Delimiters* are discarded characters. They may be put anywhere in a configuration file but they are always ignored. Their aims are to delimit words (i.e., fields and values) and to make the configuration file clearer.

2.2.2 Flexibility

The fields and values can be introduced in many ways. First, many delimiters are supported:

- blank space ( ),
- tabulation (	),
- line break,

- equal sign (=),
- colon (:),
- semicolon (;),
- coma (,), and
- vertical bar (|).

For example,

```
x_min = 12.5
Delta_x = 0.5
Nx = 100
```

is equivalent to

```
x_min 12.5
Delta_x == 0.5
Nx: 100
```

Recall that delimiters can only be used to delimit words, and are discarded otherwise. It means that a field or a value cannot contain a delimiter. The fact that the colon is a delimiter may raise a problem under Windows where drives are called `C:`, `D:`, . . . In the current version of Polyphemus, full paths (that is, with the drive name) should not be used under Windows. If you need a workaround, please contact the Polyphemus teams at `polyphemus@cerea.enpc.fr`.

Fields and values go by pair, but they can be placed anywhere. In particular, several fields may be put on a single line:

```
x_min = 12.5    Delta_x = 0.5    Nx = 100
y_min = -6.2    Delta_y = 1.     Ny = 230
```

The order in which the fields are placed may or may not be important. In most Polyphemus configuration files, the order does not matter. Then

```
x_min = 12.5    Delta_x = 0.5    Nx = 100
y_min = -6.2    Delta_y = 1.     Ny = 230
```

is the same as

```
y_min = -6.2    Delta_y = 1.     Ny = 230
Nx = 100        x_min = 12.5     Delta_x = 0.5
```

Recommandation – Use equal sign '=' between a field and its value if the value is a number and use semi-colon ':' if the value is a string. Example:

```
x_min = 12.5
Output_directory: /home/user/path
```

2.2.3 Comments

Comment lines may be added. They start with '#' or with '%':

```
# Path where results are written.
Output_directory: /home/user/path
```

They may also be put at the end of a line:

```
Output_directory: /home/user/path  # Path where results are written.
```

Recommandation – Prefer '#' for comments, so as to be consistent with Polyphemus default configuration files.

2.2.4 Markups

In order to avoid duplications in a configuration file, Polyphemus features a markup management. A markup is denoted with surrounding '<' and '>', e.g. <`path`>. A markup is automatically replaced with its value whenever it is found. Its value should be provided somewhere in the configuration file with a proper field; for instance, <`path`> refers to the field `path`. Here is a complete example:

```
Root: /home/user
Input_directory: <Root>/input/
Output_directory: <Root>/output/
```

means:

```
Input_directory: /home/user/input/
Output_directory: /home/user/output/
```

The markup can be used before its value is defined:

```
Input_directory: <Root>/input/
Output_directory: <Root>/output/
Root: /home/user # After calls to <Root>. This is legal.
```

Any field may be used as a markup. The user may define any new markup (that is a new field). Moreover, several markup substitutions can be performed in a single value, and nested markups are properly handled:

```
Home: /home/user
Root: <Home>/Polyphemus/work
Number = 7
Input_directory: <Root>/input-<Number>/
```

is accepted and means:

```
Input_directory: /home/user/Polyphemus/work/input-7/
```

Notice that markups may also replace numbers and may be based on preexisting fields:

```
x_min = 12.5    Delta_x = 0.5    Nx = 100
y_min = <x_min> Delta_y = 1.     Ny = <Nx>
```

2.2.5 Sections

Fields and values may be protected inside sections. Assume that two domains are defined, one for input data and another for output data. Instead of:

```
x_min_in = 12.5    Delta_x_in = 0.5    Nx_in = 100
x_min_out = 35.8   Delta_x_out = 0.3   Nx_out = 400
```

one may prefer:

```
[input]
x_min = 12.5    Delta_x = 0.5    Nx = 100

[output]
x_min = 35.8    Delta_x = 0.3    Nx = 400
```

Conflicts are avoided and the syntax is clear. This is why most Polyphemus configuration files have sections.

Sections are enclosed by square brackets ('[' and ']').

Markups are not bound to any section.

Recommandation – Put two blank lines before each section and one blank line after:

```
( blank line )
( blank line )
[input]
( blank line )
x_min = 12.5    Delta_x = 0.5    Nx = 100


[output]

x_min = 12.5    Delta_x = 0.5    Nx = 100
```

2.2.6 Multiple Files

Several Polyphemus programs accept two configuration files as input. Providing two configuration files is then equivalent to providing a single configuration file that would contain all the lines of both files. This is useful to let several programs share a same configuration base. For instance, the simulation domain (whose description is needed by most programs) is usually defined in a configuration file that is provided to every program, in addition to a file dedicated to the specific configuration of the program.

For instance:

```
./emissions general.cfg emissions.cfg 20010506
```

launches the program `emissions` with two configuration files as input: (1) the configuration file `general.cfg` shared with other programs and notably defining the domain description, (2) a specific configuration file, `emissions.cfg`, that includes options for emission generation.

Markups defined in one configuration file can be used in the other file. Note however that each section must be defined in one file only.

2.2.7 Dates

Date formats are:

```
YYYY                 # Year.
YYYY-MM              # With the month.
YYYY-MM-DD           # With the day.
YYYY-MM-DD_HH        # With the hour.
YYYY-MM-DD_HH-II     # With the minute.
YYYY-MM-DD_HH-II-SS  # With the second.
```

Months range from `01` to `12`. Days range from `01` to `31`. Hours range from `00` to `23`. Minutes and seconds range from `00` to `59`.

If the month is not specified (format `YYYY`), then the month is set to `01` (January). If the day is not specified (formats `YYYY` and `YYYY-MM`), it is set to `01` (first day of the month). If the hour, the minute or the second is not specified, it is set to zero (`00`).

Hyphens and underscores may be replaced with any character that is neither a delimiter (see Section 2.2.2) nor a cipher. They can also be removed. Examples:

```
19960413
1996-04-13_20h30
1996/04/13@2030
```

Recommandation – Use hyphens around the month and around minutes. Use an underscore between the day and the hour (`YYYY-MM-DD_HH-II-SS`).

2.2.8 Booleans

Booleans are supported in configuration files and can be specified in any of the following ways:

```
true   t   yes   y
false  f   no    n
```

This is case unsensitive: e.g., `True` or `NO` are valid.

2.3 Running Programs

2.3.1 Compiling Programs

Along with all programs are provided makefiles, in the same directory. Edit these makefiles to change the compiler. Main variables are the C++ compiler `CC`, the Fortran compiler `F77`, the linker `LINK`, and maybe the libraries `LIBS` and the include paths `INCPATH`.

2.3.2 Running a Program

Most programs require one or two input configuration files, and sometimes one or two dates (beginning and end dates, see Section 3.2.3). Most programs provide help when launched without any input file. Here is an example with the program `bio`[†1]:

[†1] Further details about specific programs are provided in chapter 3.

```
~/Polyphemus/preprocessing/bio/> ./bio

Usage:
  ./bio [main configuration file] [secondary config file] [first date] [second date/interval
  ./bio [main configuration file] [first date] [second date/interval]
  ./bio [main configuration file] [secondary config file] [first date]
  ./bio [first date] [second date/interval]
  ./bio [first date]

Arguments:
  [main configuration file] (optional): main configuration file. Default: bio.cfg
  [secondary configuration file] (optional): secondary configuration file. Default: "".
  [first date]: beginning date in any valid format.
  [second date]: end date in any valid format.
  [interval] (optional): Interval in format NdMh or Nd-Mh or Nd or Mh where N
  is the number of days and M the number of hours. Default: 1d.

Note:
The end date, whether it is given directly or computed by adding the time
interval to the beginning date, is always considered as excluded.
```

Program `bio` takes from one to four arguments. Below are four possible calls:

```
./bio 2001-04-22
./bio bio.cfg 2001-04-22
./bio bio.cfg 2001-01-22 2001-04-23
./bio ../general.cfg bio.cfg 2001-04-22
```

The first three calls are equivalent. The fourth one involves two configuration files. The program `bio` behaves as if these two configuration files were merged. It means that the fields required by the program may be put in any of these two files. Markups defined in one file can be expanded in the other file. The only constraint is that each section should appear in a single file only.

2.3.3 Sharing Configuration

The command line:

```
./bio ../general.cfg bio.cfg 2001-04-22 1d
```

with the two configuration files `general.cfg` and `bio.cfg`, is the advocated line. The configuration file `general.cfg` gathers information that may be needed by several programs in the `preprocessing` directory (`meteo`, `attenuation`, `luc-usgs`, etc.). Such a configuration file is provided with `Polyphemus/preprocessing/general.cfg`:

```
[general]

Home: /u/cergrene/0/bordas
Directory_computed_fields: <Home>/B/data
Directory_ground_data: <Directory_computed_fields>/ground
Programs: <Home>/codes/Polyphemus-HEAD
```

```
[domain]

Date: 2001-01-02_00-00-00
Delta_t = 3.0
x_min = -10.0   Delta_x = 0.5   Nx = 65
y_min = 40.5    Delta_y = 0.5   Ny = 33
Nz = 5
Vertical_levels: <Programs>/levels.dat
```

The simulation domain and the simulation dates are defined. In addition, markups (`Directory_computed_fields`, `Directory_ground_data` and `Programs`) are introduced and may be referred by other configuration files such as `meteo.cfg`.

Actually most configuration files (`meteo.cfg`, `luc-usgs.cfg`, `emissions.cfg`, etc.) provided in Polyphemus, along with the programs, are examples that refer to the markups defined in `general.cfg`. Essentially three markups are defined in `general.cfg`:

- `Directory_computed_fields`: where output results (i.e., fields computed by preprocessing programs) are stored.
- `Directory_ground_data`: where ground data (land use cover, roughness) is stored.
- `Programs`: path to Polyphemus preprocessing directory.

Polyphemus configuration files are written so that mainly changes in `general.cfg` should be needed to perform a reference simulation. In `general.cfg`, one changes the paths (markups) to the preprocessing programs (`Programs`) and to the output results (`Directory_computed_fields` and `Directory_ground_data`), and one chooses its simulation domain. Other configuration files provide paths to input data (meteorological files, emissions data, etc.) and fine options.

2.3.4 Notes about Models

To launch a simulation you have to compile and execute a C++ program which differs from preprocessing programs. After preprocessing steps, the simulation is made of a driver (on top of the model itself), a model and its modules (if any). See Section 1.1 for a short description of the flowchart. The program of the simulation looks like `driver/polair3d.cpp`: it is a short C++ code that declares the driver, the model and the modules.

You may have to modify this program in case you change the model, the driver or a module. In that case, duplicate `driver/polair3d.cpp` (or another example) and modify it according to the notes below. Actually it is likely that the model/driver combination is already in use in one of the examples: have a look in `driver/*.cpp`.

First determine which model you need, depending on your simulation target:

- for a passive simulation: `Polair3DTransport` or `CastorTransport`;
- for a simulation with chemistry for gaseous species: `Polair3DChemistry` or `CastorChemistry`;
- for a simulation with aerosol species: `Polair3DAerosol`;
- for a simulation with gaseous species and data assimilation: `Polair3DChemistryAssimConc`;
- for a simulation at local scale using an Eulerian model: one of Polair3D models with driver `StationaryDriver`;

- for a simulation with a plume Gaussian model: `GaussianPlume`, or `GaussianPlume_aer` if there are aerosol species;
- for a non-stationary simulation at local scale with a Gaussian model: `GaussianPuff`, or `GaussianPuff_aer` if there are aerosol species;
- for a simulation with point sources, you can use the driver `PlumeInGridDriver` as a model, in order to improve the way the dispersion of the pollutants inside a cell is modelled.

To set the model, just modify the definition of `ClassModel`:

```
typedef MyModel<Argument(s)> ClassModel;
```

For instance:

```
typedef Polair3DAerosol<real, AdvectionDST3<real>,
    DiffusionROS2<real>, Decay<real> > ClassModel;
```

If you change a model, you may also change the modules (a model may need less modules or no module at all: remove them is necessary). The modules are all (template) arguments of the model (`AdvectionDST3<real>`, `DiffusionROS2<real>` and `Decay<real>`, in the previous example), except for `real` that should not be changed. The order in which the modules are provided matters: it is always advection, diffusion and chemistry, or transport (single module) and chemistry. See Section 2.5.5 for the modules you can use with the model you chose.

Then, in your main C++ program, declare the right driver. You may replace `BaseDriver` with a new driver at this line (in `driver/polair3d.cpp`):

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

Finally make sure to include all models, modules, drivers and output savers you use (at the beginning of the file – statements `#include "...cxx"`). The makefile may need changes too if the module uses Fortran functions. In particular, chemistry modules `ChemistryRADM` and `ChemistryRACM` need Fortran routines, make sure that they are included in `SRC77`:

- for `ChemistryRACM`: LU_decompose.f, LU_solve.f, angzen.edf.f, chem.f, dratedc.f, fexchem.f, jacdchemdc.f, kinetic.f, rates.f, roschem.f and solvlin.f (in directory `include/modules/chemistry/ChemistryRACM`).
- for `ChemistryRADM`: LU_decompose.f, LU_solve.f, angzen.edf.f, chem.f, fexchem.f, jacdchemdc.f, kinetic.f, roschem.f and solvlin.f (in directory `include/modules/chemistry/ChemistryRADM`).

Aerosol module `AerosolRACM_SIREAM` also needs Fortran routines, but a specific makefile is provided for simulations using this module.

If you are not confident with your own changes, have a look at the examples: it is likely that you find a close combination there. In case you try an unusual combination, you may contact `polyphemus@cerea.enpc.fr`.

The directory named `driver/example` provides examples of configuration and data files to use with the programs. These examples should be launched *in* directory `driver`. Their outputs will then be stored in `driver/results`, so make sure that this directory exists **before** you start the simulation (indeed Polyphemus programs do not create directories before saving results).

2.4 Useful Tools

A few useful tools are provided in directory `Polyphemus/utils`. Here is a brief explanation of their aim and their use.

2.4.1 Information about Binary Files

Two programs provided in `Polyphemus/utils` are meant to provide information about the content of one or several binary files. It is highly recommended to use these programs to check the output files of preprocessing programs and drivers/models (e.g. in Section 2.5.6).

These two programs perform the same thing but on binary files with different floating precision:

- `get_info_float` gives the minimum, maximum and mean of binary files in single precision.
- `get_info_double` gives the minimum, maximum and mean of binary files in double precision.

It is assumed that the binary file to be analyzed by `get_info_float` or `get_info_double` contains only floating point numbers. No extra data such as headers should be in the file. Output binary files from preprocessing programs and from drivers/models satisfy this condition and can be properly read by `get_info_float` or `get_info_double`. Note that Polyphemus programs usually generate single precision files: it is very likely that one only uses `get_info_float`.

Using `get_info_float` or `get_info_double` is straightforward:

```
$ get_info_float Temperature.bin

Minimum: 257.621
Maximum: 300.882
Mean: 282.262

$ get_info_float Temperature.bin Pressure.bin

-- File "Temperature.bin"
Minimum: 257.621
Maximum: 300.882
Mean: 282.262

-- File "Pressure.bin"
Minimum: 56369.2
Maximum: 102496
Mean: 87544.1
```

2.4.2 Differences between Two Binary Files

There are two different types of programs to compute statistics about the differences between two files:

- `get_diff_precision` where `precision` is `float` or `double`. They return statistics about the difference between two files. As for `get_info_precision`, the files should only contain floating point numbers.

- `get_partial_diff_precision` where `precision` is `float` or `double`. They return statistics about the difference between two files. If these two files have the same size, `get_partial_diff_precision` does the same as `get_diff_precision`. If the files do not have the same size, only the first values (as much as possible) are compared.

Here is an example with `get_diff_float`:

```
~Polyphemus/driver/results> ../../utils/get_diff_float O3.bin O3-other.bin

                File #0         File #1
Minima:         0.0145559       0.0181665
Maxima:         136.795         175.123
Means:          71.578          65.4088
Standard dev.:  26.958          28.643

                Difference
Minimum:        -57.324
Maximum:        66.9219
Mean:           6.16919
Standard dev.:  14.4999

Correlation between files #0 and #1: 0.865696
```

2.4.3 MM5 Files

It can be useful to get information from MM5 files, in particular to modify the configuration file `MM5-meteo.cfg` (see Section 3.4.5). To do so, two programs are provided:

- `MM5_var_list` gives a list of all variables stored in a MM5 file. It also gives miscellaneous information about the file. Information provided can be needed in preprocessing step (program `MM5-meteo` – Section 3.4.5): number of space steps, time step and projection type.

- `get_info_MM5` gives the minimum, maximum, mean and standard deviation of a variable stored in a MM5 file (use program `MM5_var_list` to know what variables are stored in the file).

For instance, the output of `MM5_var_list` for the file `MM5-2004-08-09` used in the Eulerian test case (see Section A) is:

```
Metadata (-999 means unknown):

OUTPUT FROM PROGRAM MM5 V3 : 11
TERRAIN VERSION 3 MM5 SYSTEM FORMAT EDITION NUMBER : 1
TERRAIN PROGRAM VERSION NUMBER : 6
TERRAIN PROGRAM MINOR REVISION NUMBER : 0
COARSE DOMAIN GRID DIMENSION IN I (N-S) DIRECTION : 76
COARSE DOMAIN GRID DIMENSION IN J (E-W) DIRECTION : 86
MAP PROJECTION. 1: LAMBERT CONFORMAL, 2: POLAR STEREOGRAPHIC, 3: MERCATOR : 1
IS COARSE DOMAIN EXPANDED?, 1: YES, 0: NO : 0
EXPANDED COARSE DOMAIN GRID DIMENSION IN I DIRECTION : 76
EXPANDED COARSE DOMAIN GRID DIMENSION IN J DIRECTION : 86
```

```
GRID OFFSET IN I DIR DUE TO COARSE GRID EXPANSION : 0
GRID OFFSET IN J DIR DUE TO COARSE GRID EXPANSION : 0
DOMAIN ID : 1
MOTHER DOMAIN ID : 1
NEST LEVEL (0: COARSE MESH) : 0
DOMAIN GRID DIMENSION IN I DIRECTION : 76
DOMAIN GRID DIMENSION IN J DIRECTION : 86
I LOCATION IN THE MOTHER DOMAIN OF THE DOMAIN POINT (1,1) : 1
J LOCATION IN THE MOTHER DOMAIN OF THE DOMAIN POINT (1,1) : 1
DOMAIN GRID SIZE RATIO WITH RESPECT TO COARSE DOMAIN : 1
                                                      : 1
REGRID Version 3 MM5 System Format Edition Number : 2
REGRID Program Version Number : 16
REGRID Program Minor Revision Number : 1
COARSE DOMAIN GRID DISTANCE (m) : 36000
COARSE DOMAIN CENTER LATITUDE (degree) : 47
COARSE DOMAIN CENTER LONGITUDE (degree) : 6
CONE FACTOR : 0.715567
TRUE LATITUDE 1 (degree) : 60
TRUE LATITUDE 2 (degree) : 30
POLE POSITION IN DEGREE LATITUDE : 90
APPROX EXPANSION (m) : 360000
GRID DISTANCE (m) OF THIS DOMAIN : 36000
I LOCATION IN THE COARSE DOMAIN OF THE DOMAIN POINT (1,1) : 1
J LOCATION IN THE COARSE DOMAIN OF THE DOMAIN POINT (1,1) : 1
I LOCATION IN THE COARSE DOMAIN OF THE DOMAIN POINT (IX,JX) : 76
J LOCATION IN THE COARSE DOMAIN OF THE DOMAIN POINT (IX,JX) : 86
TERRAIN DATA RESOLUTION (in degree) : 0.0833333
LANDUSE DATA RESOLUTION (in degree) : 0.0833333

MM5 Version 3 MM5 System Format Edition Number : 1
MM5 Program Version Number : 6
MM5 Program Minor Revision Number : 1
FOUR-DIGIT YEAR OF START TIME : 2004
INTEGER MONTH OF START TIME : 8
DAY OF THE MONTH OF THE START TIME : 9
HOUR OF THE START TIME : 0
MINUTES OF THE START TIME : 0
SECONDS OF THE START TIME : 0
TEN THOUSANDTHS OF A SECOND OF THE START TIME : 0
MKX: NUMBER OF LAYERS IN MM5 OUTPUT : 25
TIMAX:   SIMULATION END TIME (MINUTES) : 5760
TISTEP:  COARSE-DOMAIN TIME STEP IN SECONDS : 100
TAPFRQ:  TIME INTERVAL (MINUTES) THAT DATA WERE SAVED FOR GRIN : 60

Outputs:

Name      Dim. 1   2   3   4   Stag. Ord. Units            Description
U         3    76  86  25      D     YXS  m/s              U COMPONENT OF
```

```
                                                           HORIZONTAL WIND
V         3    76  86  25      D     YXS  m/s              V COMPONENT OF
                                                           HORIZONTAL WIND
T         3    76  86  25      C     YXS  K                TEMPERATURE
Q         3    76  86  25      C     YXS  kg/kg            MIXING RATIO
CLW       3    76  86  25      C     YXS  kg/kg            CLOUD WATER MIXING RATIO
RNW       3    76  86  25      C     YXS  kg/kg            RAIN WATER MIXING RATIO
ICE       3    76  86  25      C     YXS  kg/kg            CLOUD ICE MIXING RATIO
SNOW      3    76  86  25      C     YXS  kg/kg            SNOW MIXING RATIO
GRAUPEL   3    76  86  25      C     YXS  kg/kg            GRAUPEL MIXING RATIO
RAD TEND  3    76  86  25      C     YXS  K/DAY            ATMOSPHERIC RADIATION TENDENCY
W         3    76  86  26      C     YXW  m/s              VERTICAL WIND COMPONENT
PP        3    76  86  25      C     YXS  Pa               PRESSURE PERTURBATION
PSTARCRS  2    76  86          C     YX   Pa               (REFERENCE) SURFACE PRESSURE
                                                           MINUS PTOP
GROUND T  2    76  86          C     YX   K                GROUND TEMPERATURE
RAIN CON  2    76  86          C     YX   cm               ACCUMULATED CONVECTIVE
                                                           PRECIPITATION
RAIN NON  2    76  86          C     YX   cm               ACCUMULATED NONCONVECTIVE
                                                           PRECIPITATION
TERRAIN   2    76  86          C     YX   m                TERRAIN ELEVATION
MAPFACCR  2    76  86          C     YX   (DIMENSIONLESS)  MAP SCALE FACTOR
MAPFACDT  2    76  86          D     YX   (DIMENSIONLESS)  MAP SCALE FACTOR
CORIOLIS  2    76  86          D     YX   1/s              CORIOLIS PARAMETER
RES TEMP  2    76  86          C     YX   K                INFINITE RESERVOIR SLAB
                                                           TEMPERATURE
LATITCRS  2    76  86          C     YX   DEGREES          LATITUDE (SOUTH NEGATIVE)
LONGICRS  2    76  86          C     YX   DEGREES          LONGITUDE (WEST NEGATIVE)
LAND USE  2    76  86          C     YX   category         LANDUSE CATEGORY
TSEASFC   2    76  86          C     YX   K                SEA SURFACE TEMPERATURE
PBL HGT   2    76  86          C     YX   m                PBL HEIGHT
REGIME    2    76  86          C     YX   (DIMENSIONLESS)  PBL REGIME
SHFLUX    2    76  86          C     YX   W/m^2            SENSIBLE HEAT FLUX
LHFLUX    2    76  86          C     YX   W/m^2            LATENT HEAT FLUX
UST       2    76  86          C     YX   m/s              FRICTIONAL VELOCITY
SWDOWN    2    76  86          C     YX   W/m^2            SURFACE DOWNWARD SHORTWAVE
                                                           RADIATION
LWDOWN    2    76  86          C     YX   W/m^2            SURFACE DOWNWARD LONGWAVE
                                                           RADIATION
SWOUT     2    76  86          C     YX   W/m^2            TOP OUTGOING SHORTWAVE
                                                           RADIATION
LWOUT     2    76  86          C     YX   W/m^2            TOP OUTGOING LONGWAVE
                                                           RADIATION
SOIL T 1  2    76  86          C     YX   K                SOIL TEMPERATURE IN LAYER  1
SOIL T 2  2    76  86          C     YX   K                SOIL TEMPERATURE IN LAYER  2
SOIL T 3  2    76  86          C     YX   K                SOIL TEMPERATURE IN LAYER  3
SOIL T 4  2    76  86          C     YX   K                SOIL TEMPERATURE IN LAYER  4
SOIL M 1  2    76  86          C     YX   m^3/m^3          TOTAL SOIL MOIS IN LYR  1  4
SOIL M 2  2    76  86          C     YX   m^3/m^3          TOTAL SOIL MOIS IN LYR  2  4
SOIL M 3  2    76  86          C     YX   m^3/m^3          TOTAL SOIL MOIS IN LYR  3  4
SOIL M 4  2    76  86          C     YX   m^3/m^3          TOTAL SOIL MOIS IN LYR  4  4
SOIL W 1  2    76  86          C     YX   m^3/m^3          SOIL LQD WATER IN LYR  1   4
SOIL W 2  2    76  86          C     YX   m^3/m^3          SOIL LQD WATER IN LYR  2   4
SOIL W 3  2    76  86          C     YX   m^3/m^3          SOIL LQD WATER IN LYR  3   4
SOIL W 4  2    76  86          C     YX   m^3/m^3          SOIL LQD WATER IN LYR  4   4
CANOPYM   2    76  86          C     YX   m                CANOPY MOISTUR E CONTENT
WEASD     2    76  86          C     YX   mm               WATER EQUIVALENT SNOW DEPTH
SNOWH     2    76  86          C     YX   m                PHYSICAL SNOW DEPTH
SNOWCOVR  2    76  86          C     YX   fraction         FRACTIONAL SNOW COVER
ALB       2    76  86          C     YX   fraction         ALBEDO
GRNFLX    2    76  86          C     YX   W m{-2}          GROUND HEAT FLUX
VEGFRC    2    76  86          C     YX   fraction         VEGETATION COVERAGE
SEAICE    2    76  86          C     YX   (DIMENSIONLESS)  SEA ICE FLAG
SFCRNOFF  2    76  86          C     YX   mm               SURFACE RUNOFF
UGDRNOFF  2    76  86          C     YX   mm               UNDERGROUND RUNOFF
T2        2    76  86          C     YX   K                2-meter Temperature
Q2        2    76  86          C     YX   kg kg{-1}        2-meter Mixing Ratio
U10       2    76  86          C     YX   m s{-1}          10-meter U Component
V10       2    76  86          C     YX   m s{-1}          10-meter V Component
ALBD      2    27  2                 CA   PERCENT          SURFACE ALBEDO
SLMO      2    27  2                 CA   fraction         SURFACE MOISTURE AVAILABILITY
SFEM      2    27  2                 CA   fraction         SURFACE EMISSIVITY AT 9 um
SFZ0      2    27  2                 CA   cm               SURFACE ROUGHNESS LENGTH
THERIN    2    27  2                 CA   100*cal cm^-2
                                          K^-1 s^1/2       SURFACE THERMAL INERTIA
SFHC      2    27  2                 CA   J m^-3 K^-1      SOIL HEAT CAPACITY
SCFX      1    27                    CA   fraction         SNOW COVER EFFECT
SIGMAH    1    25              H     S    sigma            VERTICAL COORDINATE

Total number of time steps read in the file: 97
```

For each variable is provided:

- its name;
- its number of dimensions;
- its length along dimension 1 (if applicable);
- its length along dimension 2 (if applicable);
- its length along dimension 3 (if applicable);
- its length along dimension 4 (if applicable);
- the position at which the variable is given (`Stag.`): dot points (`D`, corner of the grid squares) or cross points (`C`, center of the grid squares);
- its dimensions ordering;
- its unit (or `(DIMENSIONLESS)`);

- a short description.

Then you can use the program `get_info_MM5` to have statistical data about one of the variables only. Note that some variables have a blank space in their name so in that case you need to put the name between quotes to use `get_info_MM5`. If the name has no blank spaces, quotes are not necessary but can be used.

```
~/TestCase/raw_data/MM5> get_info_MM5 MM5-2004-08-09 'GROUND T'

Min: 271.911
Max: 327.747
Mean: 294.112
Std. dev.: 6.46779

~/TestCase/raw_data/MM5> get_info_MM5 MM5-2004-08-09 ALB

Min: 0.0738
Max: 0.8
Mean: 0.122658
Std. dev.: 0.0501975
```

Note that `get_info_MM5` only gives information on *one* field stored in the MM5 file.

2.4.4 Script `call_dates`

The script `call_dates` allows to call a program (in particular for preprocessing) over several consecutive days. Launch it without arguments to get help:

```
~/Polyphemus/utils> ./call_dates

Script "call_dates" calls a program over a range of dates.
Usage:
  "call_dates" [program] {arguments} [first date] [second date / number of days]
Arguments:
  [program]: program to be launched over the range of dates.
  {arguments}: arguments. Any occurence of %D is replaced with the date;
               otherwise the date is assumed to be the last argument.
  [first date]: first date of the range of dates.
  [second date / number of days]: last date of the range of dates
                                  or number of days of this range.
```

Below is an example:

```
~/Polyphemus/utils> call_dates echo "Current date:" 20060720 20060722

------------------------------------------------------------------------------
nice time echo Current date: 20060720
Current date: 20060720
0.00user 0.00system 0:00.00elapsed 0%CPU (0avgtext+0avgdata 0maxresident)k
0inputs+0outputs (0major+176minor)pagefaults 0swaps
```

```
-----------------------------------------------------------------------------
nice time echo Current date: 20060721
Current date: 20060721
0.00user 0.00system 0:00.00elapsed 0%CPU (0avgtext+0avgdata 0maxresident)k
0inputs+0outputs (0major+177minor)pagefaults 0swaps
-----------------------------------------------------------------------------
nice time echo Current date: 20060722
Current date: 20060722
0.00user 0.00system 0:00.00elapsed 200%CPU (0avgtext+0avgdata 0maxresident)k
0inputs+0outputs (0major+177minor)pagefaults 0swaps
-----------------------------------------------------------------------------
```

For each day, the command that is launched is shown (note that `nice time` has been prepended) and its output is displayed below.

2.4.5 Other Utilities

Many other utilities (to be compiled with `make` – `utils/makefile` includes all programs) are provided to manipulate binary and text files. Below are short descriptions of these utilities:

- `add_float`: adds two single-precision binary files;
- `add_nb_float`: adds a number to a single-precision binary file;
- `double_to_float`: converts a double-precision binary file to a single-precision binary file;
- `float_to_text`: converts a single-precision binary file to a text file;
- `mult_nb_float`: multiplies a single-precision binary file by a given number;
- `reverse`: switch from big-endian to little-endian, or vice versa;
- `subtraction_float`: subtracts two single-precision binary files;
- `text_to_float`: converts a text file to a single-precision binary file.

To get further help on a program, launch this program without arguments. It will print help on screen.

2.5 Setting Up a Simulation

This section is a quick overview of how a simulation should be set up. It is not meant to and cannot replace the chapters about preprocessing, models, modules, . . .

2.5.1 Suggested Directory Tree

It is advocated not to modify Polyphemus code, including the configuration files provided with it. The whole Polyphemus directory should not be modified (except maybe makefiles). Copy the configuration files you need in a dedicated directory, modify the new configuration files in this directory, and run Polyphemus programs from this directory. Your directory tree may look like:

```
Polyphemus-{version}/driver/
                    /include/
                    /postprocessing/
                    /preprocessing/
                    /utils/
MyStudy/configuration/
       /data/emissions/
            /meteo/
            /[...]
       /results/reference/
               /new_emissions/
               /[...]
```

where `MyStudy` contains Polyphemus configurations files set for the study (`configuration` with `general.cfg`, `meteo.cfg`, ... in it), data generated by preprocessing programs (directory `data`), and output results from the chemistry-transport model (`results`, with results from different runs).

Notice that Polyphemus directory includes the version number (or the date). This is very useful in order to properly track simulations. In directory `MyStudy`, you should add a file called `version` which should contain Polyphemus version (and maybe the version of other tools).

