iucm Documentation

Release 0.2.1

Philipp Sommer

Dec 06, 2018

Contents

docs	
package	

This model simulates urban growth and transformation with the objective of minimising the energy required for transportation. This user manual describes its technical implementation as the `iucm` python package.

Here we provide the principal steps on how to *install* and *use* the model, as well as a complete documentation of the *python API* and the *command line interface*.

The scientific background will be published in a separate journal article.

CHAPTER 1

Documentation

1.1 How to install

1.1.1 Installation using conda

We highly recommend to use conda for the installation of IUCM. Packages have been built for python 2.7 and 3.6 for windows, OSX and Linux.

Just download a miniconda installer, add the conda-forge channel to your configurations and install iucm from the chilipp channel:

```
conda config --add channels conda-forge
conda install -c chilipp iucm
```

1.1.2 Installation via pip

After having installed the necessary *Requirements*, install iucm from PyPi.org via:

```
$ pip install iucm
```

1.1.3 Installation from scratch

After having installed the necessary *Requirements*, clone the *Github repository* and install it via:

```
$ python setup.py install
```

1.1.4 Requirements

This package depends on

- python >= 2.7
- Cython
- numpy
- scipy
- xarray
- psyplot
- netCDF4
- funcargparse
- model-organization

To install all the necessary Packages in a conda environment *iucm*, type:

```
$ conda create -n iucm -c conda-forge cython psyplot netCDF4 scipy
$ conda activate iucm
$ pip install model-organization
```

1.2 Getting started

The iucm package uses the model-organization framework and thus can be used from the command line. The corresponding subclass of the `model_organization.ModelOrganizer` is the *`iucm.main.IUCMOrganizer`* class.

In this section, we provide a small starter example that transforms a fictitious city by moving 125‘000 inhabitants. Additional to the already mentioned *requirements*, this tutorial needs the psy-simple plugin and the pyshp package.

After *having installed the package* you can setup a new project with the *iucm setup* command via

```
In [1]: !iucm setup . -p my_first_project
INFO:iucm.my_first_project:Initializing project my_first_project
```

To create a new experiment inside the project, use the *iucm init* command:

```
In [2]: !iucm -id my_first_experiment init -p my_first_project
INFO:iucm.my_first_experiment:Initializing experiment my_first_experiment of project␣
↪my_first_project
```

Running the model, only requires a netCDF file with absolute population data. The x-coordinates and y-coordinates must be in metres.

1.2.1 Transforming a city

For the purpose of demonstration, we simply create a random input file with 2 city centers on a 25km x 25km grid at a resolution of 1km.

```
In [3]: import numpy as np
   ...: import xarray as xr
   ...: import matplotlib.pyplot as plt
   ...: import psyplot.project as psy
   ...: np.random.seed(1234)
```

(continues on next page)

(continued from previous page)

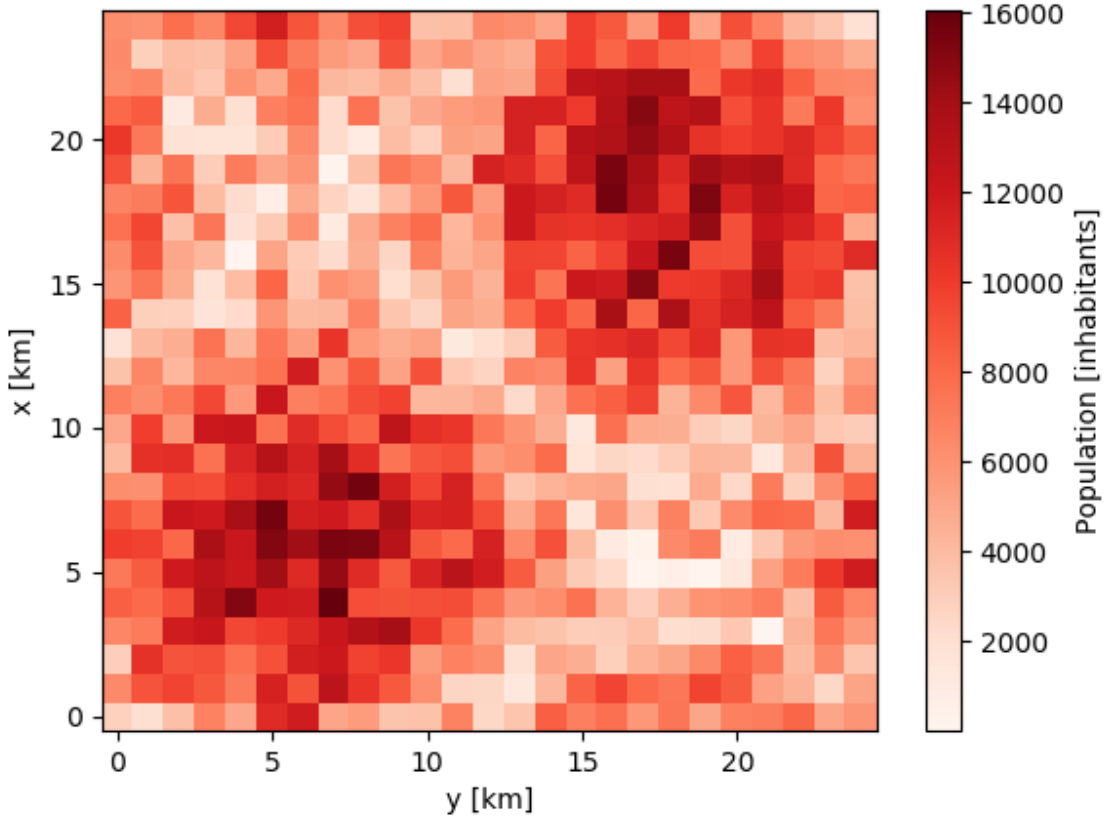
```
   ...:

In [4]: sine_vals = np.sin(np.linspace(0, 2 * np.pi, 25)) * 5000
   ...: x2d, y2d = np.meshgrid(sine_vals, sine_vals)
   ...: data = np.abs(x2d + y2d) + np.random.randint(0, 7000, (25, 25))
   ...:

In [5]: population = xr.DataArray(
   ...:     data,
   ...:     name='population',
   ...:     dims=('x', 'y'),
   ...:     coords={'x': xr.Variable(('x', ), np.arange(25, dtype=float),
   ...:                              attrs={'units': 'km'}),
   ...:             'y': xr.Variable(('y', ), np.arange(25, dtype=float),
   ...:                              attrs={'units': 'km'})},
   ...:     attrs={'units': 'inhabitants', 'long_name': 'Population'})
   ...:

In [6]: population.plot.pcolormesh(cmap='Reds');

In [7]: population.to_netcdf('input.nc')
```



Now we create a new scenario where we transform the city by moving stepwise 500 inhabitants around. For this, we need a forcing file which we can create using the *iucm preproc forcing* command:

```
In [8]: !iucm -v preproc forcing -steps 50 -trans 500
INFO:iucm.my_first_experiment:Creating forcing data...
DEBUG:iucm.my_first_experiment:Saving output to /home/docs/checkouts/readthedocs.org/
↪user_builds/iucm/checkouts/stable/docs/my_first_project/experiments/my_first_
↪experiment/input/forcing.nc
DEBUG:iucm.my_first_experiment:    development_file: None
DEBUG:iucm.my_first_experiment:    development_steps: [100]
DEBUG:iucm.my_first_experiment:    trans_size: 500.0
DEBUG:iucm.my_first_experiment:    total_steps: 100
DEBUG:iucm.my_first_experiment:    movement: 0
```

This now did create a new netCDF file with two variables

```
In [9]: xr.open_dataset(
   ...:     'my_first_project/experiments/my_first_experiment/input/forcing.nc')
   ...:
Out[9]:
<xarray.Dataset>
Dimensions:   (step: 100)
Coordinates:
  * step      (step) int64 1 2 3 4 5 6 7 8 9 10 ... 92 93 94 95 96 97 98 99 100
Data variables:
    change    (step) float64 ...
    movement  (step) int64 ...
```

that is also registered as forcing file in the experiment configuration

```
In [10]: !iucm info -nf
id: my_first_experiment
project: my_first_project
expdir: experiments/my_first_experiment
timestamps:
  init: '2018-12-06 07:27:44.900603'
  setup: '2018-12-06 07:27:42.966066'
  preproc: '2018-12-06 07:27:47.394419'
indir: experiments/my_first_experiment/input
preproc:
  forcing:
    development_file: null
    development_steps:
    - 100
    trans_size: 500.0
    total_steps: 100
    movement: 0
forcing: experiments/my_first_experiment/input/forcing.nc
```

The change variable in this forcing file describes the number of people that are moving within each step. In our case, this is just an alternating series of 500 and -500 since we take 500 inhabitants from one grid cell and move it to another.

Having prepared this input file, we can run our experiment with the *iucm run* command:

```
In [11]: !iucm -id my_first_experiment configure -s run -i input.nc -t 50 -max 15000
INFO:iucm.model:Saving to /home/docs/checkouts/readthedocs.org/user_builds/iucm/
↪checkouts/stable/docs/my_first_project/experiments/my_first_experiment/outdata/my_
↪first_experiment_1-50.nc...
```

The options here in detail:

-id my_first_experiment Tells iucm the experiment to use. The `-id` option is optional. If omitted, iucm uses the last created experiment.

configure -s This subcommand modifies the configuration to run our model in serial (see *iucm configure*)

run The *iucm run* command which tells iucm to run the experiment. The options here are

-t 50 Tells to model to make 50 steps

-max 15000 Tells the model that the maximum population is 15000 inhabitants per grid cell

The output now is a netCDF file with 50 steps:

```
In [12]: ds = xr.open_dataset(
   ....:     'my_first_project/experiments/my_first_experiment/'
   ....:     'outdata/my_first_experiment_1-50.nc')
   ....:

In [13]: ds
Out[13]:
<xarray.Dataset>
Dimensions:         (en_variables: 5, probabilistic: 1, step: 50, x: 25, y: 25)
Coordinates:
  * x               (x) float64 0.0 1.0 2.0 3.0 4.0 ... 20.0 21.0 22.0 23.0 24.0
  * y               (y) float64 0.0 1.0 2.0 3.0 4.0 ... 20.0 21.0 22.0 23.0 24.0
  * step            (step) int64 1 2 3 4 5 6 7 8 9 ... 42 43 44 45 46 47 48 49 50
  * en_variables    (en_variables) object 'k' 'dist' 'entrop' 'rss' 'own'
Dimensions without coordinates: probabilistic
Data variables:
    population      (step, x, y) float64 ...
    cons            (step) float64 ...
    dist            (step) float64 ...
    entrop          (step) float64 ...
    rss             (step) float64 ...
    cons_det        (step) float64 ...
    cons_std        (step) float64 ...
    left_over       (step) float64 ...
    nscenarios      (step) float64 ...
    cons_2d         (step, x, y) float64 ...
    dist_2d         (step, x, y) float64 ...
    entrop_2d       (step, x, y) float64 ...
    rss_2d          (step, x, y) float64 ...
    cons_det_2d     (step, x, y) float64 ...
    cons_std_2d     (step, x, y) float64 ...
    left_over_2d    (step, x, y) float64 ...
    nscenarios_2d   (step, x, y) float64 ...
    scenarios_2d    (step, x, y) float64 ...
    weights         (step, en_variables, probabilistic) float64 ...
```

With the output for the population, energy consumption and other variables. In the last step we also see, that the new population has mainly be added to the city centers in order to minimize the transportation energy:

```
In [14]: fig = plt.figure(figsize=(14, 6))
   ....: fig.subplots_adjust(hspace=0.5)
   ....:

# plot the energy consumption
In [15]: ds.cons.psy.plot.lineplot(
   ....:     ax=plt.subplot2grid((4, 2), (0, 0), 1, 2),
   ....:     ylabel='{desc}', xlabel='%(name)s');
```

