Asim Documentation

Release 0.1

Matěj Laitl

May 05, 2014

Contents

Chapter 1

Asim Project

1.1 Building & Installing

Current dependencies:

- Ceygen 0.3 or newer
- PyBayes, latest tip of the *experimental* branch
- Cython 0.19.1 or newer

`python setup.py build` to build. See `python setup.py build_ext -h` for possible options for the important build_ext subcommand.

Type `python setup.py test` to run the simulation and assimilation in a twin experiment.

Type `python setup.py install` to install.

1.2 Launching

Simple simulation can be starting by calling:

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

Assimilation can be started using

```
python -c 'import asim.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() methods may take optional parameters, study the sources.

Chapter 2

Dispersion Model

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

class `asim.dispmodel.iface.Location(`*x*, *y*, *z*`)`

Bases: `object`

Describes a point in 3D space.

class `asim.dispmodel.iface.Nuclide(`*dose_mju*, *dose_mjue*, *dose_a*, *dose_b*, *dose_e*, *dose_cb*, *half_life*, *depo_vg*`)`

Bases: `object`

Nuclide describes properties of one radionuclide.

`half_life` **= None**

nuclide half-life in seconds

`lambda_` **= None**

decay constant $\lambda = \frac{\ln 2}{\text{half\_life}}$

class `asim.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 3)

`release_rate(`*time*`)`

Return release rate in Bq/s of all radionuclides at given time. Rate of radionuclide `inventory()[i]` (page 3) is `release_rate(time)[i]` (page 3).

Return type array of `floats`

`location()`

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

Return type `Location` (page 3)

class `asim.dispmodel.iface.MeteoModel`

Bases: `object`

Model of weather.

`mixing_layer_height_at`(*loc*, *time*)
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_speed_at`(*loc*, *time*)
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`

`wind_direction_at`(*loc*, *time*)
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`

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

class `asim.dispmodel.iface.DispersionModel`
Bases: `object`

Model of propagation of radionuclides in atmosphere.

`propagate`(*meteo_model*, *source_model*)
Propagate model to next time step.

Parameters

- **meteo_model** (`MeteoModel` (page 3)) – model of weather the calculation should use
- **source_model** (`SourceModel` (page 3)) – model of the source of the release the calculation should use

`concentration_at`(*loc*)
Compute radioactive concentration at given location (`Location` (page 3)) and model's current time. [Bq/m^3]

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

`dose_at`(*loc*)

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

`deposition_at`(*loc*)

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

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

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
```

`asim.dispmodel.nuclides.NOBLE_DEPO_VG` = **0.0**
constant: deposition of noble radionuclides

`asim.dispmodel.nuclides.AERO_DEPO_VG` = **0.003**
constant: deposition of aero radionuclides

`asim.dispmodel.nuclides.ELIOD_DEPO_VG` = **0.015**
constant: deposition of eliod radionuclides

`asim.dispmodel.nuclides.db` = **{'Ar-41': <Nucl. half_life=6560.0s>}**
structure holding nuclide database

Chapter 4

Puff Model (Dispersion Model Implementation)

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

class `asim.dispmodel.puffmodel.Puff`(*time_step*, *inventory*, *Q*, *loc*, *time*)

Bases: `asim.dispmodel.iface.DispersionModel` (page 4)

One puff. Makes sense only as a part of `PuffModel` (page 10).

Parameters

- **time_step** (*int*) – number of seconds the integration step lasts. `propagate()` (page 4) advances *time_step* seconds
- **inventory** (array of `Nuclide` (page 3) objects) – radionuclides present in this release
- **Q** (array of `floats`) – per-nuclide initial activities in Bacquerels
- **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`(*loc*)

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

`unit_dose_at`(*loc*)

Same as `dose_at()` (page 9), but assume unit (1 Bacquerel) Puff activity

`_rad_decay`(*i*)

Return a coefficient that can be used to account for radioactive decay loses of the i-th radionuclide during current time_step

`_dry_depo`(*i*)

Return a coefficient that can be used to account for dry deposition loses of the i-th radionuclide during current time_step

`_unit_concentration_at`(*x*, *y*, *z*)

Same as `concentration_at()`, but assume unitary activity (1 Bacquerel). Does not account for radioactive decay or dry deposition.

`_reflection_at_mix_layer(z)`
Return vertical reflection at the top of the mixing layer.