You may also want to copy configuration files in your output directory. For instance, you may copy `meteo.cfg` in directory `MyStudy/data/meteo/` so as to know with which configuration your meteorological data were generated.

2.5.2 Roadmaps

Roadmaps with "Polair3D" Models

In short, the main steps to set up an Eulerian simulation with model `Polair3D` are:

1. generation of ground data (land use cover, roughness height) – `preprocessing/ground`;
2. preprocessing of meteorological fields – `preprocessing/meteo`;
3. other preprocessing steps if relevant (deposition velocities, emissions, ...);
4. compiling the right combination of model, module(s) and driver (see Sections 2.5.5 and 2.3.4).

Passive tracer Below is a possible sequence of programs to be launched to perform a basic passive simulation:

```
preprocessing/ground/luc-glcf
preprocessing/ground/roughness
preprocessing/meteo/MM5-meteo
preprocessing/meteo/Kz_TM
driver/polair3d-transport
```

Program `polair3d-transport` is not provided with Polyphemus. It should be built with Polyphemus components: `BaseDriver` (driver), `Polair3DTransport` (model), `AdvectionDST3` (module), `DiffusionROS2` (module). See Section 2.3.4 for details.

Photochemistry Below is a possible sequence of programs to be launched to perform a photochemistry simulation:

```
preprocessing/ground/luc-glcf
preprocessing/ground/roughness
preprocessing/meteo/MM5-meteo
preprocessing/meteo/Kz_TM
preprocessing/emissions/emissions
preprocessing/bio/bio
preprocessing/dep/dep
preprocessing/ic/ic
preprocessing/bc/bc
driver/polair3d
```

Roadmap with “Castor” Models

The roadmap with “Castor” models is very similar to the one with “Polair3D” models except that raw data and preprocessing programs to modify them are often different.

Photochemistry Below is a possible sequence of programs to be launched to perform a photochemistry simulation:

```
preprocessing/ground/ground-castor
preprocessing/meteo/MM5-meteo-castor
preprocessing/emissions/chimere_to_castor
preprocessing/bio/bio-castor
preprocessing/dep/dep-emberson
preprocessing/ic/ic
preprocessing/bc/bc-inca
driver/castor
```

Roadmaps with Gaussian Models

In short, the main steps to set up a Gaussian simulation are:

1. generation of meteorological data: no program is available to do it, but as only little information is required this should be quite easy. Examples of meteorological files are provided in `driver/example/gaussian/gaussian-meteo.dat` and `driver/example/gaussian/gaussian-meteo_aer.dat`.
2. preprocessing: `discretization` to generate source files for line emission and `gaussian-deposition` or `gaussian-deposition_aer` to compute deposition velocities and scavenging coefficients(without or with aerosol species respectively). For more details, see Section 3.9
3. compiling the right combination of model (`GaussianPlume`, `GaussianPlume_aer`, `GaussianPuff`, `GaussianPuff_aer`) and driver (`PlumeDriver` or `PuffDriver`).

2.5.3 Mandatory Data in Preprocessing

ECMWF Fields

In ECMWF files, it is recommended to have the following fields (with their Grib codes):

- Volumetric soil water layer 1 (39),
- Volumetric soil water layer 2 (40),
- Volumetric soil water layer 3 (41),
- Volumetric soil water layer 4 (42),
- Temperature [3D] (130),
- U velocity [3D] (131),
- V velocity [3D] (132),
- Specific humidity [3D] (133),
- Snow depth (141),
- Stratiform precipitation (Large-scale precipitation) [accumulated] (142),
- Convective precipitation [accumulated] (143),
- Snowfall (convective + stratiform) [accumulated] (144),
- Surface sensible heat flux [accumulated] (146),
- Surface latent heat flux [accumulated] (147),
- Logarithm of surface pressure (152),
- Boundary layer height (159),
- Total cloud cover (164),
- 2 meter temperature (167),
- Surface solar radiation downwards [accumulated] (169),
- Surface solar radiation [accumulated] (176),
- East-West surface stress [accumulated] (180),
- North-South surface stress [accumulated] (181),
- Evaporation [accumulated] (182),
- Low cloud cover (186),
- Medium cloud cover (187),
- High cloud cover (188),
- Skin temperature (235),
- Forecast albedo (243),
- Cloud liquid water content [3D] (246),
- Cloud ice water content [3D] (247),
- Cloud cover [3D] (248).

Not all data may be required, depending on the programs you actually run.

2.5.4 Mandatory Data for Models

The table below presents all variables needed by various models (and the name under which they appear in the data configuration files). Note that additional data can be necessary to add initial conditions, boundary conditions, source terms (volume emissions, surface emissions) or loss terms (deposition velocities, scavenging). In the table, Gaussian represents any Gaussian model (`GaussianPlume`, `GaussianPuff`, `GaussianPlume_aer` or `GaussianPuff_aer`) as they all need the same data.

Table 2.1: Mandatory data for each models.

Model	Data necessary.
CastorTransport	Temperature, Pressure, Altitude, AirDensity, MeridionalWind ZonalWind VerticalDiffusion.
CastorChemistry	The same as CastorTransport and SpecificHumidity, LiquidWaterContent, Attenuation.
Polair3DTransport	MeridionalWind (for advection), ZonalWind (for advection), VerticalDiffusion (for diffusion), Horizontal_diffusion (if `Isotropic_diffusion` is set to no; this value is given in the main configuration file), Temperature (if `With_air_density` is set to yes or for microphysical scavenging model), Pressure (if `With_air_density` is set to yes or for microphysical scavenging model).
Polair3DChemistry	The same as Polair3DTransport and SpecificHumidity, Attenuation.
Polair3DChemistryAssimConc	The same as Polair3DChemistry.
Polair3DAerosol	The same as Polair3DChemistry and LiquidWaterContent, SnowHeight.
Gaussian	Temperature, Wind_angle, Wind (wind module), Inversion_height, Stability.

All data for Eulerian models are outputs of meteorological preprocessing programs:

- `meteo`, `Kz`, `attenuation` (and `Kz_TM` if you use Troen & Mahrt parameterization for vertical diffusion), for models of type "Polair3D" while using raw meteorological data from ECMWF.

- `MM5-meteo` (and `Kz_TM` if you use Troen & Mahrt parameterization for vertical diffusion) for models of type “Polair3D” while using raw meteorological data from model MM5;
- `MM5-meteo-castor` (and `Kz_TM` if you use Troen & Mahrt parameterization for vertical diffusion) for models of type “Castor” while using raw meteorological data from model MM5.

2.5.5 Models / Modules Compatibilities

Models of type “Polair3D” require two transport modules (one for advection and one for diffusion), while models of type “Castor” only require one transport module (which deals with advection and diffusion). This does not mean that a module could not be shared by both models (although there is no common module in current Polyphemus version).

Table 2.2 and Table 2.3 present a summary of the compatibility between models and modules. Note that Gaussian models are not included in these tables because they don’t need any module.

Table 2.2: Compatibility between models and transport modules.

	AdvectionDST3	DiffusionROS2	TransportPPM
Polair3DTransport	`X`	`X`	
Polair3DChemistry	`X`	`X`	
Polair3DAerosol	`X`	`X`	
Polair3DChemistryAssimConc	`X`	`X`	
CastorTransport			`X`
CastorChemistry			`X`

	SplitAdvectionDST3	GlobalAdvectionDST3	GlobalDiffusionROS2
Polair3DTransport	`X`	`X`	`X`
Polair3DChemistry	`X`	`X`	`X`
Polair3DAerosol	`X`	`X`	`X`
Polair3DChemistryAssimConc	`X`	`X`	`X`
CastorTransport			
CastorChemistry			

Table 2.3: Compatibility between models and chemistry modules.

	Castor	RACM	RADM	SIREAM	Decay
Polair3DChemistry		`X`	`X`	`X`	`X`
Polair3DAerosol				`X`	`X`
Polair3DChemistryAssimConc		`X`	`X`		
CastorChemistry	`X`				

In Table 2.3, module names are shortened to be displayed on one line: `Castor` is actually `ChemistryCastor`, `RACM` is `ChemistryRACM`, `RADM` is `ChemistryRADM`, `SIREAM` is `AerosolRACM_SIREAM`.

As for drivers, BaseDriver is the simplest and the most used of them. The other drivers available are:

- `StationaryDriver`: for local scale Eulerian simulations.
- `PlumeDriver`: for Gaussian plume model (with or without aerosol species).

- `PuffDriver`: for Gaussian puff model (with or without aerosol species).
- Data assimilation drivers: `OptimalInterpolationDriver` (optimal interpolation), `EnKFDriver` (ensemble Kalman filter), `RRSQRTDriver` (reduced rank square root filter), `FourDimVarDriver` (4D-Var), to be associated with `Polair3DAssimConc` model.
- Drivers for the verification of adjoint coding in variational assimilation: `AdjointDriver`, `GradientDriver`, `Gradient4DVarDriver`.

2.5.6 Checking Results

It is highly recommended to check the fields generated by Polyphemus programs: meteorological fields, deposition velocities, output concentrations, ...

First you can check the size of the binary files. The results are saved as floating point numbers with single precision. This means that most results file must be of size $4 \times N_t \times N_z \times N_y \times N_x$ bytes where N_x and N_y are the space steps along x and y directions respectively, N_z is the number of vertical levels of the field and N_t is the number of time steps. Note that, N_t is not the number of time steps of the simulation but the number of time steps *for which concentrations are saved.*

Second you should check that the fields have reasonable values using the programs from directory `utils`, mainly `get_info_float` (see Section 2.4.1). The command line to use `get_info_float` is:

```
get_info_float results/O3.bin
```

And the output looks like:

```
Minimum: 0.0563521
Maximum: 169.219
Mean:  91.3722
```

Chapter 3

Preprocessing

This chapter introduces all preprocessing programs. It details the input files (data files and configuration files) of every program, and it describes their output files. In Section 3.2, configurations and features shared by almost all programs are explained.

3.1 Remark

In the descriptions of preprocessing programs, there are references to functions like `ComputePressure`, `ComputeAttenuation_LWC`, etc. These functions are part of AtmoData and are described in AtmoData scientific documentation [Njomgang et al., 2005].

3.2 Introduction

3.2.1 Running Preprocessing Programs

Most preprocessing programs:

- accept one or two configuration files as arguments;
- process data daily unless specified otherwise (see Section 3.2.3);
- *append* their results at the end of binary files (if they already exist) or create them. Note that they cannot create the directory so you have to make sure it exists before launching a preprocessing program.

For instance, program `meteo` processes meteorological data over one day. To generate data from day 2001-05-19 to day 2001-05-21, one should launch:

```
./meteo ../general.cfg meteo.cfg 2001-05-19
./meteo ../general.cfg meteo.cfg 2001-05-20
./meteo ../general.cfg meteo.cfg 2001-05-21
```

Another option is to use the script `call_dates` (see Section 2.4.4). In that case, to generate data from day 2005-05-19 to day 2005-05-21, one should launch:

```
meteo> ../utils/call_dates ./meteo ../general.cfg meteo.cfg 20050519 20050521
```

or

```
meteo> ../utils/call_dates ./meteo ../general.cfg meteo.cfg 20050519 3
```

Remember that *the results are appended at the end of the output files if they already exist.* If you decide to recompute your fields from the first day, you have to *first remove* old output binary files.

In order to know what are the arguments of a program, you may launch it without arguments. For instance:

```
emissions> ./emissions

Usage:
  ./emissions [main configuration file] [secondary configuration file] [date]
  ./emissions [main configuration file] [date]
  ./emissions [date]

Arguments:
  [main configuration file] (optional): main configuration file. Default: emissions.cfg
  [secondary configuration file] (optional): secondary configuration file.
  [date]: date in any valid format.
```

3.2.2 Configuration

Almost all programs require the description of the domain over which computations should be performed. Since this configuration is shared by many programs, it is put in a common configuration file called `general.cfg`. An example of such a file is `Polyphemus/preprocessing/general.cfg`, whose content is quoted below:

```
[general]

Home: /u/cergrene/0/bordas
Directory_computed_fields: <Home>/B/data
Directory_ground_data: <Directory_computed_fields>/ground
Programs: <Home>/codes/Polyphemus-HEAD


[domain]

Date: 2001-01-02
Delta_t = 3.0
x_min = -10.0   Delta_x = 0.5   Nx = 65
y_min = 40.5    Delta_y = 0.5   Ny = 33
Nz = 5
Vertical_levels: <Programs>/levels.dat
```

Entries in section `[general]` are markups provided for convenience. See Section 2.3.3 for further explanations.

The section `[domain]` contains the domain (in space and time) description.

[domain]	
`Date`	The date at which the simulation (of the chemistry-transport model) is starting. It is also the date at which meteorological data (processed by Polyphemus – output from programs `meteo` or `MM5-meteo`) starts. As a consequence, any program that needs to read this meteorological data refers to this date. The date must be in a format described in Section 2.2.7.
`Delta_t`	Time step in hour of *output meteorological data* processed by Polyphemus.
`x_min`	Abscissa of the center of the lower-left cell. It is usually in longitude (degrees).
`Delta_x`	Step length along x, usually in degrees (longitude).
`Nx`	Number of cells along x (integer).
`y_min`	Ordinate of the center of the lower-left cell. It is usually in latitude (degrees).
`Delta_y`	Step length along y, usually in degrees (latitude).
`Ny`	Number of cells along y (integer).
`Nz`	Number of vertical levels (integer).
`Vertical_levels`	Path to the file that defines vertical levels interfaces in m.

3.2.3 Dates

Many preprocessing programs require a starting and end date. In that case, the starting date must always be provided in command line, but there are three possibilities for the end date:

1. provide an end date in any valid format (see Section 2.2.7), e.g.

   ```
   ./MM5-meteo ../general.cfg MM5-meteo.cfg 2004-08-09_12-00-00 2004-08-11_06-00-00
   ```

2. provide a duration in format `NdMh` for a duration of N *days* and M *hours*. Alternatively, you can put a duration of `Mh` for M *hours* or `Nd` for N *days*. Valid duration can be, for instance, `3d5h` or `5d` or `14h`. E.g.

   ```
   ./MM5-meteo ../general.cfg MM5-meteo.cfg 2004-08-09_12-00-00 1d12h
   ```

3. provide no end date nor duration. In that case a duration of one day is used. E.g.

   ```
   ./MM5-meteo ../general.cfg MM5-meteo.cfg 2004-08-09_12-00-00
   ```

Please note that in all three cases the end date is excluded. The advantage of it is that, if for whatever reason you want to use the preprocessing program several times in a row, you can use the end date as the starting date for the next time.

Then the number of iterations is computed from the interval between the starting and end dates and from the time step you provided in the general configuration file.

The first date given in command line is the date at which the preprocessing starts *this time*. It might differ from the date given in `general.cfg` if the preprocessing program is launched several times in a row, for example because the meteorological data file does not cover the whole time span of the simulation.

3.2.4 Data Files

Polyphemus reads ECMWF Grib files, MM5 files, NetCDF files (for Mozart 2), text files and binary files. All files generated by Polyphemus are text files or binary files.

Unless specified otherwise, all binary files store single-precision floating-point numbers. They do not contain any header. Each binary file only stores the values of a single field. Four-dimensional fields are stored this way:

```
Loop on time t
   Loop on z
      Loop on y
         Loop on x
```

Let this storage be symbolized by $\{t, z, y, x\}$. Dimensions t, z, y and x always appear in this order. For instance, three-dimensional fields may be stored in formats $\{z, y, x\}$ or $\{t, y, x\}$, $\{t, z, x\}$ or $\{t, z, y\}$.

3.3 Ground Data

Computing ground data is the first step of a preprocessing as they are necessary to process meteorological fields. All programs related to ground-data generation are available in `Polyphemus/preprocessing/ground`.

The first step should be program `luc-usgs` or `luc-glcf` depending on what raw data you have. Land use data may come from the US Geological Survey (USGS) or from the Global Land Cover Facility (GLCF).

3.3.1 Land Use Cover – GLCF: `luc-glcf`

In order to prepare land use cover from GLCF, one should use program `luc-glcf`. It is recommended to download the global land use cover file at 1 km resolution provided in latitude–longitude coordinates. At the time this documentation is written, the file is available[†1] at:

`ftp://ftp.glcf.umiacs.umd.edu/glcf/Global_Land_Cover/Global/`
`gl-latlong-1km-landcover/gl-latlong-1km-landcover.bsq.gz` (single line, no white space – you may use `wget` to download it, or copy and paste the URL in your favorite browser).

You need to uncompress this file (e.g., `gunzip gl-latlong-1km-landcover.bsq.gz`).

Finally you have to fill the configuration file `luc-glcf.cfg`. Note that the default values in section `[GLCF]` are for file `gl-latlong-1km-landcover.bsq`: no need to change them if you downloaded this recommended file.

[paths]	
`Database_luc-glcf`	Directory where the raw data from GLCF can be found (directory where `gl-latlong-1km-landcover.bsq` lies).
`LUC_in`	Name of the file containing raw data (i.e., `gl-latlong-1km-landcover.bsq` or its new name if you renamed it).
`Directory_luc-glcf`	Output directory.
`LUC_out`	Output filename. The default filename `LUC-glcf.bin` is recommended for clarity.

[†1] In case the file has been moved, try to find it from `http://glcf.umiacs.umd.edu/data/landcover/`, or even from GLCF homepage `http://glcf.umiacs.umd.edu/index.shtml`.

	`[GLCF]`
`Step`	Space step in degrees in GLCF input file.
`x_min`	Minimum longitude in the input file (degrees).
`y_min`	Minimum latitude in the input file (degrees).
`Nx`	Number of cells along longitude in the input file.
`Ny`	Number of cells along latitude in the input file.
`Nc`	Number of land use categories.

The output land-cover file is in format $\{c, y, x\}$ where c stands for (land use) category. Table 3.3 presents land-use categories as they are computed with `luc-glcf`.

Table 3.3: Land-use categories in GLCF description.

Value	Label
0	Water.
1	Evergreen Needleleaf Forest.
2	Evergreen Broadleaf Forest.
3	Deciduous Needleleaf Forest.
4	Deciduous Broadleaf Forest.
5	Mixed Forest.
6	Woodland.
7	Wooded Grassland.
8	Closed Shrubland.
9	Open Shrubland.
10	Grassland.
11	Cropland.
12	Bare Ground.
13	Urban and Built.

Program `luc-glcf` does not require any date as an input in command line. To launch `luc-glcf`, just type:

```
./luc-glcf ../general.cfg luc-glcf.cfg
```

3.3.2 Land Use Cover – USGS: `luc-usgs`

For a simulation over Europe, program `luc-usgs` requires two files found at `http://edcsns17.cr.usgs.gov/glcc/`:

- USGS Land Use/Land Cover Scheme for Eurasia in Lambert Azimuthal Equal Area Projection (optimized for Europe) available at `http://edcftp.cr.usgs.gov/pub/data/glcc/ea/lamberte/eausgs2_0le.img.gz` in compressed format.

- USGS Land Use/Land Cover Scheme for Africa in Lambert Azimuthal Equal Area Projection available at `http://edcftp.cr.usgs.gov/pub/data/glcc/af/lambert/afusgs2_0l.img.gz` in compressed format.

The configuration file `luc-usgs.cfg` requires:

`[paths]`	
`Database_luc-usgs`	Directory where the raw data from USGS can be found.
`LUC_in_ea`	Input file containing raw data for Eurasia (eausgs2_0le.img).
`LUC_in_af`	Input file containing raw data for Africa (afusgs2_0l.img).
`Directory_luc-usgs`	Output directory.
`LUC_out`	Output file name. The default filename `LUC-usgs.bin` is recommended for clarity.
`[USGS]`	
`Step`	Space step in meters.
`lon_origin_ea`	Longitude of the center of lower-right cell for Eurasia.
`lat_origin_ea`	Latitude of the center of the lower-right cell for Eurasia.
`lon_origin_af`	Longitude of the center of the lower-right cell for Africa.
`lat_origin_af`	Latitude of the center of the lower-right cell for Eurasia.
`lon_upper_left_ea`	Longitude of the center of the upper-left cell for Eurasia.
`lat_upper_left_ea`	Latitude of the center of the upper-left cell for Eurasia.
`lon_upper_left_af`	Longitude of the center of the upper-left cell for Africa.
`lat_upper_left_af`	Latitude of the center of the upper-left cell for Africa.
`Nx_ea`	Number of cells along longitude in the input file for Eurasia.
`Nx_af`	Number of cells along longitude in the input file for Africa.
`Ny_ea`	Number of cells along latitude in the input file for Eurasia.
`Ny_af`	Number of cells along latitude in the input file for Africa.
`Nc`	Number of land categories.
`Sea_index`	Index of the sea in land categories (Remember that indices start at 0).

The output land-cover file is in format $\{c, y, x\}$ where c stands for (land use) category.

Table 3.5 presents land-use categories as they are computed with `luc-usgs`, that is to say with indices starting at 0.

Table 3.5: Land-use categories in USGS description (indices start at 0).

Value	Label
0	Urban and Built-Up Land.
1	Dryland Cropland and Pasture.
2	Irrigated Cropland and Pasture.
3	Mixed Dryland/Irrigated Cropland and Pasture.
4	Cropland/Grassland Mosaic.
5	Cropland/Woodland Mosaic.
6	Grassland.
7	Shrubland.
8	Mixed Shrubland/Grassland.
9	Savanna.
10	Deciduous Broadleaf Forest.
11	Deciduous Needleleaf Forest.
12	Evergreen Broadleaf Forest.
13	Evergreen Needleleaf Forest.

14	Mixed Forest.
15	Water Bodies.
16	Herbaceous Wetland.
17	Wooded Wetland.
18	Barren or Sparsely Vegetated.
19	Herbaceous Tundra.
20	Wooded Tundra.
21	Mixed Tundra.
22	Bare Ground Tundra.
23	Snow or Ice.

Program `luc-usgs` does not require any date as an input in command line. To launch `luc-usgs`, just type:

```
./luc-usgs ../general.cfg luc-usgs.cfg
```

3.3.3 Conversions: `luc-convert`

The output of `luc-glcf` or `luc-usgs` are land use cover described with GLCF or USGS categories. It is often useful to convert these descriptions to another set of land use categories. This means, for example, summing up the contributions of sparsely vegetated and bare ground tundra (USGS categories #19 and #22) to estimate the proportion of barren land in Wesely description (category #8). An input category may also be split into several output categories. In practice, one may want to convert in Wesely or Zhang land use cover using `luc-convert`. In particular it is necessary to convert land data from USGS or GLCF to Zhang categories before computing deposition velocities with program `dep` (see Section 3.5.1).

In addition to the domain definition (Section 3.2.2), below is the information required in the configuration file (or configuration files) for `luc-convert`:

[paths]	
`Database_luc-convert`	Directory where the input file (input land use categories) is located.
`File_in`	Input file name (in `Database_luc-convert`).
`Directory_luc-convert`	Directory where the output file (output land use categories) should be stored.
`File_out`	Output file name (in `Directory_luc-convert`).
[dimensions]	
`Nc_in`	Number of land categories in the input format.
`Nc_out`	Number of land categories in the output format.
[coefficients]	
Correspondence matrix between input land categories and output land categories. Each line corresponds to an input category. Each line contains: the index of the category (or any number: this first column is not read) and the distribution of the input category in all output categories (columns). The distribution is a set of numbers in $[0, 1]$ whose sum should be 1.	

Several configuration files are provided to convert GLCF or USGS categories to Wesely or Zhang categories: `glcf_to_wesely.cfg`, `glcf_to_zhang.cfg`, `usgs_to_wesely.cfg` and `usgs_to_zhang.cfg`.

The output land-cover file is in format $\{c, y, x\}$ where c stands for (land use) category.

The conversion can be launched with:

```
./luc-convert ../general.cfg usgs_to_wesely.cfg
```

3.3.4 Roughness: `roughness`

After land use cover has been computed, roughness data can be estimated, using program `roughness`.

	`[domain]`
`Nx`	Number of grid points along longitude.
`Ny`	Number of grid points along latitude.
	`[paths]`
`LUC_file`	File where the land use cover data are stored (e.g., computed using `luc-glcf` or `luc-usgs`).
`Directory_roughness`	Directory where the output file will be stored.
`Roughness_out`	Output file name.
	`[data]`
`Roughness_data_file`	Path to the file giving the roughness of land categories. This file should be a text file with three columns: the land category index (starting at 0), the roughness height (in m) and the category name. Two examples are provided: `roughness-glcf.dat` and `roughness-usgs.dat`

The program may be launched with:

```
./roughness ../general.cfg roughness.cfg
```

Section `[domain]` is in `general.cfg` and the other sections are read in `roughness.cfg`.

3.3.5 LUC for emissions: `extract-glcf`

This program is only necessary in order to generate anthropogenic emissions from EMEP inventories (see Section 3.6.2). The output is the land category (read from GLCF global land-use classification, `gl-latlong-1km-landcover.bsq`) over a region that covers the simulation domain – hence where emissions should be generated. Make sure the output domain (described in section `[subdomain]` of the configuration file) entirely contains your simulation domain.

The configuration file should contain (see example `extract-glcf.cfg`):

	`[paths]`
`File_GLCF`	GLCF input file (global). It is the same as the one used for `luc-glcf`.
`File_out`	Output file. That is, the file given in section `[LUC]` of `emissions.cfg`.
	`[GLCF]`
`x_min`	Minimum latitude of the GLCF domain (whole world).
`Nx`	Number of steps along the latitude.

`y_max`	Maximum longitude of the GLCF domain (whole world).
`Nx`	Number of steps along the latitude.
`Step`	Space step (in degrees) of the input GLCF file. The output file will have the same resolution.
	`[subdomain]`
`x_min`	Minimum latitude of the subdomain.
`Nx`	Number of steps along the latitude.
`y_min`	Minimum longitude of the subdomain.
`Nx`	Number of steps along the latitude.

The program may be launched with

```
./extract-glcf ../general.cfg extract-glcf.cfg
```

Note that the file `../general.cfg` is not compulsory providing no markup from it is used in `extract-glcf.cfg`.

The output is a binary file of integers between 0 and 13.

3.4 Meteorological Fields

3.4.1 Program `meteo`

Program `Polyphemus/preprocessing/meteo/meteo` reads ECMWF Grib files and generates meteorological fields required by chemistry-transport models. Most fields are interpolated from ECMWF grid to a regular grid (latitude/longitude in the horizontal, altitudes in meters in the vertical). It is assumed that ECMWF input data are stored in daily Grib files. That is why this program (as well as `attenuation` and `Kz`) only processes data daily and requires only one date as an input in command line.

Note that `meteo` needs as input data the land use cover which can be built using programs in `preprocessing/ground`.

The reference configuration files for `meteo` is `Polyphemus/preprocessing/meteo/meteo.cfg` together with `Polyphemus/preprocessing/general.cfg`. In addition to the domain definition, below are options of `meteo`:

	`[paths]`
`Database_meteo`	Directory in which ECMWF input files may be found. It is assumed that ECMWF file names are in format `ECMWF-YYYYMMDD` where `YYYY` is the year, `MM` the month and `DD` the day. If program `meteo` is launched for a day D, ECMWF data files for days D-1 and D must be available. Data for day D-1 are needed to process cumulated data (e.g., solar radiation).
`Roughness_in`	Path to the binary file that describes roughness heights (in meters) in ECMWF grid cells. Its format is $\{y, x\}$. It is needed only if option `Richardson_with_roughness` is activated.
`Directory_meteo`	Directory where output meteorological files are stored.
	`[ECMWF]`
`Date`	Date at which the meteorological file begins. It is referred as `&D` which is the date given in command line because it is supposed that ECMWF Grib files store data daily.

`t_min`	First hour stored in the Grib file.
`Delta_t`	Time step (in hour) of data stored in every ECMWF file.
`Nt`	Number of time steps stored in every ECMWF file.
`x_min`	Longitude in degrees of the center of the lower-left cell in ECMWF grid.
`Delta_x`	Step length (in degrees) along longitude of ECMWF grid.
`Nx`	Number of cells along longitude (integer) in ECMWF grid.
`y_min`	Latitude in degrees of the center of the lower-left cell in ECMWF grid.
`Delta_y`	Step length (in degrees) along latitude of ECMWF grid.
`Ny`	Number of cells along latitude (integer) in ECMWF grid.
`Nz`	Number of vertical layers (integer) in ECMWF grid.
	`[meteo]`
`Richardson_with_roughness`	Should the surface Richardson number be computed taking into account roughness height?
	`[accumulated_data]`
`Accumulated_time`	For data storing values cumulated in time (e.g., solar radiation), length number of time steps over which data is cumulated.
`Accumulated_index`	Start index of the first complete cycle of cumulated data. Data is then cumulated from `t_min` plus `Accumulated_index` times `Delta_t`.

To launch the program, just type :

```
./meteo ../general.cfg meteo.cfg 2001-04-22
```

The program basically reads data in the ECMWF Grib file and interpolates it in time and space to Polyphemus grid. ECMWF data is described in `meteo.cfg` and Polyphemus grid is described in `general.cfg`. For the sake of simplicity, it is recommended to work with ECMWF files containing data for one day: all Polyphemus programs work on a daily basis. Program `meteo` should be called for each day (preferably from 0h to 24h), that is, for each available ECMWF file (except the first one – see below). If ECMWF files are not provided on a daily basis, it is recommended to contact the Polyphemus team at `polyphemus@cerea.enpc.fr`.

In order to process the ECMWF file for a given day, the ECMWF file for the previous day must be available. Indeed, ECMWF files contain data that is accumulated over several time steps (like rain), and values from previous steps (including from the previous day) must be subtracted to get the actual value of the field.

Here is the list of input data needed in ECMWF files (with their Grib code): surface temperature (167), skin temperature (235), surface pressure (152), temperature (130), specific humidity (133), liquid water content (246), medium cloudiness (187), high cloudiness (188), meridional wind (132), zonal wind (131), zonal friction velocity (180), meridional friction velocity (181), solar radiation (169), boundary layer height (159), soil water content (39), sensible heat (146), evaporation (182).

The list of output variables is: pressure, surface pressure, temperature, surface temperature, skin temperature, Richardson number, surface Richardson number, specific humidity, liquid water content, solar radiation, photosynthetically active radiations (direct beam, diffuse and total), zonal wind, meridional wind, wind module, wind friction module, boundary layer height, soil water content, evaporation, sensible heat and first-level wind module.

Inside `meteo`, ECMWF variables are read and decumulated (in time) if necessary. Pressures at ECMWF levels are computed with `ComputePressure` and altitudes are computed with `ComputeInterfHeight` and `ComputeMiddleHeight`. Richardson number is then estimated with `ComputeRichardson`. All input fields are then interpolated on the vertical. Finally photosynthetically active radiation are estimated, based on solar radiation and zenith angle (`ZenithAngle`).

To get the complete set of input meteorological data for a transport model, one should then launch `attenuation`, `Kz` and maybe `Kz_TM`.

3.4.2 Program `attenuation`

Program `Polyphemus/preprocessing/attenuation` should be launched after program `meteo`. It computes cloud attenuation for photolysis rates. It also computes cloud related data, such as cloud height. Even for passive simulations this program should be launched.

The reference configuration files for `attenuation` is `Polyphemus/preprocessing/meteo/meteo.cfg` together with `Polyphemus/preprocessing/general.cfg`. In addition to the domain definition and to the entries of `meteo.cfg` introduced in Section 3.4.1, below are options for `attenuation`:

	`[paths]`
`Directory_attenuation`	Directory where the output of program `attenuation` is stored.
	`[attenuation]`
`Type`	Parameterization to be used to compute cloud attenuation. Put `1` to use `RADM` parameterization or put `2` to use `ESQUIF` parameterization.
	`[clouds]`
`Min_height`	Minimum cloud basis height in m.

Just like in program `meteo`, ECMWF data is read and interpolated. Then the relative humidity and the critical relative humidity are computed respectively with `ComputeRelativeHumidity` and `ComputeCriticalRelativeHumidity`. The cloud fraction is computed with `ComputeCloudFraction`. For it the cloudiness and cloud height are diagnosed using `ComputeCloudiness` and `ComputeCloudHeight`. Finally attenuation coefficients are computed with `ComputeAttenuation_LWC` (RADM parameterization) or `ComputeAttenuation_ESQUIF` (ESQUIF parameterization).