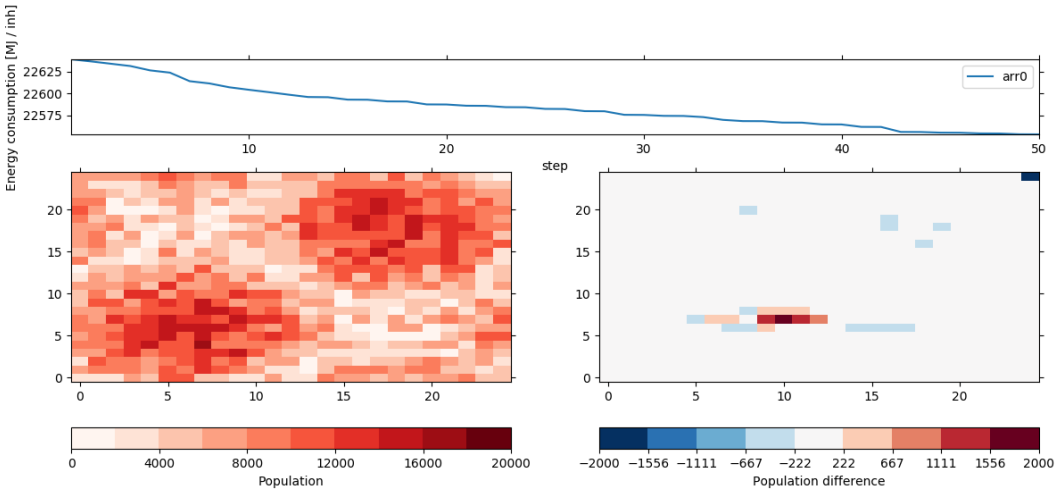
(continues on next page)

(continued from previous page)

```
   ....:

In [16]: ds.population[-1].psy.plot.plot2d(
   ....:     ax=plt.subplot2grid((4, 2), (1, 0), 3, 1),
   ....:     cmap='Reds', clabel='Population');
   ....:

In [17]: (ds.population[-1] - population).psy.plot.plot2d(
   ....:     ax=plt.subplot2grid((4, 2), (1, 1), 3, 1),
   ....:     bounds='roundedsym', cmap='RdBu_r',
   ....:     clabel='Population difference');
   ....:
```



As we can see, the model did move the population of sparse cells to locations where the population is higher, mainly to decrease the average distance between two individuals within the city.

The run settings are now stored in the configuration of the experiment, which can be seen via the *iucm info* command:

```
In [18]: !iucm info -nf
id: my_first_experiment
project: my_first_project
expdir: experiments/my_first_experiment
timestamps:
  init: '2018-12-06 07:27:44.900603'
  setup: '2018-12-06 07:27:42.966066'
  preproc: '2018-12-06 07:27:47.394419'
  run: '2018-12-06 07:28:00.576606'
  configure: '2018-12-06 07:27:51.294724'
indir: experiments/my_first_experiment/input
preproc:
  forcing:
    development_file: null
    development_steps:
    - 100
    trans_size: 500.0
    total_steps: 100
    movement: 0
forcing: experiments/my_first_experiment/input/forcing.nc
```

(continues on next page)

(continued from previous page)

```
run:
  ncells: 4
  selection_method: consecutive
  update_method: forced
  categories:
  - null
  - 15000.0
  probabilistic: 0
  max_pop: 15000.0
  coord_transform: 1.0
  steps: 50
  step_date:
    50: '2018-12-06 07:27:51.785721'
  vname: population
outdir: experiments/my_first_experiment/outdata
outdata:
- experiments/my_first_experiment/outdata/my_first_experiment_1-50.nc
input: experiments/my_first_experiment/input/input.nc
```

1.2.2 Probabilistic model

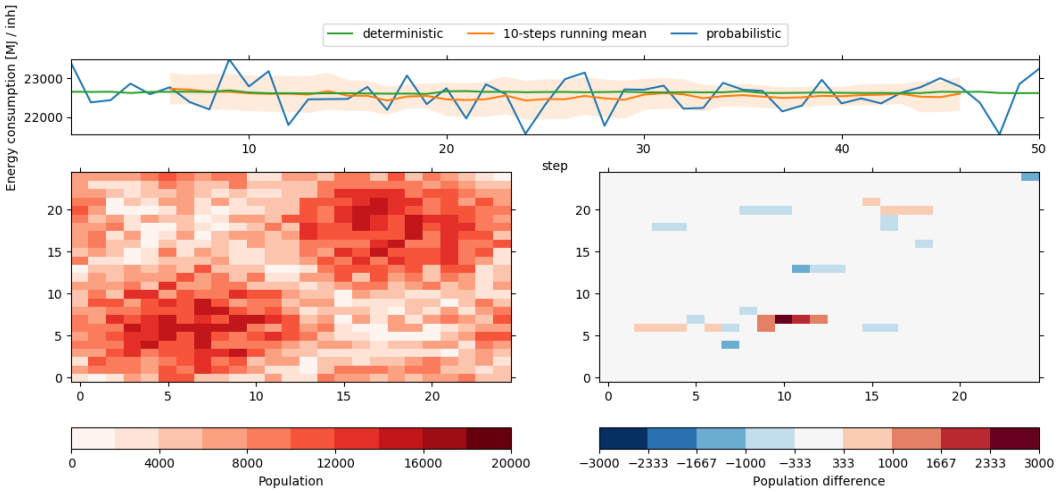
The default IUCM settings use a purely deterministic methodology based on the regression by [LeNechet2012]. However, to take the errors of this model into account, there exists a probabilistic version that is simply enabled via the `-prob` (or `--probabilistic`) argument, e.g. via

```
In [19]: !iucm run -nr -prob 1000 -t 50
INFO:iucm.model:Saving to /home/docs/checkouts/readthedocs.org/user_builds/iucm/
↪checkouts/stable/docs/my_first_project/experiments/my_first_experiment/outdata/my_
↪first_experiment_1-50.nc...
```

Instead of simply moving population from one cell to another, it distributes the population to multiple cells based on their probability to lower the energy consumption for the city.

```
In [20]: ds = xr.open_dataset(
   ....:     'my_first_project/experiments/my_first_experiment/'
   ....:     'outdata/my_first_experiment_1-50.nc')
   ....:

In [21]: plot_result()
```



As we can see, the results are not as smooth as the deterministic results, because now the energy consumption is based on a probabilistic set of regression weights (see *iucm.energy_consumption.random_weights()*). On the other hand, the deterministic energy consumption (stored as variable *cons_det* in the output file) corresponds pretty much to the deterministic version of our experiment setup above, as well as the running mean. And indeed, if we would drastically increase the number of probabilistic scenarios, we would approximate this energy consumption curve.

Note: The energy consumption in the output file is for the probabilistic setting determined by the mean energy consumption for all random scenarios. The *cons_det* variable on the other hand is always determined by the weights in [LeNechet2012] (see *iucm.energy_consumption.weights_LeNechet*)

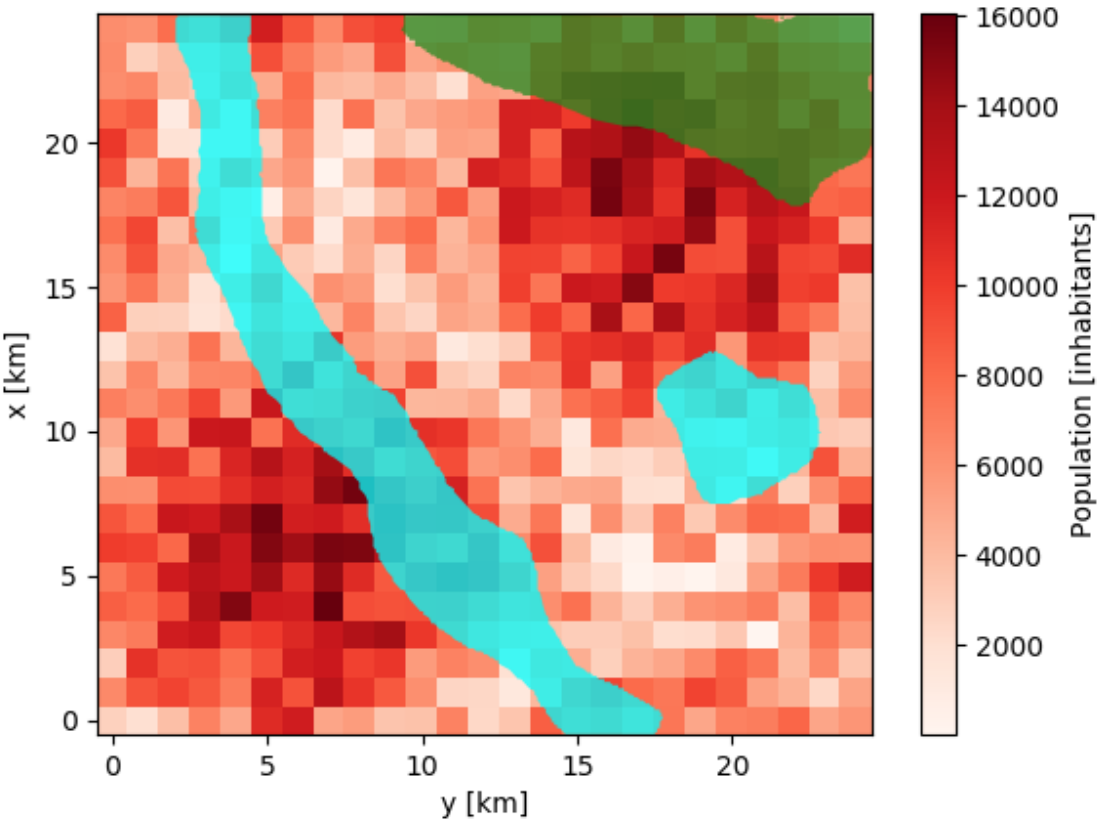
1.2.3 Masking areas

Each city has several areas that should not be filled with population, such as rivers, parks, etc. For example we assume a river, a lake and a forest in our city (see `the zipped shapefile`)

```
In [22]: population.plot.pcolormesh(cmap='Reds');

In [23]: from shapefile import Reader
   ....: reader = Reader('masking_shapes.shp')
   ....:

In [24]: from matplotlib.patches import Polygon
   ....: ax = plt.gca()
   ....: for shape_rec in reader.iterShapeRecords():
   ....:     color = 'forestgreen' if shape_rec.record[0] == 'Forest' else 'aqua'
   ....:     poly = Polygon(shape_rec.shape.points, facecolor=color, alpha=0.75)
   ....:     ax.add_patch(poly)
   ....:
```



IUCM has three options, how to handle these areas:

ignore The cells and the population that are touched by these shapes are completely ignored

mask The cells are masked for keeping their population constant

max-pop The maximum population in the cells that are touched by the shapes is lowered by the fraction that the shape cover in each cell

All three methods can easily be applied using the *iucm preproc mask* command.

Note: Using this feature requires pyshp to be installed and the shapefile must be defined on the same coordinate system as the input data!

