PumHa Documentation

Release 1.0.0

Chloe Sumner, Elen Kalda, Marcin Kirsz

Nov 02, 2017

CONTENTS

PumHa package simulates the population dynamics of hares and pumas in a user-specified landscape. The population densities depend on a number of parameters, such as birth, death and diffusion rates, also on a rate at which pumas eat hares. For more in-depth mathematical formulation, see PumaPopulation and HarePopulation classes in the pumha.pop module.

PumHa is written in Python programming language, it is compatible with any version of Python 2.7 or higher. Provided the package is correctly installed (see *How to install*), it can be run in variety of operating systems (see *System compatibility and requirements* for a list of operating systems in which the package has been tested).

The package was developed using GitHub revision control mechanism and the tests were created using Python's unittest framework. The code complies with PEP 257 and PEP 8 conventions. The documentation was generated using Sphinx and it is located in the `docs` folder inside the project directory.

CHAPTER ONE

HOW TO INSTALL

To install PumHa package on a Linux machine (see *System compatibility and requirements* for known compatibility):

1. Change directory to your install directory (or create one).
2. Copy repository from Github by running the command:

```
git clone https://github.com/ad1v7/PumHa
```

Alternatively, if you are lucky to have a tar.gz package in your computer (in fact you would be very lucky because only 4 people out of 7+ billions have it!), you can do the following:

3. Or extract archive content using:

```
tar zxvf pumha.tar.gz
```

where pumha.tar.gz is replaced with your archive name

4. Make sure you are in a directory which contains setup.py and use pip:

```
pip install .
```

You might need to run above command as super user (root):

```
sudo pip install .
```

If you can't run it as a root, you can try:

```
pip install --user .
```

In that case pip will install command line script into:

```
~/.local/bin
```

directory. This is the case for Scientific Linux and Ubuntu.

If `~/.local/bin` is not in your `$PATH` (run `echo $PATH` to check it), you can export it running:

```
export PATH=$PATH:~/.local/bin
```

You may want to add above line to `~/.profile` so `~/.local/bin` is added to the path at login.

If you want to use the package with python 3 you might need to run pip3 in place of pip for above commands.

CHAPTER

TWO

HOW TO USE

Once you have correctly installed the package, you can run the simulation from any directory by typing into the command line:

```
pumha <landscape_file> [<config_file>]
```

`<config_file>` is a JSON type configuration file that contains all the information about the parameter values. You are welcome to play around with those values! You can find the default configuration file `default.dat` in `...installation_path/pumha/data` directory (you can copy-paste it into your simulation directory, if you want).

If configuration file is not provided, the program will display warning and will continue using the default values form `default.dat` file. If configuration file is not present or was accidentally deleted, it can be regenerated by running a simulation without specifying config file:

```
pumha <landscape_file>
```

Therefore the simulation can be run without specifying a configuration file, but you must provide a `<landscape_file>`. This file must be a bitmask ASCII file with the first row giving the dimensions of the landscape and rest of the rows representing landscape (1 for land square and 0 for water), as an example:

```
5 7

1 1 1 1 1 1 1
1 1 1 1 0 0 1
1 0 0 0 0 0 0
1 1 0 0 0 1 1
1 1 0 0 0 1 1
```

There are some example landscapes in the `...installation_path/pumha/data` directory.

Once you have run the simulation, an output folder with a time stamp `PumHa_out_Y-m-d-H-M-S` will be created in your simulation directory. In that directory there will be a file `average_densities.dat` in which there are three columns, the first column giving a timestep value and rest two showing the average values of hare and puma densities on the whole landscape respectively. The average values are calculated for the whole landscape, including the water squares.

Rest of the files in the output folder are ppm image files that describe the densities of pumas and hares on a given timestep. Each pixel represents a square on landscape grid. Blue pixels denote water. On the pixels representing land, red denotes puma density and green hare density (so the more red/green the square, the higher is puma/hare population on that square). The population values are scaled relative to the highest density of animals that appeared in the whole simulation on a single square.

CHAPTER

THREE

HOW TO RUN TESTS

To run the tests, go into the directory which contains `setup.py` and run the following command:

```
python setup.py test
```

Depending on how you have installed the package, you might need to run the tests as root:

```
sudo python setup.py test
```

Testing requires nose which will be installed by pip automatically together with other dependencies.