Output files are:

- the rain intensity (`Rain.bin`) in $\mathrm{mm\,h^{-1}}$,
- the convective rain intensity (`ConvectiveRain.bin`) in $\mathrm{mm\,h^{-1}}$,
- the 3D cloud attenuation coefficient (in $[0, 2]$),
- the cloud height in m.

3.4.3 Program `Kz`

Program `Kz` computes the vertical diffusion coefficients (needed in almost all applications) using Louis parameterization [Louis, 1979]. It should be launched after `attenuation`.

The reference configuration files for `Kz` is `Polyphemus/preprocessing/meteo/meteo.cfg` together with `Polyphemus/preprocessing/general.cfg`. In addition to the domain definition and to the entries of `meteo.cfg` introduced in Section 3.4.1 and 3.4.2, below are options for `Kz`:

	`[paths]`
`File_Kz`	Name of the file where the vertical diffusion coefficients (output) are stored.
	`[Kz]`
`Min`	Lower threshold for vertical diffusion in $\mathrm{m}^2\,\mathrm{s}^{-1}$.
`Max`	Higher threshold for vertical diffusion in $\mathrm{m}^2\,\mathrm{s}^{-1}$.
`Apply_vert`	If set to `no`, the lower threshold is applied only at the top of the first layer, otherwise it is applied to all levels.

This programs mainly computes the vertical diffusion coefficients with a call to `ComputeLouisKz`. Simple corrections are also performed to take into account convective conditions.

The output is a 3D time-dependent field (format $\{t, z, y, x\}$) of vertical diffusion coefficients (in $\mathrm{m}^2\,\mathrm{s}^{-1}$). Along the vertical, the coefficients are defined on the interfaces. So the size of the field (for each day) is $Nt \times (Nz+1) \times Ny \times Nx$. It is stored in the path given by entry `File_Kz`.

3.4.4 Program `Kz_TM`

Program `Kz_TM` "overwrites", in the boundary layer height, the vertical diffusion coefficients computed with Louis parameterization, with coefficients computed according to Troen & Mahrt parameterization [Troen and Mahrt, 1986]. It should be launched either after `Kz` or after `MM5-meteo`.

The reference configuration files for `Kz_TM` is `Polyphemus/preprocessing/meteo/meteo.cfg` or `Polyphemus/preprocessing/meteo/MM5-meteo.cfg` together with `Polyphemus/preprocessing/general.cfg`. In addition to the domain definition in `general.cfg`, below are the entries for `Kz_TM`:

	`[path]`
`LUC_file`	Path to the binary file that describes land use cover over the *output* grid (described in section `[domain]`). This file must be in format $\{l, y, x\}$ (l is the land category) and it must contain proportions (in $[0, 1]$) of each land category in every grid cell.
`Sea_index`	Index of sea in land categories (remember that indices start at 0). It is 0 for GLCF description and 15 for USGS description.
`Roughness_file`	Path to the binary file that describes roughness heights (in meters) in output grid cells. Its format is $\{y, x\}$. It is needed only if option `Flux_diagnosed` is activated.
`Directory_meteo`	Directory where output meteorological files are stored.
`File_Kz`	Name of the file where the vertical diffusion coefficients as computed with the Louis parameterization are stored.
`Directory_Kz_TM`	Name of the directory where the vertical diffusion coefficients (output) are stored (the filename being `Kz_TM`).
	`[Kz]`
`Min`	Lower threshold for vertical diffusion in $\mathrm{m}^2\,\mathrm{s}^{-1}$.
`Max`	Higher threshold for vertical diffusion in $\mathrm{m}^2\,\mathrm{s}^{-1}$.

`Apply_vert`	If set to `no`, the lower threshold is applied only at the top of the first layer, otherwise it is applied to all levels.
`p`	Coefficient used in Troen and Mahrt parameterization (see Troen and Mahrt [1986]).
`C`	Coefficient used in Troen and Mahrt parameterization (see Troen and Mahrt [1986]).
`SBL`	Ratio between the surface layer and the atmospheric boundary layer (0.1 in Troen and Mahrt [1986]).
`Ric`	Critical Richardson number used to estimate the atmospheric boundary layer height (in case `BL_diag` is set to `2`).
`Fluxes_diagnosed`	Should the friction module, the evaporation and the sensible heat be diagnosed? If not, they are read in input data (which is recommended).
`BL_diag`	What kind of diagnosis is used to estimate the boundary layer height? Put `1` to use Troen and Mahrt diagnosis [Troen and Mahrt, 1986]; put `2` to rely on a critical Richardson number; and put `3` to use ECMWF (or MM5) boundary layer height (so, there is no diagnosis – this option is more robust and it is recommended).
`TM_stable`	The vertical diffusion as computed by Troen and Mahrt parameterization is applied only within the boundary layer. It is possible to further restrict its application: if `TM_stable` is set to `no`, the parameterization is not applied in stable conditions. In this case, the Troen and Mahrt parameterization is only applied in unstable boundary layer.

Several meteorological fields are computed with `ComputePotentialTemperature`, `ComputeSaturationHumidity` and `ComputeSurfaceHumidity_diag`. If fluxes are not diagnosed, the Monin-Obukhov length is computed with `ComputeLMO`. Then the boundary layer height may be diagnosed with `ComputePBLH_TM` (Troen & Mahrt parameterization) or `ComputePBLH_Richardson` (critical Richardson number). Finally the vertical diffusion coefficients are computed with `ComputeTM_Kz`.

The main output is a 3D time-dependent field (format $\{t, z, y, x\}$) of vertical diffusion coefficients (in $\mathrm{m}^2\,\mathrm{s}^{-1}$). Along the vertical, the coefficients are defined on the interfaces. So the size of the field (for each day) is $Nt \times (Nz + 1) \times Ny \times Nx$. It is stored in `Kz_TM.bin` in the directory given by entry `Directory_Kz_TM`. The surface relative humidity is saved in `SurfaceRelativeHumidity.bin`. Depending on the options, additional fields may be saved, such as the Monin-Obukhov length in file `LMO.bin`.

3.4.5 Program `MM5-meteo`

Program `Polyphemus/preprocessing/meteo/MM5-meteo` processes MM5 data and generates meteorological fields required by chemistry-transport models. Most fields are interpolated from MM5 grid to a regular grid (latitude/longitude in the horizontal, altitudes in meters in the vertical).

Note that `MM5-meteo` needs as input data the land use cover which can be built using programs in `preprocessing/ground`.

Note for ECMWF users: program `MM5-meteo` is equivalent to what is performed by `meteo`, `attenuation` and `Kz` successively. Similarly to ECMWF files, `Kz_TM` can be used afterwards.

Program `MM5-meteo` can be launched as follows:

```
./MM5-meteo ../general.cfg MM5-meteo.cfg 2004-08-09_09-00-00
./MM5-meteo ../general.cfg MM5-meteo.cfg 2004-08-09_09-00-00 2004-08-10_09-00-00
./MM5-meteo ../general.cfg MM5-meteo.cfg 2004-08-09_09-00-00 1d
```

The configuration file `MM5-meteo.cfg` contains several options:

	`[paths]`
`Database_MM5_meteo`	Directory in which MM5 input files may be found. If `&D` appears in the file name, it is replaced by `MM5-YYYY-MM-DD` where `YYYY` is the year, `MM` the month and `DD` the day.
`Directory_meteo`	Directory where output meteorological files are stored.
`Directory_attenuation`	Directory where the output of program `attenuation` is stored.
	`[MM5]`
`t_min`	First hour stored in every MM5 file.
`Delta_t`	Time step (in hour) of data stored in every MM5 file.
`Nt`	Number of time steps stored in every MM5 file.
`x_min`	Index in MM5 coordinates of the center of the lower-left cell in MM5 grid. This is most likely `0.5`.
`Delta_x`	Index (MM5 coordinates) increase along longitude of MM5 grid. This is most likely `1`.
`Nx`	Number of cells (or dot points) along longitude (integer) in MM5 grid.
`y_min`	Index in MM5 coordinates of the center of the lower-left cell in MM5 grid. This is most likely `0.5`.
`Delta_y`	Index (MM5 coordinates) increase along latitude of MM5 grid. This is most likely `1`.
`Ny`	Number of cells (or dot points) along latitude (integer) in MM5 grid.
`Nz`	Number of vertical layers (integer) in MM5 grid.
`projection_type`	Type of projection. `1` corresponds to Lambert conformal conic, `2` to Mercator and `3` to stereographic.
	`[accumulated_rain]`
`Prev_accumulated_rain`	Is the rain accumulated from the previous day?
	`[attenuation]`
`Type`	Parameterization to be used to compute cloud attenuation. Put `1` to use `RADM` parameterization or put `2` to use `ESQUIF` parameterization.
	`[clouds]`
`Min_height`	Minimum cloud basis height in m.
	`[Kz]`
`Min`	Lower threshold for vertical diffusion in $\mathrm{m}^2\,\mathrm{s}^{-1}$.
`Max`	Higher threshold for vertical diffusion in $\mathrm{m}^2\,\mathrm{s}^{-1}$.
`Apply_vert`	If set to `no`, the lower threshold is applied only at the top of the first layer, otherwise it is applied to all levels.

The program basically reads data in MM5 output file and interpolates it in time and space to Polyphemus grid. MM5 file is described in `MM5-meteo.cfg` and Polyphemus grid is described in `general.cfg`. Note that each time a field is loaded by `MM5-meteo`, all time steps are loaded in memory. Note that the fields are released from memory when unused, but you may still need a lot of memory for big MM5 output files.

The program first computes the altitude of MM5 layers, and converts the Polyphemus grid coordinates (latitude/longitude) to MM5 grid coordinates (Lambert, Mercator or stereographic) for interpolations. Interpolations on the horizontal are performed in MM5 grid for efficiency. The pressure is computed based on MM5 fields. The winds are rotated: this gives meridional and zonal winds. The Richardson number is then computed (`ComputeRichardson`). The relative humidity and the critical relative humidity are computed respectively with `ComputeRelativeHumidity` and `ComputeCriticalRelativeHumidity`. The cloud fraction is computed with `ComputeCloudFraction`. For it the cloudiness and cloud height are diagnosed using `ComputeCloudiness` and `ComputeCloudHeight`. Finally attenuation coefficients are computed with `ComputeAttenuation_LWC` (RADM parameterization) or `ComputeAttenuation_ESQUIF` (ESQUIF parameterization). The vertical diffusion coefficients are computed with `ComputeLouisKz` [Louis, 1979]. Finally photosynthetically active radiation are estimated, based on solar radiation and zenith angle (`ZenithAngle`).

Among output files one may find:

- the pressure and the surface pressure in Pa,
- the temperature, the surface temperature and the skin temperature in K,
- the meridional and zonal winds (`MeridionalWind.bin` and `ZonalWind.bin`) in $\mathrm{m\,s^{-1}}$,
- the Richardson number and the surface Richardson number,
- the boundary layer height in m,
- the vertical diffusion coefficients (time-dependent 3D field, defined on layer interfaces on the vertical, `Kz_Louis.bin`) in $\mathrm{m^2\,s^{-1}}$,
- the specific humidity in $\mathrm{kg\,kg^{-1}}$,
- the liquid water content in $\mathrm{kg\,kg^{-1}}$,
- the cloud attenuation coefficients (3D field, `Attenuation.bin`) in $[0, 2]$,
- the solar radiation intensity (`SolarRadiation.bin`) in $\mathrm{W\,m^{-2}}$,
- the rain intensity (`Rain.bin`) in $\mathrm{mm\,h^{-1}}$,
- the convective rain intensity (`ConvectiveRain.bin`) in $\mathrm{mm\,h^{-1}}$,
- the cloud height in m.

3.4.6 Program `MM5-meteo-castor`

Program `MM5-meteo-castor` processes MM5 data and generates meteorological fields required by chemistry-transport model Castor.

	[domain]
Vertical_levels	File containing the parameters alpha and beta used to compute the pressure at various levels and the altitudes. Note that for most preprocessing programs, this field designates file `preprocessing/levels.dat` but not for this specific application, for which you can use file `preprocessing/meteo/hybrid_coefficients.dat`.
	[paths]
Database_MM5_meteo	Directory in which MM5 input files may be found. If `&D` appears in the file name, it is replaced by `YYYY-MM-DD` where `YYYY` is the year, `MM` the month and `DD` the day.
Roughness_file	Path to the binary file that describes roughness heights (in meters) per month in output grid cells. Note that this file is not the output of program `roughness`.
Directory_meteo	Directory where output meteorological files are stored.
	[MM5]
t_min	First hour stored in every MM5 file.
Delta_t	Time step (in hour) of data stored in every MM5 file.
Nt	Number of time steps stored in every MM5 file.
x_min	Index in MM5 coordinates of the center of the lower-left cell in MM5 grid. This is most likely `0.5`.
Delta_x	Index (MM5 coordinates) increase along longitude of MM5 grid. This is most likely `1`.
Nx	Number of cells (or dot points) along longitude (integer) in MM5 grid.
y_min	Index in MM5 coordinates of the center of the lower-left cell in MM5 grid. This is most likely `0.5`.
Delta_y	Index (MM5 coordinates) increase along latitude of MM5 grid. This is most likely `1`.
Ny	Number of cells (or dot points) along latitude (integer) in MM5 grid.
Nz	Number of vertical layers (integer) in MM5 grid.
projection_type	Type of projection. `1` corresponds to Lambert conformal conic, `2` to Mercator and `3` to stereographic.
Horizontal_interpolation	Type of horizontal interpolation used. `MM5` corresponds to MM5 coordinates and `latlon` to latitude/longitude coordinates.
Dot_coordinates	File containing coordinates of dot points. Used if `Horizontal_interpolation` is set to `latlon`.
	[meteo]
Relative_humidity_threshold	Minimum relative humidity above which cloud are formed.
Low_cloud_top_max	Low clouds maximum height (in m).
	Kz]

`Min_dry`	Minimum value of Kz in PBLH for dry conditions (in $\mathrm{m\,s^{-2}}$).
`Min_wet`	Minimum value of Kz in PBLH for cloudy conditions (in $\mathrm{m\,s^{-2}}$).
`Min_above_PBLH`	Minimum value of Kz above PBLH (in $\mathrm{m\,s^{-2}}$).
`Max`	Maximum value for Kz (in $\mathrm{m\,s^{-2}}$).

Among output files one may find:

- the altitude in meters,
- the air density (`AirDensity.bin`)
- the pressure in Pa (`Pressure.bin`),
- the temperature and temperature at 2 m in K (`Temperature.bin` and `Temperature_2m.bin`),
- the meridional wind , zonal wind, convective velocity and wind module at 10 m (`MeridionalWind.bin`, `ZonalWind.bin`, `ConvectiveVelocity.bin` and `WindModule_10m.bin`) in $\mathrm{m\,s^{-1}}$,
- the boundary layer height in m (`PBLH.bin`),
- the vertical diffusion coefficients using Troen and Mahrt parameterization (`Kz.bin`) in $\mathrm{m^2\,s^{-1}}$,
- the specific humidity in $\mathrm{kg\,kg^{-1}}$ (`SpecificHumidity.bin`),
- the surface relative humidity (`SurfaceRelativeHumidity.bin`,
- the liquid water content in $\mathrm{kg\,kg^{-1}}$ (`LiquidWaterContent.bin`),
- the cloud attenuation coefficients (`Attenuation.bin`)
- the soil moisture (`SoilMoisture.bin`),
- the aerodynamic resistance (`AerodynamicResistance.bin`),
- the friction velocity in $\mathrm{m\,s^{-1}}$ (`FrictionModule.bin`),

3.5 Deposition Velocities

Deposition velocities are generated on the basis of meteorological fields and land data. The programs must be launched after meteorological and ground preprocessing.

The computation of deposition velocities for Gaussian models is presented in Section 3.9.2.

3.5.1 Program `dep`

The program `dep` computes deposition velocities according to Wesely [1989] or Zhang et al. [2003].

In addition to `general.cfg`, the program reads the configuration in `dep.cfg`. In this file, paths to several files generated by programs `meteo` or `MM5-meteo` are given.

	`[paths]`
`SurfaceTemperature`	File where surface temperature is stored.
`SurfaceRichardson`	File where surface Richardson number is stored.
`SolarRadiation`	File where solar radiation is stored.
`WindModule`	File where wind module is stored.
`PAR`	File where photosynthetically active radiation is stored.
`PARdiff`	File where the diffuse part of the photosynthetically active radiation is stored.
`PARdir`	File where the direct beam part of photosynthetically active radiation is stored.
`SpecificHumidity`	File where (3D) specific humidity is stored.
`SurfacePressure`	File where surface pressure is stored.
`FrictionVelocity`	File where friction velocity is stored.
`CanopyWetness`	File where canopy wetness is stored.
`Rain`	File where rain is stored.
`RoughnessHeight`	File where roughness height is stored.
`Type`	Configuration file that describes land use cover (see below for details about this file).
`Data`	File containing the data for species. This file should contain: the species name, the molecular weight ($\mathrm{g\,mol^{-1}}$), Henry constant, diffusivity, reactivity, alpha [Zhang et al., 2003], beta [Zhang et al., 2003], Rm. An example for RADM/RACM is available in `preprocessing/dep/input/species_data.txt`.
`Directory_dep`	Directory where the output files are stored.
	`[Species]`
`Ns`	Number of species for which data are provided. This should be the number of columns in `preprocessing/dep/input/species_data.txt`.
	`[Options]`
`CellRoughness`	If this option is set to `yes`, the roughness height used in calculations only depends on the model cell (and not on the land use category). In this case, it uses the data file whose path is given in entry `RoughnessHeight` (section `[paths]`). If the options is set to `no` (recommended), the roughness height depends on the land use category (see entry `Type`).
`Ra`	Parameterization used to compute the aerodynamic resistance. You can choose between `fh` (heat flux), `fm` (momentum flux) or `diag` (diagnostic).
`Rb`	Parameterization used to compute the quasi-laminar sublayer resistance. You can choose between `friction` and `diag`.
`Rc`	Parameterization used to compute the canopy resistance (Zhang et al. [2003] or Wesely [1989]).
`Save_resistance`	Should Ra, Rb and Rc be saved? This may take a lot of storage space: put `no` if you do not work on the deposition parameterizations.

Entry `Type` is the path to a configuration file whose entries should be:

`File`	Path to the file describing the land use cover. The number of categories in the file is deduced from its size, but it must be consistent with the data provided in the following entries (`Midsummer`, etc.)
`Midsummer`	Data file for midsummer (see below for details).
`Autumn`	Data file for autumn (see below for details).
`Late_autumn`	Data file for late autumn (see below for details).
`Snow`	Data file for snow (see below for details).
`Spring`	Data file for spring (see below for details).

The data files mentioned above for the five "seasons" must contain a column for each land use category with 22 parameters in each column. You may modify these files or create new files only if you are well aware of deposition parameterizations. With Polyphemus, a set of 5 files is provided for convenience, and any beginner should use them. They are suited for land use categories as defined in Zhang et al. [2002].

A key step is therefore to generate a land use description with these categories (referred as Zhang categories). The recommended program to generate this file is `luc-convert` which is described in Section 3.3.3. You should use this program to convert GLCF or USGS land cover to Zhang categories.

Please note that the program `dep` chooses which land use file to use according to *the month of the beginning date only.* Therefore, if you need deposition velocities for a date range during which the "season" changes, make sure to launch different simulations for the different seasons.

The program may be launched this way:

```
./dep ../general.cfg dep.cfg 20040809
```

3.5.2 Program `dep-emberson`

The program `dep-emberson` is used to compute deposition velocities for Castor model, using Emberson parameterization.

	`[paths]`
`Altitude`	File where altitude is stored.
`SurfaceTemperature`	File where surface temperature is stored.
`SurfaceRelativeHumidity`	File where surface relative humidity is stored.
`FrictionVelocity`	File where the friction velocity is stored.
`Attenuation`	File where the attenuation is stored.
`AerodynamicResistance`	File where the aerodynamic resistance is stored.
`LUC_file`	File containing the land use cover.
`Nc`	Number of land use cover categories.
`Nveg`	Number of vegetation classes.
`Land_data`	File containing land data in Chimere format.
`Species_data`	File containing the data for species (molecular weight, Henry constant, reactivity).
`Directory_dep`	Directory where the output files are stored.
	`[Species]`
`Ns`	Number of species for which data are provided.

The program must be launched with:

```
./dep-emberson ../general.cfg dep-emberson.cfg 20040809
```

3.6 Emissions

Emissions are generated on the basis of land data (anthropogenic emissions) and meteorological fields (biogenic emissions). The programs must be launched after meteorological and ground preprocessing.

For Gaussian models, a preprocessing step may also be required in case line emissions are included (see Section 3.9.1).

3.6.1 Mapping Two Vertical Distributions: `distribution`

Program `distribution` may be used to define the distribution of emissions along the vertical. It reads the vertical distribution of emissions in some input grid and maps this distribution on an output vertical grid. Thus it generates a file with the vertical distribution of emissions in the output grid. It is based on AtmoData function `ComputeVerticalDistribution`.

Running this program is not compulsory. Even if the vertical distribution of emissions is required to compute anthropogenic emissions (program `emissions`), the vertical distribution can be generated by other means (including "by hand").

	`[domain]`
`Nz`	Number of output vertical levels.
`Vertical_levels`	Path to the text file that stores the altitudes (in m) of output level interfaces (hence `Nz+1` values are read).
	`[EMEP]`
`Nz_in`	Number of input vertical levels.
`Vertical_levels`	Path to the text file that stores the altitudes (in m) of input level interfaces (hence `Nz_in+1` values are read).
`Vertical_distribution`	Path to the file with the input vertical distribution of emissions. This file should contain one line per emission sector. Each line contains the percentage of emissions at ground level (first column) and the percentage of emissions in each vertical level (`Nz_in` following columns).
`Polair_vertical_distribution`	Path to the output file where the output vertical distribution of emissions should be stored. The format is the same as in file `Vertical_distribution`.
`Nsectors`	Number of activity sectors.

3.6.2 Anthropogenic Emissions (EMEP): `emissions`

Program `emissions` processes an EMEP emission inventory and generates (anthropogenic) surface and volume emissions needed by Polair3D.

First you must download Expert emissions inventories from `http://webdab.emep.int/`. Download emissions for CO, NH3, NNMVOC, NOx, SOx, PM2.5 and PMcoarse and make sure to have a file for each species called `CO.dat`, `NH3.dat`, `NMVOC.dat`, `NOX.dat`, `SOX.dat`, `PM2.5.dat` and `PMcoarse.dat`.

Download files with the following options:

- for all countries;
- for the year of your choice (up to 2004);
- for all activity sectors (SNAP), note that emissions for the eleventh EMEP sector are better estimated with program `bio` (see Section 3.6.3) but that if they are included in the inventory, they are ignored;
- in format “Grid (50 km × 50 km), Semicolon-Separated”;
- for one species at a time.

In addition to the domain definition (Section 3.2.2), program `emissions` reads a configuration file such as `emissions.cfg`:

	`[paths]`
`Directory_surface_emissions`	Directory where the computed surface emissions are stored.
`Directory_volume_emissions`	Directory where the computed volume emissions are stored. This should be different from `Directory_surface_emissions` since files for surface emissions and volume emissions have the same names (species names).
	`[EMEP]`
`Polair_vertical_distribution`	File where the vertical distribution of emissions is stored. This file should contain one line per emission sector. Each line contains the percentage of emissions at ground level (first column) and the percentage of emissions in each vertical level (`Nz` following columns).
`Input_directory`	Directory containing EMEP emissions inventory.
`Hourly_factors`	File defining hourly factors (see below).
`Weekdays_factors`	File defining weekdays factors (see below).
`Monthly_factors`	File defining monthly factors (see below).
`Time_zones`	File defining the time zone for various countries.
`Nx_emep`	Number of cells along longitude (integer) in EMEP grid.
`Ny_emep`	Number of cells along latitude (integer) in EMEP grid.
`Ncountries`	Maximum code number of the countries covered by the inventory. If a code number in the inventory is greater than or equal to `Ncountries`, an error message is thrown.
`Species`	Names of inventory species.
`Nsectors`	Number of activity sectors.
`Urban_ratio`	Emission ratio for urban areas (see below).
`Forest_ratio`	Emission ratio for forest (see below).
`Other_ratio`	Emission ratio for other areas (see below).
	`[LUC]`
`File`	Path to land use cover file.
`x_min`	Longitude in degrees of the center of the lower-left cell in LUC grid.
`Delta_x`	Step length (in degrees) along longitude of LUC grid.

`Nx`	Number of cells along longitude (integer) in LUC grid.
`y_min`	Latitude in degrees of the center of the lower-left cell in LUC grid.
`Delta_y`	Step length (in degrees) along latitude of LUC grid.
`Ny`	Number of cells along latitude (integer) in LUC grid.
	`[Species]`
`N`	Number of emitted species.
`Aggregation`	Aggregation matrix file (relations of the emitted species to the real chemical species).
`Speciation_directory`	Directory in which, for each inventory species `XXX`, a file `XXX.dat` contains the speciation to real chemical species as function of the emission sector (columns).
`Deposition_factor_NH3`	Part of emitted NH_3 which is deposited right away.

The program `emissions` reads EMEP emissions inventory, multiplies them by temporal factors and interpolates them on Polair3D grids. EMEP emissions are read with AtmoData function `ReadEmep`. The spatial interpolation is performed with `EmepToLatLon`.

LUC file This file gives the land categories in GLCF description in a *domain that must contain your simulation domain* – but this LUC domain should not be too large because of computational costs. You can generate this file using program `extract-glcf` (see Section 3.3.5).

Urban, forest and "other" ratios Ratios `Urban_ratio`, `Forest_ratio` and `Other_ratio` enable to distribute emissions of an EMEP cell according to the type of land (urban, forests and other categories). For instance, in an EMEP cell, emissions are distributed so that the ratio between total urban emissions and total emissions is `Urban_ratio` on top of the sum of `Urban_ratio`, `Forest_ratio` and `Other_ratio`.

Temporal Factors EMEP emissions are provided as annual values. They are multiplied by temporal factors to estimate their time evolution (as function of month, week day and hour) in all emission sectors and in all countries.

Here are examples on how these factors should be provided:

- `monthly_factors.dat` gives the factors for each country index (`CC`), each activity sector (`SNAPsector`) and each month.

```
# Formate: CC SNAPsector  JAN    FEB    MAR    APR    MAY    JUN    JUL  ...
2  1  1.640  1.520  1.236  1.137  0.798  0.459  0.393 ..
```

- `weekdays_factors.dat` gives the factors for each country index (`CC`), each activity sector (`SNAPsector`) and each day of the week.

```
# Formate: CC SNAPsector  MON      TUE      WED      Thu      FRI      SAT      SUN
2  1  1.0159   1.0348   1.0399   1.0299   1.0298   0.9651   0.8846
```

- `hourly_factors.dat` gives the factor for each activity sector (`SNAPsector`) and each hour.

```
#  1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  ...
1  1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 ...
```

In addition to these factors a file called `time_zones.dat` is necessary. It gives the various countries in EMEP inventories and their time zone offset to GMT. Please note that the list of countries in the inventories may vary without warnings. If it happens, the code should raise an error and tell you what country code in `time_zones.dat` is unknown.

3.6.3 Biogenic Emissions for Polair3D Models: `bio`

Program `bio` computes biogenic emissions on the basis of meteorological fields and land use cover.

In addition to the domain definition (Section 3.2.2), below is the information required in the configuration for `bio` (see example `bio.cfg`):

	`[paths]`
`SurfaceTemperature`	Binary file where the surface temperature is stored.
`PAR`	Binary file where the photosynthetically active radiation is stored.
`LUC_file`	Binary file where the land use cover is stored.
`Land_data`	Data file giving emission factors for isoprene, terpenes and NO_x for all land categories defined in `LUC_file`. In this file, each line (which is not empty or does not start with "#") provides data for one land use category. For such a line, the first 55 characters are discarded: you may put the category number and description for convenience. Then four columns are read with the biomass density ($g\,m^{-2}$), and the emission factors for isoprene, terpenes and NO_x (in this order). Two examples are provided with `land_data_glcf.dat` and `land_data_usgs.dat`, to be used in combination with land use cover generated by `luc-glcf` or `luc-usgs` respectively.
`Directory_bio`	Directory where output biogenic emissions are stored.
	`[biogenic]`
`Delta_t`	Time step (in hours) for the output biogenic emissions. For simulations with Polair3D, anthropogenic emissions and biogenic emissions must have the same time step (that is, usually one hour).
`Rates`	Should emission rates be saved? These rates are not needed by chemistry-transport models.
`Terpenes`	Names of the species included in terpenes emissions. For RACM Stockwell et al. [1997], put `API` and `LIM`.
`Terpenes_ratios`	Distribution of terpenes emissions among species (entry `Terpenes`).

Biogenic emissions are computed according to Simpson et al. [1999]. Meteorological data is first interpolated in time so that its time step is `Delta_t` (section `[biogenic]`). Emission rates are then computed using AtmoData function `ComputeBiogenicRates` and emissions using `ComputeBiogenicEmissions`.

3.6.4 Biogenic Emissions for Castor Models: `bio-castor`

Program `bio-castor` is slightly different from `bio`, in particular regarding the data provided.

	[paths]
`SurfaceTemperature`	Binary file where the surface temperature is stored.
`WindModule_10m`	Binary file where the wind module at 10 m is stored.
`Attenuation`	Binary file where attenuation data are stored.
`SoilMoisture`	Binary file where the moisture of the ground is stored.
`ConvectiveVelocity`	Binary file where the convective velocity is stored.
`Land_data`	Land data in Chimere format.
`Directory_bio`	Directory where output biogenic emissions are stored.
	[biogenic]
`Minimum_wind_velocity`	Minimum value of `WindModule_10m`.
`Terpenes`	Species between which terpene emissions are distributed.
`Terpenes_ratios`	Ratio of the terpene emissions for each of the above species.

3.6.5 Sea Salt Emissions: `sea-salt`

Program `sea_salt` computes the emissions of sea-salt aerosols. Its options and parameters are given in `sea_salt.cfg`.

	[paths]
`Surface_wind_module_file`	Binary where the wind module at surface is stored.
`Directory_sea_salt`	Directory where sea-salt emissions are stored.
	[sea_salt]
`Threshold_radius`	Radius above which Monahan parameterization is used (in µm).
`Delta_t`	Time step for sea-salt emissions computation.
	[LUC]
`File`	File containing land use cover.
`Nb_luc`	Number of land categories.
`Sea_index`	Index of sea in land categories (recall that indices start at 0).
	[PM]
`Section_computed`	Should diameter classes bounds be computed? Otherwise they are read in `File_sections`.
`Diameter_min`	Minimum diameter if diameter classes bounds are computed.
`Diameter_max`	Maximum diameter if diameter classes bounds are computed.
`Nsections`	Number of diameter classes.
`File_sections`	File containing the diameter classes bounds if they are not computed.

3.7 Initial Conditions: `ic`

Climatological concentrations from Mozart 2 [Horowitz et al., 2003] are used to generate initial concentrations for photochemistry simulation with Polair3D.

Program `ic` has been tested with Mozart 2 output files downloaded on NCAR data portal at `http://cdp.ucar.edu`.

To download any data from the NCAR Community Data Portal you need to register. This is quite easy and fast but there is a second step. You also have to ask for an access to Mozart data specifically. This takes longer as the application has to be reviewed by someone but it should go without problems if you say that you need Mozart data to generate initial and boundary conditions for a CTM.

The data can be found in “ACD: Atmospheric Chemistry Models, Data Set and Visualization Tools”. If you registered, in this section of the site, you should be able to access “MOZART (Model for OZone And Related chemical Tracers)”. This opens a page with various informations about Mozart and in particular, in “Nested Collections” a link named “MOZART-2 MACCM3 Standard Simulation (v2.1)”. Click on this link.