Ignoring the areas

Ignoring the shapes will set the grid cells that are touched by the given shapefiles to `NaN`, i.e. *not a number*. Input cells that contain this value are completely ignored in the simulation. For our shapefile and input data here, the result would look like

```
In [25]: fig, axes = plt.subplots(1, 2)

In [26]: plotter = population.psy.plot.plot2d(
```

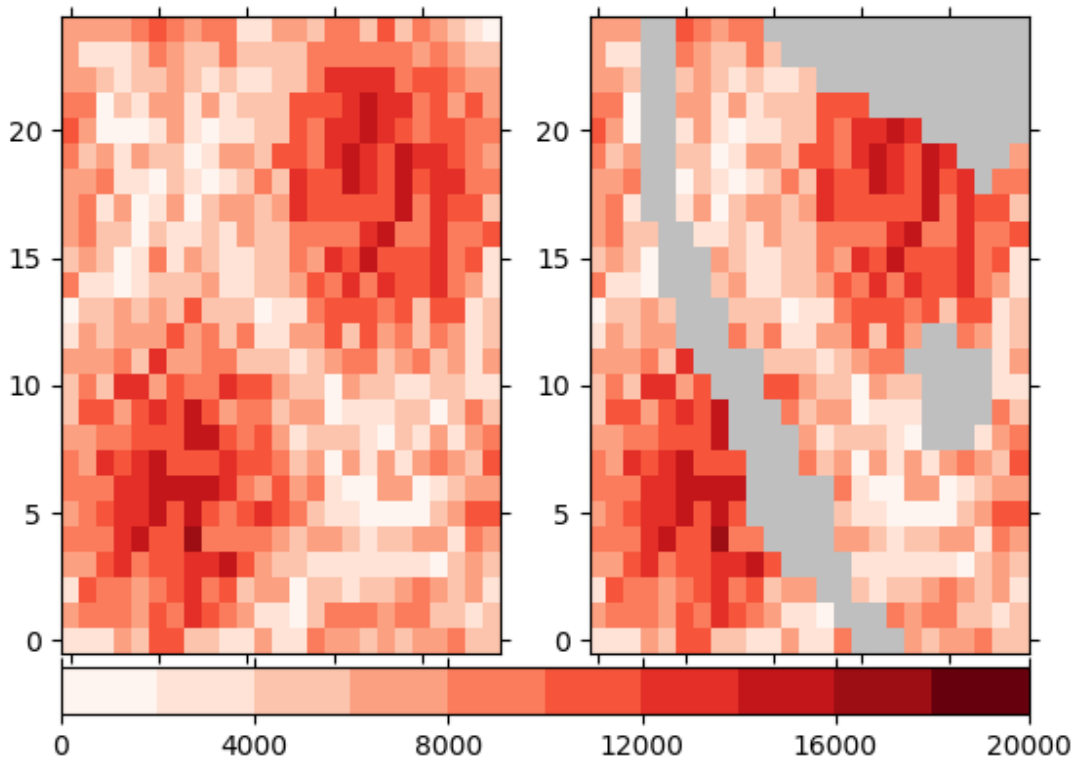
(continues on next page)

(continued from previous page)

```
   ....:     ax=axes[0], cmap='Reds', cbar='')
   ....:

In [27]: !iucm preproc mask masking_shapes.shp -m ignore

In [28]: sp = psy.plot.plot2d('input.nc', name='population', ax=axes[1],
   ....:                      cmap='Reds', cbar='fb', miss_color='0.75')
   ....: sp.share(plotter, keys='bounds')
   ....:
```

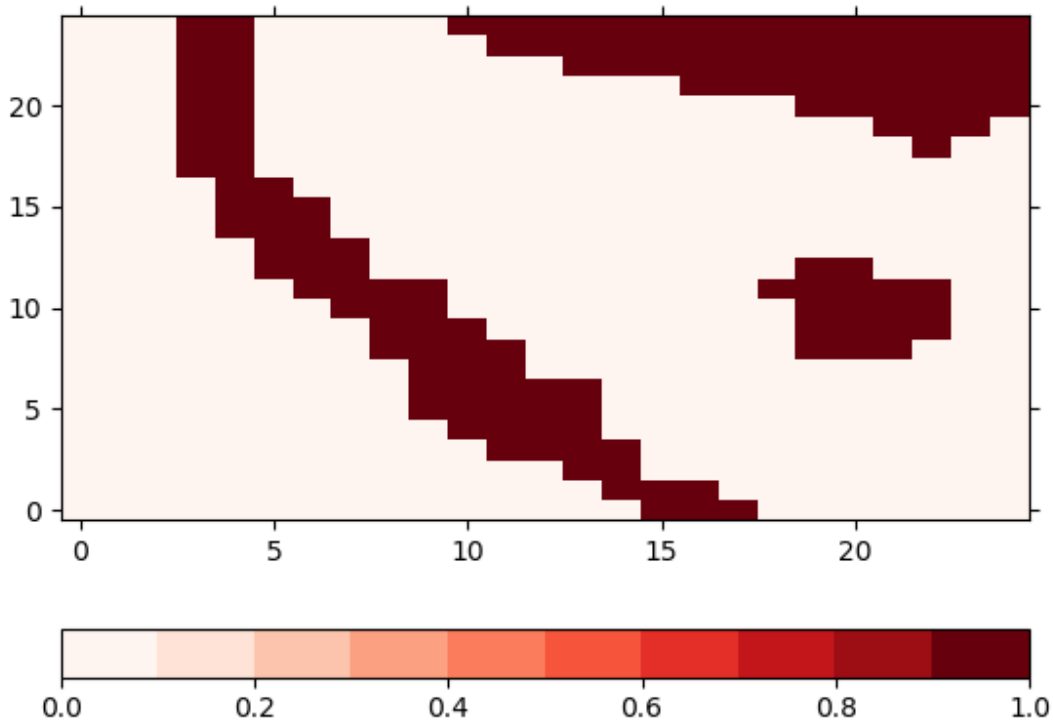


Masking the areas

Masking the areas means, that the population data in the grid cells that touch the given cells is not changed but it is considered in the calculation of the energy consumption. The input file for the model has a designated variable named *mask* for that. The population data for non-zero grid cells in this variable will be kept constant. In our case, the resulting *mask* variable in looks like this

```
In [29]: !iucm preproc mask masking_shapes.shp -m mask

In [30]: sp = psy.plot.plot2d('input.nc', name='mask', cmap='Reds')
```

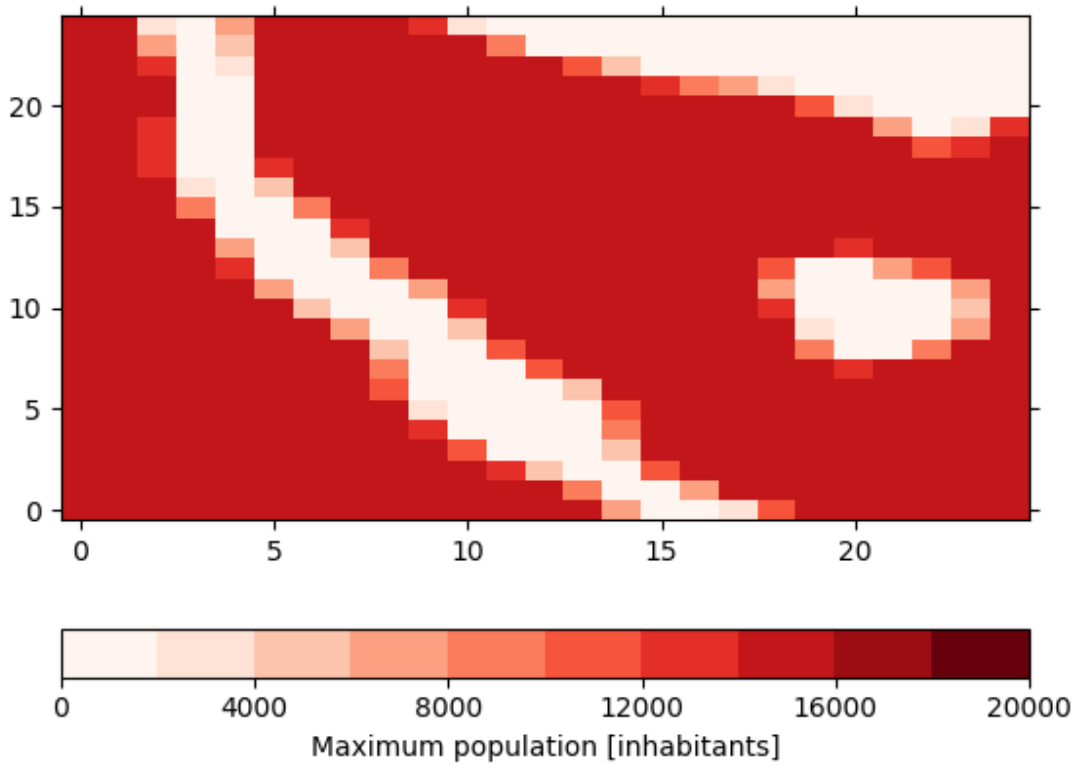


Adjusting the maximum population

This is the default method and is the best method represent the shape files in the model. Instead of masking the data, we lower the amount of the maximum possible population in the grid cells. For this, the shapefile is rasterized at high resolution (by default 100-times the resolution of the input file) and the we calculate the percentage for each coarse grid cell that is covered by the shape. The result will then be stored in the *max_pop* variable in the input dataset which defines the maximum population for each grid cell. In our case, this variable looks like

```
In [31]: !iucm preproc mask masking_shapes.shp

In [32]: sp = psy.plot.plot2d('input.nc', name='max_pop', cmap='Reds',
   ....:                      clabel='{desc}')
   ....:
```



Note: This method is a pure python implementation that does not have any other dependencies than matplotlib and pyshp. Due to this, it might be slow for large shapefiles or large input files. In this case, we recommend to use gdal_rasterize for creating the high resolution rastered shape file and gdalwarp for interpolating it to the input grid. In our case here, this would look like

```
gdal_rasterize -burn 1.0 -tr 0.01 0.01 masking_shapes.shp hr_rastered_shapes.tif
gdalwarp -tr 1.0 1.0 -r average hr_rastered_shapes.tif covered_fraction.tif
gdal_calc.py -A covered_fraction.tif --outfile=max_pop.nc --format=netCDF --calc="(1-
↪A)*15000"
```

And then merge the file `'max_pop.nc'` into `'input.nc'` as variable `'max_pop'`.

1.3 Command Line API Reference

1.3.1 iucm setup

Perform the initial setup for the project

```
usage: iucm setup [-h] [-p str] [-link] str
```

Positional Arguments

str The path to the root directory where the experiments, etc. will be stored

Named Arguments

-p, --projectname The name of the project that shall be initialized at *root_dir*. A new directory will be created namely `root_dir + '/' + projectname`

-link If set, the source files are linked to the original ones instead of copied

Default: False

1.3.2 iucm init

Initialize a new experiment

```
usage: iucm init [-h] [-p str] [-d str]
```

Named Arguments

-p, --projectname The name of the project that shall be used. If None, the last one created will be used

-d, --description A short summary of the experiment

Notes

If the experiment is None, a new experiment will be created

Notes

If the experiment is None, a new experiment will be created

1.3.3 iucm set-value

Set a value in the configuration

```
usage: iucm set-value [-h] [-a] [-P] [-g] [-p str] [-b str] [-dt str]
                      level0.level1.level...=value
                      [level0.level1.level...=value ...]
```

Positional Arguments

level0.level1.level...=value The key-value pairs to set. If the configuration goes some levels deeper, keys may be separated by a `'.'` (e.g. `'namelists.weathergen'`). Hence, to insert a `'.'`, it must be escaped by a preceeding `''`.