CHAPTER

FOUR

SYSTEM COMPATIBILITY AND REQUIREMENTS

The package was tested on:

```
Scientific Linux release 7.3 (Nitrogen)
Ubuntu 16.04.3 LTS
Ubuntu 14.04 LTS
Windows 10 Home
```

The package is likely to work on other systems as well, but there is no guarantee to that. Also, if you are using an operating system which is not listed above, the installation procedure may also differ from the one outlined in this document.

The package requires following dependencies:

- numpy >=1.9.2
- simplejson>=3.8.1
- scipy>=0.15.1
- tqdm>=4.19.4
- jsonschema>=2.6.0
- docopt>=0.6.2

The package has been tested with the minimum required version, but it is likely that the package will work with older versions as well.

Above packages should be installed automatically when using pip (as described in the section *How to install*). However, if there are some issues with the installation, they can be installed separately using pip:

```
sudo pip install  package_name
```

or if root is not available:

```
pip install --user package_name
```

CHAPTER

FIVE

KEY DESIGN DECISIONS

This section discusses some design and implementation decisions.

5.1 Usage properties

PumHa package has been designed keeping the ease of usage in mind. All the additional packages required for the code to run can easily installed with pip (see *How to install*). Since it can be installed as a Python module, it can be easily used as a part of other simulation softwares.

Every simulation produces an output folder with a timestamp in its name, making it easy to keep track of previously run simulations.

5.2 Handling input

The package has nice buit-in mechanisms for handling invalid input data:

- If a configuration file is not in a JSON format or has invalid input values, the program terminates the simulation and generates a new configuration file in a correct format, giving the user an opportunity to "try again" by changing parameter values in a correct configuration file.
- Classes Landscape and Configuration that deal with user input, have built-in error checks, that can handle majority of cases.
- If the user does not have configuration file or has deleted the default one, it is simple to generate a new one - simply run the program without specifying a configuration file!
- The program will display an error message and terminates if no landscape file is provided or if the landscape file is not in a correct format, since there is no point in running a simulation on a landscape that is not the one user wanted to simulate.

5.3 Class structure

The code is modular and loosely coupled and it is hence easy to extend it and make changes locally, without having to rewrite methods in different modules or classes. With the choice of variable names, the code aims to be as self-documenting as possible.

The modular structure and use of inheritance in the pumha.pop module allows modules and classes to be used in different projects. The population class has methods relevant to all populations, a user can create their own subclass with corresponding methods (that perhaps use different mathematical formalism). Setting up a simulation is very simple, requiring only at the least a landscape file and one population to be specified. Hence it is simple to create custom tailored simulations.

Though it is possible to extend the code to include several populations, the output functions are specific to the case of two populations. There is a function that checks the number of populations in the simulation and if it is other than two, it displays a message and continues the simulation without providing output.

To make the simulation faster, the methods responsible for the density updates only loop over land squares. For standard landscapes this implementation can reduce the total simulation time around four times.

5.4 Output and visualisation

The output file that lists average densities at given timesteps has the timestep value, hare density and puma density written as three columns respectively, making it simple to plot.

In out visualisation implementation, both puma and hare densities on a given time step are shown on one PPM file, one pixel corresponding to one square on a grid, blue without any red or green representing water, green hare density and red puma density (for more information about the output, see *How to use*). The RGB values representing the puma and hare densities are equal to the actual value of the density at the square. The colours are scaled using the maximum value of the density found during the simulation.

Since a line in a plain PPM file must be no longer than 70 characters, all the RGB values are written into an array of strings, each element in an array corresponging to a pixel. Those strings are then written on a file, four pixels per line (since that is the maximum amount of pixels that could fit to one line if both puma and hare densities are 5-digit numbers).

CHAPTER

SIX

PUMHA

6.1 pumha package

6.1.1 pumha.env module

Environment module.

The module contains one class:

```
Landscape
```

The module creates a Landscape object which holds all the landscape-related information, such as the actual landscape grid array, information about the number of neighbouring dry squares to each square and indices of land squares.

class pumha.env.**Landscape**(*filename*)

Bases: `object`

Class for instantiating a simulation landscape.

Class checks that a valid landscape file exists, then loads it into a numpy array. The array will be padded with zeros around the given landscape. The number of land squares (represented by 1) around every square is then calculated for each array element and this information is saved into a new numpy array, so this can be used in future calculations. Finally, a list of indices is provided for land elements. When updating the population densities, this list of indices is used to avoid having to loop over water squares.