At the time, the direct link is `http://cdp.ucar.edu/browse/browse.htm?uri=http://dataportal.ucar.edu/metadata/acd/mozart/mozart2/mozart_v2_1_maccm3.thredds.xml`, but if this changes a search for “MOZART MACCM3” should lead you to the page.

There are 38 files available in NCAR Community Data Portal (from hc0040.nc to hc0077.nc). One file gathers data for ten consecutive days. File hc0040.nc start at 26 December. Data in those files have been generated for a typical year, which means they can be use to generate boundary conditions for any year. See Equation 3.1to know how to determine the file you need given the date for which you want to generate the initial conditions.

In addition to the domain definition (Section 3.2.2), below is the information required in the configuration for `ic` (see example `ic.cfg`):

	`[ic_input_domain]`
`Date_ic`	Date for which initial conditions are generated.
`Nt`	Number of time steps in Mozart 2 files.
`Delta_t`	Time step of Mozart 2 files (in hours).
`Nx`	Number of grid points along latitude in Mozart 2 files (integer).
`Ny`	Number of grid points along longitude in Mozart 2 files (integer).
`Nz`	Number of vertical levels in Mozart 2 files (integer).
`Database_ic`	Directory where the Mozart 2 files are available. Mozart 2 filenames are in form `h00xx.nc` where `xx` is computed by the program according to the date `Date_ic`.
	`[ic_files]`

`Species`	File providing correspondence between the name of species in Mozart 2 files and the name of species in simulation. In this file, the first column contains Mozart 2 species. After each Mozart 2 species name, the corresponding output species (e.g., RACM species) is put, if any. If Mozart 2 species gathers two output species, put the names of all output species followed by their proportion in Mozart 2 bulk species. For instance, the line `C4H10 HC5 0.4 HC8 0.6` splits Mozart 2 species `C4H10` into `HC5` (40%) and `HC8` (60%). Three examples are provided: `preprocessing/bc/species_v1.dat`, `preprocessing/bc/species_v2.dat` and `preprocessing/bc/species_v3.dat`.
`Molecular_weight`	File providing the molecular weights of output species.
`Directory_ic`	Directory where the output initial conditions must be stored.

The name of the Mozart 2 files must be in the form `h00xx.nc` where `xx` is computed as shown in Equation 3.1.

$$xx = 40 + int\left[\frac{N_d + 6}{10}\right] \tag{3.1}$$

with N_d the number of days since the beginning of the year (0 for first January) and $int(x)$ represents the integral part of x.

Output results are in $\mu\text{g}\,\text{m}^{-3}$.

In case your Mozart 2 files do not satisfy this format (this may happen if Mozart files are updated on the NCAR data portal), you may modify the code or contact Polyphemus team at `polyphemus@cerea.enpc.fr`.

3.8 Boundary Conditions

3.8.1 Boundary Conditions for Gaseous Species: `bc`

Boundary condition for gaseous species are generated using Mozart 2 files. See Section 3.7 on how to get those files.

In addition to the domain definition (Section 3.2.2), below is the information required in the configuration for `bc` (see example `bc.cfg`):

	`[bc_input_domain]`
`Nt`	Number of time steps in Mozart 2 files.
`Delta_t`	Time step of Mozart 2 files (in hours).
`Nx`	Number of grid points along latitude in Mozart 2 files (integer).
`Ny`	Number of grid points along longitude in Mozart 2 files (integer).
`Nz`	Number of vertical levels in Mozart 2 files (integer).
	`[bc_files]`
`Directory_bc`	Directory where the output boundary conditions must be stored.

`Species`	File providing correspondence between the name of species in Mozart 2 files and the name of species in simulation. In this file, the first column contains Mozart 2 species. After each Mozart 2 species name, the corresponding output species (e.g., RACM species) is put, if any. If Mozart 2 species gathers two output species, put the names of all output species followed by their proportion in Mozart 2 bulk species. For instance, the line `C4H10 HC5 0.4 HC8 0.6` splits Mozart 2 species `C4H10` into `HC5` (40%) and `HC8` (60%). Three examples are provided: `preprocessing/bc/species_v1.dat`, `preprocessing/bc/species_v2.dat` and `preprocessing/bc/species_v3.dat`.
`Molecular_weight`	File providing the molecular weights of output species.

Program `bc` processes an entire Mozart 2 output file. If this file contains concentrations for 10 days, the program generates boundary conditions for 10 days.

The program must be launched with:

```
./bc bc.cfg ../general.cfg /net/libre/adjoint/mallet/mozart/h0067.nc
```

The last argument is the path to the Mozart 2 file. You have to select the file to use according to date as what is shown in Section 3.7.

The results are in $\mu\mathrm{g\,m^{-3}}$. They are stored as `&f_&c.bin` where `&f` is replaced by the name of the species and `&c` by the direction associated with the boundary condition (x, y or z). For example, the concentrations in `O3_x.bin` are interpolated at both ends of the domain along x, for all grid points along y and z.

3.8.2 Boundary Conditions for Aerosol Species: `bc-gocart`

Boundary conditions for aerosol species are obtained using Gocart model[†2] thanks to the program `bc-gocart`.

Gocart format and conventions

Gocart model usually provides files with the following naming convention:

file name	signification
yyyymm.XX.vs.g	6-hourly concentrations in $\mathrm{g\,m^{-3}}$.
yyyymm.XX.vs.g.day	daily averaged concentrations in $\mathrm{g\,m^{-3}}$.
yyyymm.XX.vs.g.avg	monthly averaged concentrations in $\mathrm{g\,m^{-3}}$.

where *yyyymm* is the year and month (e.g., 200103), *XX* is the Gocart species, which can be either SU (sulfur), CC (carbonaceous), DU (dust), SS (sea-salt), and *vs* is the version (e.g., STD.tv12).

Gocart species may have further speciations:

- **SU** (sulfur): Total 4, 1-DMS, 2-SO2, 3-SO4, 4-MSA.
- **CC** (BC+OC): Total 4, 1-hydrophobic BC, 2-hydrophobic OC, 3-hydrophilic BC, 4-hydrophilic OC.

[†2] `http://code916.gsfc.nasa.gov/People/Chin/gocartinfo.html`

- **DU** (dust): Total 5, 1-Re=0.1-1, 2-Re=1-1.8, 3-Re=1.8-3, 4-Re=3-6, 5-Re=6-10 µm. The first group (0.1-1 µm) contains the following subgroups:
  - 0.10-0.18 µm (fraction = 0.01053)
  - 0.18-0.30 µm (fraction = 0.08421)
  - 0.30-0.60 µm (fraction = 0.25263)
  - 0.60-1.00 µm (fraction = 0.65263)
- **SS** (sea-salt): Total 4, 1-Re=0.1-0.5, 2-Re=0.5-1.5, 3-Re=1.5-5, 4-Re=5-10 µm.

The data format of Gocart files is “direct access binary, 32 bits, big endian”. As an example, here is how they should be read in Fortran 77 language:

```
dimension Q(imx,jmx,lmx)
do k=1,nstep
 do n=1,nmx
  read(unit) nt1,nt2,nn,Q
 end do
enddo
```

where

- imx = total number of longitudinal grid (144),
- jmx = total number of latitudinal grid (91),
- lmx = total number of vertical layers (version dependent),
- nmx = total number of species (4 or 5, see species list above),
- nt1 = yyyymmdd after 2000 (year-month-day, e.g., 20010201), or yymmdd before 2000 (e.g., 970101)
- nt2 = hhmmss (hour-minute-second, e.g., 120000)
- nn = tracer number (see species list above)
- Q = 3-dimensional concentration of tracer nn
- nstep = total time step (e.g., in 200101, nstep=4*31 for 4-times/day, nstep=31 for daily average files, and nstep=1 for monthly average files).

Important

- If you plan to read Gocart data on your own, do not forget to translate files from big endian to little endian if necessary.
- The conventions and format of Gocart files may change in the future.

Fields resolution

The horizontal resolution of Gocart fields is 2 degree latitude × 2.5 degree longitude, except at the poles where latitudinal resolution is 1 degree. In other words the longitude interval is $[-180 : 2.5 : 177.5]$ (144 cells) and the latitude one is $[-89.5 - 88 : 2 : 8889.5]$ (91 cells).

The vertical resolution is given as a given number of vertical sigma levels. The number of vertical levels depends of the year :

- **1980-1995**: 20 sigma layers centered at 0.993936, 0.971300, 0.929925, 0.874137, 0.807833, 0.734480, 0.657114, 0.578390, 0.500500, 0.424750, 0.352000, 0.283750, 0.222750, 0.172150, 0.132200, 0.100050, 0.0730000, 0.0449750, 0.029000, 0.00950000
- **1996-1997**: 26 vertical sigma layers centered at 0.993935, 0.971300, 0.929925, 0.875060, 0.812500, 0.745000, 0.674500, 0.604500, 0.536500, 0.471500, 0.410000, 0.352500, 0.301500, 0.257977, 0.220273, 0.187044, 0.157881, 0.132807, 0.111722, 0.0940350, 0.0792325, 0.0668725, 0.0565740, 0.0447940, 0.0288250, 0.00997900
- **2000-2002**: 30 vertical sigma layers centered at 0.998547, 0.994147, 0.986350, 0.974300, 0.956950, 0.933150, 0.901750, 0.861500, 0.811000, 0.750600, 0.682900, 0.610850, 0.537050, 0.463900, 0.393650, 0.328275, 0.269500, 0.218295, 0.174820, 0.138840, 0.109790, 0.0866900, 0.0684150, 0.0539800, 0.0425750, 0.0335700, 0.0239900, 0.0136775, 0.00501750, 0.00053000

Gocart files processing

Gocart files are handled by `bc-gocart` program which takes 4 arguments:

```
./bc-gocart ../general.cfg bc-gocart-CC.cfg 200101.CC.STD.tv15.g.day 200101
```

where

- `../general.cfg` is the general configuration file,
- `bc-gocart-CC.cfg` is the configuration file for CC Gocart species,
- `200101.CC.STD.tv15.g.day` is the Gocart file
- `200101` is the date of Gocart file, this file corresponds to daily carbonaceous values during month of January 2001.

The gocart configuration file `bc-gocart-CC.cfg` provides all necessary informations to read Gocart fields and how to translate them into `polair3d` species.

	`[paths]`
`Temperature`	Meteorological file of temperature.
`Directory_bc`	Directory where output will be written.
	`[bc_input_domain]`
`x_min`	Minimum longitude in Gocart resolution.
`y_min`	Minimum latitude in Gocart resolution.
`Delta_x`	Gocart longitude resolution.
`Delta_y`	Gocart latitude resolution.
`Nx`	Number of grid cells in the longitude Gocart axe.
`Ny`	Number of grid cells in the latitude Gocart axe.
`Nz`	Number of Gocart vertical layers.

`Sigma_levels`	File where are written the center of Gocart sigma levels.
`Scale_height`	Scale height in meter.
`Surface_pressure`	Surface pressure in atm.
`Top_pressure`	Pressure at top of Gocart level (in atm).

There are two more sections in configuration file.

The first one is `[input_species]`. Each non blank line of this section corresponds to one speciation of Gocart species, e.g. CC is sub-divided in CC-1, CC-2, CC-3, CC-4. The range after the delimiter ":" is the aerosol size range (in SIμm) to which this sub-species apply. Most of the time this is the whole aerosol size range of `polair3d` model (e.g. 0.1 – 10.0), but in the case of dust (DU) each sub-species may correspond to a precise part of the `polair3d` aerosol size range, see configuration file `bc-gocart-DU.cfg` for an illustration.

The second section is `[output_species]`. Each non blank line of this section corresponds to one aerosol species of `polair3d` model. The columns after ":" delimiter correspond to the Gocart sub-species. Therefore the number of line in previous section must equal the number of column after ":" delimiter. The numbers in these columns are the fraction (between 0.0 – 1.0) of given Gocart sub-species that will contribute to given model species. As an example in `bc-gocart-CC.cfg` the first line

```
PBC:    1.      0.      1.      0.
```

means that sub-species CC-1 and CC-3 will fall into PBC `Polair3D` species, and nowhere else. In the same way the following line

```
PPOA:   0.      0.4     0.      0.4
```

means that PPOA species is composed of 40% of CC-2 and 40% of CC-4.

Important The Gocart files are proceeded month by month.

- The beginning date of computation is the one provided in `../general.cfg` if the beginning month is equal to the Gocart month, beginning of Gocart month otherwise.
- The end date of computation is provided by the length of file `Temperature`. If this length exceeds the Gocart month, the end date is set to end of Gocart month.
- If some boundary files already exist, the program `bc-gocart` will not overwrite them but append its result to each.

For example if you want to compute boundary conditions between 15th of April to 15th of June 2001, you would launch `bc-gocart` three times:

```
./bc-gocart ../general.cfg bc-gocart-CC.cfg 200104.CC.STD.tv15.g.day 200104
./bc-gocart ../general.cfg bc-gocart-CC.cfg 200105.CC.STD.tv15.g.day 200105
./bc-gocart ../general.cfg bc-gocart-CC.cfg 200106.CC.STD.tv15.g.day 200106
```

The file given in `Temperature` field of configuration file `bc-gocart-CC.cfg` must respectively start and end exactly at dates 2001-04-15 and 2001-06-15.

The python script `bc-gocart.py` provides an easy way to compute boundary conditions without worrying about how many times to launch `bc-gocart`. In the last example, one should simply launch:

```
./bc-gocart.py ../general.cfg 04 06
```

where 04 and 06 respectively stands for the first and last month to treat. Pay attention that some paths must be supplied inside this script (paths of Gocart configuration and data files) for it to work.

Remarks

Gocart **does not** provide any boundary conditions for nitrate and ammonia, you have to compute them on your own. Nevertheless a quick way to compute boundary conditions for ammonia is to apply electroneutrality to already computed aerosol boundary conditions from Gocart (or whatever else in fact). This can be done by `bc-nh4` program which takes two arguments:

```
./bc-nh4    ../general.cfg bc-nh4.cfg
```

The electroneutrality equation is set in configuration file `bc-nh4.cfg`.

3.9 Preprocessing for Gaussian Models

3.9.1 Program `discretization`

The aim of this program is to discretize a line emission in the case of a continuous source (plume source) or an instantaneous one (puff source). It reads a line source given by two points or more, and gives in return the discretized source. The output data is a list of point sources whose coordinates have been calculated given the line coordinates and the number of points to discretize the line, or the source velocity in the case of a moving source (for puff sources only).

The program `discretization` is launched with one configuration file. The reference configuration file is `discretization.cfg`. It contains the following information:

	`[trajectory]`
`Trajectory_file`	Path to the data file that contains the line coordinates.
`Np`	Number of points to calculate on the discretized trajectory. Used only when the source is continuous or when it is not moving.
`Delta_t`	Time step to calculate the discretized trajectory in the case of a moving source.
	`[source]`
`Source_type`	Source type: `puff` or `continuous`.
`Species_name`	Name of the species emitted by the source (only one species for one line source).
	`[plume-source]`
`Rate`	Source rate (in mass per second) of the line source.
`Velocity`	Velocity of the gas or aerosol emitted by the source (in $\mathrm{m\,s^{-1}}$).
`Temperature`	Temperature of the gas or aerosol emitted by the source (Celsius degrees).
`Section`	Section of the source in m^2.
	`[puff-source]`
`Quantity`	Total mass released on the line source (mass unit).
`Source_velocity`	Source velocity (in $\mathrm{km\,h^{-1}}$) (0. for non mobile sources).
`tinit`	Release time for the first trajectory point (in s).
	`[output]`
`With_comment`	Are comments written?
`Source_file`	Path to the data file where the list of sources will be written.

The associated data file (reference: `line-emission.dat`) contains the coordinates of the line source to be discretized. The line source is a continuous line made of segments. Each segment is defined by two end points. The data file contains three columns corresponding to the coordinates (X, Y, Z in meters) of all end points. It contains at least the coordinates of two points.

This is an example of data file, defining a straight line emission between two points:

```
#X(m)   Y(m)    Z(m)
0.      0.     30.
20.     0.     30.
```

The output data file contains a list of point sources. All points have the same source data and their coordinates have been calculated by the program. It is presented as a list of sections named `[source]`, each section containing the coordinates and other data for one point source.

3.9.2 Programs `gaussian-deposition` and `gaussian-deposition_aer`

The aim of these programs is to calculate the scavenging coefficient and the deposition velocity of the species. The program `gaussian-deposition` is used when all species are gaseous species, and `gaussian-deposition_aer` is used when some or all species are aerosol species. The input data are meteorological data and species data, and the output file is a file containing meteorological data and the scavenging coefficients and deposition velocities of all species. This file can be used as input meteorological file for the programs `plume` and `puff` for gaseous species, or `plume_aer` and `puff_aer` in the case of aerosol species.

Program gaussian-deposition

Configuration File The program `gaussian-deposition` is launched with one configuration file and two input files. The configuration file contains the path to the two input files and to the output file. The reference configuration file is `gaussian-deposition.cfg`. It contains the following information:

	`[data]`
`Species`	Path to the data file that contains the species data.
`Meteo`	Path to the data file that contains the meteorological data.
	`[scavenging]`
`Type`	Parameterization to be used to calculate the scavenging coefficients.
	`[deposition]`
`Type`	Parameterization to be used to calculate the deposition velocities.
	`[output]`
`With_comment`	Are comments written in the output file? (put yes or no).
`Output_file`	Path to the file where the output data are written.

The parameterization type for the scavenging coefficient can be chosen between:

- `none`: the scavenging coefficient is set to 0. for all species
- `constant`: the scavenging coefficient is constant for one given species and entered in the

species file.

- `belot`: the scavenging coefficient is calculated with a Belot parameterization. In that case, the input data are a rainfall rate given in the meteorological data file, and coefficients a and b given for each species in the species file.

Concerning the deposition velocity, the type can be chosen between:

- `none`: the deposition velocity is set to 0. for all species
- `constant`: the deposition velocity is constant for one given species and is given in the species file.

Input Files There are two input data files for this program: the meteorological data file (reference: `meteo.dat`) and the species file (reference: `species.dat`).

1. Meteorological data file: it contains as many sections as there are meteorological situations. For each situation, meteorological data (temperature, wind ...) are given as described in section 5.1.5. They will be written unchanged in the output file which will be the meteorological data file of the main program. Other meteorological data might be needed, depending on the chosen parameterization to compute scavenging coefficients and deposition velocities. Currently, the only parameterization that needs other information is the Belot parameterization. If the type `belot` is chosen for the calculation of the scavenging coefficient, a rainfall rate must be provided (in $\mathrm{mm\,h^{-1}}$). If the chosen type is `constant` or `none`, the rainfall rate or other information can be provided but will be ignored by the program. So, the meteorological data file finally looks like this:

```
[situation]

# Temperature (Celsius degrees)
Temperature = 10.

# Wind angle (degrees)
Wind_angle = 30.

# Wind speed (m/s)
Wind = 3.0

#Inversion height (m)
Inversion_height = 1000.

# Stability class
Stability = D

# Rainfall rate (mm/hr)
Rainfall_rate = 1.
```

In this example, there is only one meteorological situation described. Others can be added simply by adding similar sections `[situation]` at the end of the file.

2. Species data file: it contains several sections, but not all are needed for the preprocessing. The needed sections are:

   - `[species]` Contains the list of all species.
   - `[scavenging]` Contains the list of the species for which scavenging occurs. The scavenging coefficient of the others is set to 0.
   - `[deposition]` Contains the list of the species for which deposition occurs. The deposition velocity of the others is set to 0.
   - `[scavenging_constant]` This section is needed when the type of parameterization chosen for the scavenging is `constant`. It contains the name of a species followed by the value of its scavenging coefficient (in s^{-1}). Only one species per line must be provided. All species listed in the section `[scavenging]` must be present (the order is not important), the others will be ignored.
   - `[scavenging_belot]` This section is needed when the type of parameterization chosen for the scavenging is `belot`. It contains the name of a species followed by two values corresponding to the coefficients a and b respectively in the Belot parameterization. Only one species per line must be provided. All species listed in the section `[scavenging]` must be present, the others will be ignored.
   - `[deposition_constant]` This section is needed when the type of parameterization chosen for the deposition is `constant`. It contains the name of a species followed by the value of its deposition velocity (in $m\,s^{-1}$). Only one species per line must be provided. All species listed in the section `[deposition]` must be present, the others will be ignored.

   A species file might look like this:

```
[species]

Caesium       Iodine


[scavenging]

Iodine        Caesium

[deposition]

Caesium        Iodine


[scavenging_constant]

Caesium: 1.e-4
Iodine: 1.e-4
```

```
[scavenging_belot]

Caesium: 2.8e-05  0.51
Iodine: 7e-05  0.69


[deposition_constant]

Caesium: 0.05e-2
Iodine: 0.5e-2
```

Output File The output data file contains as many sections as there are meteorological situations. Each section `[situation]` contains the temperature, wind angle, wind speed, inversion height and stability class that are provided. In addition, it contains the list of all species followed by their scavenging coefficient, and the list of all species followed by their deposition velocity. It looks like this:

```
[situation]

# Temperature (Celsius degrees)
Temperature = 10

# Wind angle (degrees)
Wind_angle = 30.

# Wind speed (m/s)
Wind = 3.

# Inversion height (m)
Inversion_height = 1000

# Stability class
Stability = D

# Scavenging coefficient of the species (s^-1)
 Scavenging_coefficient =
 Caesium  6.36257e-05  Iodine  0.000212514

# Deposition velocity of the species (m/s)
 Deposition_velocity =
 Caesium  0.0005  Iodine  0.005
```

Program gaussian-deposition_aer

The program `gaussian-deposition_aer` works the same way as the program `gaussian-deposition`, except that there are some more information specific to the aerosol species. The input and output files are the same as described in the section about `gaussian-deposition`, so in this section we will only describe the data that are added to the files described previously. One input file is needed in addition to the meteorological data and species data files. It is the diameter file (reference: `diameter.dat`) which contains the diameters of the aerosol particles.

Configuration File In the configuration file, the following information are added:

	`[data]`
`Diameter`	Path to the data file that contains the particle diameters.
	`[scavenging]`
`Type_aer`	Parameterization to be used to calculate the scavenging coefficients for aerosol species.
`Value`	Values to be used for a Slinn parameterization (choose between `best_estimate` and `conservative`.
	`[deposition]`
`Type_aer`	Parameterization to be used to calculate the deposition velocities for aerosol species.

The parameterization type for the scavenging coefficient of aerosol species can be chosen between:

- `none`: the scavenging coefficient is set to 0. for all aerosol species
- `constant`: the scavenging coefficient is constant for one given diameter and entered in the species file.
- `slinn`: the scavenging coefficient is calculated with a Slinn parameterization. In that case, the only input data that are used are the rainfall rate and the particle diameters.

Concerning the deposition velocity, the type can be chosen between:

- `none`: the deposition velocity is set to 0. for all species
- `constant`: the diffusive part of the deposition velocity is constant for one given diameter and entered in the species file. The gravitational settling velocity is calculated for each particle, given the density and the diameter (provided in the species file) and the pressure and temperature (provided in the meteorological data file).

Input Files

1. Meteorological data file: it is the same as the one for `gaussian-deposition`. If the parameterization type for the deposition velocity calculation is `constant`, the pressure must be provided (in Pa).

2. Diameter file: it contains the list of particle diameters (in µm). The first number is the diameter of index 0, the second of index 1, and so on. This is an example of diameter file:

```
#Diameter (micrometer)
[diameter]
0.1
1.
```

The diameter of index 0 corresponds to the value 0.1 µm, the diameter of index 1 to the value 1. µm and so on. When referring to a given diameter in the other data files, one has to give the corresponding index. Note that there is only one diameter file for all aerosol species. Therefore all particulate species are assumed to have the same diameter distribution. The diameter file can also be the main configuration file. In that case, the section `[diameter]` is simply added to the main configuration file.

3. Species file: it is the same as described before, but the sections described for `gaussian-deposition` concern only gaseous species. All data concerning aerosol species are added in the following sections:

   - `[aerosol_species]` Contains the list of all aerosol species.
   - `[scavenging_constant_aer]` This section is needed when the type of parameterization chosen for the scavenging for aerosol species is `constant`. In that case, the scavenging coefficient is assumed to be constant for one particle diameter. So the section contains the index of one diameter followed by the corresponding value of the scavenging coefficient (in s^{-1}). Only one diameter per line must be provided.
   - `[deposition_constant_aer]`This section is needed when the type of parameterization chosen for the deposition of aerosol species is `constant`. It contains the index of a diameter followed by the value of its deposition velocity (in $m\,s^{-1}$). Only one diameter per line must be provided.
   - `[density_aer]` It contains the density of the aerosol species. That is, the name of each aerosol species followed by the corresponding density (in $kg\,m^{-3}$). Only one species per line must be provided. This section is needed in order to calculate the gravitational settling velocity of a particle. The calculated deposition velocity of one species of a given diameter is therefore a combination of the diffusive part given in the section `[deposition_constant_aer]` and the gravitational settling velocity calculated by the program.

   Note that while some gaseous species might not be concerned by scavenging or deposition, the loss processes are assumed to occur for all aerosol species. Therefore, there is no need of a section containing the species for which scavenging or deposition occur in the case of aerosol species, as it is the case for gaseous species. Here is an example of species file containing the sections dedicated to aerosol species:

```
[aerosol_species]

aer1
aer2
aer3


[scavenging_constant_aer]

# Scavenging coefficient for aerosol species ( Unit: seconds^(-1) )
# Depends on the diameter (first value: diameter index in file diameter.dat).
# Only one diameter per line.

0: 1.e-4
```

```
1: 2.e-4


[deposition_constant_aer]

# Dry deposition velocity (diffusive part) of the species (Unit: m/s)
# Depends on the diameter
# Only one diameter per line.

0: 0.05e-2
1: 0.5e-2


[density_aer]

# Particle density (aerosol species) (kg/m^3)
# Only one species per line.

aer1: 1.88
aer2: 1.
aer3: 4.93
```

Output File The output file is the same file as the one for `gaussian-deposition`, except that the scavenging coefficients and deposition velocities of aerosol species are also written. One coefficient corresponds to a given species of a given diameter. It is written as “species-name”-”diameter-index” followed by the value of the corresponding scavenging coefficient (or deposition velocity). The following example corresponds to a case with two gaseous species named “gas1” and “gas2” and three aerosol species named “aer1”, “aer2” and “aer3”. The diameter file is the same as displayed before, that is, contains two diameters. The output file looks like this:

```
[situation]

# Temperature (Celsius degrees)
Temperature = 10

# Pressure (Pa)
Pressure = 101325

# Wind angle (degrees)
Wind_angle = 30

# Wind speed (m/s)
Wind = 3

# Inversion height (m)
Inversion_height = 1000

# Stability class
Stability = D
```

```
# Scavenging coefficient of the gaseous species (s^-1)
Scavenging_coefficient =
gas1  0.0001    gas2  0.0001

# Deposition velocity of the gaseous species (m/s)
 Deposition_velocity =
 gas1  0.0005     gas2  0.005

# Scavenging coefficient of the aerosol species (s^-1)
 Scavenging_coefficient_aer =
 aer1-0  5.95238e-05      aer1-1  5.95238e-05    aer2-0  5.95238e-05
 aer2-1  5.95238e-05      aer3-0  5.95238e-05    aer3-1  5.95238e-05

# Deposition velocity of the aerosol species (m/s)
 Deposition_velocity_aer =
 aer1-0  2.11708     aer1-1  6.69479      aer2-0  1.54404
 aer2-1  4.88268     aer3-0  3.42833      aer3-1  10.8413
```

The value following “aer1-0” corresponds to the calculated coefficient for the species “aer1” and the diameter of index 0, that is, in the case of our diameter file, the diameter equal to 0.1 µm. The value following “aer1-1” corresponds to the coefficient for the species “aer1” and the diameter of index 1, that is, equal to 1 µm, and so on.

Chapter 4

Drivers

4.1 BaseDriver

BaseDriver is configured with a file which contains the displaying options for the simulation.

`[display]`	
`Show_iterations`	If activated, each iteration is displayed on screen.
`Show_date`	If activated, the starting date of each iteration is displayed on screen in format `YYYY-MM-DD HH:II` (notations from Section 2.2.7).

4.2 PlumeDriver

It is the driver dedicated to the Gaussian plume model. The associated configuration file is the same as the one for the BaseDriver, and it is usually part of the model configuration file described in Section 5.1. The associated input data file describes the meteorological data (reference: `gaussian-meteo.dat`) for gaseous species and `gaussian-meteo_aer.dat` for aerosol and/or gaseous species. The meteorological data file contains the meteorological data that are needed. It can be the output file of the preprocessing program `gaussian-deposition`.

The meteorological data file describes one or several meteorological situations. For each situation, the driver calls the model to calculate the concentrations, that is, the stationary solution for the given meteorological situation. It is associated with two models: the GaussianPlume model for gaseous species only (described in Section 5.1) and the GaussianPlume_aer model which is the same model for aerosol and/or gaseous species (see Section 5.2).

4.3 PuffDriver

It is the driver dedicated to the Gaussian puff model. The associated configuration file is the same as the one for the BaseDriver, and it is usually part of the model configuration file described in Section 5.3. The associated input data file describes the meteorological data. It is the same file as for the plume model.

For each meteorological situation, the driver calculates the concentrations that depend on time. That is, for a given situation, it makes a loop on time and calls the model at each time step to calculate the current concentrations. It is associated with two models: the GaussianPuff model for gaseous species only (described in Section 5.3) and the GaussianPuff_aer model which is the same model for aerosol and/or gaseous species (see Section 5.4).

4.4 StationaryDriver

This driver, as the Gaussian drivers presented before, is used to perform a simulation at local scale, the only difference being that in that case an Eulerian model is used.

An additional section `[stationary]` is necessary in the configuration file:

	`[stationary]`
`Nt`	Number of stationary steps.
`Delta_t`	Time-step between stationary steps.

4.5 MonteCarloDriver

The Monte Carlo driver performs several simulations with perturbed input data. The input data are perturbed by the `PerturbationManager` (see Section 4.11). The simulation outputs are saved with the unit saver of type `domain_ensemble_forecast` (see Section 4.9.2). The configuration file for this driver should contain:

	`[MonteCarlo]`
`Number_ensemble`	The number of samples.
	`[perturbation_management]`
`Configuration_file`	Name of the file that contains perturbations configuration (see Section 4.11 about `PerturbationManager`).

4.6 PlumeInGridDriver

Warning: This part is about a recent development in Polyphemus, that has not been fully tested yet. The Plume in Grid model is present in this version for demonstration purposes, but it is not stable yet.

The base files for Plume in Grid model are `PlumeInGridDriver.hxx` and `PlumeInGridDriver.cxx`. Basically, in spite of its name, it is a model, but it can be used as a driver. The Plume in Grid model uses both an Eulerian model and a Gaussian Puff model. It processes major point emissions first with the Puff model, then feeds the puffs back to the Eulerian model when their size is large enough. Apart from this special treatment of point sources, the Eulerian simulation is performed as if BaseDriver were used. The Eulerian model can be, for example, Polair3dAerosol, Polair3dChemistry, or CastorChemistry.

Use basic configuration files for the Eulerian model. The changes to make are:

- in the main configuration file, there is a new section named `[gaussian]` where the name of a configuration file for the puff model has to be provided (field `file_gaussian`).

- in the data configuration file, there is a new section `[gaussian_meteo]` that provides more meteorological fields than the ones used for Eulerian model. It consists in cloudiness and solar radiation data that are used to compute Pasquill stability class. In case those fields are not available, put whatever value you want for those fields and use the Doury parameterization for `sigma`, so that stability class will not be used.

- in the data configuration file, there is a new section `[plume-in-grid]` that provides the file containing the sources to be treated with plume in grid model (field `[file_source]`. It has the same form as a point source file.

Note that there is no point emission for the Eulerian model, since the source file is directly read by the plume in grid model (it is the file provided in section `[plume-in-grid]` of the data file). Hence, if you set the option `With_point_emission` to "yes" and provide the same configuration file for point source, the source file will be read and treated twice: once by the Eulerian model and once by the Plume in Grid model.