Named Arguments

-a, --all If True/set, the information on all experiments are printed

Default: False

-P, --on-projects If set, show information on the projects rather than the experiment

Default: False

-g, --globally If set, show the global configuration settings

Default: False

-p, --projectname The name of the project that shall be used. If provided and *on_projects* is not True, the information on all experiments for this project will be shown

-b, --base A base string that shall be put in front of each key in *values* to avoid typing it all the time

Default: “”

-dt, --dtype Possible choices: ArithmeticError, AssertionError, AttributeError, BaseException, BlockingIOError, BrokenPipeError, BufferError, BytesWarning, ChildProcessError, ConnectionAbortedError, ConnectionError, ConnectionRefusedError, ConnectionResetError, DeprecationWarning, EOFError, Ellipsis, EnvironmentError, Exception, False, FileExistsError, FileNotFoundError, FloatingPointError, FutureWarning, GeneratorExit, IOError, ImportError, ImportWarning, IndentationError, IndexError, InterruptedError, IsADirectoryError, KeyError, KeyboardInterrupt, LookupError, MemoryError, ModuleNotFoundError, NameError, None, NotADirectoryError, NotImplemented, NotImplementedError, OSError, OverflowError, PendingDeprecationWarning, PermissionError, ProcessLookupError, RecursionError, ReferenceError, ResourceWarning, RuntimeError, RuntimeWarning, StopAsyncIteration, StopIteration, SyntaxError, SyntaxWarning, SystemError, SystemExit, TabError, TimeoutError, True, TypeError, UnboundLocalError, UnicodeDecodeError, UnicodeEncodeError, UnicodeError, UnicodeTranslateError, UnicodeWarning, UserWarning, ValueError, Warning, ZeroDivisionError, __build_class__, __debug__, __doc__, __import__, __loader__, __name__, __package__, __spec__, abs, all, any, ascii, bin, bool, breakpoint, bytearray, bytes, callable, chr, classmethod, compile, complex, copyright, credits, delattr, dict, dir, divmod, enumerate, eval, exec, exit, filter, float, format, frozenset, getattr, globals, hasattr, hash, help, hex, id, input, int, isinstance, issubclass, iter, len, license, list, locals, map, max, memoryview, min, next, object, oct, open, ord, pow, print, property, quit, range, repr, reversed, round, set, setattr, slice, sorted, staticmethod, str, sum, super, tuple, type, vars, zip

The name of the data type or a data type to cast the value to

1.3.4 iucm get-value

Get one or more values in the configuration

```
usage: iucm get-value [-h] [-ep] [-pp] [-a] [-P] [-g] [-p str] [-nf] [-k]
                      [-b str] [-arc]
                      level0.level1.level... [level0.level1.level... ...]
```

Positional Arguments

level0.level1.level... A list of keys to get the values of. If the key goes some levels deeper, keys may be separated by a `'.'` (e.g. `'namelists.weathergen'`). Hence, to insert a `','`, it must be escaped by a preceeding `''`.

Named Arguments

-ep, --exp-path If True/set, print the filename of the experiment configuration

Default: False

-pp, --project-path If True/set, print the filename on the project configuration

Default: False

-a, --all If True/set, the information on all experiments are printed

Default: False

-P, --on-projects If set, show information on the projects rather than the experiment

Default: False

-g, --globally If set, show the global configuration settings

Default: False

-p, --projectname The name of the project that shall be used. If provided and *on_projects* is not True, the information on all experiments for this project will be shown

-nf, --no-fix If set, paths are given relative to the root directory of the project

Default: False

-k, --only-keys If True, only the keys of the given dictionary are printed

Default: False

-b, --base A base string that shall be put in front of each key in *values* to avoid typing it all the time

Default: “”

-arc, --archives If True, print the archives and the corresponding experiments for the specified project

Default: False

1.3.5 iucm del-value

Delete a value in the configuration

```
usage: iucm del-value [-h] [-a] [-P] [-g] [-p str] [-b str] [-dtype DTYPE]
                      level0.level1.level... [level0.level1.level... ...]
```

Positional Arguments

level0.level1.level... A list of keys to be deleted. If the key goes some levels deeper, keys may be separated by a `'.'` (e.g. `'namelists.weathergen'`). Hence, to insert a `','`, it must be escaped by a preceeding `''`.

Named Arguments

-a, --all If True/set, the information on all experiments are printed

Default: False

-P, --on-projects If set, show information on the projects rather than the experiment

Default: False

-g, --globally If set, show the global configuration settings

Default: False

-p, --projectname The name of the project that shall be used. If provided and *on_projects* is not True, the information on all experiments for this project will be shown

-b, --base A base string that shall be put in front of each key in *values* to avoid typing it all the time

Default: “”

-dtype

1.3.6 iucm info

Print information on the experiments

```
usage: iucm info [-h] [-ep] [-pp] [-gp] [-cp] [-a] [-nf] [-P] [-g] [-p str]
                 [-k] [-arc]
```

Named Arguments

-ep, --exp-path If True/set, print the filename of the experiment configuration

Default: False

-pp, --project-path If True/set, print the filename on the project configuration

Default: False

-gp, --global-path If True/set, print the filename on the global configuration

Default: False

-cp, --config-path If True/set, print the path to the configuration directory

Default: False

-a, --all If True/set, the information on all experiments are printed

Default: False

-nf, --no-fix If set, paths are given relative to the root directory of the project

Default: False

-P, --on-projects If set, show information on the projects rather than the experiment

Default: False

-g, --globally If set, show the global configuration settings

Default: False

-p, --projectname The name of the project that shall be used. If provided and *on_projects* is not True, the information on all experiments for this project will be shown

-k, --only-keys If True, only the keys of the given dictionary are printed

Default: False

-arc, --archives If True, print the archives and the corresponding experiments for the specified project

Default: False

1.3.7 iucm unarchive

Extract archived experiments

```
usage: iucm unarchive [-h] [-ids exp1,[exp2[,...]]] [exp1,[exp2[,...]]] ...]]
                      [-f str] [-a] [-pd] [-P] [-d str] [-p str]
                      [-fmt { 'gztar' | 'bztar' | 'tar' | 'zip' }] [--force]
```

Named Arguments

-ids, --experiments The experiments to extract. If None the current experiment is used

-f, --file The path to an archive to extract the experiments from. If None, we assume that the path to the archive has been stored in the configuration when using the `archive()` command

-a, --all If True, archives are extracted completely, not only the experiment (implies `project_data = True`)

Default: False

-pd, --project-data If True, the data for the project is extracted as well

Default: False

-P, --replace-project-config If True and the project does already exist in the configuration, it is updated with what is stored in the archive

Default: False

-d, --root An alternative root directory to use. Otherwise the experiment will be exctracted to

1. the root directory specified in the configuration files (if the project exists in it) and *replace_project_config* is False
2. the root directory as stored in the archive

-p, --projectname The projectname to use. If None, use the one specified in the archive

-fmt The format of the archive. If None, it is inferred

--force If True, force to overwrite the configuration of all experiments from what is stored in *archive*. Otherwise, the configuration of the experiments in *archive* are only used if missing in the current configuration

Default: False

1.3.8 iucm configure

Configure the project and experiments

```
usage: iucm configure [-h] [-g] [-p] [-i str] [-f str] [-s] [-n int or 'all']
                      [-update-from str]
```

Named Arguments

-g, --globally If True/set, the configuration are applied globally (already existing and configured experiments are not impacted)

Default: False

-p, --project Apply the configuration on the entire project instance instead of only the single experiment (already existing and configured experiments are not impacted)

Default: False

-i, --ifile The input file for the project. Must be a netCDF file with population data

-f, --forcing The input file for the project containing variables with population evolution information. Possible variables in the netCDF file are *movement* containing the number of people to move and *change* containing the population change (positive or negative)

-s, --serial Do the parameterization always serial (i.e. not in parallel on multiple processors). Does automatically impact global settings

Default: False

-n, --nprocs Maximum number of processes to when making the parameterization in parallel. Does automatically impact global settings and disables *serial*

-update-from Path to a yaml configuration file to update the specified configuration with it

1.3.9 iucm preproc

iucm preproc forcing

Create a forcing file from a predescribed population path

```
usage: iucm preproc forcing [-h] [-df str] [-o str]
                            [-t col1,col2,col3,... [col1,col2,col3,... ...]]
                            [-steps int] [-m float] [-trans float] [-p str]
                            [-nd]
```

Named Arguments

-df, --development-file The path to a csv file containing (at least) one column with the projected population development

-o, --output The name of the ouput forcing netCDF file. By default: `'<expdir>/input/forcing.nc'`

-t, --date-cols The names of the date columns in the *development_file* that shall be used to generate the date-time information. If not given, the date will simply be a range from 1 to *steps* times the length of the projected population development from *development_file*

-steps The numbers of model steps between on step of the projected development in *development_file*

Default: 1

-m, --movement The people moving randomly during on model step

Default: 0

-trans, --trans-size The forced size of the transformation additionally to the development from *development_file*

Default: 0

-p, --population-col The name of the column with population data. If not given, the last one is used

-nd, --no-date-parsing If True, then *date_cols* is simply interpreted as an index and no date-time information is parsed

Default: False

iucm preproc mask

Mask grid cells based on a shape file

This command calculates the maximum population for the model based on a masking shape file. The given shape file is rasterized at high resolution (by default, 100 times the resolution of the input file) and the fraction for each grid cell that is covered by that shape file is calculated.

```
usage: iucm preproc mask [-h] [-m {'max-pop', 'mask', 'ignore'}] [-i str]
                         [-v str] [-overwrite] [-max float] [-r int]
                         str
```

Positional Arguments

str The path to a shapefile

Named Arguments

-m, --method Default: “max-pop”. Determines how to handle the given shapes.

max-pop The maximum population per grid cell is lowered by the fraction of the cell that is covered by the given shape. This will adjust the *max_pop* variable in the input file *ifile*

mask The population of the grid cells in the input data that are touched by the given shapes will be kept constant during the simulation. This will adjust the *mask* variable in the input file *ifile*

ignore The grid cells in the input data that are touched by the given grid cells are put to NaN and their population is not considered during the simulation. This will adjust the input population data (i.e. variable *vname*) directly

Default: “max-pop”

-i, --ifile The path of the input file. If not specified, the value of the configuration is used

-v, --vname The variable name to use. If not specified and only one variable exists in the dataset, this one is used. Otherwise, the `'run.vname'` key in the experiment configuration is used

-overwrite If True and the target variable exists already in the input file *ifile* (and method is not ‘ignore’), this variable is overwritten

Default: False

-max, --max-pop The maximum population. If not specified, the value in the configuration is used (only necessary if `method=='max-pop'`)

-r, --hr-res The resolution of the high resolution file, relative to the resolution of *ifile* (only necessary if `method=='max-pop'`)

Default: 100

Notes

Note that the shapefile and the input file have to be defined on the same coordinate system! This function is not super efficient, for large data files we recommend using gdal_rasterize and gdalwarp.

Preprocess the data

```
usage: iucm preproc [-h] {forcing,mask} ...
```

1.3.10 iucm run

Run the model for the given experiment

```
usage: iucm run [-h] [-i str] [-f str] [-v str] [-t int]
                [-sm { 'consecutive' | 'random' }]
                [-um { 'categorical' | 'random' | 'forced' }] [-n int]
                [-c float1,float2,...] [-pctls] [-nr] [-ot int] [-seed int]
                [-stop-en-change float] [-agg-stop-steps int] [-agg-steps int]
                [-prob int] [-max float] [-ct float] [-cp str]
```

Named Arguments

-i, --ifile The input file. If not specified, the *input* key in the experiment configuration is used

-f, --forcing The forcing file (necessary if `update_method=='forced'`). If not specified, the *forcing* key in the experiment configuration is used

-v, --vname The variable name to use. If not specified and only one variable exists in the dataset, this one is used. Otherwise, the `'run.vname'` key in the experiment configuration is used

-t, --steps The number of time steps

Default: 50

-sm, --selection-method The name of the method on how the data is selected. The default is consecutive. Possible selection methods are

consecutive: Always the *ncells* consecutive cells are selected.

random: *ncells* random cells in the field are updated.

-um, --update-method The name of the update method on how the selected cells (see *selection_method* are updated). The default is categorical. Possible update methods are

categorical: The selected cells are updated to the lower level of the next category.

random: The selected cells are updated to a randum number within the next category.

forced: A forcing file is used (see the *forcing* parameter).