Variables **filename** (*string*) – name of file holding the landscape array

find_dry_squares()

Count the number of dry squares around each array element.

Assigns to every element of an array a value equal to the sum of it's neighbours multiplied by the kernel (see example). Since land squares have value 1 and water squares have value 0, multiplying cardinal neighbours by one and summing gives the total land in the cardinal directions.

Example:

```
Land:    0 1 0    Kernel:    0 1 0
         0 1 1               1 0 1
         0 0 0               0 1 0
```

For entry (1,1), the kernel will multiply elements (0,1), (1,0), (1,2), (2,1) by 1 (from the kernel) and everything else by 0. In the land this corresponds to:

```
(1*1), (0*1), (1*1), (0*1) = 1 + 0 + 1 + 0 = 2
```

We calculate this value just once and store it to reduce computation.

Returns array of summed neighbours

Return type integer array

find_land_squares_indices()

Return tuples of all non-zero elements of landscape.

Find the non-zero elements of the landscape array and then transpose them in to an array of tuples. This allows for just iterating over the land elements in later calculations, significantly reducing the computation.

Parameters **filename** (*string*) – name of file containing land array

Returns list of indices for zon-zero (land) landscape array elements

Return type [int, int] list

load_landscape(*filename*)

Load the landscape as a numpy array from a file.

Loads an array of 1-s for land and 0-s for water in to a numpy array from the parsed filename. The array should start on the second line of the file (the first line contains the size, so it is skipped in the loading). The method pads the array with a border of 0-s, so that the land is always surrounded by water. Before loading the landscape, the mehtod checks that the file can be loaded as a numpy array and then ensures that all entries are either 1 or 0. If either of these checks fails, the simulation will terminate.

Parameters **filename** (*string*) – name of file containing land array

Returns padded landscape array

Return type integer array

6.1.2 pumha.main module

Pumas and hares simulation.

Usage: pumha <landscape_file> [<config_file>] pumha (-h | –help | –version)

The program requires landscape file in the following format:

```
4 3

0 1 1 0
0 1 0 0
0 1 1 0
```

The first line in the input file specifies size of the landscape grid. Land is represented by 1 and water by 0. The program can simulate arbitrary size grids subject to hardware restrictions.

If config file is not provided, the program will display warning and will continue using default values.

Arguments:

```
landscape_file  required argument
config_file     optional config file
```

Options:

```
-h --help    Show this screen and exit.
--version    Print current version
```

`pumha.main.`**`main`**`()`

Entry point function for the PumHa program.

The function parses user input from the terminal and then sets up, configures and runs simulation using values in the config file.

6.1.3 pumha.pop module

Population module.

The module contains two classes:

```
Configuration
Population
```

and two subclasses of the Population class:

```
HarePopulation(Population)
PumaPopulation(Population)
```

The Configuration class consists of several methods for handling and parsing the input files and Population class with its subclasses are responsible for doing all the maths in the density change dynamics.

`class` `pumha.pop.`**`Configuration`**`(`*config_file*`)`

Bases: `object`

Class for loading simulation parameters.

Class checks that a valid configuration file has been parsed as an argument and if not, creates a default one. If a config file is given by the user, it is checks if it contains a valid JSON object for the simulation, then loads it as the configuration data and does few more data validation checks.

Variables **`filename`** (*`string`*) – name of file holding the configuration JSON

`create_config``(`*config_file*`)`

Create a default configuration file with some standard values.

This method is primarily used as a default when no file is parsed to a simulation, but is also called when a config file is passed with an error, so users will have a working file which they can edit with their own settings.

Parameters **`config_file`** (*`String`*) – Name of file containing coniguration

`load_from_file``(`*config_file*`)`

Load json config file.

Load the configuration file as a JSON object and check that the loaded object has all the correct keys.

Parameters **`config_file`** (*`String`*) – Name of file containing coniguration

`valid_config``(`*config_file*`)`

Validate format of configuration file.

Checks that the configuration file is a JSON file in the correct format by comparing it to an expected schema.

Parameters **`config_file`** (*`String`*) – Name of file containing coniguration

`class` `pumha.pop.`**`HarePopulation`**`(`*Landscape*, *birth=0.08*, *death=0.04*, *diffusion=0.2*, *min_ro=0.0*, *max_ro=5.0*, *dt=0.4*`)`

Bases: *`pumha.pop.Population`*

Hare population class with its specific update method.