In addition, the Gaussian puff model needs the usual configuration files. However, few of their information are actually used, since most information are directly provided by the Plume in Grid model.

The most important information given in `puff.cfg` is the time step `Delta_t`. 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:

$$N = int\left[\frac{\Delta t_{Eulerian}}{\Delta t_{Gaussian}}\right] \tag{4.1}$$

You are strongly advised to use a time step for the Gaussian model smaller than the one for the Eulerian model in order to perform several iterations of the Gaussian model. Other information about the domain are still read but not used, except the land category that is used for Briggs parameterization.

In addition, it also reads all options and parameterizations. 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. Note that there is no need to provide a meteorological file or a source file, since those information are fed to the puff model by the Plume in Grid model. The species file is still read. It is advised to use the same as the species file for Eulerian model. Levels file and saver file are still read but not used.

4.7 Data Assimilation Drivers

4.7.1 AssimilationDriver

It is the base driver from which all data assimilation drivers are derived. Data assimilation is the concept and methods that estimate model state from diverse available sources, e.g. model simulations, observations and statistics information, aiming at and validated by a better prediction. Data assimilation methods can roughly be catalogued into variational and sequential ones. For the former the variational principle applies. The objective can be defined by the discrepancy between model simulation and a block of observations, usually combined with a priori background knowledge. This can be theorized and solved efficiently by optimal control theory (`FourDimVarDriver`). The sequential methods make use of observations instantaneously. This is a filtering process, and filter theory (linear or nonlinear) applies (`OptimalInterpolationDriver`, `EnKFDriver` and `RRSQRTDriver`).

A typical data assimilation system consists of three components: model (physics), data (observation), and assimilation algorithm. The data assimilation drivers organize model and data to perform assimilations. The associated configuration file is an extension of that of the model configuration file exemplified in Section 5.8. For data part, it has an additional section `[observation_management]`,

[observation_management]	
`Configuration_file`	Path to the file containing the configuration of the observation management. In the distribution (directory `driver/example/assimilation/`), choose between observation.cfg (to use observations) and observation-sim.cfg (to use simulated observations).

The value of `Configuration_file` can be set to `observation.cfg` if you use `GroundObservationManager` (see Section 4.10.1) and to `observation-sim.cfg` if you use `SimObservationManager` (see Section 4.10.2).

The data assimilation experiments are controlled by the following options.

[domain]	
`Nt`	Number of time steps for the whole simulation (assimilation and prediction).
	[data_assimilation]
`Nt_assimilation`	Number of time steps for the assimilation period.

`Nt` is supposed to be greater than or equal to `Nt_assimilation`. From time step #`0` to time step #`Nt_assimilation-1` , assimilation is performed; and from step #`Nt_assimilation` to step #`Nt-1`, prediction is performed.

In many cases such as data assimilation and ensemble prediction, perturbed model simulations are needed. Perturbations are managed by `PerturbationManager` (see Section 4.11) reading an additional section,

[perturbation_management]	
`Configuration_file`	Name of the file that contains perturbation configurations.

4.7.2 OptimalInterpolationDriver

It is the driver dedicated to data assimilation applications using optimal interpolation algorithm. The optimal interpolation algorithm estimates model state status by minimizing the error variance of the estimation (called *analysis* in data assimilation terminology). It searches for a linear combination between *background* state (model simulations) and the *background departures*. The background departures are defined as the discrepancies between observations and background state. It involves with observation managements (described in Section 4.10) and storage managements of forecast and analysis results (see for instance Section 4.9.4). The background error covariance matrix can be either diagonal or generated by Balgovind correlation functions (see Section 5.8).

4.7.3 EnKFDriver

It is the driver dedicated to data assimilation applications using ensemble Kalman filter algorithm. It consists of two steps: forecast and analysis. It differs from optimal interpolation in that the *background error covariance* is flow-dependent and approximated by an ensemble of perturbed model forecast. The algorithm parameters are set in section `[EnKF]`.

[EnKF]	
`Number_ensemble`	The number of samples in the ensemble.
`With_observation_perturbation`	If observations are perturbed for consistent statistics for analyzed ensemble.
`With_positivity_requirement`	If positivity of the assimilated species concentrations is required.

The generation of the ensemble is detailed in `PerturbationManager` configurations (see Section 4.11).

4.7.4 RRSQRTDriver

It is the driver dedicated to data assimilation applications using reduced rank square root Kalman filter algorithm (RRSQRT). It consists of two steps: forecast and analysis. The *background error covariance* is flow-dependent and approximated by an explicit low rank representation. The algorithm parameters are set in section `[RRSQRT]`.

[RRSQRT]	
`Number_analysis_mode`	The expected rank number (column number) of the square root (mode matrix) of forecast error covariance matrix.
`Number_model_mode`	The number of the columns of the square root of model error covariance matrix to be added to the mode matrix.
`Number_observation_mode`	The number of the columns of the square root of observation error covariance matrix to be added to the analyzed mode matrix.
`Propagation_option`	The option for the forecast of the columns of mode matrix. Only finite difference is supported.
`Finite_difference_perturbation`	Perturbation coefficient for mode forecast using finite difference method; set to 1.

Model perturbations are employed to generate the columns of the square root of model error covariance matrix (see `PerturbationManager` configurations in Section 4.11).

4.7.5 FourDimVarDriver

It is the driver dedicated to data assimilation applications using four-dimensional variational assimilation algorithm (4D-Var). The assimilation period is from time step `0` to `Nt_assimilation-1`. The optimal model state at initial time step is obtained by minimizing an objective function which is the background departure plus discrepancy between model simulations and observations during the assimilation period. The model is supposed to be perfect, thus no model error terms are considered. The gradient of the objective function is calculated efficiently using adjoint model of the underlying model. The algorithm parameters are set in section `[4DVar]`.

[4DVar]	
`Display_precision`	Display precision for optimization results.
`Jb_file`	Name of the file that saves background departure during optimization.

`Jo_file`	Name of the file that saves observation discrepancy during optimization.
`Gradient_norm_file`	Name of the file that saves gradient norms during optimization.
`With_trajectory_management`	If the trajectory of model integration is saved to disk for adjoint integration.
`Trajectory_delta_t`	Trajectory time step in seconds.
`Trajectory_file`	Name of the file that saves model trajectory.

The parameters for numerical optimization algorithm are set in section `[optimizer]`

`[optimizer]`	
`Type`	Type of optimization solver; only BFGS is supported.
`Maximal_iteration`	The number of the maximal iteration for numerical optimization.
`Display_iterations`	If the optimization results during iteration are displayed.

4.8 Drivers for the verification of adjoint coding

The three drivers `AdjointDriver`, `GradientDriver`, `Gradient4DVarDriver` are dedicated to the verifications of adjoint model. The gradient of a given objective function calculated by adjoint model is compared with the gradient calculated by finite difference. The following ratio is checked

$$\rho = \frac{J(x + \alpha h) - J(x)}{\alpha \langle \nabla_x J, h \rangle}$$

where J is the objective function, x is the control variable, h is the perturbation direction, α is the perturbation coefficient, and $\nabla_x J$ is the gradient calculated by adjoint model, $< \quad >$ denotes inner product. With $\alpha \to 0$, the ratio ρ is supposed to approach to 1 with high precision, then becomes unstable due to round-off errors. In practice, $\alpha = D^{-i}, i \in [m, m+1, \ldots, n]$, where D is the decreasing factor (typical values are 2 and 10), m is the integer for largest perturbation (typical value is 0), and n is the integer for smallest perturbation.

4.8.1 AdjointDriver

The objective function is chosen to be the model output of a given grid point in model domain with respect to initial model status. The corresponding gradient can therefore be interpreted as the sensitivity. This driver aims at the verification of adjoint code obtained by automatic differentiation of underlying model code using O∂yssée (version 1.7). The following options in section `[adjoint]` provide flexible control of the verification.

`[adjoint]`	
`Point_species_name`	Species name of the selected point in model domain for sensitivity calculation.
`Point_nx`	x-index of the selected point for sensitivity calculation.
`Point_ny`	y-index of the selected point for sensitivity calculation.
`Point_nz`	z-index of the selected point for sensitivity calculation.
`Norm_perturbation_vector`	Norm of the initial perturbation vector.
`With_random_perturbation`	With random directions for the perturbation?

`Decreasing_root`	Decreasing factor of the sequence of perturbation vectors (D).
`Start_index`	Index for the calculation of the first decreasing ratio (m).
`End_index`	Index for the calculation of the last decreasing ratio (n).
`With_left_finite_difference_checking`	Checking left-side finite difference results?
`Display_sensitivity`	Display sensitivity results for the decreasing perturbation sequences?

Option `With_trajectory_management`, `Trajectory_delta_t`, `Trajectory_file` are similar to those for `4DVar` in Section 4.7.5.

4.8.2 GradientDriver

The objective function is chosen to be the norm of the difference between model simulations and synthetic observations. This driver aims at the verification of the backward integration algorithm of adjoint model for gradient calculations. Options `Norm_perturbation_vector`, `With_random_perturbation`, `Decreasing_root`, `Start_index`, `End_index`, and `With_left_finite_difference_checking` in section `[adjoint]` have the same meanings as those in Section 4.8.1. Option `Display_cost` indicates whether the values of the objective function are displayed when perturbation decreases accordingly.

4.8.3 Gradient4DVarDriver

The objective function is chosen to be the observation discrepancy in the 4DVar objective function. This driver aims at the verification of the adjoint code of observation operator. All the options for this driver are same as those in Section 4.8.2

4.9 Output Savers

4.9.1 BaseOutputSaver

The saver `BaseOutputSaver` is configured with a file that contains one or several sections `[save]`. Each section is associated with one element of a list of output-saver units managed by `BaseOutputSaver`.

According to the value of `Type` in every section, different saver units are called. Note however that a `group` attribute can be set in `BaseOutputSaver` (the default being `all`, and the other choices being `forecast` and `analysis`) and that only savers with the same group are called.

Some parameters must be provided for any kind of savers:

`[save]`	
`Species`	Chemical species to be saved. If it is set to `all`, concentrations for all species are saved.
`Date_beg`	The date from which the concentrations are saved. If concentrations are averaged, the first step at which concentrations are actually saved if not `Date_beg`, but `Date_beg` plus the number of steps over which concentrations are averaged. If the value - 1 is supplied, `Date_beg` is set at the start of the simulation.
`Date_end`	The last date at which concentrations may be saved. If the value - 1 is supplied, `Date_end` is set at the end of the simulation.
`Interval_length`	The number of steps between saves.

`Type`	The type of saver, see Table 4.13 for details.
`Output_file`	The full path of output files, in which `&f` will be replaced by the name of the chemical species. Note that the directory in which the files are written must exist before the simulation is started.

Note that `Species`, `Date_beg`, `Date_end`, `Interval_length` must appear before `Type`. After `Type`, put additional options relevant for the chosen output saver.

Here is a list of all types of saver units available at the moment:

Table 4.13: Types of saver

`domain`	To save entire vertical layers.
`domain_aer`	The same as `domain` but for aerosol species.
`domain_assimilation`	The same as `domain` but for data assimilation applications.
`nesting`	To perform nested simulations.
`nesting_aer`	The same as `nesting` but for aerosol species.
`subdomain`	To save concentrations only for an horizontal subdomain.
`subdomain_aer`	The same as `subdomain` but for aerosol species.
`wet_deposition`	To save entire wet deposition fluxes.
`dry_deposition`	To save entire dry deposition fluxes.
`wet_deposition_aer`	The same as `wet_deposition` but for aerosol species.
`dry_deposition_aer`	The same as `dry_deposition` but for aerosol species.
`backup`	The backup gas species in order to restart.
`backup_aer`	The same as `backup` but for aerosol species.

4.9.2 SaverUnitDomain and SaverUnitDomain_aer

The output saver `SaverUnitDomain` defines an output-saver unit when `Type` is set to either “domain”, or “domain_ensemble_forecast” and “domain_ensemble_analysis” in case of ensemble applications. This output saver requires additional parameters presented in the table below.

	`[save]`
`Levels`	A list of integers that determines the vertical layers to be saved. Note that 0 is the first layer. Remember that the heights you specified in the file `levels.dat` are those of the level interfaces, while concentration are saved in the middle of each levels.
`Averaged`	Should concentrations be averaged over `Interval_length`? If not, instantaneous concentrations are saved.
`Initial_concentration`	Should initial concentrations be saved? This option is only available if concentrations are not averaged.

For aerosol species, the saver should be `SaverUnitDomain_aer` and the `Type` “domain_aer”. The section `[save]` is very similar to the one for gaseous species, except that you have to specify for which diameters the concentrations are saved. Hence, the list of species to be saved looks like this:

```
Species: aer1_{0} aer 1_{2} aer2_{0-1}
```

In that case, the species named “aer1” is to be saved for the diameter of indices 0 and 2, and “aer2” for the diameters of indices 0 and 1.

In `Output_file`, `&f` will be replaced by the species name and `&n` by the bin index. You can use any symbol which is not a delimiter (or even nothing) to separate the species name from the bin index, even though `&f_&n.bin` is the advised form.

If `Species` is set to “all” the concentrations will be saved for all aerosol species and for all diameters.

4.9.3 SaverUnitSubdomain and SaverUnitSubdomain_aer

These saver units allow the user to save concentrations only over an horizontal subdomain (for example, if they perform a simulation over the whole of Europe but only want the concentrations over one country or region). Their `Type` is “subdomain” and “subdomain_aer” respectively. The user must provide between which indices for x and y they want to save concentrations. The specific parameters for these saver units are:

`[save]`	
`Levels`	A list of integers that determines the vertical layers to be saved. Note that 0 is the first layer. Remember that the heights you specified in the file `levels.dat` are those of the level interfaces, while concentration are saved in the middle of each levels.
`Averaged`	Should concentrations be averaged over `Interval_length`? If not, instantaneous concentrations are saved.
`Initial_concentration`	Should initial concentrations be saved? This option is only available if concentrations are not averaged.
`i_min`	Minimum latitude index of the subdomain.
`i_max`	Maximum latitude index of the subdomain.
`j_min`	Minimum longitude index of the subdomain.
`j_max`	Maximum longitude index of the subdomain.

4.9.4 SaverUnitDomain_assimilation

The output saver `SaverUnitDomain_assimilation` defines an output-saver unit similar to `SaverUnitDomain`, except that it requires additional parameters presented in the table below.

`[save]`	
`Date_file`	The full path name of the file that stores the date sequences of the assimilation results.

The `group` attribute of the output saver `SaverUnitDomain_assimilation` is set to “analysis”, whereas the `group` attributes of other saver units are set to “forecast” by default. Its `Type` is “domain_assimilation”.

4.9.5 SaverUnitDomain_prediction

The output saver `SaverUnitDomain_prediction` defines an output-saver unit similar to `SaverUnitDomain`. The `group` attribute of the output saver `SaverUnitDomain_prediction` is set to “prediction”. Its `Type` is “domain_prediction”. It works in a similar way to `SaverUnitDomain_assimilation`, except that it is designed for the storage of model predictions starting from analyzed model state at the end of the assimilation period.

4.9.6 SaverUnitNesting and SaverUnitNesting_aer

The saver units `SaverUnitNesting` and `SaverUnitNesting_aer` are used to perform nested simulations. That means that the results of a first simulation on a large domain are interpolated and saved at the boundary of a subdomain and are then used as boundary conditions for a second simulation on the subdomain.

The two simulations are quite "normal" except that:

- **For the first one:** additional concentrations have to be saved using `SaverUnitNesting` and/or `SaverUnitNesting_aer`.
- **For the second one:** boundary conditions from the first simulation have to be provided the usual way in `polair3d-data.cfg`.

Refer to the files in `driver/example/nesting` for concrete examples. Files ending with "-nesting" are for the first simulation and files ending with "-nested" are for the second one.

If the saver unit is of `Type` "nesting" or "nesting_aer", the additional parameters needed in the section `[save]` are presented in the table below.

[save]	
`x_min`	Origin of the subdomain along x.
`Delta_x`	Step along x for the subdomain.
`Nx`	Number of points along x for the subdomain.
`y_min`	Origin of the subdomain along y.
`Delta_y`	Step along y for the subdomain.
`Ny`	Number of points along y for the subdomain.
`levels`	File giving the interfaces of the layers for the subdomain.
`Nz`	Number of layers in the subdomain.

In `Output_file` `&f` and `&n` are replaced as for `SaverUnitDomain` or `SaverUnitDomain_aer` and `&c` is replaced by the direction along which the boundary conditions were interpolated (that means that `&c` is replaced by `x`, `y` or `z`).

4.9.7 SaverUnitPoint and SaverUnitPoint_aer

The saver units `SaverUnitPoint` and `SaverUnitPoint_aer` are used to save concentrations at a list of given points. There are two possible types: `indices_list` saves concentrations at given indices in the simulation grid (provided in the main configuration file in section `[domain]`) and `coordinates_list` saves concentrations at given (cartesian) coordinates, and is available for Gaussian models only. In case the saver is of type `coordinates_list` the simulation grid is still read but is not used, as concentrations are computed directly at each point of interest.

The list of points where concentrations are to be saved has to be specified in the section `[save]` of the saver configuration file. It has to be written just after the line containing the field `Output_file`. The list begins with the field `Indices` in case the saver type is `indices_list` and `Coordinates` if the type is `coordinates_list`. In both cases, the list must end with a line containing the field `Point_file`, which is used to specify a file name where the list of all points where concentrations are saved is to be written during the simulation. This looks like:

```
Type: indices_list
Levels: 0 2
```

```
Output_file: <Results>/&f.bin

Indices:

0   0
5   15
20  25
30  30

Point_file: <Results>/point.txt
```

One line corresponds to one point. There can be either two or three indices. In case there are two indices on the line, the first one corresponds to index along y, the second one to the index along x, and concentrations are saved at this point for each vertical level specified in the field `Levels`. In the previous example, the file `<Results>/point.txt` will be created during the simulation and look like:

```
# z   y    x

  0   0    0
  2   0    0
  0   5    15
  2   5    15
  0   20   25
  2   20   25
  0   30   30
  2   30   30
```

In case there are three indices on a line, concentrations are saved only at the specified point, no matter what the field `Levels` contains. The first value on the line corresponds to index along z, the second to index along y and the third to index along x. Note that lines containing two or three values can be entered in any order. The output binary file containing concentrations will simply follow the order given in the file `<Results>/point.txt` for each time step. It means that the resulting binary file will be of size $N_t \times N_{point}$ where N_t is the number of time steps to be saved, and N_{point} is the number of points where concentrations are saved.

To save coordinates instead of indices, one simply has to change the type to `coordinates_list`, to add the field `Levels_coordinates` to specified values of z where concentrations are to be saved, and to write the field `Coordinates` instead of `Indices`. The field `Levels` is still read but not used. Here is an example:

```
Levels: 0
Levels_coordinates: 1.5

Output_file: <Results>/&f.bin

Coordinates:

470.0  535.0
```

```
470.7  535.0
470.5  535.1
1.5 470.8 535.4
2.5 470.8 535.4
4.5 470.8 535.4
7.5 470.8 535.4
10.5 470.8 535.4
13.5 470.8 535.4
17.5 470.8 535.4

Point_file: <Results>/point.txt
```

Note that coordinates are entered in meters, first z, then y, then x, or just y then x. In the previous example, every point for which only two coordinates are entered is saved at 1.5 meters above ground. For one point, one wished to save concentrations at different heights above ground, so heights have been explicitly written.

When dealing with aerosol species, one just has to put `indices_list_aer` or `coordinates_list_aer` instead of `indices_list` or `coordinates_list` respectively.

4.9.8 SaverUnitWetDeposition and SaverUnitDryDeposition

The output savers `SaverUnitWetDeposition` and `SaverUnitDryDeposition` define output-saver units when `Type` is set to “wet_deposition” or “dry_deposition” and both require additional parameters presented in the table below.

[save]	
`Averaged`	Should concentrations be averaged over `Interval_length`? If not, instantaneous concentrations are saved.
`Initial_concentration`	Should initial concentrations be saved? This option is only available if concentrations are not averaged.

If `Species` is set to “all” the deposition fluxes will be saved for all scavenged or dry deposited species.

4.9.9 SaverUnitWetDeposition_aer and SaverUnitDryDeposition_aer

The output savers `SaverUnitWetDeposition_aer` and `SaverUnitDryDeposition_aer` define output-saver units when `Type` is set to “wet_deposition_aer” or “dry_deposition_aer”. The section `[save]` is very similar to the one for gaseous species, except that you have to specify for which diameters the deposition fluxes are saved. Hence, the list of species to be saved looks like this:

```
Species: aer1_{0} aer 1_{2} aer2_{0-1}
```

In that case, the species named “aer1” is to be saved for the diameter of indices 0 and 2, and “aer2” for the diameters of indices 0 and 1.

In `Output_file`, `&f` will be replaced by the species name and `&n` by the bin index. You can use any symbol which is not a delimiter (or even nothing) to separate the species name from the bin index, even though `&f_&n.bin` is the advised form.

If `Species` is set to “all” the deposition fluxes will be saved for all aerosol species and for all scavenged or dry deposited diameters.

4.9.10 SaverUnitBackup and SaverUnitBackup_aer

The output savers `SaverUnitBackup` and `SaverUnitBackup_aer` are respectively similar to `SaverUnitDomain` and `SaverUnitDomain_aer` output-savers: they save gas and aerosol concentrations over the entire domain. The difference is that latter output-savers are intended for post-treatment whereas formers for eventually be able to restart a simulation as if it had not stopped.

Therefore, all gas and aerosol concentrations are saved and not averaged, only one simulation time step is saved and each backup overwrites the latter one. A "date" file stores the current date and iteration of backup files.

All backup files are buffered in files with extension ".buf". These buffers are only intended for the case the simulation breaks during the backup saving. In this case the "date" file will contain the message "!! BACKUP SAVING NOT FINISHED !!" which means that buffer files have to be used instead of backup ones. These buffer files are only needed at run time and are removed at the end of simulation.

The output savers `SaverUnitBackup` and `SaverUnitBackup_aer` are configured with a `[save]` section:

```
[save]

Type: backup
Interval_length: 10
Output_file: backup/&f.bin
Date_file: backup/date_backup

[save]

Type: backup_aer
Interval_length: 10
Output_file: backup/&f_&n.bin
Date_file: backup/date_backup_aer
```

The backup output-savers are selected by setting `type` to `backup` or `backup_aer`. The number of time steps between two backups is set in `Interval_length`. In order to be able to correctly restart a simulation this number has to be large enough compared to that of other output-savers. Furthermore the backup time must overlap the save time of other output-savers. For example if a simulation makes averaged savings every six time steps, the backup `Interval_length` has to be a multiple of six and at least six.

How to restart? Open the "date" file, pick up the backup date and replace the beginning date of `polair3d.cfg` main configuration file with it. Then go into `polair3d-data.cfg` and change the `initial_condition` and `initial_condition_aer` sections so that they points to the backup files. Modify the `[save]` sections in order to keep previous output files.

Remark A restart must give exactly the same results as if the simulation had not stopped. Nevertheless vector concentrations are stored in memory in double precision whereas backups are written on disk in simple precision so that on restart you cannot avoid roundoff errors between simple and double precision.

4.10 Observation Managers

The observation managers deal with available observational data at different locations and dates. These managers are designed to prepare for applications related to observation treatments, especially for data assimilation. The observation operator are implemented for the mapping from observation space into model space. For a given date, these managers retrieve observation data values and the corresponding statistical information, e.g. observational error covariances.

4.10.1 GroundObservationManager

The `GroundObservationManager` is dedicated to ground observation managements.

	`[general]`
`Species`	Name of observed species. The current version deals with only one observed species.
`Error_variance`	Error variance for the observed species.
`With_spatial_interpolation`	Should observations be interpolated at adjacent model grid points?
`With_perturbation`	Should the observation be perturbed?
`Perturbation_scale`	If `With_perturbation` is set to “yes”, gives the amplitude of the perturbation.
	`[stations]`
`Nstations`	Total number of stations.
`Stations_file`	File containing station information (code, name, latitude, longitude and altitude). `&s` in path names is replaced by species name specified in `[general]` section.
`Input_directory`	Directory where the observations are stored.

4.10.2 SimObservationManager

The `SimObservationManager` is dedicated to synthetic observation managements. Library NewRan is needed for random number generations. Note that NewRan is not included in the distribution, and it is the user’s duty to install NewRan. The associated configuration file is an extension of that of `GroundObservationManager`. The additional sections are mainly for data specifications of the binary data files.

	`[simulation_manager]`
`Simulation_option`	Specifies how observations are provided. The current version deals only with observations at ground stations.
`Input_file`	Files containing the observation data. They usually are generated by certain reference run of Polair3D. `&s` in path names is replaced by species name.
`Date_min`	Starting date for the simulation results in data files.
`Delta_t`	Time step in seconds for the simulation results in data files.
`Levels`	Levels for the simulated data in files.
`Initial_concentration`	Flag that indicates whether initial concentrations are included in data file.

4.11 Perturbation Manager

The `PerturbationManager` is dedicated to perturbation managements. It reads the perturbation configurations, and performs perturbations. The concerning fields are then updated according to perturbation results for new model simulations in diverse applications such as data assimilation and ensemble predictions. Library NewRan is needed for random number generations.

The perturbation fields are defined in configuration file of which the name can be read from section `[perturbation_management]` in the driver configuration file.

[general]	
`Fields`	Perturbation field list that depends on species, e.g. DepositionVelocity, PhotolysisRate, SurfaceEmission, and BoundaryCondition.
`Rand_seed`	Seed assignment for NewRan library, either a given seed number (in]0, 1[), or name of the directory that contains NewRan seed files, or a random seed number (in]0, 1[) that depends on the current CPU time.
`Field_maximum_spread`	Every random number for field perturbations cannot exceed the mean plus or minus 'Field_maximum_spread' times the standard deviation.
`Observation_maximum_spread`	Every random number for observation perturbations cannot exceed the mean plus or minus 'Observation_maximum_spread' times the standard deviation.

For fields listed in section `[AdditionalField]` e.g. `Attenuation` and `VerticalDiffusionCoefficient`, the perturbations do not depend on species. For each additional field, or each species of species-dependent fields, the perturbation is performed according to lognormal (LN) or normal (N) law with given standard derivation. The probability distribution types are listed in the `PDF` column, and the standard derivations are listed in the `Parameter` column. Sometimes there are two additional columns for species-dependent fields indicating correlated species, and correlation coefficient. The correlation coefficients can only be set to 1.

Chapter 5

Models

There are three major types of models: Gaussian models (see Section 5.1, 5.2, 5.3 and 5.4), Polair3D models (see Section 5.5, 5.6, and 5.7) and Castor models (see Section 5.9). All variants of a model have the same principles but can deal with various applications and phenomena.

Polair3D models were the first implemented in Polyphemus. They allow, as well as Castor models, to compute the advection and diffusion of pollutants at a large scale and can integrate various additional phenomena (such as photochemical chemistry or deposition). Gaussian models have been added to perform simulation at a local scale of the effect of a continuous (plume) or instantaneous (puff) source of pollutant.

As for now, Castor models only deal with gaseous species, while the other models deal with gaseous or aerosol species.

5.1 GaussianPlume

Model `GaussianPlume` is the Gaussian plume model for gaseous species only. The associated program to be run is `plume` and it is configured with one configuration file (`plume.cfg`) and four data files (`plume-source.dat`, `plume-level.dat`, `gaussian-meteo.dat` and `gaussian-species.dat`). The configuration file provides the paths to the four other files. Basically, given a series of continuous point sources, it calculates the concentration of each species along a specified grid. There are several output files, one for each species, that are binary files. The way results are saved is described in an additional configuration file which corresponds to the file described in Section 4.9 (reference: `plume-saver.cfg`).

In these configuration files, there are entries that are not relevant for the Gaussian model but that must be provided anyway. In descriptions of configuration files (below), they are described as `irrelevant`.

5.1.1 Configuration File: `plume.cfg`

`[domain]`	
`Date_min`	**Irrelevant.** Provide a date.
`Delta_t`	**Irrelevant.** Provide any number.
`Nt`	**Irrelevant.** Provide an integer.
`x_min`	Abscissa in meter of the center of the lower-left cell.
`Delta_x`	Step length along x (in m).
`Nx`	Number of cells along x (integer).
`y_min`	Ordinate in meter of the center of the lower-left cell.

Delta_y	Step length along y (in m).
Ny	Number of cells along y (integer).
Nz	Number of vertical levels (integer).
Vertical_levels	Path to the file that defines vertical levels heights.
Land_category	Land category (choose between `rural` and `urban`). Relevant only when standard deviations are computed with Briggs parameterization.
Time	Choose whether it is nighttime (`night`) or daytime (`day`). Relevant only when there is biological decay, or when Doury sigma parameterization is used.
Species	Path to the file that defines involved species.
	[gaussian]
With_plume_rise	Is plume rise taken into account?
With_radioactive_decay	Is radioactive decay taken into account?
With_biological_decay	Is biological decay taken into account?
With_scavenging	Is scavenging taken into account?
With_dry_deposition	Is dry deposition taken into account?
Sigma_parameterization	Parameterization used to compute standard deviations (`Briggs` for Briggs parameterization, `Doury` for Doury parameterization, and `similarity_theory` for a parameterization based on similarity theory).
File_meteo	Path to the file containing the meteorological data.
File_source	Path to the file that describes the sources.
	[deposition]
Deposition_model	Model used to take dry deposition into account (`Chamberlain` for Chamberlain model, `Overcamp` for Overcamp model).
Nchamberlain	Number of points to calculate the Chamberlain integral (integer). Relevant only when dry deposition with Chamberlain model is taken into account.
	[output]
Configuration_file	Path to the configuration for the output saver.

Note: The Chamberlain integral for the calculation of dry deposition is discretized and approximated as a sum. The integer that is provided corresponds to the number of terms of the sum in the plume model. In the puff model, it is incremented at each time step, so as to have a number of points consistent with the range of the integral (that is, not to have too many points to discretize an integral whose range is very small).

5.1.2 Source Description: `plume-source.dat`

Sources are described in a single configuration file containing as many sections as there are sources. Each section named `[source]` is associated with a new source. Each section contains the following information: (1) the emission rate (in mass per seconds – the mass unit does not matter: the model will stick to it), (2) the velocity of emitted pollutants ($\mathrm{m\,s^{-1}}$), (3) the temperature of pollutants when emitted (°C), (4) the section area (in $\mathrm{m^2}$) of the source (most likely a stack), (5) the abscissa of the source (m), (6) the ordinate of the source (m), (7) the height of the source (m), and (8) the species that is emitted.

A typical source file looks like this:

```
[source]

# Source coordinates (meters)
Abscissa: 100.
Ordinate: 100.
Altitude: 0.5
# Species name
Species_name: Iodine

# Source rate (mass/s)
Rate = 56.5
# Source velocity (m/s)
Velocity = 10.1
# Source temperature (Celsius degrees)
Temperature = 60.
# Source section (m^2)
Section = 5.725
```

In this example, only one section is provided, but other sources may be added simply by adding the corresponding sections at the end of the file. One species may have several sources. The source file can contain a list of point sources provided by the user or a discretized line source. In that case, it corresponds to the output file of the discretization preprocessing program `discretization`.

5.1.3 Vertical Levels: `plume-level.dat`

Vertical levels are defined in a single data file. They are defined by their interfaces. This means that the file contains `Nz+1` heights, where `Nz` is the number of levels specified in the main configuration file. The concentrations are computed at layers mid-points.