-n, --ncells The number of cells that shall be changed during 1 step. The default value is 4

-c, --categories The values for the categories to use within the models

-pctls, --use-pctls If True, interprete *categories* as percentiles instead of real population density

Default: False

-nr, --no-restart If True, and the run has already been conducted, restart it. Otherwise the previous run is continued

Default: False

-ot, --output-step Make an output every *output_step*. If None, only the final result is written to the output file

-seed The random seed for numpy to use. Specify this parameter for the experiment to guarantee reproducability

-stop-en-change The minimum of required relative energy consumption change. If the mean relative energy consumption change over the last *agg_stop_steps* steps is below this number, the run is stopped

-agg-stop-steps The number of steps to aggregate over when calculating the mean relative energy consumption change. Does not have an effect if *stop_en_change* is None

Default: 100

-agg-steps Use only every *agg_steps* energy consumption for the aggregation when checking the *stop_en_change* criteria

Default: 1

-prob, --probabilistic The number of probabilistic scenarios. For each scenario the energy consumption is calculated and the final population is distributed to the cells with the ideal energy consumption. Set this to 0 to only use the weights by [LeNechet2012]. If this option is None, the value will be taken from the configuration with a default of 0 (i.e. no probabilistic run).

-max, --max-pop The maximum population for each cell. If None, the last value in *categories* will be used or what is stored in the experiment configuration

-ct, --coord-transform The transformation factor to transform the coordinate values into kilometres

-cp, --copy-from If not None, copy the run settings from the other given experiment

1.3.11 iucm postproc

iucm postproc rolling

Calculate the rolling mean for the energy consumption

This postprocessing function calculates the rolling mean for the energy consumption

```
usage: iucm postproc rolling [-h] [-w int] [-o str] [-od str]
```

Named Arguments

-w, --window	Size of the moving window. This is the number of observations used for calculating the statistic. Each window will be a fixed size. If None, it will be taken from the experiment configuration with a default value of 50.
-o, --output	A filename where to save the output. If not given, it is not saved but may be later used by the `evolution()` method
-od, --odir	The name of the output directory

iucm postproc map

Make a movie of the post processing

```
usage: iucm postproc map [-h] [-o str] [-od str] [-t int] [-diff] [-t0 int]
                         [-p str] [-save str] [-simple]
```

Named Arguments

-o, --output	The name of the output file
	Default: "map.pdf"
-od, --odir	The name of the output directory
-t, --time	The timestep to plot. By default, the last timestep is used
	Default: -1
-diff	If True/set, visualize the difference to the *t0* (by default, the first step) is used, instead of the entire data
	Default: False
-t0	If *diff* is set, the reference step for the difference
	Default: 0
-p, --project-file	The path to a filename containing a file that can be loaded via the `psyplot.project.Project.load_project()` method
-save, --save-project	The path to a filename where to save the psyplot project
-simple, --simple-plot	If True/set, use a non-georeferenced plot. Otherwise, we use the cartopy module to plot it
	Default: False

iucm postproc movie

Make a movie of the post processing

```
usage: iucm postproc movie [-h] [-fps FPS] [-dpi DPI] [-o str] [-od str]
                           [-diff] [-t0 int] [-p str] [-save str] [-simple]
                           [-t t1[;t2[;t31,t32,[t33]]]
                           [t1[;t2[;t31,t32,[t33]]] ...]]
```

Named Arguments

-fps The number of frames per second. Default: 10

-dpi The dots per inch

-o, --output The name of the output file

Default: "movie.gif"

-od, --odir The name of the output directory

-diff If True/set, visualize the difference to the *t0* (by default, the first step) is used, instead of the entire data

Default: False

-t0 If *diff* is set, the reference step for the difference

-p, --project-file The path to a filename containing a file that can be loaded via the `psyplot.project.Project.load_project()` method

-save, --save-project The path to a filename where to save the psyplot project

-simple, --simple-plot If True/set, use a non-georeferenced plot. Otherwise, we use the cartopy module to plot it

Default: False

-t, --time **The time steps to use.** A semicolon (`';'`) separated string. A single value in this string represents one number, ranges can also be used via a separation by comma (`','`). Hence, `'2009;2012,2015'` will be converted to `[2009,2012, 2013, 2014]` and `2009;2012,2015,2` to `[2009, 2012, 2015]`

iucm postproc evolution

Plot the evolution of DIST, RSS, ENTROP and Energy consumption

```
usage: iucm postproc evolution [-h] [-o str] [-od str]
                               [-t t1[;t2[;t31,t32,[t33]]]
                               [t1[;t2[;t31,t32,[t33]]] ...]] [-roll]
```

Named Arguments

-o, --output The name of the output file

Default: "plots.pdf"

-od, --odir The name of the output directory

-t, --time **The time steps to use.** A semicolon (`';'`) separated string. A single value in this string represents one number, ranges can also be used via a separation by comma (`','`). Hence, `'2009;2012,2015'` will be converted to `[2009,2012, 2013, 2014]` and `2009;2012,2015,2` to `[2009, 2012, 2015]`

-roll, --use-rolling If True, use the rolling mean for the energy consumption

Default: False

Postprocess the data

```
usage: iucm postproc [-h] [-ni] {rolling,map,movie,evolution} ...
```

Named Arguments

-ni, --no-input If True/set, the initial input file is ignored

Default: False

1.3.12 iucm archive

Archive one or more experiments or a project instance

This method may be used to archive experiments in order to minimize the amount of necessary configuration files

```
usage: iucm archive [-h] [-d str] [-f str]
                    [-fmt { 'gztar' | 'bztar' | 'tar' | 'zip' }] [-p str]
                    [-ids exp1,[exp2[,...]]]] [-P] [-na] [-np]
                    [-e str [str ...]] [-k] [-rm] [-n] [-L]
```

Named Arguments

-d, --odir The path where to store the archive

-f, --aname The name of the archive (minus any format-specific extension). If None, defaults to the projectname

-fmt Possible choices: bztar, gztar, tar, zip

The format of the archive. If None, it is tested whether an archived with the name specified by *aname* already exists and if yes, the format is inferred, otherwise `'tar'` is used

-p, --projectname If provided, the entire project is archived

-ids, --experiments If provided, the given experiments are archived. Note that an error is raised if they belong to multiple project instances

-P, --current-project If True, *projectname* is set to the current project

Default: False

-na, --no-append It True and the archive already exists, it is deleted

Default: False

-np, --no-project-paths If True, paths outside the experiment directories are neglected

Default: False

-e, --exclude Filename patterns to ignore (see `glob.fnmatch.fnmatch()`)

-k, --keep If True, the experiment directories are not removed and no modification is made in the configuration

Default: False

-rm, --rm-project If True, remove all the project files

Default: False

-n, --dry-run If True, set, do not actually make anything

Default: False

-L, --dereference If set, dereference symbolic links. Note: This is automatically set for `fmt=='zip'`

Default: False

1.3.13 iucm remove

Delete an existing experiment and/or projectname

```
usage: iucm remove [-h] [-p [str]] [-a] [-y] [-ap]
```

Named Arguments

-p, --projectname The name for which the data shall be removed. If set without, argument, the project will be determined by the experiment. If specified, all experiments for the given project will be removed.

-a, --all If set, delete not only the experiments and config files, but also all the project files

Default: False

-y, --yes If True/set, do not ask for confirmation

Default: False

-ap, --all-projects If True/set, all projects are removed

Default: False

The main function for parsing global arguments

```
usage: iucm [-h] [-id str] [-l] [-n] [-v] [-vl str or int] [-nm] [-E]
            {setup,init,set-value,get-value,del-value,info,unarchive,configure,
↪preproc,run,postproc,archive,remove}
            ...
```

1.3.14 Named Arguments

-id, --experiment **experiment: str** The id of the experiment to use. If the *init* argument is called, the *new* argument is automatically set. Otherwise, if not specified differently, the last created experiment is used.

-l, --last	If True, the last experiment is used Default: False
-n, --new	If True, a new experiment is created Default: False
-v, --verbose	Increase the verbosity level to DEBUG. See also *verbosity_level* for a more specific determination of the verbosity Default: False
-vl, --verbosity-level	The verbosity level to use. Either one of `'DEBUG'`, `'INFO'`, `'WARNING'`, `'ERROR'` or the corresponding integer (see pythons logging module)
-nm, --no-modification	If True/set, no modifications in the configuration files will be done Default: False
-E, --match	If True/set, interprete *experiment* as a regular expression (regex) und use the matching experiment Default: False

1.4 Python API Reference

IUCM - The Integrated Urban Complexity Model

A model to increase the population in an efficient way with respect to transportation energy

1.4.1 Submodules

iucm.dist module

Module for calculating the average distance between two individuals

Functions

dist(population, x, y[, dist0, indices, ...])	Calculate the average distance between two individuals

iucm.dist.**dist**(*population*, *x*, *y*, *dist0=-1*, *indices=None*, *increase=None*)

Calculate the average distance between two individuals

This function calculates

$$d = \frac{\sum_{i,j=1}^{N} d_{ij} P_i P_j}{P_{tot}(P_{tot} - 1)}$$

after [LeNechet2012], where P_i is the population of the i-th grid cell, P_{tot} is the total population and d_{ij} is the distance between the i-th and j-th grid cell

Parameters

- **population** (*1D np.ndarray of dtype float*) – The population data for each grid cell. If `dist0>=0` then this array must hold the population of the previous step corresponding to *dist0*. Otherwise it should be the real population.
- **x** (*1D np.ndarray of dtype float*) – The x-coordinates for each element in *population*

- **y** (*1D np.ndarray of dtype float*) – The y-coordinates for each element in *population*
- **dist0** (*float, optional*) – The previous average distance between individuals. If this is given, *increase* and *indices* must not be None and *population* must represent the population of the previous step. Specifying this value, significantly speeds up the calculation.
- **indices** (*1D np.ndarray of dtype int*) – The indices of the grid cells, where the population has been increased.
- **increase** (*1D np.ndarray of dtype float*) – The increase in the grid cells corresponding to the given *indices*

Returns The average distance between two individuals.

Return type float

iucm.energy_consumption module

Energy consumption module of the iucm package

This module, together with the *dist_numerator* package contains the necessary functions for computing the energy consumption of a city. The main API function is the *energy_consumption()* function. Note that the *OWN* value is set to the value for Stuttgart, Germany. You should change it if you want to model another city. The parameters of this module come from [LeNechet2012].

