Asim Documentation

Release 0.1

Matěj Laitl

September 03, 2012

Contents

Chapter 1

Asim Project

1.1 Launching

Simple simulation can be starting by calling:

`python -c 'import simulation.simple as s; s.main()`

Assimilation can be started using

`python -c 'import assimilation.twin as t; t.main()`

Calling something like *python simulation/simple.py* doesn't work for Cython build because it circumvents binary version of the simulation.simple module. The main() methofs may take optional parameters, study the sources.

1.2 Cythoning

For dramatic performance improvements you can Cythonize Python modules into C and then compile them forming binary Python modules. You need to have recent Cython (0.17 beta3 or newer) and a compiler. Cythoning is performed by calling `python setup.py build_ext -i`. Every time you change source files you need to re-Cython. Beware, Python prefers binary modules to .py modules, because of that command `python setup.py uncython` to clean any binary modules and generated C.

Chapter 2

Dispersion Model

Module with interfaces (empty classes) for core dispersion model objects.

class `dispmodel.iface.DispersionModel`

Bases: `object`

Model of propagation of radionuclides in atmosphere.

`concentration_at()`

Compute radioactive concentration at given location (`Location` (page 3)) and model's current time. TODO: units.

Returns view on 1D array with per-nuclide concentration. You may *not* modify values in the view. The view may become invalid when this or other method is called on this instance again

`deposition_at()`

Compute combined dry and wet radioactive deposition at given location (`Location` (page 3)) and model's current time. TODO: units.

Returns view on 1D array with per-nuclide deposition. You may *not* modify values in the view. The view may become invalid when this or other method is called on this instance again

`dose_at()`

Compute radioactive dose at given location (`Location` (page 3)) and model's current time in Sieverts.

Returns view on 1D array with per-nuclide dose. You may *not* modify values in the view. The view may become invalid when this or other method is called on this instance again

`propagate()`

Propagate model to next time step.

Parameters

- **meteo_model** (`MeteoModel` (page 4)) – model of weather the calculation should use
- **source_model** (`SourceModel` (page 4)) – model of the source of the release the calculation should use

class `dispmodel.iface.Location`
Bases: `object`

Describes a point in 3D space.

class `dispmodel.iface.MeteoModel`
Bases: `object`

Model of weather.

`dispersion_xy()`
Return horizonal dispersion coefficient for a puff that is currently at location *loc*, time *time* and has flewn *total_ distance* meters in total.

`dispersion_z()`
Return vertical dispersion coefficient for a puff that is currently at location *loc*, time *time* and has flewn *total_ distance* meters in total.

`mixing_layer_height_at()`
Return height of the mixing layer in meters at given location and time.

Parameters

- **loc** (`Location` (page 3)) – where the height is to be computed
- **time** (*int*) – time in seconds since start of the simulation

Return type `float`

`wind_direction_at()`
Return direction of the wind in radians at given location and time.

Parameters

- **loc** (`Location` (page 3)) – where the direction is to be computed
- **time** (*int*) – time in seconds since start of the simulation

Return type `float`

`wind_speed_at()`
Return speed of the wind in m/s at given location and time.

Parameters

- **loc** (`Location` (page 3)) – where the speed is to be computed
- **time** (*int*) – time in seconds since start of the simulation

Return type `float`

class `dispmodel.iface.Nuclide`
Bases: `object`

Nuclide describes properties of one radionuclide.

class `dispmodel.iface.SourceModel`
Bases: `object`

Model of the source of an atmospheric radioactive release.

`inventory()`
Return an array of radionuclides this release contains.

Return type array of `Nuclides` (page 4)

`location()`

Return location of the release. All three coordinates are to be used.

Return type `Location` (page 3)

`release_rate()`

Return release rate in Bq/s of all radionuclides at given time. Rate of radionuclide `inventory()[i]` (page 4) is `release_rate(time)[i]` (page 5).

Return type array of `floats`

class `dispmodel.iface._memoryviewslice`

Bases: `dispmodel.iface.memoryview`

Internal class for passing memoryview slices to Python

Chapter 3

Nuclide Database

A simple database of radionuclides.

To be used as:

```
>>> import nuclides
>>> argon = nuclides.db["Ar-41"]
>>> argon.half_life
6560.0
```

class `dispmodel.nuclides._memoryviewslice`

Bases: `dispmodel.nuclides.memoryview`

Internal class for passing memoryview slices to Python

Chapter 4

Puff Model (Dispersion Model Implementation)

Implementation of the dispersion model based on the concept of puffs.

class `dispmodel.puffmodel.Puff`

Bases: `dispmodel.iface.DispersionModel` (page 3)

One puff. Makes sense only as a part of `PuffModel` (page 9).

Parameters

- **time_step** (*int*) – number of seconds the integration step lasts. `propagate()` (page 3) advances *time_step* seconds
- **inventory** (array of `Nuclide` (page 4) objects) – radionuclides present in this release
- **Q** (array of `floats`) – per-nuclide initial activities TODO: in what units?
- **loc** (`Location` (page 3)) – initial location of the puff; Puff references it (no copy)
- **time** (*int*) – total time in seconds since start of the simulation

`dose_at()`

Computes dose at location *loc* using m/mju method

class `dispmodel.puffmodel.PuffModel`

Bases: `dispmodel.iface.DispersionModel` (page 3)

Implementation of the dispersion model based on the concept of `puffs` (page 9).