5.1.4 Species: `gaussian-species.dat`

Species are listed in the section `[species]` of a data file (the same as the species data file used in the preprocessing program `gaussian-deposition`, see Section 3.9.2). When radioactive or biological decay is taken into account, a section containing the half-life times of the species has to be provided. The section `[half_life]` contains the list of all species followed by their half-life time in days for radioactive decay (put 0. in the case of non-radioactive species). Provide only one species per line. The section `[half_life_time]` contains two values following each species name, the first corresponding to its biological half-life time (in s) during daytime and the second to the value during nighttime (put 0. in the case of non biological species). Here is an example:

```
[species]

# List of the species
Caesium     Iodine    bio1


[half_life]
```

```
# Half-life of the species (Unit: days)
# 0 coresponds to non-radioactive species
Caesium: 1.1e4
Iodine: 8.04
bio1: 0.

[half_life_time]

# Half-life of the species ( Unit: seconds )
# First value: day, second value: night
# 0 corresponds to non-radioactive species.
Caesium: 0. 0.
Iodine:  0. 0.
bio1: 1000 500
```

In that case we have two radioactive species, "Caesium" and "Iodine", and one biological species, "bio1".

If scavenging is taken into account, sections `[scavenging]` and `[scavenging_constant]` must be added. The section `[scavenging]` contains the name of all species for which scavenging occur and `[scavenging_constant]` their constants in s^{-1}.

5.1.5 Meteorological data file: `gaussian-meteo.dat`

This file contains basic meteorological information needed to run Gaussian models. In case there are scavenging and deposition, it is the output file of preprocessing program `gaussian-deposition` described in Section 3.9.2. Information that are always needed are:

- Temperature (°C)
- Wind angle (° from x axis)
- Wind speed ($\mathrm{m\,s^{-1}}$)
- Inversion height (m)

These information are provided inside a section `[situation]`, and the meteorological data file contains as many sections are there are situations. If the inversion height is set to 0., it is not taken into account. Otherwise, reflections on the inversion layer are performed. This value can be set equal to the boundary layer height, if known, during day and to 0. during night. It can also be set to 0. all the time if no other value is known.

```
[situation]

# Temperature (Celsius degrees)
Temperature = 15

# Wind angle (degrees)
Wind_angle = -100

# Wind speed (m/s)
```

```
Wind = 0.5

# Inversion height (m)
Inversion_height = 1000
```

In case the Briggs parameterization for standard deviation is used, a stability class between 'A' (very unstable) and 'F' (very stable) has to be provided.

```
# Stability class
Stability = A
```

In case the standard deviations are computed with similarity theory, a lot more information have to be provided:

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

If not known, the Coriolis parameter can be set to $10^{-4}\mathrm{s}^{-1}$.

```
[situation]

# Boundary layer height (m)
Boundary_height = 500.0

# Friction velocity (m/s)
Friction_velocity = 0.37

# Convective velocity (m/s)
Convective_velocity = -0.81

# Monin Obukhov Length (m)
LMO = 120.0

# Coriolis parameter (/s)
Coriolis = 1.4e-04
```

5.2 GaussianPlume_aer

It is the Gaussian plume model for aerosol species. The corresponding program is `plume_aer`. It can be run when there are aerosol species only, or both aerosol and gaseous species. It takes the same input files as the Gaussian plume model, except that they contain in addition some sections dedicated to aerosol species. It takes in addition another input file that describes the diameters of particles (file `diameter.dat` already described in Section 3.9.2). The output files are binary files, one for each gaseous species and one for each couple (aerosol species, diameter). The way results are saved is described in an additional configuration file (reference: `plume-saver_aer.cfg`) described in Section 4.9.

5.2.1 Configuration File: `plume_aer.cfg`

It is exactly the same file as the configuration file described in Section 5.1. The only data that may differ are the paths to the input files.

5.2.2 Source Description: `plume-source_aer.dat`

It is the same file as the source file for gaseous species, except that obviously some (or all) emitted species will be particulate species. The corresponding sections are named `[aerosol_source]`.

5.2.3 Vertical Levels: `plume-level.dat`

It is the same file as in Section 5.1.

5.2.4 Species: `gaussian-species_aer.dat`

The section `[species]` lists the gaseous species, and the section `[aerosol_species]` lists the aerosol species. In the case of radioactive or biological decay, the sections are the same as described in Section 5.1 and contain the half-life times of both gaseous and aerosol species.

5.2.5 Diameters: `diameter.dat`

See Section 3.9.2.

5.2.6 Meteorological data: `gaussian-meteo.dat`

See Section 5.1.5.

5.3 GaussianPuff

Model `GaussianPuff` is the Gaussian puff model for gaseous species only. The associated program to be run is `puff` and it is configured with one configuration file (`puff.cfg`) and four data files (`puff.dat`, `puff-level.dat`, `gaussian-meteo.dat` and `gaussian-species.dat`). The configuration file provides the paths to the four other files. Basically, given a series of instantaneous puffs emitted at different times, it calculates the concentration of each species along a specified grid. There are several output files, one for each species, that are binary files. (same as in the Gaussian plume model, and fully described in Section 4.9).

5.3.1 Configuration File: `puff.cfg`

	`[display]`
`Show_date`	**Irrelevant.** Provide any Boolean.
	`[domain]`
`Date_min`	**Irrelevant.** Provide any date (see Section 2.2.7).
`Delta_t`	Time step of the simulation (in seconds).
`Nt`	Number of time steps (integer).
`x_min`	Abscissa in meter of the center of the lower-left cell.
`Delta_x`	Step length along x (in m).
`Nx`	Number of cells along x (integer).

y_min	Ordinate in meter of the center of the lower-left cell.
Delta_y	Step length along y (in m).
Ny	Number of cells along y (integer).
Nz	Number of vertical levels (integer).
Vertical_levels	Path to the file that defines vertical levels heights.
Land_category	Land category (choose between `rural` and `urban`).
Time	Choose whether it is nighttime (`night`) or daytime (`day`). Relevant only when there is biological decay.
Species	Path to the file that defines involved species.
[gaussian]	
With_radioactive_decay	Is radioactive decay taken into account?
With_biological_decay	Is biological decay taken into account?
With_scavenging	Is scavenging taken into account?
With_dry_deposition	Is dry deposition taken into account?
Sigma_parameterization	Parameterization used to compute standard deviations (`Briggs` for Briggs parameterization, `Doury` for Doury parameterization).
File_meteo	Path to the file containing the meteorological data.
File_puff	Path to the file that contains the puff data.
[deposition]	
Deposition_model	Model used to take dry deposition into account (`Chamberlain` for Chamberlain model, `Overcamp` for Overcamp model)
Nchamberlain	Number of points to calculate the Chamberlain integral (integer). Relevant only when dry deposition with Chamberlain model is taken into account.
[output]	
Configuration_file	Path to the configuration for the output saver.

Note: Contrary to the Gaussian plume model, only two parameterizations to compute standard deviations are currently available: Doury and Briggs models.

5.3.2 Puff Description: `puff.dat`

Puffs are described in a single configuration file containing as many sections as there are puffs. Each section named `[source]` is associated with a new source. Each section contains the following information: (1) the time when the puff is released (s) (from the beginning of the simulation) (2) the total mass emitted (in mass unit – the mass unit does not matter: the model will stick to it),(3) the abscissa of the point of emission (m), (4) the ordinate of the point of emission (m), (5) the height of the point of emission (m), and (6) the species that is emitted.

```
[source]

# Source coordinates (meters)
Abscissa: 0.
Ordinate: 5.
Altitude: 25.
# Species name
Species_name: Iodine
```

```
# Release time (seconds)
Release_time: 0.
# Total mass released (mass)
Quantity: 1.
```

One species may have several puffs. The puff file can contain a list of puffs provided by the user or a discretized line source or trajectory. In that case, it corresponds to the output file of the discretization preprocessing program `discretization`.

5.3.3 Vertical Levels, Species and Meteorological data

They are exactly the same files as those described in Section 5.1, except that sigma parameterization with similarity theory is not yet available for puff model, so there is no need of the associated meteorological data.

5.4 GaussianPuff_aer

It is the Gaussian puff model for aerosol species. It can be run when there are aerosol species only, or both aerosol and gaseous species. It takes the same input files as the Gaussian puff model, except that they contain in addition some sections dedicated to aerosol species. It takes in addition another input file that describes the diameters of particles (file `diameter.dat` already described in the Section 3.9.2). The output files are binary files, one for each gaseous species and one for each couple (species, diameter).

5.4.1 Configuration File: `puff_aer.cfg`

It is exactly the same file as the configuration file described in Section 5.3. The only data that may differ are the paths to the input files.

5.4.2 Source Description: `puff_aer.dat`

It is the same file as the puff file for gaseous species described in Section 5.3, except that obviously some (or all) emitted species will be particulate species. The corresponding sections are named `[aerosol_source]`.

5.4.3 Vertical Levels, Species, Meteo and Diameters

Vertical level file and gaussian meteo file have been described in Section 5.1 and diameter files is the same as in Section 3.9.2. Species file is the same file as described for the plume model for aerosol species (Section 5.2.4).

5.5 Polair3DTransport

The model Polair3DTransport is configured with three configuration files (`polair3d.cfg`, `polair3d-data.cfg` and `polair3d-saver.cfg`) and two data files (`levels.dat` and `species.dat`). The main configuration file (`polair3d.cfg`) provides the paths to the four other files.

5.5.1 Main Configuration File: `polair3d.cfg`

	`[domain]`
`Date_min`	Starting date in any legal format (see Section 2.2.7). The date can therefore include seconds.
`Delta_t`	Time step in seconds.
`Nt`	Number of iterations of the simulation (integer).
`x_min`	Abscissa of the center of the lower-left cell. Provide a longitude (in degrees) or, in case Cartesian coordinates are chosen, an abscissa in meters.
`Delta_x`	Step length along x, in degrees (longitude) or in meters (for Cartesian coordinates).
`Nx`	Number of cells along x (integer).
`y_min`	Ordinate of the center of the lower-left cell. Provide a latitude (in degrees) or, in case Cartesian coordinates are chosen, an ordinate in meters.
`Delta_y`	Step length along y, in degrees (latitude) or in meters (for Cartesian coordinates).
`Ny`	Number of cells along y (integer).
`Nz`	Number of vertical levels (integer).
`Vertical_levels`	Path to the file that defines vertical levels interfaces.
`Cartesian`	If activated, coordinates are Cartesian and in meters. Otherwise, coordinates are latitudes and longitudes in degrees.
`Species`	Path to the file that defines involved species and their chemical properties.
	`[options]`
`With_advection`	Are species advected?
`With_diffusion`	Are species diffused?
`With_air_density`	If activated, vertical wind is diagnosed from $\mathrm{div}(\rho V) = 0$ where ρ is the air density and V the wind, and the diffusion term is $\mathrm{div}\left(\rho K \nabla \frac{c}{\rho}\right)$ where c is the concentration and K is the diffusion matrix. If this option is not activated, it is assumed that ρ is constant and therefore disappears from the previous equations.
`With_initial_condition`	Are initial conditions provided for given species? If not, initial concentrations are set to zero.
`With_boundary_condition`	Are boundary conditions available for given species?
`With_deposition`	Is dry deposition taken into account?
`With_point_emission`	Are point emissions provided?
`With_surface_emission`	Are emissions at ground provided?
`With_additional_surface_emission`	Are additional emissions at ground provided?
`With_volume_emission`	Are volume emissions provided?

`Scavenging_model`	Which scavenging model is applied? If `none`, the scavenging is not taken into account. Otherwise, the following model is applied: `constant` for constant scavenging coefficient, `belot` for the Belot model (of the form $a\,{p_0}^b$, where p_0 is the rain intensity in $\mathrm{mm\,h^{-1}}$) or `microphysical` for the scavenging model based on microphysical properties of species.
`Collect_dry_flux`	Are the dry deposition fluxes are collected in order to postprocess them if dry deposition is taken into account?
`Collect_wet_flux`	Are the wet deposition fluxes are collected in order to postprocess them if wet deposition is taken into account?
	`[data]`
`Data_description`	Path to the configuration file that describes input data.
`Horizontal_diffusion`	Horizontal diffusion coefficient in $\mathrm{m^2\,s^{-1}}$.
`Isotropic_diffusion`	If activated, horizontal diffusion is set equal to vertical diffusion.
	`[output]`
`Configuration_file`	Path to the configuration for the output saver.

5.5.2 Data Description: `polair3d-data.cfg`

This configuration file describes input data files (binary files). It is divided into sections: for deposition, for meteorological fields, etc. A section roughly looks like this:

```
[meteo]

Date_min: 2004-08-09
Delta_t = 10800.

Fields: MeridionalWind ZonalWind Temperature Pressure Rain CloudHeight Attenuation\
SpecificHumidity
Filename: /u/cergrene/a/ahmed-dm/TestCase-1.0/data/meteo/&f.bin

VerticalDiffusion: /u/cergrene/a/ahmed-dm/TestCase-1.0/data/meteo/Kz_TM.bin
```

It is assumed that all binary files start at the same date, and this date is `Date_min` (see dates formats in Section 2.2.7). The time step is `Delta_t`, in seconds.

Then a list of fields is provided after `Fields`. These are fields that the model needs, and their names are determined by the model. Below, all fields required by the model (depending on its options) are listed. A generic path (full file name) is then provided (entry `Filename`). In this path, the shortcut '`&f`' refers to a field name. In the previous example, the full path to the temperature is `/u/cergrene/a/ahmed-dm/TestCase-1.0/data/Temperature.bin`. In the specific case of boundary conditions, the shortcut '`&c`' is replaced by x, y and z.

If a few fields are not stored in a file with a generic path, their specific paths can be provided after the entry `Filename`. This is the case for `VerticalDiffusion` in the previous example.

Note that:

1. entries `Fields`, `Filename` and additional paths *must* be at the *end of the section*, and *in this order*;

2. *at least one element* (possibly not a required field) must be provided to `Fields` and at least one element (possibly not a path) to `Filename`; for instance:

   ```
   Fields: ---
   Filename: --- # means no generic path.
   ```

   but:

   ```
   Fields:    # Illegal: one element required.
   Filename:     # Illegal: one element required.
   ```

In most sections, `Fields` is used to specify all chemical species involved in the process, e.g.:

```
[deposition]

Date_min: 2001-01-02
Delta_t = 10800.

Fields: O3 NO NO2 H2O2 HCHO PAN HONO SO2 HNO3 OP1 PAA ORA1
Filename: /u/cergrene/A/mallet/2001/data/dep-2005-01-19/&f.bin

ALD   /u/cergrene/A/mallet/2001/data/dep-2005-01-19/ALD-modified.bin
CO    0.002
```

Notice that `CO` is not associated with a path but with a numerical value. This is a feature: a binary file may be replaced with a numerical value. In this case, the field (in the example, CO deposition velocity) is set to a constant value (in every cell and at every time step). This works with any field, including meteorological fields (section `[meteo]`). This feature is often used to set constant boundary conditions.

In `polair3d-data.cfg`, several sections are required. Several sections have to be included only if given options are activated. In the following table, all possible sections are listed, with their entries.

Section	Entries	Comments
`[initial_condition]`	`Fields`, `Filename`	If initial conditions are activated (`With_initial_condition`).
`[boundary_condition]`	`Date_min`, `Delta_t`,`Fields`, `Filename`	If boundary conditions are activated (`With_boundary_condition`).
`[meteo]`	`Date_min`, `Delta_t`, `Fields`, `Filename`	Required fields are: `MeridionalWind` and `ZonalWind` if advection is activated, `VerticalDiffusion` if diffusion is activated, and `Temperature` and `Pressure` in case air density is taken into account.
`[deposition]`	`Date_min`, `Delta_t`, `Fields`, `Filename`	If deposition is activated (`With_deposition`).
`[point_emission]`	`file`	Path to the file which defines the point emissions (described below). If point emissions are activated (`With_point_emissions`).

`[surface_emission]`	`Date_min`, `Delta_t`, `Fields`, `Filename`	If surface emissions are activated (`With_surface_emission`).
`[additional_surfa...]`	`Date_min`, `Delta_t`, `Fields`, `Filename`	If surface additional emissions are activated (`With_additional_surface_emission`). This is mostly useful for biogenic emissions. Note that a species with additional surface emissions must have emissions in `[surface_emission]`. You might need to add the given species (say `ISO`) in section `[surface_emission]` with zero emissions (a line like `ISO: 0`).
`[volume_emission]`	`Date_min`, `Delta_t`, `Nz`, `Fields`, `Filename`	If volume emissions are activated (`With_volume_emission`). `Nz` is the number of levels in which pollutants are emitted.
`[scavenging]`	`Fields`	If the scavenging model is not set to “none” (`Scavenging_model`).

The file for point emissions have to contain a section `[source]` for each point emissions, with the following features:

- its location: `Abscissa` and `Ordinate` are given in degrees (or in meters, in case Cartesian coordinates are chosen) and `Altitude` is the vertical height in meters. Notice that the emission is released in the cell containing the location of the point emission.
- the emitted species is filled after `Species`. Only one species is possible for each section `[source]`.
- the type `Type` may be `continuous` or `puff` for instantaneous release. The `continuous` emission is described with entries `Rate` which corresponds to the quantity averaged on the duration of the release, `Date_beg` and `Date_end` which are the beginning and ending dates of the emissions (with format as described in Section 2.2.7). The `puff` emission is described with entries `Quantity` and `Date`.

It may look like this:

```
[source]

Abscissa: 5.2
Ordinate: 48.5
Altitude: 10.

Species: NO

Type: continuous
Rate: 1.
Date_beg: 2001-04-22_00-05
Date_end: 2001-04-22_00-07


[source]

Abscissa: 10.3
```

```
Ordinate: 48.
Altitude: 80.

Species: SO2

Type: puff
Quantity: 1.
Date: 2001-04-22_00-05
```

5.5.3 Vertical Levels and Species

Vertical levels are defined in a single data file. They are defined by their interfaces. This means that the file contains `Nz+1` heights, where `Nz` is the number of levels specified in the main configuration file. The concentrations are computed at layers mid-points.

Species are listed in the section `[species]` of a configuration file. In addition, some scavenging models needs extra data:

- The `constant` model requires a section `[scavenging_coefficient]` which contains a threshold of rain to apply scavenging (in $\mathrm{mm\,h^{-1}}$) and the name of the species with its associated scavenging coefficient (in $\mathrm{s^{-1}}$); for instance:

  ```
  [scavenging_coefficient]

  # Scavenging is applied above the following threshold over rain [mm / h].
  Scavenging_rain_threshold: 1.

  # Scavenging coefficient of the species: [s^{-1}]
  NO2 1.e-4       SO2 1.e-4
  ```

  Notice that if the previous lines are replaced by

  ```
  # Scavenging coefficient of the species: [s^{-1}]
  all    1.e-4
  ```

  the same scavenging coefficient will be used for all scavenged species.

- The `belot` model has the following expression $a\,{p_0}^b$ where coefficients a and b have to be provided for every species in a section `[belot]`; for instance:

  ```
  [belot]

  # Coefficients a and b for the Belot parameterization ($a * {p_0}^b$)
  #  where po is the rain intensity  [mm / h].

  # species       a       b
    all           1.e-05  0.8
  ```

- In case the `microphysical` model is used, Henry constants (in $\mathrm{mol\,L^{-1}\,atm^{-1}}$) and gas-phase diffusivities (in $\mathrm{cm^2\,s^{-1}}$) should be provided. Henry constants are listed in section `[henry]`; for instance:

```
[henry]

# Henry constant: [mol / L / atm]

O3      1.e-2   NO      2.e-3   NO2     1.e-2   H2O2    1.e5
HCHO    6.e3    ALD     15.     PAN     3.6     HONO    1.e5
SO2     1.e5    HNO3    1.e14   OP1     2.4e2   PAA     5.4e2
ORA1    4.e6    CO      1.e3    N2O5    1.e14
```

Gas-phase diffusivities are provided in the same way in section `[diffusivity]`.

5.6 Polair3DChemistry

Model Polair3DChemistry is configured with three configuration files (`polair3d.cfg`, `polair3d-data.cfg` and `polair3d-saver.cfg`) and two data files (`levels.dat` and `species.dat`). The main configuration file (`polair3d.cfg`) provides the paths to the four other files.

A configuration for Polair3DChemistry is an extension of the configuration for Polair3DTransport. In this section, the description is limited to Polair3DChemistry additional configuration. See Section 5.5 for the rest of the configuration.

5.6.1 Main Configuration File: `polair3d.cfg`

In addition to fields introduced in Section 5.5.1, the following fields are read by Polair3DChemistry:

[options]	
`With_chemistry`	Should chemistry occur?
`With_photolysis`	Should photolysis occur?
`With_forced_concentrations`	If activated, the concentrations of a few species are set to values read in files.
`Source_splitting`	If activated, source splitting is used within chemistry integration. Advection and diffusion fluxes are included in the chemistry integration as sources. This slightly increases the memory requirements but is recommended for numerical stability.

5.6.2 Data Description: `polair3d-data.cfg`

In addition to the configuration described in Section 5.5.2, a section `[photolysis_rates]` may be required (if the chemical mechanism includes photolysis reactions). Photolysis rates depend on days, time angle, latitude and altitude. During the time integration, they are linearly interpolated in all cells.

Section	Entries	Comments
`[photolysis_rates]`	`Date_min`	Starting date of photolysis rates.
	`Delta_t`	Time step in days.
	`Ndays`	Number of steps.
	`Time_angle_min`	Starting time angle in hours.
	`Delta_time_angle`	Time angle step in hours.
	`Ntime_angle`	Number of time angles.

`Latitude_min`	First latitude in degrees.
`Delta_latitude`	Step along latitude in degrees.
`Nlatitude`	Number of latitude steps.
`Altitudes`	List of altitudes in meters at which photolysis rates are provided.
`Fields, Filename`	Photolysis reaction names and the paths to the files in which photolysis rates are stored.

5.6.3 Vertical Levels and Species

Section 5.5.3 is relevant for Polair3DChemistry, and in particular the file giving the levels is exactly the same. As for species, a section `[molecular_weight]` lists the molecular weights (in $\mathrm{g\,mol^{-1}}$) of *all* species. If photolysis reactions are involved, the section `[photolysis_reaction_index]` is required. This section provides all reaction names and their indices in the list of reactions. Below is an example.

```
[photolysis_reaction_index]

NO2     0       O3O1D   1       O3O3P   2       HONO    3
HNO3    4       HNO4    5       NO3NO   6       NO3NO2  7
H2O2    8       HCHOmol 9       HCHOrad 10      ALD     11
MHP     12      HOP     13      PAA     14      KETONE  15
GLYform 16      GLYmol  17      MGLY    18      UDC     19
ORGNIT  20      MACR    21      HKET    22
```

The previous section is quoted from `Polyphemus/driver/species.dat` and is consistent with RACM (as implemented in ChemistryRACM – Section 6.2.1).

5.7 Polair3DAerosol

Polair3DAerosol is configured with three configuration files (`polair3d.cfg`, `polair3d-data.cfg` and `polair3d-saver.cfg`) and two data files (`levels.dat` and `species.dat`). The main configuration file (`polair3d.cfg`) provides the paths to the four other files.

A configuration for Polair3DAerosol is an extension of the configuration for Polair3DChemistry. In this section, the description is limited to Polair3DAerosol additional parameters. See Section 5.6 for the rest of the configuration.

5.7.1 Main Configuration File: `polair3d.cfg`

In addition to fields introduced in Section 5.6.1, the following fields are read by Polair3DAerosol.

	`[domain]`
`Bin_bounds`	The bounds of the diameter classes for aerosol species. Note that the classes are the same for each aerosol species.
	`[options]`
`With_initial_condition_aerosol`	Are initial conditions provided for given aerosol species? If not, initial concentrations are set to zero.

`With_boundary_condition_aerosol`	Are boundary conditions available for given aerosol species?
`With_pH`	Does the aerosol module returns cloud droplet pH?
`Lwc_cloud_threshold`	Liquid water content threshold above which a cloud is diagnosed in the cell.
`Fixed_aerosol_density`	Fixed aerosol density in kg m^{-3} used in the model.
`With_deposition_aerosol`	Is dry deposition taken into account for aerosol species?
`Compute_deposition_aerosol`	If set to yes, deposition velocities for aerosol species are computed with land data, otherwise they are read in files. Only needed if dry deposition is taken into account.
`With_point_emission_aerosol`	Are point emissions provided for aerosol species?
`With_surface_emission_aerosol`	Are emissions at ground provided for aerosol species?
`With_volume_emission_aerosol`	Are volume emissions provided for aerosol species?
`With_scavenging_aerosol`	Is there scavenging for aerosol species?
`With_in_cloud_scavenging`	Is there in cloud scavenging for aerosol species?
`Collect_dry_flux_aerosol`	Are the dry deposition fluxes are collected in order to postprocess them if dry deposition is taken into account?
`Collect_wet_flux_aerosol`	Are the wet deposition fluxes are collected in order to postprocess them if wet deposition is taken into account?

The bin bounds are presented as follow:

```
Bin_bounds:
# diameter of the particle classes in micrometers.
0.0 0.1 1 1.5 2 5
```

Note that these values are the bounds of the various diameter classes and that therefore there is one more value than there are classes.

5.7.2 Data Description: `polair3d-data.cfg`

In addition to the sections described in Section 5.6.2, some parameters may be necessary:

Section	Entries	Comments
`[initial_condition_aerosol]`	`Fields`, `Filename`	If initial conditions are activated (`With_initial_condition_aerosol`).
`[boundary_condition_aerosol]`	`Date_min`, `Delta_t`, `Fields`, `Filename`	If boundary conditions are activated (`With_boundary_condition_aerosol`).
`[deposition_velocity_aerosol]`	`Fields`, `Filename`	If deposition is activated (`With_deposition_aerosol`) and deposition velocities are not computed (`Compute_deposition_aerosol` set to `no`).
`[point_emission_aerosol]`	`file`	Path to the file which defines the point emissions. If point emissions are activated (`With_point_emissions_aerosol`).
`[surface_emission_aerosol]`	`Date_min`, `Delta_t`, `Fields`, `Filename`	If surface emissions are activated (`With_surface_emission_aerosol`).
`[volume_emission_aerosol]`	`Date_min`, `Delta_t`, `Nz` `Fields`, `Filename`	If volume emissions are activated (`With_volume_emission_aerosol`). `Nz` is the number of levels in which pollutants are emitted.

5.7.3 Vertical Levels and Species

Section 5.6.3 is relevant for Polair3DAerosol. In addition, there is at least a section added in the file `species.dat`:

```
[aerosol_species]

PMD   PBC   PNA   PSO4  PNH4 PNO3  PHCL  PARO1
PARO2 PALK1 POLE1 PAPI1 PAPI2 PLIM1 PLIM2 PPOA  PH2O
```

5.8 Polair3DChemistryAssimConc

`Polair3DChemistryAssimConc` is dedicated for a state space formulation of the underlying dynamical model. The stochastic modeling is implemented for diverse applications such as data assimilation.

`Polair3DChemistryAssimConc` is configured with three configuration files (`polair3d.cfg`, `polair3d-data.cfg` and `polair3d-saver.cfg`) and two data files (`levels.dat` and `species.dat`). The four files other than the main configuration file (`polair3d.cfg`) are the same as those for `Polair3DChemistry`. The main configuration file is an extension of that of `Polair3DChemistry`.

The additional parameters are:

	`[state]`
`Species`	List of species included in model state vector. All species must be on the same line.
`Levels`	List of vertical levels of model domain included in model state vector. All levels must be on the same line.
	`[data_assimilation]`
`Error_covariance_model`	Stochastic model for model and background error covariance. With option set to `Balgovind`, the corresponding error covariance matrix is calculated using Balgovind correlation function; with option set to `diagonal_constant`, the corresponding error covariance matrix is a diagonal matrix of which the diagonal elements are error variances.
`Background_error_variance`	Error variance for background concentrations. The unit for the option value is $\mu g\,m^{-3}$.
`Balgovind_scale_background`	Balgovind scale for background error covariance. The model grid interval is chosen to be the unit for option values.
`Model_error_variance`	Error variance for model simulations (in $\mu g\,m^{-3}$).
`Balgovind_scale_model`	Balgovind scale for model error covariance. The model grid interval is chosen to be the unit for option values.

The data file is the same as in Section 5.6.2, the species and levels files are the same as those presented in Section 5.6.3.

5.9 CastorTransport

5.9.1 Main Configuration File: `castor.cfg`

Model `CastorTransport` is based on IPSL model Chimere. Its option are provided in a configuration file:

	`[domain]`
`Date_min`	Starting date in any legal format (see Section 2.2.7). The date can therefore include seconds.
`Delta_t`	Time step in seconds.
`Nt`	Number of iterations of the simulation (integer).
`x_min`	Abscissa of the center of the lower-left cell. Provide a longitude (in degrees) or, in case Cartesian coordinates are chosen, an abscissa in meters.
`Delta_x`	Step length along x, in degrees (longitude) or in meters (for Cartesian coordinates).
`Nx`	Number of cells along x (integer).
`y_min`	Ordinate of the center of the lower-left cell. Provide a latitude (in degrees) or, in case Cartesian coordinates are chosen, an ordinate in meters.
`Delta_y`	Step length along y, in degrees (latitude) or in meters (for Cartesian coordinates).
`Ny`	Number of cells along y (integer).
`Nz`	Number of vertical levels (integer).
`Vertical_levels`	Path to the file that defines vertical levels interfaces. This field is read but is not used.
`Species`	Path to the file that defines involved species and their chemical properties.
	`[options]`
`With_transport`	Is transport taken into account?
`With_initial_condition`	Are initial conditions used?
`Interpolated_initial_condition`	If set to yes, initial conditions are interpolated from boundary conditions, otherwise they are read in binary files.
`With_boundary_condition`	Are boundary conditions provided?
`With_deposition`	Is deposition taken into account?
`With_volume_emission`	Are volume emissions taken into account?
	`[data]`
`Data_description`	Path to the configuration file that describes input data.

5.9.2 Data Description: `castor-data.cfg`

The data description is very similar to that of Polair3DTransport (see Section 5.5.1), except that the data can be different.

Section	Entries	Comments
[initial_condition]	Fields, Filename	If initial conditions are activated (With_initial_condition) and not interpolated (Interpolated_initial_condition set to no).
[boundary_condition]	Fields, Filename	If boundary conditions are activated (With_boundary_condition).
[meteo]	Date_min, Delta_t, Fields, Filename	Required fields are: Temperature, Pressure, Altitude, AirDensity, MeridionalWind, ZonalWind and VerticalDiffusion.
[deposition]	Date_min, Delta_t, Fields, Filename	If deposition is activated (With_deposition).
[volume_emission]	Date_min, Delta_t, Nz, Fields, Filename	If volume emissions are activated (With_volume_emission). Nz is the number of levels in which pollutants are emitted.

5.9.3 Vertical Levels and Species

A file containing vertical levels similar to the one for "Polair3D" models is read but is not useful. Give any such file.

Species file has two sections:

- `[species]` which contains all species managed by the simulation.
- `[species_ppm]` which contains all species for which an upwind scheme is not used.

5.10 CastorChemistry

Model `CastorChemistry` is derived from `CastorTransport` and all data presented in Section 5.9 are necessary for this model too. In addition some other parameters are needed.

5.10.1 Main Configuration File: `castor.cfg`

	[options]
With_chemistry	Is chemistry taken into account?
	[chemistry]
Reaction_file	Data file containing the reactions.
Stoichiometry_file	Data file containing the stoichiometry of the reactions.
Photolysis_file	
Rates_file	

5.10.2 Data Description and Species

Data Description The only difference with what is described in Section 5.9.2 is that more meteorological fields are necessary (see Section 2.5.4).

Species The species file is exactly the same as the one presented in Section 5.9.3.

5.10.3 Chemistry Files

When model `CastorChemistry` is used, four files are necessary to describe the chemistry:

- `Reaction_file` which gives the reactions between the species managed in the format

  ```
  2   CO   OH     2  HO2  CO2
  ```

  The first column gives n the number of species reacting, the n following columns give the name of the species reacting, the following columns are the number and names of the species resulting from the reaction.

- `Stoichiometry_file`
- `Photolysis_file`
- `Rates_file`

5.11 PlumeInGridDriver

The plume in grid model has been described in Section 4.6. As it is mainly used as a driver, it has been described along with other drivers. However, it is basically a model. This allows to use advanced methods using PlumeInGridDriver as a model.

Chapter 6

Modules

6.1 Transport modules

6.1.1 AdvectionDST3

Module `AdvectionDST3` is the transport module associated to advection for `Polair3D`. It is based on a third-order “direct space-time” scheme with a Koren-Sweby flux limiter. The data needed are the wind components and boundary conditions if they are available.