References

Classes

EnVariables(k, dist, entrop, rss, own)	A tuple containing the values that setup the energy consumption

Data

K	Intercept of energy consumption
OWN	number of cars per 100 people. The default is 0, i.e. the value is ignored.
std_err_LeNechet	The standard errors of the weights used in [LeNechet2012] (obtained through private communication via R.
wDIST	weight for average distance of two individuals on energy consumption
wENTROP	weight for entropy on energy consumption
wOWN	weight for car owner ship on energy consumption
wRSS	weight for rank-size rule slope on energy consumption
weights_LeNechet	The weights for the variables to calculate the energy_consumption from

Functions

energy_consumption(population, x, y[, ...])	Compute the energy consumption of a city
entrop(population, size)	Compute the entropy of a city

Continued on next page

Table 4 – continued from previous page

random_weights(weights)	Draw random weights
rss(population)	Compute the Rank-Size slope coefficient

class iucm.energy_consumption.**EnVariables**(*k*, *dist*, *entrop*, *rss*, *own*)

Bases: tuple

A tuple containing the values that setup the energy consumption

Parameters

- **k** (*float*) – Intercept of energy consumption
- **dist** (*float*) – Average distance between two individuals
- **entrop** (*float*) – Entropy
- **rss** (*float*) – Rank-Size Slope
- **own** (*float*) – weight for car owner ship on energy consumption

Attributes

dist	Alias for field number 1
entrop	Alias for field number 2
k	Alias for field number 0
own	Alias for field number 4
rss	Alias for field number 3

See also:

iucm.model.Output, *iucm.model.Output2D*, *iucm.model.PopulationModel.state*, *iucm.model.PopulationModel.allocate_output*

Create new instance of EnVariables(k, dist, entrop, rss, own)

dist
Alias for field number 1

entrop
Alias for field number 2

k
Alias for field number 0

own
Alias for field number 4

rss
Alias for field number 3

iucm.energy_consumption.**K = -346.5**
Intercept of energy consumption

iucm.energy_consumption.**OWN = 0**
number of cars per 100 people. The default is 0, i.e. the value is ignored. Another possible value that has been previously used might be 37.7, the value for Stuttgart

iucm.energy_consumption.**energy_consumption**(*population*, *x*, *y*, *dist0=-1*, *slicer=None*, *indices=None*, *increase=None*, *weights=EnVariables(k=-346.5*, *dist=279.0*, *entrop=21700.0*, *rss=-9343.0*, *own=17.36)*)

Compute the energy consumption of a city

Compute the energy consumption according to [LeNechet2012] via

$$E = -346 + 17.4 \cdot OWN + 279 \cdot DIST - 9340 \cdot RSS + 21700 \cdot ENTROP$$

Parameters

- **population** (*1D np.ndarray*) – The 1D population data
- **x** (*1D np.ndarray*) – The x coordinates information for each cell in *population* in kilometers
- **y** (*1D np.ndarray*) – The y coordinates information for each cell in *population* in kilometers
- **dist0** (*float, optional*) – The previous average distance between two individuals (see *iucm.dist.dist()* function). Speeds up the computation significantly
- **slicer** (slice or boolean array, optional) – The slicer that can be use to access the changed cells specified by *increase*
- **indices** (*1D np.ndarray of dtype int, optional*) – The indices corresponding to the increase in *increase*
- **increase** (*1D np.ndarray, optional*) – The changed population which will be added on *population*. Specifying this and *dist0* speeds up the computation significantly instead of using the *population* alone. Note that you must then also specify *slicer* and *indices*
- **weights** (EnVariables) – The multiple regression coefficients (weights) for the calculating the energy consumption. If not given, the (0-dimensional) weights after [LeNechet2012] (*weights_LeNechet*, see above equation) are used.

Returns

- *np.ndarray of dtype float* – The energy consumption. The shape of the array depends on the given *weights*
- *float* – The average distance between two individuals (DIST)
- *float* – The entropy ENTROP
- *float* – The rank-size-slope RSS

See also:

OWN(), *entrop()*, *rss()*, dist_numerator()

iucm.energy_consumption.**entrop**(*population*, *size*)

Compute the entropy of a city

Compute the entropy of a city after [LeNechet2012] via

$$ENTROP = \frac{\sum_{i=1}^{size} \frac{p_i}{P_{sum}} \log \frac{p_i}{P_{sum}}}{\log size}$$

Parameters

- **population** (*1D np.ndarray*) – The population data (must not contain 0!)
- **size** (*int*) – The original size of the *population* data (including 0)

Returns The entropy value

Return type float

iucm.energy_consumption.**random_weights**(*weights*)
Draw random weights

This functions draws random weights and fills the arrays in the given *weights* with them. Weights are drawn using normal distributions defined through the *weights_LeNechet* and *std_err_LeNechet*.

Parameters **weights** (EnVariables) – The arrays to fill

Notes

weights are modified inplace!

iucm.energy_consumption.**rss**(*population*)
Compute the Rank-Size slope coefficient

The rank-size coefficient $a > 0$ is calculated through a linear fit after [LeNechet2012] with

$$\log \frac{p_k}{p_1} = -a \log k$$

where p_k is the :population of the math:k-th ranking cell.

Parameters **population** (*1D np.ndarray*) – The population data (must not contain 0!)

Returns The rank-size coefficient

Return type float

iucm.energy_consumption.**std_err_LeNechet = EnVariables(k=10500.0, dist=74.88, entrop=9172.0**
The standard errors of the weights used in [LeNechet2012] (obtained through private communication via R. Cremades)

iucm.energy_consumption.**wDIST = 279.0**
weight for average distance of two individuals on energy consumption

iucm.energy_consumption.**wENTROP = 21700.0**
weight for entropy on energy consumption

iucm.energy_consumption.**wOWN = 17.36**
weight for car owner ship on energy consumption

iucm.energy_consumption.**wRSS = -9343.0**
weight for rank-size rule slope on energy consumption

iucm.energy_consumption.**weights_LeNechet = EnVariables(k=-346.5, dist=279.0, entrop=21700.0**
The weights for the variables to calculate the energy_consumption from the multiple regression after [LeNechet2012]

iucm.main module

Main module of the iucm package

This module defines the *IUCMOrganizer* class that is used to create a command line parser and to manage the configuration of the experiments

Classes

IUCMOrganizer([config])	A class for organizing a model

Functions

main()	Call the main() method of the

class iucm.main.**IUCMOrganizer**(*config=None*)

Bases: model_organization.ModelOrganizer

A class for organizing a model

This class is indended to have hold the basic functions for organizing a model. You can subclass the functions setup, init to fit to your model. When using the model from the command line, you can also use the setup_parser() method to create the argument parsers

Parameters **config** (*model_organization.config.Config*) – The configuration of the organizer

Attributes

commands	Built-in mutable sequence.
name	str(object='') -> str
paths	Built-in mutable sequence.
postproc_funcs	A mapping from postproc commands to the corresponding function
preproc_funcs	A mapping from preproc commands to the corresponding function

Methods

get_population_vname(ds)	
make_map(ds[, output, odir, time, diff, t0, ...])	Make a movie of the post processing
make_movie(ds[, output, odir, diff, t0, ...])	Make a movie of the post processing
plot_evolution(ds[, output, odir, close, ...])	Plot the evolution of DIST, RSS, ENTROP and Energy consumption
postproc([no_input])	Postprocess the data
preproc(**kwargs)	Preprocess the data
preproc_forcing([development_file, output, ...])	Create a forcing file from a predescribed population path
preproc_mask(shapefile[, method, ifile, ...])	Mask grid cells based on a shape file
rolling_mean(ds[, window, output, odir])	Calculate the rolling mean for the energy consumption
run([ifile, forcing, vname, steps, ...])	Run the model for the given experiment

commands = ['setup', 'init', 'set_value', 'get_value', 'del_value', 'info', 'unarchive

get_population_vname(*ds*)

make_map(*ds*, *output='map.pdf'*, *odir=None*, *time=-1*, *diff=False*, *t0=0*, *project_file=None*, *save_project=None*, *simple_plot=False*, *close=True*, ***kwargs*)

Make a movie of the post processing

Parameters

- **ds** (*xarray.Dataset*) – The dataset for the plot or a list of filenames
- **output** (*str*) – The name of the output file
- **odir** (*str*) – The name of the output directory
- **time** (*int*) – The timestep to plot. By default, the last timestep is used

- **diff** (*bool*) – If True/set, visualize the difference to the *t0* (by default, the first step) is used, instead of the entire data
- **t0** (*int*) – If *diff* is set, the reference step for the difference
- **project_file** (*str*) – The path to a filename containing a file that can be loaded via the psyplot.project.Project.load_project() method
- **save_project** (*str*) – The path to a filename where to save the psyplot project
- **simple_plot** (*bool*) – If True/set, use a non-georeferenced plot. Otherwise, we use the cartopy module to plot it
- **close** (*bool*) – If True, close the project at the end

Other Parameters **‘‘**kwargs‘‘** – Any other keyword that is passed to the psyplot.project.Project.export() method

make_movie(*ds*, *output='movie.gif'*, *odir=None*, *diff=False*, *t0=None*, *project_file=None*, *save_project=None*, *simple_plot=False*, *close=True*, *time=None*, ***kwargs*)

Make a movie of the post processing

Parameters

- **ds** (*xarray.Dataset*) – The dataset for the plot or a list of filenames
- **output** (*str*) – The name of the output file
- **odir** (*str*) – The name of the output directory
- **diff** (*bool*) – If True/set, visualize the difference to the *t0* (by default, the first step) is used, instead of the entire data
- **t0** (*int*) – If *diff* is set, the reference step for the difference
- **project_file** (*str*) – The path to a filename containing a file that can be loaded via the psyplot.project.Project.load_project() method
- **save_project** (*str*) – The path to a filename where to save the psyplot project
- **simple_plot** (*bool*) – If True/set, use a non-georeferenced plot. Otherwise, we use the cartopy module to plot it
- **close** (*bool*) – If True, close the project at the end
- **time** (*list of int*) – The time steps to use for the movie

Other Parameters **‘‘**kwargs‘‘** – Any other keyword for the matplotlib.animation.FuncAnimation class that is used to make the plot

name = 'iucm'

paths = ['expdir', 'src', 'data', 'input', 'outdata', 'outdir', 'plot_output', 'projec

plot_evolution(*ds*, *output='plots.pdf'*, *odir=None*, *close=True*, *time=None*, *use_rolling=False*)

Plot the evolution of DIST, RSS, ENTROP and Energy consumption

Parameters

- **ds** (*xarray.Dataset*) – The dataset for the plot or a list of filenames
- **output** (*str*) – The name of the output file
- **odir** (*str*) – The name of the output directory
- **close** (*bool*) – If True, the created figures are closed in the end
- **time** (*list of int*) – The time steps to use for the movie

- **use_rolling** (*bool*) – If True, use the rolling mean for the energy consumption

Returns Information on the output

Return type dict

postproc(*no_input=False*, ***kwargs*)
Postprocess the data

Parameters **no_input** (*bool*) – If True/set, the initial input file is ignored

postproc_funcs
A mapping from postproc commands to the corresponding function

preproc(***kwargs*)
Preprocess the data

Parameters ****kwargs** – Any keyword from the *preproc* attribute with kws for the corresponding function, or any keyword for the *main()* method

preproc_forcing(*development_file=None*, *output=None*, *date_cols=None*, *steps=1*, *movement=0*, *trans_size=0*, *population_col=None*, *no_date_parsing=False*)
Create a forcing file from a predescribed population path

Parameters

- **development_file** (*str*) – The path to a csv file containing (at least) one column with the projected population development
- **output** (*str*) – The name of the ouput forcing netCDF file. By default: `'<expdir>/input/forcing.nc'`
- **date_cols** (*list of str*) – The names of the date columns in the *development_file* that shall be used to generate the date-time information. If not given, the date will simply be a range from 1 to *steps* times the length of the projected population development from *development_file*
- **steps** (*int*) – The numbers of model steps between on step of the projected development in *development_file*
- **movement** (*float*) – The people moving randomly during on model step
- **trans_size** (*float*) – The forced size of the transformation additionally to the development from *development_file*
- **population_col** (*str*) – The name of the column with population data. If not given, the last one is used
- **no_date_parsing** (*bool*) – If True, then *date_cols* is simply interpreted as an index and no date-time information is parsed

preproc_funcs
A mapping from preproc commands to the corresponding function

preproc_mask(*shapefile*, *method='max-pop'*, *ifile=None*, *vname=None*, *overwrite=False*, *max_pop=None*, *hr_res=100*)
Mask grid cells based on a shape file

This command calculates the maximum population for the model based on a masking shape file. The given shape file is rasterized at high resolution (by default, 100 times the resolution of the input file) and the fraction for each grid cell that is covered by that shape file is calculated.