Parameters z (*float*) – height where the reflection should be computed

`_reflection_at_ground(z)`
Return reflection from the ground, account for exponential loss

Parameters z (*float*) – height where the reflection should be computed

`_gauss_legendre_quadrature(loc, l, nuclide)`
Compute Gamma flux rate using Gauss-Legendre quadrature in 3D for unit concentration

Parameters

- **loc** (`Location` (page 3)) – location of the centre of the area where to compute
- **l** (*int*) – index of the nuclide into the self.inventory array
- **nuclide** (`Nuclide` (page 3)) – nuclide object at index l (Cython GIL work-around)

`_gamma_flux_rate_arg(r, th, phi, base, l, nuclide)`
Return integrand for the Gauss-Legendre quadrature, spheric coordinates, for unit concentration

Parameters

- **base** (`Location` (page 3)) – center of the spherical coord system
- **r** (*float*) – radius from loc
- **th** (*float*) – vertical angle from loc
- **phi** (*float*) – horizontal angle from loc
- **l** (*int*) – index of the nuclide into the self.inventory array
- **nuclide** (`Nuclide` (page 3)) – nuclide object at index l (Cython GIL work-around)

class `asim.dispmodel.puffmodel.PuffModel(time_step, puff_sampling_step, source_model)`
Bases: `asim.dispmodel.iface.DispersionModel` (page 4)

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

Parameters

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

`unit_puff_dose_at(`*loc*, *out=None*`)`
Similar to `dose_at()`, but return per-puff doses (i.e. a matrix rather than an array) and assume that each puff has unitary (1 Bacquerel) activity

Parameters out – optional matrix to write the results to must have shape

(puff count, nuclide count)

`puff_activities(`*out=None*`)`
Return activity of each puff in the puff model in Bacquerels

Parameters out – optional matrix to write the results to must have shape

(puff count, nuclide count)

`_create_kill_puffs(`*source_model*`)`
Kill puffs sentenced to death (those too far away or with activity below threshold) and eventually create a new one according to the source_model.

Chapter 5

Simulation of the Puff Model

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

class `asim.simulation.simple.StaticCompoLocSourceModel`

Bases: `asim.dispmodel.iface.SourceModel` (page 3)

Source model that provides static implementation of `inventory()` (page 3) and `location()` (page 3).

class `asim.simulation.simple.SimulatedSourceModel(`*puff_sampling_step*, *activities*`)`

Bases: `asim.simulation.simple.StaticCompoLocSourceModel` (page 13)

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 `asim.simulation.simple.PasquillsMeteoModel(`*stability_category*`)`

Bases: `asim.dispmodel.iface.MeteoModel` (page 3)

Abstract meteo model based on Pasquills stability category

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

class `asim.simulation.simple.StaticMeteoModel(`*meteo_array*, *stability_category*, *grid_step_xy=3*, *grid_step_time=3600*`)`

Bases: `asim.simulation.simple.PasquillsMeteoModel` (page 13)

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(`*q11*, *q12*, *q21*, *q22*, *x1*, *y1*, *x2*, *y2*, *x*, *y*`)`

Bilinear interpolation.

`asim.simulation.simple.read_receptor_locations(`*path*`)`

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 `asim.simulation.simple` (page 13) and `asim.dispmodel.puffmodel` (page 9)

class `asim.assimilation.twin.AssimilationSourceModel`(*puff_sampling_step*)

Bases: `asim.simulation.simple.StaticCompoLocSourceModel` (page 13)

Source model used for assimilation of released activity.

Parameters time_step (*int*) – inverval between rate changes

class `asim.assimilation.twin.AssimilationMeteoModel`(*stability_category*, *static_ws*, *static_wd*)

Bases: `asim.simulation.simple.PasquillsMeteoModel` (page 13)

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 `asim.assimilation.twin.ObservationCPdf`(*emp*, *receptors*, *meteo_model*, *params*)

Bases: `pybayes.pdfs.CPdf`

(wind speed, wind direction, dose) 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`(x, *cond=None*)

Evalate cpdf in point x given condition *cond*.

Parameters

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

class `asim.assimilation.twin.LogProposalCPdfFunc`(*alpha*, *receptor_count*)

Bases: `asim.support.newton.Function`

Derivative of the logarithm of the optimal proposal conditional probability density function, i.e: $log(p(Q|y_t))$

Used to find maxima of the proposal cpdf using Newton method.

class `asim.assimilation.twin.LogProposalCPdfDerivative(`*func*`)`
Bases: `asim.support.newton.Function`

Derivative of the logarithm of the optimal proposal conditional probability density function, i.e: $log(p(Q|y_t))$

Used to find maxima of the proposal cpdf using Newton method.

class `asim.assimilation.twin.ActivityProposalCPdf(`*emp*, *rv*, *cond_rv*`)`
Bases: `pybayes.pdfs.CPdf`

$p(Q_t|y_{1:t}, l_{1:t})p(l_t|l_{t-1}, a_t, b_t)$ where l_t is deterministic given a_t, b_t and l_{t-1} is sourced from the model. The actual conditioning variable is therefore only (a_t, b_t)

Chapter 7

Math Wrappers (Cythonization helpers)

Chapter 8

Graph Plotters

`asim.support.plotters.plot_trajectories(`*ax*, *TRAJECTORIES*, *alphas=None*, *colors=None*`)`
TRAJECTORIES = numpy.zeros((pC, tC, 2)) #0 = x, 1 = y

`asim.support.plotters.plot_stations(`*ax*, *locations*, *add_ center=True*`)`
Plot points representing stations to axis *ax*

Parameters locations – iterable of Location objects with station locations

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

Chapter 9

Indices and tables

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

Index

Symbols

A

C

D

E

G

H

I

L

M

N

O

P

R

S

U

W