Please note that Courant-Friedrichs-Lewy (CFL) condition is not verified and that the user should choose the mesh dimensions and the time-step of simulations very carefully. In order to enforce the CFL, you may use module `SplitAdvectionDST3` instead.

6.1.2 SplitAdvectionDST3

Module `SplitAdvectionDST3` is the same as `AdvectionDST3` except that

- it uses directional splitting;
- it performs automatic subcycling in order to satisfy the CFL.

6.1.3 GlobalAdvectionDST3

Module `GlobalAdvectionDST3` is the same as `AdvectionDST3` for global scale (boundary conditions are not used).

6.1.4 DiffusionROS2

Module `DiffusionROS2` is the transport module associated to diffusion for `Polair3D`. It is based on a second-order Rosenbrock method. Fortran routines are used to perform all numerical computations.

6.1.5 GlobalDiffusionROS2

Module `GlobalDiffusionROS2` is the same as `DiffusionROS2` for global scale.

6.1.6 TransportPPM

Module `TransportPPM` is the numerical solver for transport used in Castor model. It uses piecewise parabolic method (PPM) for advection but can also use an upwind scheme for some species.

In the species file associated with castor there are two sections: `[species]` and `[ppm_species]`. For all species in `[species]` but not in `[ppm_species]` an upwind scheme will be used.

6.2 Chemistry Modules

6.2.1 ChemistryRACM

Module `ChemistryRACM` is the most common photochemical module used with Polair3D. It implements RACM (Stockwell et al. [1997]) and uses a second-order Rosenbrock method for time integration. Computations are performed by Fortran routines (automatically generated by the chemical preprocessor SPACK) and a C++ program is used as a frame to launch all these calculations.

It only deals with gaseous species and manages 72 species, 237 reactions (including 23 photolysis reactions).

6.2.2 ChemistryRADM

Module `ChemistryRADM` is quite similar to `ChemistryRACM`. RACM has actually been derived from RADM.

RADM manages 61 species, 157 reactions involving those species and 21 photolysis reactions.

6.2.3 ChemistryCastor

Module `ChemistryCastor` is the default chemical module for `Castor`. It involves 44 species and 118 reactions. It is based on several data files which must be provided: `Reaction_file`, `Stoichiometry_file`, `Photolysis_file` and `Rate_file`. See Section 5.10.3 for more details about these files.

6.2.4 Decay

This chemistry module is used for species (gaseous or particulate) which have a radioactive or biological decay, that is to say a natural decrease in their concentrations over time. This decay requires two more options in the configuration file (`polair3d.cfg`).

[options]	
`With_time_dependence`	If set to yes, the value of the half-life time for each species depend on the time of the day (see below).
`With_filiation_matrix`	If set to yes, decay and filiation are represented by a matrix (see below).

Note that `With_time_dependence` and `With_filiation_matrix` cannot be both set to yes at the same time.

Use of One Value of Decay The first possibility is that each species has a half-life time which is given in `species.dat`. In that case `With_time_dependence` and `With_filiation_matrix` are both set to no. The variation of concentration due to decay only is described in equation (6.1) where $T_{1/2}$ is the species half-life time in days and t_0 is a reference time. If a species has no decay, its half-life time is set to 0, and this is interpreted by `Decay` as the fact that concentration does not vary due to decay.

$$C(t) = C(t_0)\exp\left(-\frac{(t-t_0)\ln 2}{T_{1/2}}\right) \tag{6.1}$$

The parameters needed are provided in `species.dat` as follow.

```
[species]

Sp1 Sp2 Sp3 Sp4

[aerosol_species]

Aer1 Aer2

[half_life]

# Half-lives in days, put 0 for species without decay.
Sp1 300
Sp2 216
Sp3 0
Sp4 41

[half_life_aerosol]

# Half-lives in days, put 0 for species without decay.
Aer1   250
Aer2   120
```

Use of Two Values of Decay Another option is that each species has two values of $T_{1/2}$, one for the day and one for the night. This is in particular the case for species which have a biological effect. As before, for a species without decay, both half-life times are set to 0. The equation involved is very similar to equation (6.1), except that the value of $T_{1/2}$ can vary. In that case `With_time_dependence` is set to yes and `With_filiation_matrix` to no.

The parameters needed are provided in `species.dat`.

```
[species]

Sp1 Sp2 Sp3 Sp4

[aerosol_species]

Aer1 Aer2

[half_life_time]

# Half-lives in days, put 0 for species without decay.
# First value for day, second for night.
```

```
Sp1 300 500
Sp2 216 300
Sp3 0 0
Sp4 41 72

[half_life_time_aerosol]

# Half-lives in days, put 0 for species without decay.
# First value for day, second for night.
Aer1: 250 350
Aer2: 120 180
```

`Decay` tests whether it is day or night and chooses the value of half-life time to use.

Use of a Filiation Matrix The last solution is that a single matrix (called filiation matrix) is specified for all gaseous species (and one for all aerosol species), which takes into account both decay and the fact that a species can react to form other species. As a result, the evolution of the concentration due to chemistry only is described in equation (6.2). In that case `With_time_dependence` is set to no and `With_filiation_matrix` to yes.

$$C^{n+1}(x,y,z) = AC^n(x,y,z) \tag{6.2}$$

where A is the $s \times s$ *filiation matrix* and $C^n(x,y,z) = \begin{pmatrix} c_0^n(x,y,z) \\ \dots \\ c_i^n(x,y,z) \\ \dots \\ c_{s-1}^n(x,y,z) \end{pmatrix}$ with $c_i^n(x,y,z)$ the concentration of species i at time-step n in point of coordinate (x,y,z) and s the number of species involved.

The parameters are specified as follows, in `species.dat`

```
[species]

Sp1 Sp2 Sp3 Sp4

[aerosol_species]

Aer1 Aer2

[filiation_matrix]
File: example/decay/matrix.dat

[filiation_matrix_aerosol]
File: example/decay/matrix_aer.dat
```

The $s \times s$ filiation matrix is specified in file matrix.dat as below :

```
0.7 0.05 0 0.1
0 0.8 0.1 0.05
0.1 0.1 0.6 0.1
0.15 0 0.1 0.7
```

The matrix for aerosol species is very similar to the one for gaseous species.

6.3 Aerosol Modules

6.3.1 AerosolRACM_SIREAM

To use this module, *sources* for ISORROPIA (Nenes et al. [1998]) are necessary. You can obtain them from its home page `http://nenes.eas.gatech.edu/ISORROPIA/`. After you obtained and extracted the files, you have to rename `ISOCOM.FOR` as `isocom.f`, `ISOFWD.FOR` as `isofwd.f`, `ISOREV.FOR` as `isorev.f` and `ISRPIA.INC` as `isrpia.inc` and to put them in directory `include/isorropia`.

And finally compile the file `polair3d-siream.cpp` using the makefile `makefile.siream` provided. A specific makefile has been provided because AerosolRACM_SIREAM redefines some Fortran routines defined by ChemistryRACM, and because the flag `POLYPHEMUS_WITH_AEROSOL_MODULE` should be activated.

AerosolRACM_SIREAM This chemistry module is used for gas and aerosol species for general purposes as air quality modeling and risk assessment. The gas chemistry is solved with the RACM (Stockwell et al. [1997]) mechanism and the aerosol dynamics by the SIREAM model (Debry et al. [2007]). When a cloud is diagnosed in one cell of the domain, then instantaneous aerosol activation is assumed and the SIREAM model is replaced by the VSRM cloud chemistry model (Fahey and Pandis [2003]).

The number of aerosol bins is directly inferred from the number of bounds provided by the `Bin_bounds` option in main configuration file (`polair3d.cfg`). Further options are required in this configuration file.

[options]	
`With_pH`	Does the aerosol module returns cloud droplet pH?
`Scavenging_model`	Which below cloud scavenging model is used?
`Lwc_cloud_threshold`	Liquid water content threshold for clouds.
`With_coagulation`	Is coagulation taken into account?
`With_condensation`	Is condensation taken into account?
`With_nucleation`	Is nucleation taken into account?
`Fixed_aerosol_density`	Fixed aerosol density in $\mathrm{kg\,m^{-3}}$ used in the module
`With_cloud_chemistry`	Is cloud chemistry taken into account?
`With_in_cloud_scavenging`	Is in-cloud scavenging taken into account?
`With_heterogeneous_reactions`	Are heterogeneous reactions taken into account?
`With_kelvin_effect`	Is Kelvin effect taken into account?
`Dynamic_condensation_solver`	Which solver is used for dynamic condensation?
`Fixed_cutting_diameter`	Fixed cutting diameter in µm.
`Sulfate_computation`	Which method is used to solve sulfate condensation? Choices are `equilibrium` or `dynamic`.
`Redistribution_method`	Which redistribution method is used?
`Nucleation_model`	Which nucleation model is used?
`With_fixed_density`	Is aerosol density fixed in the module?
`Wet_diameter_estimation`	Which method is used to compute aerosol wet diameters?

The liquid water content threshold is the amount of liquid water in the air above which a cloud is diagnosed in the cell.

This chemistry module returns the cloud droplet pH, this means that `With_pH` can be set to yes, and that `microphysical-pH` scavenging model can be used. Otherwise choosing the `microphysical-pH` scavenging model may result in crash or errors.

Note that options `With_pH`, `Lwc_cloud_threshold` and `Fixed_aerosol_density` are used by both model and module. That is to say the fixed aerosol density is the same in the model as in the module.

The fixed cutting diameter has to be given as an aerosol diameter in µm. Aerosol bins below that diameter are assumed at equilibrium, and those above that diameter are not considered at equilibrium. The criteria is the comparison between the fixed cutting diameter and the bin bounds. The aerosol bin whose bounds are surrounding the fixed cutting diameter is included in the equilibrium bins.

Dynamic condensation is intended for aerosol bins which are not at equilibrium, and therefore time resolved mass transfer has to be computed for them. The solver for dynamic condensation may be set to either `etr` or `ros2` or `ebi`. The `etr` solver is an Explicit Trapezoidal Rule second order algorithm, the `ros2` solver is the Rosenbrock implicit second order scheme (Rosenbrock [1963]), and `ebi` is an Euler Backward Iterative scheme. Each of these solvers usually needs some numerical parameters, these are gathered in the Fortran include file `paraero.inc`.

Option `With_kelvin_effect` only affects dynamic bins.

Among aerosol species, sulfate condensation may have a different treatment. If `Sulfate_computation` is set to `equilibrium` then its treatment is equivalent to other species for both equilibrium and dynamic bins. But if it is set to `dynamic` then sulfate condensation is time resolved for all bins, using an analytic solution of mass transfer equations. This method is implemented in the `sulfdyn.f` Fortran routine.

As dynamic condensation is solved with a Lagrangian scheme, a redistribution process over the fixed aerosol size grid has to be performed at the end of condensation. Two methods are possible: `number-conserving` or `interpolation`. The former conserves the relationship between mass and number concentration in each bin, the latter relaxes this relationship.

Available nucleation models are either `binary` nucleation H_2SO_4–H_2O (Vehkamäki et al. [2002]) or `ternary` nucleation H_2SO_4–H_2O–NH_3 (Arstila et al. [1999]).

In the aerosol module, the aerosol density can be either fixed or recomputed at run time according to option `With_fixed_density`. If set to yes the aerosol density will always be equal to the fixed aerosol density mentioned above, if set to no the module will recompute one density for each aerosol bins according to their chemical compositions given by the thermodynamic model.

Two parameterizations are available to compute wet diameters depending on the option `Wet_diameter_estimation`. If set to `Isorropia` the aerosol liquid water content computed by the thermodynamic model, for instance ISORROPIA, is used. If set to `Gerber`, a simpler but faster method, the Gerber formula, is used.

Note that when `Gerber` option is used, the aerosol density is fixed for all aerosol processes except condensation even if option `With_fixed_density` is set to no. In other words if run time computation of density is chosen, it will only affect condensation. Indeed when using the Gerber formula for fastness purpose there is little interest in recomputing density. Then the fixed density is that specified with `Fixed_aerosol_density` option.

6.3.2 Decay

Module `Decay` also supports aerosols. See Section 6.2.4.

Chapter 7

Postprocessing

7.1 Graphical Output

7.1.1 Installation and Python Modules

In Polyphemus, we advocate the use of Matplotlib (with NumPy), Basemap and AtmoPy. Please refer to Sections 1.2 and 1.3 for system-wide installation notes. Below are installation notes for the user.

IPython and Matplotlib

Matplotlib is a Python 2D plotting library which produces high quality figures. On its website `http://matplotlib.sourceforge.net/`, you may find help for all Matplotlib commands. You may also find many useful examples (Sections "Screenshots" and "Examples (zip)"), a complete tutorial, a useful FAQ, an interesting "cookbook/wiki" and other resources. We highly recommend the use of Matplotlib (actually imported through module `pylab` – see below) together with IPython (enhanced Python shell, `http://ipython.scipy.org/`) to benefit from a powerful interactive mode.

Once IPython, NumPy and Matplotlib installed, launch IPython with command `ipython`. IPython needs to be launched once to complete its (user) installation. You should get the prompt "`In [1]:`". If you are aware of Python, you can execute Python commands from this prompt.

Now, try to import Matplotlib:

```
In [1]: from pylab import *
```

If no error occurs, your installation is mostly complete. Quit IPython (`ctrl+d` and `RET`). Now edit Matplotlib configuration file. Under Linux or Unix, it is located in `~/.matplotlib/matplotlibrc`. Under Windows, it is in `C:\Documents and Settings\yourname\.matplotlib`. You should find the entries `numerix` and `interactive`. Edit them if necessary (warning: it is case sensitive), so that you have:

```
numerix      : numpy  # numpy, Numeric or numarray
interactive  : True   # see http://matplotlib.sourceforge.net/interactive.html
```

You may change the `backend` depending on what is installed on your computer. Polyphemus development team mostly uses the backend `TkAgg`:

```
backend      : TkAgg
```

But other interactive backends are fine. If you have any question about the backends, consult Matplotlib website.

Now we test the installation. Launch IPython and:

```
In [1]: from pylab import *

In [2]: plot([5, 8])
```

On screen, you should get a new window (a figure) with a line (first diagonal) from $(0, 5)$ to $(1, 8)$. And the prompt should still be available:

```
In [3]:
```

Make another plot:

```
In [3]: plot([6, 7])
```

You should get a new line (in green, probably). The prompt should still be there:

```
In [4]:
```

Basemap

Basemap extends Matplotlib so that one may display fields with a map in the background (World map, Europe map . . .).

You can test your Basemap installation with:

```
>>> from pylab import *
>>> from matplotlib.toolkits.basemap import Basemap
>>> m = Basemap(projection = 'cyl')
>>> m.drawcountries()
>>> m.drawcoastlines()
>>> draw()
```

This should show a World map. Other examples are available on Matplotlib website.

AtmoPy

Recall that `extract_configuration.cpp` must be compiled (in `atmopy/talos`) – see Section 1.3. Then make sure that Python will be able to find the AtmoPy directory in order to load it. In the distribution, AtmoPy is in directory `Polyphemus/include/atmopy`. When you load AtmoPy under Python (`from atmopy import *` in a program or in IPython), Python searches for the directory `atmopy` in the local directory (that is, `./atmopy`). If `atmopy` is not in the current directory, Python searches for `atmopy` in all paths of `$PYTHONPATH` (under Linux and Unix).

Hence you have two options:

1. copy directory `atmopy` (from `Polyphemus/include/`) into the current directory, or create a symbolic link to it (`ln -s /path/to/Polyphemus/include/atmopy`); or
2. update `$PYTHONPATH` so that it includes the path to `atmopy`, e.g., in Bash: `export PYTHONPATH=$PYTHONPATH:/path/to/Polyphemus/include/atmopy`, which you may put in `~/.bashrc`.

If Matplotlib and Basemap are properly installed, then AtmoPy should work. A test function is provided with AtmoPy to check:

```
>>> from atmopy import *
>>> atmopy_test()
```

If you get Figure 7.1, AtmoPy is properly installed.

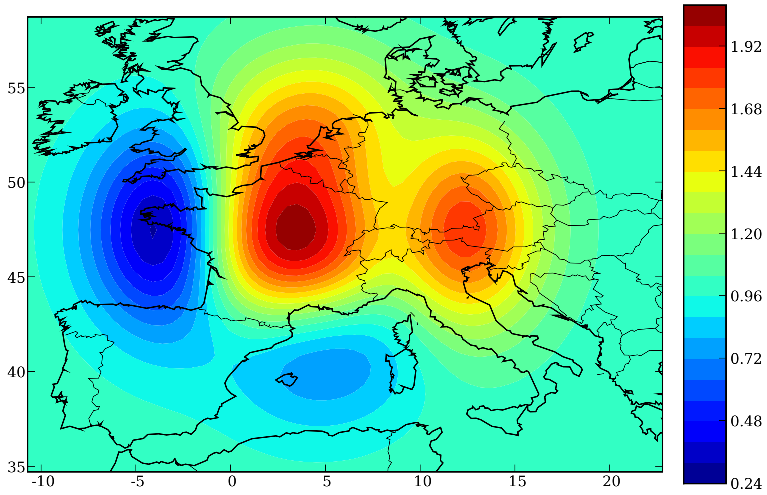


Figure 7.1: Output of function `atmopy_test`. If you get this figure, AtmoPy is properly installed.

7.1.2 A Very Short Introduction to Python and Matplotlib

In this section, several examples are meant to introduce to Python and Matplotlib. The following commands are sometimes given with comments (after character `#`). You can execute them under IPython if you wish.

Base and Lists

```
>>> x = 5
>>> y = 2 * x + 7
>>> print 2 * y # Under IPython, print is useless; just type 2 * y
34
>>> y = 7; print "y =", y # Combined commands with semicolon
y = 7
>>> a = [-1, 5, 3] # 'a' is a list
>>> print a[0] # 0 is the first index
-1
>>> a = range(10); print a
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> for i in range(3):
...     print "i =", i
i = 0
i = 1
i = 2
```

In Python, blocks are delimited by the *indentation*. For instance (without the prompt >>>):

```
y = 0; x = 0
for i in [2, 5, 15]:    # First loop
    for j in [1, 3, 9]: # Nested loop
        x += i * j      # Inside the inner loop
    y += x              # Outside the inner loop, but still inside the outer loop
print y                 # Outside the loops; the loops are "closed"
```

Arrays (NumPy)

```
>>> from numpy import * # Loads NumPy
>>> a = arange(6, dtype = 'd') # "d" means double (floating point precision)
>>> print a
[ 0.  1.  2.  3.  4.  5.]
>>> a = zeros((2, 3), dtype = 'd') # 2D array
>>> print a.shape
(2, 3)
>>> a[1, 2] = 5.
>>> a[0, :] = 10. # Fills the first line
>>> a[1, 0:2] = -1. # From column 0 to column 1 (2 is excluded)
>>> print a
[[ 10.  10.  10.]
 [ -1.  -1.   5.]]
>>> print a.sum(), a.max(), a.min(), a.mean(), a[:, 0].sum()
33.0 10.0 -1.0 5.5 9.0
>>> print 2. * a[:, 1:] - 2. # Calculation without the first column
[[ 18.  18.]
 [ -4.   8.]]
```

Matplotlib

```
>>> from numpy import *
>>> from pylab import * # Loads Matplotlib
>>> x = arange(10, dtype = 'd')
>>> y = x * x
>>> plot(y)
>>> plot(x, y)
>>> figure() # New figure
>>> plot(x, y, "k-") # "k" for "black", "-" for a solid line
>>> figure(1) # Comes back to the first figure
>>> plot(x, y, "k--") # "--" for a discontinuous line
>>> close() # Closes current figure
>>> plot(x, y, "k:") # ":" for a dotted line
>>> clf() # Clears the figure
>>> plot(x, y, "k-", label = "Simple") # "label =" is for the legend
>>> plot(x, 2. * y, "k--", label = "Double")
>>> legend()
>>> xlabel("Abscissa")
>>> ylabel("Ordinate")
>>> savefig("plot_example.eps") # Saves the figure in EPS (could be PNG or JPG)
```

7.1.3 Visualization with AtmoPy

AtmoPy provides functions to use Basemap easily and to process data (mainly statistics). It is first used to load binary files (generated in preprocessing, or output of a model or a driver).

Configuration File: `disp.cfg`

In order to load and process data in a binary file, it is convenient to use AtmoPy with a small configuration file, often called `disp.cfg`. This file describes the data to be read:

`[input]` (optional)	
`Nt`	Number of time steps to be read in the binary file. It can be less than the total number of time steps in the file. It cannot be more. If you want to load all available steps, put `0`: `Nt` will be deduced from the file size and other dimensions (`Nx`, `Ny` and `Nz`). If you do so, please check the number of steps that are actually read by AtmoPy; if the number of steps is surprising, check `Nx`, `Ny` and `Nz` in your configuration file.
`x_min`	Abscissa (longitude) of the center of the lower-left cell. It is primarily used to load a background map in figures.
`Delta_x`	Space step along x (longitude). It is primarily used to load a background map in figures.
`Nx`	Number of cells along x.
`y_min`	Ordinate (latitude) of the center of the lower-left cell. It is primarily used to load a background map in figures.
`Delta_y`	Space step along y (latitude). It is primarily used to load a background map in figures.
`Ny`	Number of cells along y.
`Nz`	Number of vertical layers.
`file`	Path to the binary file containing the data.

Here is an example of such a configuration file where the data to be read is in `results/O3.bin` (e.g., ozone at ground level):

```
[input]

Nt = 121
x_min    = -10.0  Delta_x = 0.5  Nx = 67
y_min    = 35     Delta_y = 0.5  Ny = 46
Nz = 1

file: results/O3.bin
```

Note that `general.cfg`, `polair3d.cfg`, . . . contain similar entries. You may simply copy and paste these entries. The number of time steps and vertical layers might be different. For instance, `polair3d.cfg` contains the number of model layers, not necessary the number of levels in the target file.

Python Commands: Loading and Processing Data

In IPython, AtmoPy first reads the configuration file (`disp.cfg`):

```
>>> from atmopy import * # Loads AtmoPy
>>> from atmopy.display import * # Loads AtmoPy submodule display
>>> d = getd("disp.cfg") # d is a 4D array
```

You may overwrite the entries in `disp.cfg`:

```
>>> d = getd("disp.cfg", filename = "results/NO.bin") # Loads another file
                                                       # without editing disp.cfg
>>> d = getd("disp.cfg", Nt = 0) # Overwrites Nt
>>> d = getd("disp.cfg", filename = "results/NO.bin", Nt = 0)
>>> d = getd("disp.cfg", filename = "results/NO.bin", Nt = 0, Nz = 2)
```

The array `d` has four dimensions: `Nt`×`Nz`×`Ny`×`Nx`. Hence `d[10, 0, 2, 9]` refers to data at the time step #10 (11th time step since indices start at 0), in the first level, in horizontal cell with indices 2 along y and 9 along x. Another example is `d[15, 0]` which is a 2D array (dimensions: y, x) of data at 16th time step and in the first layer.

A few examples show the way data can be manipulated:

```
>>> d = getd("disp.cfg", filename = "results/O3.bin")
>>> d_ref = getd("disp.cfg", filename = "results/O3-reference.bin")
>>> print d.shape # Same as d_ref.shape: Nt = 48, Nz = 1, Ny = 46, Nx = 67
(48, 1, 46, 67)
>>> print d.mean()
78.5597571528
>>> print abs(d - d_ref).mean()
16.99608674
>>> from numpy import * # Needed for sqrt (see below)
>>> print sqrt(((d - d_ref) * (d - d_ref)).mean()) # Elementwise multiplication
22.6488311576
>>> print (d[10:25] - d_ref[10:25]).min() # Selected time steps
-78.5329427719
>>> print (d[:, 0, 23, 34] - d_ref[:, 0, 23, 34]).max() # Selected cell in the
                                                         # middle of the domain
-1.01527786255
```

Python Commands: Visualization

Using AtmoPy:

```
>>> from atmopy import * # Loads AtmoPy
>>> from atmopy.display import * # Loads AtmoPy submodule display
>>> from pylab import * # Matplotlib is needed for figure() and plot()
>>> d = getd("disp.cfg") # d is a 4D array
>>> m = getm("disp.cfg") # Loads the background map; an empty figure should pop up
>>> disp(m, d[2, 0]) # Displays data at the third time step and first level
>>> disp(m, d[2, 1]) # Next vertical level
>>> figure() # Another figure
>>> disp(m, d[0, 0], vmin = 0, vmax = 200) # Data range (for the color bar)
>>> dispcf(m, d[10, 0]) # With contours
>>> dispcf(m, d[10, 0], V = 10) # With ten contours
>>> dispcf(m, d[10, 0], V = [0, 50, 100, 150, 200]) # Sets the contours
>>> disp(m, d[10, 0], interpolation = "nearest") # No interpolation
```

Figure 7.1 shows the result of `dispcf` with 25 contours.

Without `disp` or `dispcf` (in case there is no background map, e.g. at small scale):

```
>>> contourf(d[10, 0])
>>> colorbar()
```

Figure 7.2 illustrates `contourf`.

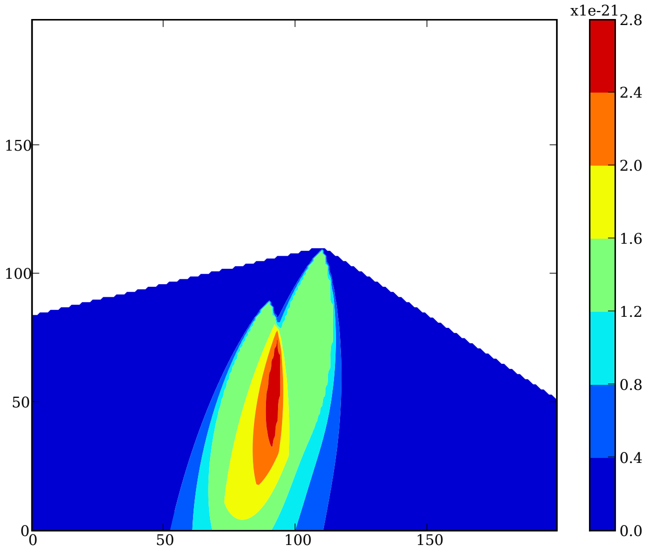


Figure 7.2: Concentration map obtained with the command `contourf`.

In addition, functions `stat.spatial_distribution` and `stat.time_evolution` may be very useful:

```
>>> d = getd("disp.cfg")
>>> print d.shape
(48, 1, 46, 67)
>>> d_max = stat.spatial_distribution(d, "max") # Takes time maxima
>>> print d_max.shape
(1, 46, 67)
```

In every cell, function `stat.spatial_distribution` takes the maximum concentration over the time. If you want to display the time averages:

```
>>> dispcf(m, stat.spatial_distribution(d, "mean")[0])
```

Function `stat.time_evolution` computes the time evolution of a spatial indicator. For instance:

```
>>> d_min = stat.time_evolution(d, "min") # Spatial minimum as function of time
>>> print d_min.shape
(48,)
>>> plot(d_min, label = "Spatial minimum")
>>> plot(stat.time_evolution(d, "max"), label = "Spatial maximum")
```

To Get Further Help

In IPython command line, you can get help this way:

```
>>> help(plot)
>>> help(stat.time_evolution)
```

In addition, all AtmoPy functions are described in a reference documentation (generated with epydoc). See AtmoPy web page: `http://www.enpc.fr/cerea/polyphemus/atmopy.html`.

Other online resources:

1. `http://diveintopython.org/`, learning Python (most useful chapters: 2, 3, 4 and 6);
2. `http://docs.python.org/`, documentations about Python;
3. `http://www.scipy.org/Tentative_NumPy_Tutorial`, introduction to NumPy;
4. `http://matplotlib.sourceforge.net/`, Matplotlib website, see Sections "Tutorial" and "Screenshots";
5. `http://www.scipy.org/`, SciPy library which includes many scientific modules (linear algebra, optimization, etc.).

7.2 Postprocessing for Gaseous Species

7.2.1 Configuration File

The configuration file `simulation.cfg` is needed for `disp.py` and `evaluation.py`.

Here is a brief explanation of the various options provided in `simulation.cfg` but more details can be found in the file itself for some options.

	`[input]`
`file`	Binary file with the results to postprocess.
`multiple_file`	Boolean stating whether several files are used.
`t_min`	Initial date of the binary file(s), in format YYYYMMDD or YYYYMMDDHH.
`Delta_t`	Time step in hours.
`Nt`	Number of time step in the binary file(s).
`x_min`	Abscissa of the center of the lower-left cell (longitude in degrees).
`Delta_x`	Step length along x, in degrees (longitude).
`Nx`	Number of cells along x (integer).
`y_min`	Ordinate of the center of the lower-left cell (latitude in degrees).
`Delta_y`	Step length along y, usually in degrees (latitude).
`Ny`	Number of cells along y (integer).
`Nz`	Number of vertical levels (integer).
`station_file`	File describing the stations.
`station_file_type`	Type of station file (Emep, Airbase, BDQA, Pioneer).
`obs_dir`	Directory where observations are stored.
	`[output]`
`station`	Name of the station for which concentrations and observations are displayed.

`t_range`	Dates for which concentrations and observations are displayed.
`concentrations`	What kind of concentrations are displayed: hourly, daily or peak concentrations?
`paired`	Should peak concentrations be paired in time?
`daily_basis`	In case daily concentrations are chosen, are observations provided on a daily basis?
`y_range`	If two numbers are provided, they define the axis range along y (and along x for scatter plots). If only one number is provided, the axis ranges are automatically set.
`scatter`	Is there a scatter plot? See the configuration file for the various options.
`meas_style`	Style for the display of measurements.
`sim_style`	Style for the display of simulated data.
`select_station`	Which stations are involved in statistical measures? Either set to `single` for a single station (defined in `station`), `all` for all stations or a couple *Field-Value* for all stations for which `Field` is equal to `Value`.
`measure`	Statistical measures to be applied to data (see configuration file for all measures available).
`cutoff`	All observations below `cutoff` are discarded.
`ratio`	All stations for which the ratio between the number of available observations and the total number of time steps is below `ratio` are discarded. For instance, if `ratio` is set to `0.3`, stations with over 70% of missing observations are discarded.
`output`	Type of the output (summary, statistics for all stations or results written in a file).
	`[file_list]`
	List of files used if `multiple_file` is set to yes.
	`[legend]`
	List of the legends associated to the files in `[file_list]` (in the same order).

7.2.2 Script `evaluation.py`

Script `evaluation.py` is meant to assess the performances of a chemistry-transport model (CTM). Results of the CTM are compared to measurements at stations and statistics on the differences are computed. The output of the script is presented on screen or can be saved in a file.

7.2.3 Script `disp.py`

This script written in Python allows to display concentrations and observations at the station `station` with regard to the time. Measurements are displayed with the style defined in `meas_style` and simulation results with `sim_style`.

7.3 Liquid water content diagnosis

The post-processing program `~/polyphemus/postprocessing/water_plume/water_plume.cpp` uses meteorological data and a concentration field of water (liquid and vapor) and diagnoses the proportion of liquid water. It is launched with two configuration files `water_plume.cfg` and `general.cfg` (which can be merged into a single configuration file) and a date.

7.3.1 Configuration File: `water_plume.cfg`

`[simulation]`	
`Date`	Simulation first day.
`t_min`	Beginning time (in seconds) since midnight.
`Delta_t`	Simulation time step.
`Nt`	Simulation number of time steps.
`PlumeWater`	File containing the simulation results (total water concentration).
`Factor`	Conversion factor to be applied to the water concentration field to convert it into $g\ m^{-3}$.
`[meteo]`	
`Path`	Path to the meteorological data files.
`Temperature`	Temperature file.
`Pressure`	Pressure file.
`SpecificHumidity`	Specific humidity file.
`LiquidWaterContent`	Liquid water content file.
`[parameters]`	
`source_temperature`	Liquid water potential temperature (in K) at source.
`source_water_content`	Total water content at the source (mass fraction).
`[output]`	
`LiquidWaterContent`	Output file name: file containing the field of liquid water mass fraction.