Parameters

- **shapefile** (*str*) – The path to a shapefile

- **method** (*{'max-pop', 'mask', 'ignore'}*) – Determines how to handle the given shapes.

  max-pop The maximum population per grid cell is lowered by the fraction of the cell that is covered by the given shape. This will adjust the *max_pop* variable in the input file *ifile*

  mask The population of the grid cells in the input data that are touched by the given shapes will be kept constant during the simulation. This will adjust the *mask* variable in the input file *ifile*

  ignore The grid cells in the input data that are touched by the given grid cells are put to NaN and their population is not considered during the simulation. This will adjust the input population data (i.e. variable *vname*) directly

- **ifile** (*str*) – The path of the input file. If not specified, the value of the configuration is used
- **vname** (*str*) – The variable name to use. If not specified and only one variable exists in the dataset, this one is used. Otherwise, the `'run.vname'` key in the experiment configuration is used
- **overwrite** (*bool*) – If True and the target variable exists already in the input file *ifile* (and method is not 'ignore'), this variable is overwritten
- **max_pop** (*float*) – The maximum population. If not specified, the value in the configuration is used (only necessary if `method=='max-pop'`)
- **hr_res** (*int*) – The resolution of the high resolution file, relative to the resolution of *ifile* (only necessary if `method=='max-pop'`)

Notes

Note that the shapefile and the input file have to be defined on the same coordinate system! This function is not super efficient, for large data files we recommend using gdal_rasterize and gdalwarp.

rolling_mean(*ds*, *window=None*, *output=None*, *odir=None*)
Calculate the rolling mean for the energy consumption

This postprocessing function calculates the rolling mean for the energy consumption

Parameters

- **ds** (*xarray.Dataset*) – The dataset with the *cons* and *cons_std* variables
- **window** (*int*) – Size of the moving window. This is the number of observations used for calculating the statistic. Each window will be a fixed size. If None, it will be taken from the experiment configuration with a default value of 50.
- **output** (*str*) – A filename where to save the output. If not given, it is not saved but may be later used by the `evolution()` method
- **odir** (*str*) – The name of the output directory

run(*ifile=None*, *forcing=None*, *vname=None*, *steps=50*, *selection_method=None*, *update_method=None*, *ncells=None*, *categories=None*, *use_pctls=False*, *no_restart=False*, *output_step=None*, *seed=None*, *stop_en_change=None*, *agg_stop_steps=100*, *agg_steps=1*, *probabilistic=None*, *max_pop=None*, *coord_transform=None*, *copy_from=None*, ***kwargs*)
Run the model for the given experiment

Parameters

- **ifile** (*str*) – The input file. If not specified, the *input* key in the experiment configuration is used
- **forcing** (*str*) – The forcing file (necessary if `update_method=='forced'`). If not specified, the *forcing* key in the experiment configuration is used
- **vname** (*str*) – The variable name to use. If not specified and only one variable exists in the dataset, this one is used. Otherwise, the `'run.vname'` key in the experiment configuration is used
- **steps** (*int*) – The number of time steps
- **selection_method** (*{ 'consecutive' | 'random' }*) – The name of the method on how the data is selected. The default is consecutive. Possible selection methods are

  consecutive: Always the *ncells* consecutive cells are selected.

  random: *ncells* random cells in the field are updated.
- **update_method** (*{ 'categorical' | 'random' | 'forced' }*) – The name of the update method on how the selected cells (see *selection_method* are updated). The default is categorical. Possible update methods are

  categorical: The selected cells are updated to the lower level of the next category.

  random: The selected cells are updated to a randum number within the next category.

  forced: A forcing file is used (see the *forcing* parameter).
- **ncells** (*int*) – The number of cells that shall be changed during 1 step. The default value is 4
- **categories** (*list of floats*) – The values for the categories to use within the models
- **use_pctls** (*bool*) – If True, interprete *categories* as percentiles instead of real population density
- **no_restart** (*bool*) – If True, and the run has already been conducted, restart it. Otherwise the previous run is continued
- **output_step** (*int*) – Make an output every *output_step*. If None, only the final result is written to the output file
- **seed** (*int*) – The random seed for numpy to use. Specify this parameter for the experiment to guarantee reproducability
- **stop_en_change** (*float*) – The minimum of required relative energy consumption change. If the mean relative energy consumption change over the last *agg_stop_steps* steps is below this number, the run is stopped
- **agg_stop_steps** (*int*) – The number of steps to aggregate over when calculating the mean relative energy consumption change. Does not have an effect if *stop_en_change* is None
- **agg_steps** (*int*) – Use only every *agg_steps* energy consumption for the aggregation when checking the *stop_en_change* criteria
- **probabilistic** (*int*) – The number of probabilistic scenarios. For each scenario the energy consumption is calculated and the final population is distributed to the cells with the ideal energy consumption. Set this to 0 to only use the weights by [LeNechet2012]. If this option is None, the value will be taken from the configuration with a default of 0 (i.e. no probabilistic run).

- **max_pop** (*float*) – The maximum population for each cell. If None, the last value in *categories* will be used or what is stored in the experiment configuration
- **coord_transform** (*float*) – The transformation factor to transform the coordinate values into kilometres
- **copy_from** (*str*) – If not None, copy the run settings from the other given experiment
- ****kwargs** – Any other keyword argument that is passed to the *main()* method

iucm.main.**main**()
Call the main() method of the *IUCMOrganizer* class

iucm.model module

Classes

Output(cons, dist, entrop, rss, cons_det, ...)	The state of the model
Output2D(cons, dist, entrop, rss, cons_det, ...)	The 2D state variables of the model
PopulationModel(data, x, y[, ...])	Class that runs and manages the population model

Data

fields	Meta information for the variables in the 1D-output of the
fields2D	Meta information for the variables in the 2D-output of the

class iucm.model.**Output**(*cons*, *dist*, *entrop*, *rss*, *cons_det*, *cons_std*, *left_over*, *nscenarios*)
Bases: tuple

The state of the model

The collections.namedtuple() defining the state of the model during a single step. Each field of this class corresponds to one output variable in the output netCDF of the *PopulationModel*. Meta information are taken from the *fields* dictionary

Parameters

- **cons** (*float*) – Energy consumption
- **dist** (*float*) – Average distance between two individuals
- **entrop** (*float*) – Entropy
- **rss** (*float*) – Rank-Size Slope
- **cons_det** (*float*) – The deterministic energy consumption based on *iucm.energy_consumption.weights_LeNechet*
- **cons_std** (*float*) – The standard deviation of the energy consumption
- **left_over** (*float*) – The left over inhabitants that could not be subtracted in the last step
- **nscenarios** (*float*) – The number of scenarios that have been changed during this step

Attributes

cons	Alias for field number 0
cons_det	Alias for field number 4
cons_std	Alias for field number 5
dist	Alias for field number 1
entrop	Alias for field number 2
left_over	Alias for field number 6
nscenarios	Alias for field number 7
rss	Alias for field number 3

See also:

Output2D, *PopulationModel.state*, *PopulationModel.allocate_output*

Create new instance of Output(cons, dist, entrop, rss, cons_det, cons_std, left_over, nscenarios)

cons
Alias for field number 0

cons_det
Alias for field number 4

cons_std
Alias for field number 5

dist
Alias for field number 1

entrop
Alias for field number 2

left_over
Alias for field number 6

nscenarios
Alias for field number 7

rss
Alias for field number 3

class iucm.model.**Output2D**(*cons*, *dist*, *entrop*, *rss*, *cons_det*, *cons_std*, *left_over*, *nscenarios*, *scenarios*)

Bases: tuple

The 2D state variables of the model

2D output variables. Each variable corresponds to the same variable as in 0-d *Output* objects, but instead of saving only the best state of the model, this output saves all scenarios

Parameters

- **cons** (*np.ndarray of dtype float*) – Energy consumption
- **dist** (*np.ndarray of dtype float*) – Average distance between two individuals
- **entrop** (*np.ndarray of dtype float*) – Entropy
- **rss** (*np.ndarray of dtype float*) – Rank-Size Slope
- **cons_det** (*np.ndarray of dtype float*) – The deterministic energy consumption based on *iucm.energy_consumption.weights_LeNechet*
- **cons_std** (*np.ndarray of dtype float*) – The standard deviations within each probabilistic scenario

- **left_over** (*float*) – The left over inhabitants that could not be subtracted in the last step
- **nscenarios** (*float*) – The weight of each scenario that has been used during this step
- **scenarios** (*np.ndarray of dtype int*) – The number of the scenario

Attributes

cons	Alias for field number 0
cons_det	Alias for field number 4
cons_std	Alias for field number 5
dist	Alias for field number 1
entrop	Alias for field number 2
left_over	Alias for field number 6
nscenarios	Alias for field number 7
rss	Alias for field number 3
scenarios	Alias for field number 8

See also:

Output, *PopulationModel.state2d*

Create new instance of Output2D(cons, dist, entrop, rss, cons_det, cons_std, left_over, nscenarios, scenarios)

cons
Alias for field number 0

cons_det
Alias for field number 4

cons_std
Alias for field number 5

dist
Alias for field number 1

entrop
Alias for field number 2

left_over
Alias for field number 6

nscenarios
Alias for field number 7

rss
Alias for field number 3

scenarios
Alias for field number 8

class iucm.model.**PopulationModel**(*data*, *x*, *y*, *selection_method='consecutive'*, *update_method='categorical'*, *ncells=4*, *categories=None*, *state=None*, *forcing=None*, *probabilistic=0*, *max_pop=None*, *use_pctls=False*, *last_step=0*, *data2modify=None*)

Bases: object

Class that runs and manages the population model

This class represents one instance of the model for one experiment and is responsible for all the computation, parallelization and input-output coordination. The major important features of this class are

***from_da()* method** A classmethod to construct the model from a `xarray.DataArray`

***step()* method** The method that is brings the model to the next step

init_step_methods* and *update_methods The available update methods how to bring the model to the next change

selection_methods The available selection methods that define the available scenarios

update_methods The available update methods that define how the population change for the given selection is pursued

***state* attribute** The current state of the model. It's an instance of the *Output* class containing all 1D-variables of the current *data* attribute

Methods

allocate_output(da, dsi, steps)	Create the dataset for the output
best_scenario(all_slices, all_indices)	Compute the best scenario
categorical_update(cell_values, slicer)	Change the values through an update to the next category
consecutive_selection()	
distribute_probabilistic(slices, nscenarios)	Redistribute the population increase to the best scenarios
from_da(da, dsi[, ofiles, osteps, ...])	Construct the model from a `psyplot.data.InteractiveArray`
get_input_ds(data, dsi, **kwargs)	Return the input dataset which can be concatenated with the output
initialize_model(da, dsi, ofiles, osteps[, mask])	Initialize the model on the I/O processor
random_selection()	
randomized_update(cell_values, slicer)	Change the values through an update to a number within the next
start_processes(nprocs)	Start *nprocs* processes for the model
step()	Bring the model to the next step and eventually write the output
stop_processes()	Stop the processes for the model
subtract_random()	Subtract the people moving during this timestep
sync_state(sl, cell_values[, state, weights])	Synchronize the `data` attribute between the processes
value_update(cell_values, slicer[, remaining])	Change the cells by using the forcing
write()	Write the current state to the output dataset
write_output([complete])	Write the data to the next netCDF file

Attributes

consumption	Energy consumption
current_step	The current step since the last output.
data	np.ndarray. The current simulation data of the model
data2modify	np.ndarray. The positions of the points in `data` that can be
dist	Denumerator of average distance between two individuals

Continued on next page

Table 15 – continued from previous page

init_step	The step initialization method from `init_step_methods` we use to
init_step_methods	Mapping from *update_method* name to the corresponding init function
left_over	Left over population that could not have been subtracted for within
movement	The population movement during this time step
ncells	`int`. The number of cells that are modified for one scenario
nprocs	`int`. The number of processes started for this model
output_written	Flag that is True if the data was written to a file during the last
population_change	The population change during this time step
procs	`multiprocessing.Process`. The processes of this model
select	The selection method from `selection_methods` we use to define the
selection_methods	Mapping from *selection_method* name to the corresponding function
state	The state as a namedtuple.
state2d	The different values from the `state` of the model for each
state2d_dict	The state as a dictionary.
state_dict	The state as a dictionary.
total_change	The total population change during this time step
total_step	The absolute current step in case the model has been restarted.
total_step_run	The current step since the initialization of the model.
update	The update method from `update_methods` we use to compute that
update_methods	Mapping from *update_method* name to the corresponding function
x	np.ndarray. The x-coordinates of the points in `data`
y	np.ndarray. The y-coordinates of the points in `data`