Parameters

- **time_step** (*int*) – `propagate()` (page 3) advances *time_step* seconds
- **puff_sampling_step** (*int*) – inteval between 2 puff releases; must be multiple of *time_step*. `Source model's` (page 4) `release_rate()` (page 5) is queried in *puff_sampling_step*-long intervals
- **release_loc** (`Location` (page 3)) – location where the puffs get released
- **source_model** (`SourceModel` (page 4)) – source term of the release

class `dispmodel.puffmodel._memoryviewslice`

Bases: `dispmodel.puffmodel.memoryview`

Internal class for passing memoryview slices to Python

Chapter 5

Simulation of the Puff Model

Run a simulation of an atmospheric radioactive realease using the puff model

class `simulation.simple.PasquillsMeteoModel`

Bases: `dispmodel.iface.MeteoModel` (page 4)

Abstract meteo model based on Pasquills stability category

Parameters **stability_category** (*int*) – Pasquills statiblity category A=0 ... F=5

class `simulation.simple.SimulatedSourceModel`

Bases: `dispmodel.iface.SourceModel` (page 4)

Source model that releases static pre-defined ammount of radionuclides.

Parameters

- **time_step** (*int*) – inverval between rate changes
- **rates** (*list*) – total activities per while time slots

class `simulation.simple.StaticMeteoModel`

Bases: `simulation.simple.PasquillsMeteoModel` (page 11)

Weather model that returns pre-computed numbers stored in .mat file.

Parameters

- **matfile** (*string*) – path to Matlab .mat file with pre-computed grid of wind speed and direction
- **stability_category** (*int*) – Pasquills statiblity category A=0 ... F=5

`_bilinear_interp()`

Bilinear interpolation.

class `simulation.simple._memoryviewslice`

Bases: `simulation.simple.memoryview`

Internal class for passing memoryview slices to Python

`simulation.simple.read_receptor_locations()`

Loads list of receptors from a file

Returns list of `Locations` (page 3) with receptor locations

Chapter 6

Asimilation Twin Experiment

Atmospheric radioactive release twin experiment using `simulation.simple` (page 11) and `dispmodel.puffmodel` (page 9)

class `assimilation.twin.AssimilationMeteoModel`

Bases: `simulation.simple.PasquillsMeteoModel` (page 11)

Meteorologic model used for assimilation of wind speed and direction.

Construct assimilation model.

Parameters stability_category (*int*) – Pasquills statiblity category A=0 ... F=5

class `assimilation.twin.ObservationCPdf`

Bases: `pybayes.pdfs.CPdf`

(wind speed, wind direction) is not a complete state - it needs to include puff locations, total flewn distances etc. Therefore ObservationCPdf references PuffModelEmpPdf to acces full state through its models attribute.

`eval_log()`

Evalate cpdf in point x given condition *cond*.

Parameters

- **x** – observation vector (wind speed, wind direction, doses measured at receptors)
- **cond** – state vector (at, bt)

class `assimilation.twin.TransitionCPdf`

Bases: `pybayes.pdfs.CPdf`

TODO: This can be now made ProdCPdf

`sample()`

Sample from this CPdf.

Parameters cond – condition: previous a_t, b_t

class `assimilation.twin._memoryviewslice`

Bases: `assimilation.twin.memoryview`

Internal class for passing memoryview slices to Python

Chapter 7

Math Wrappers (Cythonization helpers)

class `support.mathwrapper._memoryviewslice`

Bases: `support.mathwrapper.memoryview`

Internal class for passing memoryview slices to Python

`support.mathwrapper.iadd_vv()`

Hack for Cython - in-place addition of 2 vectors, same as lhs += rhs

`support.mathwrapper.imult_vs()`

Hack for Cython - in-place element-wise multiplication of 2 vectors, same as lhs *= rhs

`support.mathwrapper.imult_vv()`

Hack for Cython - in-place element-wise multiplication of 2 vectors, same as lhs *= rhs

`support.mathwrapper.matrix()`

Hack for Cython - creates a suitable container for a matrix of C doubles (Python `floats`)

`support.mathwrapper.sum_v()`

Hack for Cython - sums all elements of vector *summand*

`support.mathwrapper.vector()`

Hack for Cython - creates a suitable container for a vector of C doubles (Python `floats`)

Chapter 8

Graph Plotters

`support.plotters.get_square_map_of_given_side()`
vyrizne z mapy na ceste path a puvodnim rozmeru a1 novou mapu s rozmerem a2

`support.plotters.plot_stations()`
Plot points representing stations to axis *ax*

Parameters locations – iterable of Location objects with station locations

`support.plotters.plot_trajectories()`
TRAJECTORIES = numpy.zeros((pC, tC, 2)) #0 = x, 1 = y

Chapter 9

Indices and tables

- *genindex*
- *modindex*
- *search*

Index

R

S

T

V

W