Note that the output liquid water content is the sum of plume liquid water content (that is, emitted by the source) and of ambient liquid water content (already in the atmosphere). It is a mass fraction: g of liquid water kg of air with water vapor.

The water content diagnosis is done at each simulation time step for the whole domain. The domain description is contained in `general.cfg`. Note that you may have to change the number of vertical levels in `general.cfg`, in case not all levels where saved during the simulation.

7.4 Aerosol Postprocessing

7.4.1 Configuration File

The configuration file `simulation_aerosol.cfg` is the same as `simulation.cfg` described previously, where aerosol parameters are added:

`[input]`	
`Nbins`	Number of size bins.
`computed`	If yes, the bin bounds are computed using a logarithmic law. If no, they are given in a file.

`Dmin, Dmax`	If bin bounds are computed, the minimum and the maximum diameters.
`file_bounds`	If bin bounds are given in a file, the name of the file.
`bin_index_shift`	Number of the first bin (typically 0 or 1).
`primary`	Names of the primary species in the model.
`inorganics`	Names of the inorganic species in the model.
`organics`	Names of the organic species in the model.
`primary_names`	Real names of the primary species (to be displayed).
`inorganics_names`	Real names of the inorganic species (to be displayed).
`output_species`	Aggregated data in output (PM_{10}, $PM_{2.5}$, total mass for each chemical component, total mass and number in each bin).
`with_organics`	If yes, total masses will take into account organic species.
`graph_type`	Graphs that will be displayed when launching `graph_aerosol.py` (chemical composition, mass and number distribution, time series).
`graphs_at_station`	If yes, the graphs will be displayed for the simulation at a given station. If no, graphs will be an average over the domain defined by `i_range` and `j_range`.
`i_range`	First and last indices in x direction for the considered domain.
`j_range`	First and last indices in y direction for the considered domain.
`log_plot`	If yes, the mass and number distributions will be displayed with a log scale for diameters.
`directory_list`	List of directories where outputs are, the aggregated data will be written in a file in the same directory as the output.

The file `simulation_aerosol.cfg` is used by scripts `init_aerosol.py` and `graph_aerosol.py`.

7.4.2 Script `init_aerosol.py`

The outputs of the model for aerosols will be several files: `<species>_<number>.bin` where `<species>` is an aerosol chemical component (in `[aerosol_species]`, see Section 5.5.3) and `<number>` is the index of the size bin. But often, measurements are aggregated data:

- PM_{10} and $PM_{2.5}$ are the mass of aerosol with a diameter smaller than 10 µm and 2.5 µm respectively,
- Total mass of one chemical component.

One can also be interested by the number of particles in each size bin (granulometry), or by the mass distribution along the size bins. This will be done by the script `graph_aerosol.py`, but before you have to launch `init_aerosol.py` by the command:

```
python init_aerosol.py simulation_aerosol.cfg
```

Then you can launch `disp.py` and `evaluation.py` with species such as `PM10`, `PM2.5`, `PNA` (total mass for sodium), etc.

7.4.3 Script `graph_aerosol.py`

You can launch `graph_aerosol.py` by the command:

```
python graph_aerosol.py simulation_aerosol.cfg}
```

Then each desired graphs (specified in `graph_type` section of the configuration file) will be displayed in a different window.

Appendix A

Polyphemus Eulerian Test-Case

The test case is available on Polyphemus site[†1]. In order to use the test-case, you should download:

- The meteorological data file `MM5-2004-08-09.tar.bz2`. The file is not included in the test-case so that it can be used for various applications and has not to be downloaded each time.
- The archive `TestCase-1.1-Eulerian.tar.bz2`.

Note that you should have Polyphemus installed and working in order to use the test-case.

A.1 Preparing the Test-Case

The first step is to extract the archive `TestCase-1.1-Eulerian.tar.bz2`:

```
tar -xjvf TestCase-1.1-Eulerian.tar.bz2
```

The directory `TestCase-1.1-Eulerian`, referred to as `TestCase` in what follows, will be created. It is divided in four subdirectories:

- `data`, which contains all precomputed data.
- `raw_data`, which contains all data used for preprocessing. After preprocessing, the results are stored in `data` to be used directly during the simulation.
- `config`, where configuration files are provided.
- `results`, where the results of the simulation are stored.

In addition a file `version` is included to indicate for which version of Polyphemus the test-case is designed, and which versions of the libraries are needed.

`MM5-2004-08-09` should be extracted and then placed in `raw_data`.

```
tar -xjvf MM5-2004-08-09.tar.bz2
mv MM5-2004-08-09 TestCase/raw_data/MM5/
```

Now you have all data necessary to perform preprocessing for the ground and for meteorological data. All other data (emissions, deposition velocities . . .) are provided and ready-to-use.

[†1] `http://www.enpc.fr/cerea/polyphemus/`

A.2 Modifying the General Configuration File

The file `general.cfg` is used by all preprocessing programs and as such must be the first file you modify when performing preprocessing. Make sure to modify and use the file provided in the directory `TestCase/config`. Here is a copy of this file:

```
[general]

Home: /u/cergrene/a/ahmed-dm
Path_to_test_case: <Home>/TestCase-1.1-Eulerian
Path_to_polyphemus: <Home>/Polyphemus-1.1

Directory_raw_data: <Path_to_test_case>/raw_data
Directory_computed_fields: <Path_to_test_case>/data
Directory_ground_data: <Directory_computed_fields>/ground
Programs: <Path_to_polyphemus>/preprocessing


[domain]

Date:  2004-08-09
Delta_t = 1.0
x_min = -10.0   Delta_x = 0.5   Nx = 65
y_min = 40.5    Delta_y = 0.5   Ny = 33
Nz = 5
Vertical_levels: <Programs>/levels.dat
```

Replace `Home` by the path to your home directory, `Path_to_test_case` by the path to `TestCase` and `Path_to_polyphemus` by the path to `Polyphemus`.

Other paths needed for the simulation depend on these ones so modifying them should be sufficient. The domain is defined for a simulation over Europe. Make sure that the date is 2004-08-09 (date for which meteorological raw data is provided).

It is advised to put both `TestCase` and `Polyphemus` in your home directory, as this is what will be used below.

A.3 Computing Ground Data

Ground data are not necessary to perform the simulation but they are needed to compute the vertical diffusion using Troen and Mahrt parameterization. If you wish to use Louis parameterization, this step is not necessary and you can go to Section A.4.

A.3.1 Land Use Cover

Compile and execute `luc-usgs` (from your `Polyphemus` directory):

```
  cd Polyphemus/preprocessing/ground/
  make luc-usgs
  cd ~/TestCase/
  ~/Polyphemus/preprocessing/ground/lus-usgs config/general.cfg config/luc-usgs.cfg
```

The output on screen will be:

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

Reading LUC data... done.
Building LUC data on output grid... done.

Writing output data... done.
```

A.3.2 Roughness

The preprocessing program `roughness` needs as input data the results of `luc-usgs`.

Compile and execute `roughness`.

```
  cd Polyphemus/preprocessing/ground/
  make roughness
  cd ~/TestCase/
  ~/Polyphemus/preprocessing/ground/roughness config/general.cfg
config/roughness.cfg
```

The output on screen will be:

```
Reading configuration files... done.
Reading roughness data... done.
Writing roughness binary ... done.
```

A.4 Computing Meteorological Data

No modification to configuration file `MM5-meteo.cfg` should be necessary but make sure to use the version of this file included in directory `TestCase` and not in directory `Polyphemus`.

You can open the file and check that `Database_MM5-meteo` is the path to the file `MM5-2004-08-09`, where the date is represented by `&D`. For details about the other options available in the configuration file, see Section 3.4.5.

Then compile MM5-meteo:

```
  cd ~/Polyphemus/preprocessing/meteo/
  make MM5-meteo
```

and execute it:

```
  cd ~/TestCase/
  ~/Polyphemus/preprocessing/meteo/MM5-meteo config/general.cfg config/MM5-meteo.cfg &D
```

In the command line given before, repolace &D by the date (2004-08-09). It has not been done because the line would be too long otherwise.

The output on screen will be:

```
Reading configuration files... done.
Memory allocation for grids... done.
Memory allocation for output data fields... done.
Conversion from sigma levels to heights... done.
Converting from latlon to MM5 indices... done.
Applying transformation to read fields... done.
```

```
Computing pressure... done.
Computing surface pressure... done.
Interpolations... done.
Computing Richardson number... done.
Computing attenuation...
  + Computing relative humidity and critical relative humidity... done.
  + Computing cloud profile... done.
  + Computing attenuation... done.
Linear interpolations...
  + Attenuation
  + SpecificHumidity
  + Liquid Water content
  + CloudHeight
  + SurfaceTemperature
  + SkinTemperature
  + SoilWater
  + SensibleHeat
  + Evaporation
  + SolarRadiation
  + Rain
  + FrictionModule
  + BoundaryHeight
 done.
Computing Kz... done.
Computing PAR... done.
Writing data... done.
```

Note that in that case meteorological data has been generated for 23 hours, but emissions data are only available for this length of time, so it is not necessary to generate more meteorological data.

If you want to compute vertical diffusion using Troen and Mahrt parameterization, compile and execute `Kz_TM`.

```
  cd ~/Polyphemus/preprocessing/meteo/
  make Kz_TM
  cd ~/TestCase/
   ~/Polyphemus/preprocessing/meteo/Kz_TM config/general.cfg config/MM5-meteo.cfg &D
```

As before, replace &D by the date.

The output on screen will be:

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

Extracting fields... done.

Computing Kz... done.
Writing output files... done.
```

A.5 Launching the Simulation

A.5.1 Modifying the Configuration File

You should check and modify `polair3d.cfg` if necessary. You have to check the paths (in particular check that the data and saver files are `config/polair3d-data.cfg` and `config/polair3d-saver.cfg`) and to make sure that the date for the simulation is 20040809 (date for which the meteorological data have been computed).

A.5.2 Modifying the Data File

Check `config/polair3d-data.cfg`. If you decided to use Louis parameterization for vertical diffusion, modify the file associated to `VerticalDiffusion` in the section `[meteo]`.

As before, check the paths and dates. In particular, if the dates in any section (except for `[photolysis]`, see below) are not right, you can have an error message.

```
ERROR!
An input/output operation failed in FormatBinary<T>::
Read(ifstream& FileStream, Array<TA, N>& A).
   Unable to read 42900 byte(s). The input stream is empty.
```

Indeed, input data can be computed for several days, so the program will discard the data for the days between `Date_min` in a section of `polair3d-data` and `Date_min` for the simulation. Here, as the data has been computed for one day only, it would be as if the data files were empty, hence this error.

Remark In the case of photolysis, data are provided for a whole year and `Date_min` must be 2004-01-01_12.

A.5.3 Modifying Saver File

The file `polair3d-saver.cfg` should be ready to use. You can modify the species to save (you are advised against saving concentrations for all species). You can choose to save instantaneous concentrations or concentrations averaged over `Interval_length` by setting `Averaged` to no or yes respectively.

A.5.4 Simulation

Compile the driver.

```
cd  ~/Polyphemus/driver
make polair3d
```

Launch the simulation from `TestCase`:

```
cd ~/TestCase/
~/Polyphemus/driver/polair3d config/polair3d.cfg
```

A.6 Visualizing Results

A.6.1 Modifying Configuration File

Modify `results/disp.cfg` if necessary (in particular if you have modified `polair3d-saver.cfg`).

```
[input]

# Number of time steps for which concentrations are saved.
Nt = 22
# Domain description for x and y.
x_min    = -10.0     Delta_x = 0.5     Nx = 65
y_min    = 40.5      Delta_y = 0.5     Ny = 33
# Number of levels for which concentration are saved.
Nz = 5

file: O3.bin
```

A.6.2 Using IPython

For details see Section 7.1.3. Remember that the directory `atmopy` should be in your `$PYTHONPATH`. Launch IPython and then type in command line (comments starting with “#”have been added to explain the meaning of each line):

```
cd results/
ipython
  from atmopy.display import *    # Import to the interactive session all
                                  # functions from the module 'display'
                                  # of 'atmopy'
  m = getm('disp.cfg')            # Create the map.
  d = getd('disp.cfg')            # Create a data with the results.
  dispcf(m, d[5,0])               # Display the data for the 8th time step
                                  # and the first vertical level (remember
                                  # that indices start at 0).
```

The image obtained is Figure A.1.

You can create other data if you like to visualize concentrations for other species. In that case, the map has already been created and less information is needed to create the data. In particular it is not necessary to provide a file `disp.cfg`:

```
  d2 = getd(filename = 'NO.bin', Nt = 22, Nz = 5, Ny = 33, Nx = 65)
  disp(m, d2[5,0])
```

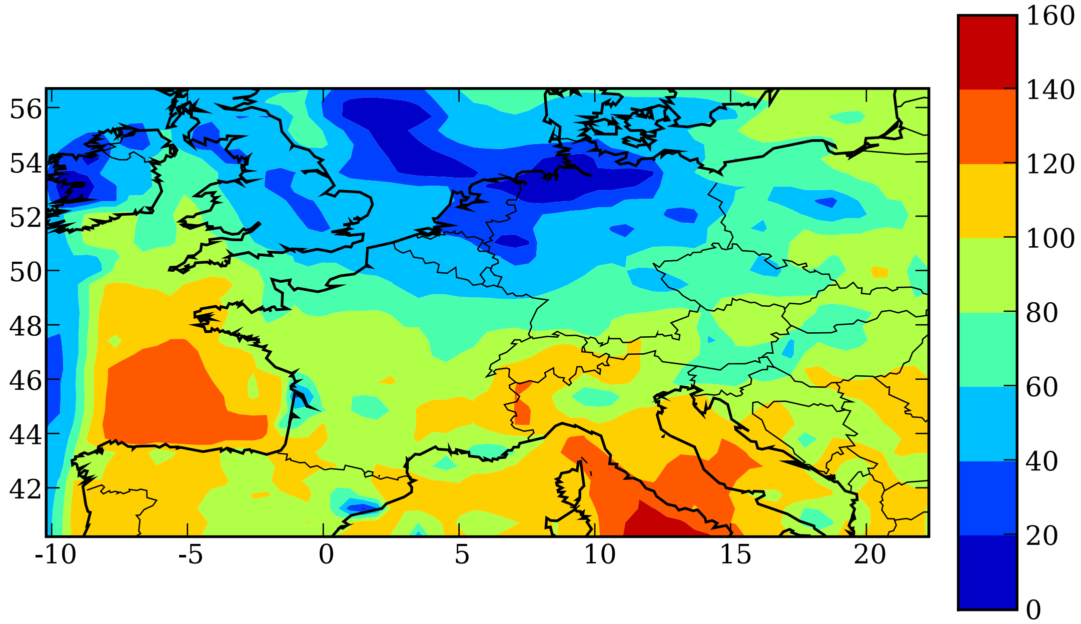


Figure A.1: Figure obtained using IPython and AtmoPy (unit is $\mu g\,m^{-3}$)

Appendix B

Polyphemus Gaussian Test-Case

This document explains how to proceed to perform simulations using the test case for Gaussian models provided with Polyphemus.

When the archive `TestCase-1.1-Gaussian.tar.bz2` is extracted a directory `TestCase-1.1-Gaussian/` is created. It is referred to below as `TestCase`.

```
tar -xjvf TestCase-1.1-Gaussian.tar.bz2
```

The subdirectory `config/` holds all configuration files necessary and the subdirectory `results/` is meant to store the results of simulations. It is divided in three subdirectories (one for each possible simulation) : `puff_line/` for the Gaussian puff model and a gaseous line source, `puff_aer/` for the puff model with point sources of gaseous and aerosol species, and `plume/` for the Gaussian plume model with gaseous species only. In each of those subdirectory a python program which allows to visualize some results easily can be found.

To launch the test cases, you do not need to modify the configuration files. In the following commands, `~/Polyphemus` and `~/TestCase` have to be replaced, if necessary, by the paths to the Polyphemus directory and test case directory respectively.

B.1 Preprocessing

Prior to use Gaussian models, you need to compute scavenging coefficients and deposition velocities for the various species. This is achieved by using `gaussian-deposition_aer`.

First compile it :

```
cd ~/Polyphemus/preprocessing/dep/
make gaussian-deposition_aer
```

Then run it from the test case directory:

```
cd ~/TestCase/config
~/Polyphemus/preprocessing/dep/gaussian-deposition_aer
                                gaussian-deposition_aer.cfg
```

The output on screen will be :

```
Reading configuration file... done.
Reading meteorological data... done.
Reading species... done.
Reading diameter... done.
Computation of the scavenging coefficients... done.
Computation of the deposition velocities..done.
Writing data... done.
```

The file `gaussian-meteo_aer.dat` has been created in the directory ∼`/TestCase/config/`. It will be used for puff simulations with aerosol species and with line source.

Note that if your simulation only involves gaseous species, you can use the preprocessing program `gaussian-deposition`. Here we use `gaussian-deposition_aer` because its output can be used for simulations with or without aerosol species.

B.2 Discretization

This step is only necessary for the simulation with a line source. Its aim is to discretize this source into a series of puffs. To do so, compile the preprocessing program `discretization`:

```
  cd ~/Polyphemus/preprocessing/emissions/
  make discretization
```

Then run it from the test case directory:

```
  cd ~/TestCase/config
  ~/Polyphemus/preprocessing/emissions/discretization
                                   discretization.cfg
```

The output on screen will be:

```
Reading configuration file... done.
Reading trajectory data...  done.
Length of the trajectory: 48.0278
Number of points on the trajectory: 28
Writing source data... done.
```

The file `puff-source-discretized.dat` has been created in the directory ∼`/TestCase/config`. It contains a series of puffs representing the discretized line source.

B.3 Simulations

B.3.1 Plume

This simulation uses the program `plume`, which is the program for the Gaussian plume model. It uses the following data:

- Gaseous species : Caesium, Iodine.
- Sources : 2 point sources for Iodine, one point source for Caesium.
- Meteorological situations : 4 situations, rotating wind with an increasing speed (0.1 $\mathrm{m\,s^{-1}}$, 2 $\mathrm{m\,s^{-1}}$, 5 $\mathrm{m\,s^{-1}}$ and 10 $\mathrm{m\,s^{-1}}$).

- Urban environment.

The simulation uses the following files :

- `plume.cfg` gives the simulation domain, the options and the paths to the other files.
- `gaussian-levels.dat` gives the vertical levels.
- `gaussian-species_aer.dat` gives the species data (species names and radioactive half-lives are used here).
- `gaussian-meteo.dat` gives all meteorological data. It does not contain scavenging coefficients or deposition velocities since the simulation will not take these processes into account. Therefore, it was not necessary to use the preprocessing program `gaussian-deposition` to create this file.
- `plume-source.dat` contains all the data on stationary sources.
- `plume-saver.cfg` contains the options and paths to save the results.

Compile the program plume :

```
cd ~/Polyphemus/driver/
make plume
```

Then execute it from ∼`/TestCase/config` :

```
cd ~/TestCase/config
~/Polyphemus/driver/plume plume.cfg
```

The output on screen will be :

```
        Temperature     Wind angle      Wind velocity   Stability
Case #0
        15              -100            0.5             D
Case #1
        10              -5              2               D
Case #2
        10              20              5               D
Case #3
        10              60              10              D
```

Results are stored in ∼`/TestCase/results/plume/`.

B.3.2 Puff with Aerosol Species

The simulation uses `puff_aer`, which is the program for puffs with aerosol species, and the following data:

- Gaseous species : Caesium, Iodine.
- Aerosol species : aer1, aer2.
- Sources : 1 point source per species.
- Meteorological situation : 1 situation with wind speed of 2 $\mathrm{m\,s^{-1}}$ and rain.

- Urban environment.

The simulation uses the following files:

- `puff_aer.cfg` gives the simulation domain, options and the paths to the other files.
- `diameter.dat` gives the aerosol diameters.
- `gaussian-levels.dat` gives the vertical levels.
- `gaussian-species_aer.dat` gives the species data (only species names are used, since all other data have been used during preprocessing).
- `gaussian-meteo_aer.dat` gives all meteorological data and data on scavenging and deposition. It was created during preprocessing (see Section B.1).
- `puff-source_aer.dat` contains all the data on gaseous and aerosol sources.
- `puff-saver_aer.cfg` contains the options and paths to save the results.

Compile the program `puff_aer`:

```
cd ~/Polyphemus/driver/
make puff_aer
```

Then execute it from `TestCase/config`:

```
cd ~/TestCase/config
~/Polyphemus/driver/puff_aer puff_aer.cfg
```

Results are stored in `~/TestCase/results/puff_aer/`.

B.3.3 Puff with Line Source

The simulation uses `puff`, which is the program for puffs with gaseous species only, and the following data:

- Gaseous species : CO2.
- Source : 1 line source.
- Meteorological situations : 1 situation.
- Urban environment.

The simulation uses the following files:

- `puff.cfg` gives the simulation domain, options and the paths to the other files. It also contains the species name.
- `gaussian-levels.dat` gives the vertical levels.
- `gaussian-meteo_aer.dat` gives all meteorological data. Scavenging coefficients and deposition velocities are not used here (since they do not correspond to the species of this simulation). Loss processes are therefore not taken into account.

- `puff-source-discretized.dat` gives data on the discretized source. It has been created using program `discretization` (see Section B.2).
- `puff-saver.cfg` gives the options and paths to save the results.

Compile the program `puff`:

```
cd ~/Polyphemus/driver/
make puff
```

Then execute it from `TestCase/config`:

```
cd ~/TestCase/config
~/Polyphemus/driver/puff puff.cfg
```

Results are stored in ∼`/TestCase/results/puff_line/`.

B.4 Result Visualization

To visualize the results of a simulation, use the interactive python interpreter IPython (launched with the command `ipython`). For details see Section 7.1.3.

B.4.1 Gaussian Plume

Launch IPython from the plume results directory:

```
cd ~/TestCase/results/plume
ipython
```

Import the modules that are needed for results visualization with the command:

```
>> import atmopy
>> from atmopy.display import *
```

Then, import the concentration field you want to visualize:

```
>> d = getd(filename = 'Iodine.bin', Nt=4, Nz=2, Ny=200, Nx=200)
```

Nt is normally the number of time steps. Here, as it is a stationary simulation, it should be equal to 1. However, as there are four meteorological situations, we have here $Nt = 4$, as each situation is similar to a time step for an unstationary simulation. It would be the same if it was an unstationary simulation (puff model) with several meteorological situations. If there are 10 time steps, and 4 meteorological situations, you will put $Nt = 40$. The first ten time steps represent the first situation, from 10 to 20 you have the concentration field for the second situation, and so on ...

To visualize the concentration over the domain for the first situation and add a colorbar, use the following commands:

```
>> contourf(d[0,0])
>> colorbar()
```

You should obtain the Figure B.1.

You can visualize the concentration field for the other meteorological situations. You should see that the wind is turning with increasing speed.

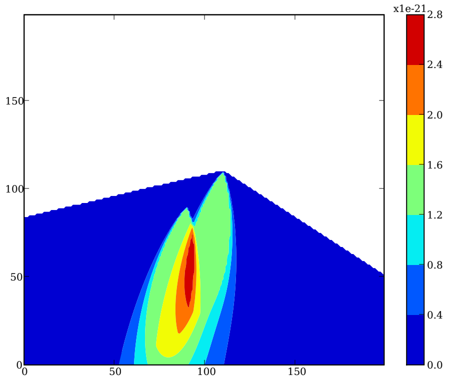


Figure B.1: Plume visualization for the first meteorological situation. Ground concentration in $\mu\,\mathrm{g}{\cdot}\mathrm{m}^{-3}$.

B.4.2 Gaussian Puff with Aerosol Species

You can go into the directory ∼`/TestCase/results/puff_aer/` and launch ipython from there, or either change directory from the ipython shell:

```
>> cd ~/TestCase/results/puff_aer/
>>
```

By doing that you ensure that you do not have to import atmopy again. However, when quitting ipython, you will be back in the directory from where it was launched.

To visualize the ground concentration on the domain at time step t, use the command:

```
>> d = getd(filename = 'aer1_0.bin', Nt=80, Nz=2, Ny=30, Nx=55)
>> contourf(d[t, 0])
```

To visualize several time steps on the same figure, just use `contourf` several times. If you want to clear the figure, use the command `clf()`. Figure B.2 gives an example of what you can obtain.

B.4.3 Gaussian Puff with Line Source

To visualize results, go to the directory ∼`/TestCase/results/puff_line/` and use the commands `getd` and `contourf`. Figure B.3 provides examples of what you can obtain.

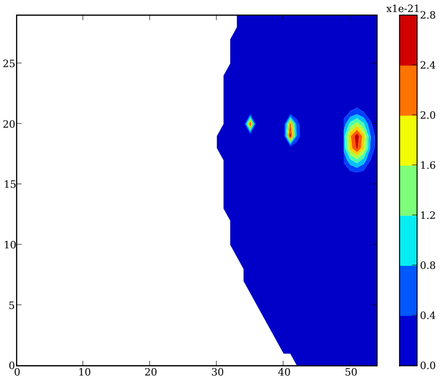
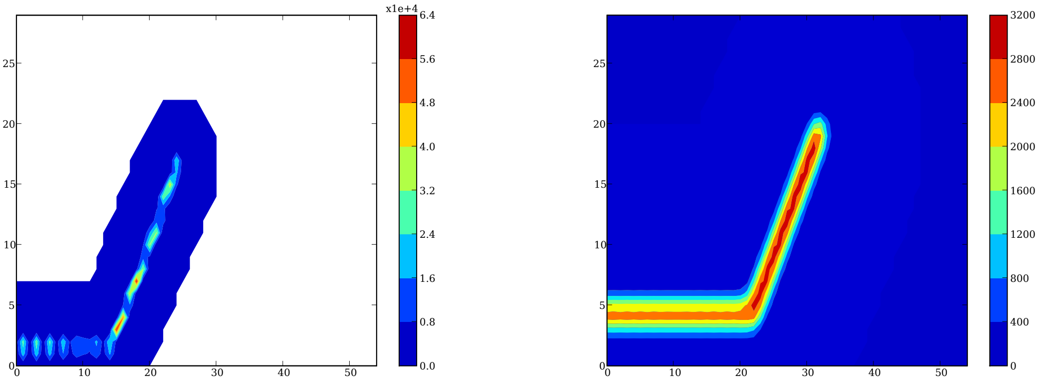
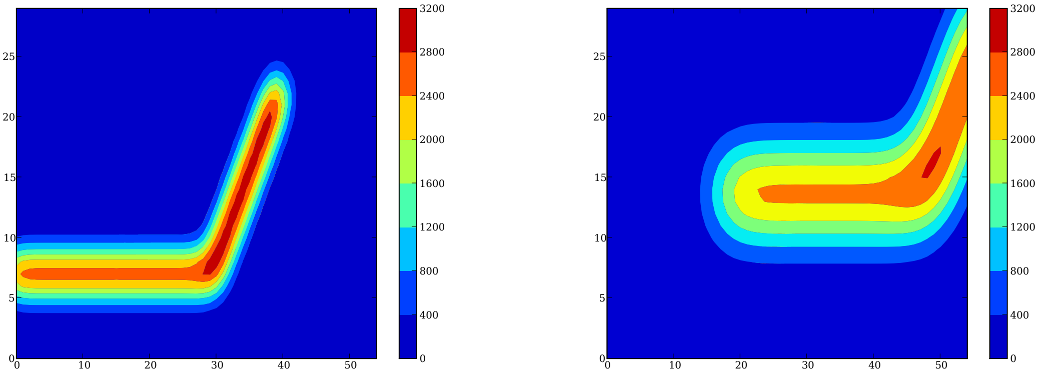


Figure B.2: Puff visualization at time steps 0, 30 and 79 for species aer1 and first diameter. Ground concentration in $\mu\,\mathrm{g}{\cdot}\mathrm{m}^{-3}$.



(a) Ground concentration at $t = 0.1$ s

(b) Ground concentration at $t = 0.3$ s



(c) Ground concentration at $t = 0.5$ s

(d) Ground concentration at $t = 0.8$ s

Figure B.3: Puff line visualization at time steps 10, 30, 50 and 79. Ground concentration in $\mu\,\mathrm{g}{\cdot}\mathrm{m}^{-3}$.

Bibliography

Arstila, H., Korhonen, P., and Kulmala, M. (1999). Ternary nucleation: kinetics and application to water-ammonia-hydrochloric acid system. *Journal of Aerosol Science*, 30(2):131–138.

Debry, E., Fahey, K., Sartelet, K., Sportisse, B., and Tombette, M. (2007). Technical Note: A new SIze REsolved Aerosol Model (SIREAM). *Atmospheric Chemistry and Physics*, 7:1,537–1,547.

Fahey, K. M. and Pandis, S. N. (2003). Size-resolved aqueous-phase atmospheric chemistry in a three-dimensional chemical transport model. *Journal of Geophysical Research*, 108(D22).

Horowitz, L. W., Walters, S., Mauzerall, D. L., Emmons, L. K., Rasch, P. J., Granier, C., Tie, X., Lamarque, J.-F., Schultz, M. G., Tyndall, G. S., Orlando, J. J., and Brasseur, G. P. (2003). A global simulation of tropospheric ozone and related tracers: description and evaluation of MOZART, version 2. *Journal of Geophysical Research*, 108(D24).

Louis, J.-F. (1979). A parametric model of vertical eddy fluxes in the atmosphere. *Boundary-Layer Meteorology*, 17:187–202.

Mallet, V., Quélo, D., Sportisse, B., Ahmed de Biasi, M., Debry, É., Korsakissok, I., Wu, L., Roustan, Y., Sartelet, K., Tombette, M., and Foudhil, H. (2007). Technical Note: The air quality modeling system Polyphemus. *Atmospheric Chemistry and Physics Discussions*, 7(3):6,459–6,486.

Nenes, A., Pandis, S. N., and Pilinis, C. (1998). ISORROPIA: A new thermodynamic equilibrium model for multiphase multicomponent inorganic aerosols. *Aquatic Geochemistry*, 4(1):123–152.

Njomgang, H., Mallet, V., and Musson-Genon, L. (2005). AtmoData scientific documentation. Technical Report 2005-10, CEREA.

Rosenbrock, H. H. (1963). Some general implicit processes for the numerical solution of differential equations. *The Computer Journal*, 5:329–330.

Simpson, D., Winiwarter, W., Börjesson, G., Cinderby, S., Ferreiro, A., Guenther, A., Hewitt, C. N., Janson, R., Khalil, M. A. K., Owen, S., Pierce, T. E., Puxbaum, H., Shearer, M., Skiba, U., Steinbrecher, R., Tarrasón, L., and Öquist, M. G. (1999). Inventorying emissions from nature in Europe. *Journal of Geophysical Research*, 104(D7):8,113–8,152.

Stockwell, W. R., Kirchner, F., Kuhn, M., and Seefeld, S. (1997). A new mechanism for regional atmospheric chemistry modeling. *Journal of Geophysical Research*, 102(D22):25,847–25,879.

Troen, I. and Mahrt, L. (1986). A simple model of the atmospheric boundary layer; sensitivity to surface evaporation. *Boundary-Layer Meteorology*, 37:129–148.

Vehkamäki, H., Kulmala, M., Napari, I., Lehtinen, K. E. J., Timmreck, C., Noppel, M., and Laaksonen, A. (2002). An improved parameterization for sulfuric acid–water nucleation rates for tropospheric and stratospheric conditions. *Journal of Geophysical Research*, 107(D22).

Wesely, M. L. (1989). Parameterization of surface resistances to gaseous dry deposition in regional-scale numerical models. *Atmospheric Environment*, 23:1,293–1,304.

Zhang, L., Brook, J. R., and Vet, R. (2003). A revised parameterization for gaseous dry deposition in air-quality models. *Atmospheric Chemistry and Physics*, 3:2,067–2,082.

Zhang, L., Moran, M. D., Makar, P. A., Brook, J. R., and Gong, S. (2002). Modelling gaseous dry deposition in AURAMS: a unified regional air-quality modelling system. *Atmospheric Environment*, 36:537–560.