Most of the other routines are related to input/output and parallelization

Parameters

- **data** (*np.ndarray*) – The 1D-data array (without NaN!) of the model
- **x** (*np.ndarray*) – The x-coordinates in *km* of each point in *data* (same shape as *data*)
- **y** (*np.ndarray*) – The y-coordinates in *km* of each point in *data* (same shape as *data*)
- **selection_method** (*{ 'consecutive' | 'random' }*) – The avaiable selection scenarios (see *selection_methods*)
- **update_method** (*{ 'categorical' | 'random' | 'forced' }*) – The avaiable update methods (see *update_methods*)
- **ncells** (*int*) – The number of cells that shall be modified for one scenario. The higher the number, the less computationally expensive is the computation
- **categories** (*list of str*) – The categories to use. If *update_method* is `'categorical'`, it describes the categories and if *use_pctls* is True, it the each category

is interpreted as a quantile in *data*

- **state** (*list of float*) – The current state of *data*. Must be a list corresponding to the *Output* class
- **forcing** (*xarray.Dataset*) – The input dataset for the model containing variables with population evolution information. Possible variables in the netCDF file are *movement* containing the number of people to move and *change* containing the population change (positive or negative)
- **probabilistic** (*int or tuple*) – The number of probabilistic scenarios. For each scenario the energy consumption is calculated and the final population is distributed to the cells with the ideal energy consumption. Set this to 0 to only use the weights by [LeNechet2012]. If tuple, then they are considered as the weights
- **max_pop** (*np.ndarray*) – A 1d-array with the maximum population for each cell in *data*. If None, the last value in *categories* will be used
- **use_pctls** (*bool*) – If True, values given in *categories* are interpreted as quantiles
- **last_step** (*int*) – If the model is restarted, the total number of already made steps (see *total_step* attribute)
- **data2modify** (*np.ndarray*) – The indices of points in *data* which are allowed to be modified. If None, all points are allowed to be modified

See also:

from_da A more convenient initialization method using a xarray.Dataset

allocate_output(*da*, *dsi*, *steps*)
Create the dataset for the output

Parameters

- **da** (*psyplot.data.IneractiveArray*) – The input data for the model
- **dsi** (*xarray.Dataset*) – The dataset *data* belongs to. If None, the psyplot.data.InteractiveArray.base attribute is used
- **steps** (*int*) – The number of steps

best_scenario(*all_slices*, *all_indices*)
Compute the best scenario

This method computes the best scenario for the given scenarios defined through the given *slices* and *indices*

Parameters

- **slices** (list of None, slice or boolean arrays) – The slicing objects for each scenario that allow us to create a view of the *data* attribute that we modify in place. If list of None, it is computed using *indices*
- **indices** (*list of list of :class`int`*) – The numpy array containing the integer position in *data* of the cells modified for each scenario

Returns

- *1-dim np.ndarray of dtype float* – The consumptions of the best scenarios for each set of weights used
- 2-dim np.ndarray of dtype float with shape (nprob, len(self.state)) – The state of the best scenario for each probabilistic scenario which can be used for the *state* attribute.

- list of `slice` or boolean array – The slicing object from *slices* that corresponds to the best scenario and can be used to create a view on the `data`
- *list of list of float* – The numbers of the modified cells for the best scenario
- *list of 2d-np.ndarrays* – The 2d variables of the `state2d` attribute

categorical_update(*cell_values*, *slicer*)
Change the values through an update to the next category

This method increases the population by updating the cells to the next (possible) category

consecutive_selection()

consumption
Energy consumption

current_step = -1
The current step since the last output. -1 means, output has just been written or the model has been initialized

data = None
np.ndarray. The current simulation data of the model

data2modify = None
np.ndarray. The positions of the points in `data` that can be modified

dist
Denumerator of average distance between two individuals

distribute_probabilistic(*slices*, *nscenarios*)
Redistribute the population increase to the best scenarios

This method distributes the population changes to the cells that have been computed as the best scenarios. It takes the input of the `best_scenario()` method

Parameters **slices** (*list*) – The slicers of the best scenarios

classmethod from_da(*da*, *dsi*, *ofiles=None*, *osteps=None*, *coord_transform=1*, ***kwargs*)
Construct the model from a `psyplot.data.InteractiveArray`

Parameters

- **data** (*xr.DataArray*) – The dataarray during the initialization
- **dsi** (*xr.Dataset*) – The base dataset of the *da*
- **ofiles** (*list of str*) – The name of the output files
- **osteps** (*list of int*) – Steps when to make the output
- **coord_transform** (*float*) – The transformation factor to transform the coordinate values into kilometres

Returns The model created ready to use

Return type *PopulationModel*

classmethod get_input_ds(*data*, *dsi*, ***kwargs*)
Return the input dataset which can be concatenated with the output

Parameters

- **data** (*xr.DataArray*) – The dataarray during the initialization
- **dsi** (*xr.Dataset*) – The base dataset of the *da*

Returns The modified *dsi*

Return type xr.Dataset

init_step = None
The step initialization method from *init_step_methods* we use to define the scenarios

init_step_methods
Mapping from *update_method* name to the corresponding init function

This property defines the init_step methods. Those methods are called at the beginning of each step on the main processor (I/O-processor). Each init_step method must accept no arguments and return a tuple with

- an `slice` object or boolean array containing the information where the data changed
- the indices of the cells in *data* that changed

initialize_model(*da*, *dsi*, *ofiles*, *osteps*, *mask=None*)
Initialize the model on the I/O processor

Parameters

- **data** (*xr.DataArray*) – The dataarray during the initialization
- **dsi** (*xr.Dataset*) – The base dataset of the *da*
- **ofiles** (*list of str*) – The name of the output files
- **osteps** (*list of int*) – Steps when to make the output
- **mask** (*np.ndarray*) – A boolean array that maps from the *data* attribute into the 2D output data array

left_over
Left over population that could not have been subtracted for within the *PopulationModel.value_update()* method and should be considered in the next step

movement
The population movement during this time step

ncells = 4
`int`. The number of cells that are modified for one scenario

nprocs
`int`. The number of processes started for this model

output_written = False
Flag that is True if the data was written to a file during the last step

population_change
The population change during this time step

procs = None
`multiprocessing.Process`. The processes of this model

random_selection()

randomized_update(*cell_values*, *slicer*)
Change the values through an update to a number within the next category

This method increases the population by updating the cells to a random value within the next (possible) category

select = None
The selection method from *selection_methods* we use to define the scenarios

selection_methods
Mapping from *selection_method* name to the corresponding function

This property defines the selection methods. Those methods are called at the beginning of each step on the main processor (I/O-processor). Each selection_step method must accept no arguments and return a tuple with

- an `slice` object or boolean array containing the information where the data changed. Alternatively it can be a list of `None` and those slicing objects will be computed from the second argument
- a 2D list of dtype integer containing the indices of the cells that for changed for the corresponding scenario

start_processes(*nprocs*)
Start *nprocs* processes for the model

state
The state as a namedtuple. You may set it with an iterable defined by the *Output* class

state2d
The different values from the *state* of the model for each of the scenarios

state2d_dict
The state as a dictionary. Mapping from state variable to the corresponding value. You may also set it with a dictionary

state_dict
The state as a dictionary. Mapping from state variable to the corresponding value. You may also set it with a dictionary

step()
Bring the model to the next step and eventually write the output

This method is the core of the entire *PopulationModel* API connecting the necessary functions to compute the next best scenario. The general structure is

1. initialize the step (see *init_step_methods*)
2. define the scenarios (see *selection_methods*)
3. choose the best scenario (see *best_scenario* and *update_methods*)
4. write the output (see *write()* method)

Depending on whether the *start_processes()* method has been called earlier, this is either done serial or in parallel

stop_processes()
Stop the processes for the model

subtract_random()
Subtract the people moving during this timestep

sync_state(*sl*, *cell_values*, *state=None*, *weights=None*)
Synchronize the *data* attribute between the processes

This method is called to synchronize the states of the model in the different processes

Parameters

- **sl** (`slice` or boolean np.ndarray) – The slicer that can be used to create a view on the *data*
- **cell_values** (*np.ndarray of dtype float*) – The values of the cells described by *sl*

- **state** (Output) – The state of the model

total_change
The total population change during this time step

total_step = -1
The absolute current step in case the model has been restarted. If -1, the model has not yet been started

total_step_run = -1
The current step since the initialization of the model. -1 means, the model has been initialized

update = None
The update method from *update_methods* we use to compute that changes for each scenario

update_methods
Mapping from *update_method* name to the corresponding function

This property defines the update methods. Each update method must accept and return a 1D numpy array of dtype float64 containing a view of the *data* attribute. The data must be modified in place!

value_update(*cell_values*, *slicer*, *remaining=None*)
Change the cells by using the forcing

This update method changes the given cells based upon the *movement* information and the *population_change* information from the `forcing` dataset

write()
Write the current state to the output dataset

write_output(*complete=True*)
Write the data to the next netCDF file

Parameters **complete** (*bool*) – If True, write the complete dataset, otherwise only until the current step

x = None
np.ndarray. The x-coordinates of the points in *data*

y = None
np.ndarray. The y-coordinates of the points in *data*

iucm.model.**fields = {'cons': {'long_name': 'Energy consumption', 'units': 'MJ / inh'},**
Meta information for the variables in the 1D-output of the *PopulationModel*

iucm.model.**fields2D = {'cons': {'long_name': 'Energy consumption', 'units': 'MJ / inh'},**
Meta information for the variables in the 2D-output of the *PopulationModel*

iucm.utils module

General utilities for the iucm package

Functions

append_doc(namedtuple_cls, doc)	Append a documentation to a namedtuple
str_ranges(s)	Convert a string of comma separated values to an iterable

iucm.utils.**append_doc**(*namedtuple_cls*, *doc*)
Append a documentation to a namedtuple

Parameters

- **namedtuple_cls** (*type*) – The type that has been created with `collections.namedtuple()`
- **doc** (*str*) – The documentation docstring

`iucm.utils.`**str_ranges**(*s*)

Convert a string of comma separated values to an iterable

Parameters **s** (*str*) – A semicolon (`';'`) separated string. A single value in this string represents one number, ranges can also be used via a separation by comma (`','`). Hence, `'2009;2012, 2015'` will be converted to `[2009,2012, 2013, 2014]` and `2009;2012,2015,2` to `[2009, 2012, 2015]`

Returns The values in s converted to a list

Return type list

iucm.version module

CHAPTER 2

License

CHAPTER 3

Acknowledgements

The authors thank Florent Le Néchet for his comments and for the provision of further statistical details about his publications. The authors thank Walter Sauf for his support on using the facilities of the German Supercomputing Center (DKRZ). The authors also wish to express their gratitude to Wolfgang Lucht, Hermann Held, Andreas Haensler, Diego Rybski and Jürgen P. Kropp for their helpful comments. PS gratefully acknowledges funding from the Swiss National Science Foundation ((ACACIA, CR10I2_146314)). RC gratefully acknowledges support from the Earth-Doc programme of the Earth League.

3.1 Indices and tables

- genindex
- modindex
- search

Bibliography

[LeNechet2012] Le Néchet, Florent. “Urban spatial structure, daily mobility and energy consumption: a study of 34 european cities.” Cybergeo: European Journal of Geography (2012).

Python Module Index

i

Index

V

W

X

Y