This class represents hare population living in some landscape, therefore it requires Landscape object as a parameter. Remaining parameters are set to defaults and can be changed later either using provided method load_config or by simply assigning required values to instance attributes. For example use see PumaPopulation.

See Also

- pumha.pop.Population
- pumha.pop.PumaPopulation

`update_density` (*populations_old*, *populations_new*)

Update density array of hare population instance.

Method updates entire density array of hare population instance based on densities of current populations living in a landscape. Only land squares in the density array are updated.

Parameters

- **`populations_old`** (*`list of Population type`*) – list of populations at current timestep
- **`populations_new`** (*`list of Extended Population type`*) – list of populations with updated density array at t+dt

`update_density_ij` (i, j, P, H)

Return updated hare density at one (ij) square.

Method implements discrete approximation of the following equation:

$$\frac{\partial H}{\partial t} = rH - aHP + k(\frac{\partial^2 H}{\partial x^2} + \frac{\partial^2 H}{\partial y^2})$$

where

- P = density of pumas
- H = density of hares
- r = birth rate of hares
- a = death rate of hares
- k = diffusion rate of hares

Parameters

- **`i`** (*`int`*) – density array row number (first row is i=0)
- **`j`** (*`int`*) – density array column number (first column is j=0)
- **`P`** (*`numpy.ndarray of float type`*) – density array of pumas
- **`H`** (*`numpy.ndarray of float type`*) – density array of hares

Returns updated density ij square

Return type float

`class` `pumha.pop.`**`Population`** (*landscape_inp*, *birth*, *death*, *diffusion*, *min_ro*, *max_ro*, *dt*)

Bases: `object`

Base class for creating specific population classes.

Class stores instance attributes and provides methods which can be used by for derived subclasses. It is not intended to be used on its own, but should be extended by specific population subclasses (e.g. PumaPopulation)

Variables

- **`min_ro`** (*`float`*) – minimum density per ij square in the density array
- **`max_ro`** (*`float`*) – maximum density per ij square in the density array
- **`birth`** (*`float`*) – birth rate for a given population
- **`death`** (*`float`*) – death rate for a given population
- **`diffusion`** (*`float`*) – diffusion rate for a given population
- **`dt`** (*`float`*) – time step in arbitrary units
- **`density`** (*`numpy.ndarray containing data with float type`*) – population density in a given landscape initialized at random

`find_density_arr`(*pop_class*, *pop_list*)

Get required population density array from a list of populations.

Returns density array of a first found element matching pop_class from a list of provided populations. If no element is found, it returns matrix of zeros.

Parameters

- **`pop_class`** (*`extended Population class`*) – required population object
- **`pop_list`** (*`list`*) – list of all populations

Returns required population density array(array of zeros if not found)

Return type numpy.ndarray of float type

`load_config`(*birth*, *death*, *diffusion*, *dt*)

Set instance attributes using provided parameters.

Parameters

- **`birth`** (*`float`*) – birth rate of a given population
- **`death`** (*`float`*) – death rate of a given population
- **`diffusion`** (*`float`*) – diffusion rate of a given population
- **`dt`** (*`float`*) – timestep in arbitrary units

`random_density`(*landscape_inp*)

Assign a random density between min and max ro to every land square.

Returns a grid with a random density assigned between minimum and maximum densities for every land square.

Parameters **`landscape_inp`** (`Landscape`) – Instance of a Landscape object

Returns a 2D array of random densities

Return type numpy.ndarray of float type

`class` `pumha.pop.`**`PumaPopulation`**(*Landscape*, *birth=0.02*, *death=0.06*, *diffusion=0.2*, *min_ro=0.0*, *max_ro=5.0*, *dt=0.4*)

Bases: *`pumha.pop.Population`*

Puma population class with its specific update method.

This class represents puma population living in some landscape, therefore it requires Landscape object as a parameter. Remaining parameters are set to defaults and can be changed later by either using provided method load_config or by simply assigning required values to instance attributes.

Example Create puma population using default values

```
>>> from pumha.pop import PumaPopulation
>>> from pumha.env import Landscape
>>> land = Landscape('my_land_file.dat')
>>> puma = PumaPopulation(land)
```

Create puma population using specifig values

```
>>> from pumha.pop import PumaPopulation
>>> from pumha.env import Landscape
>>> land = Landscape('my_land_file.dat')
>>> puma = PumaPopulation(land, birth=0.03, death=0.01)
```

See Also

pumha.pop.Population

`update_density`(*populations_old*, *populations_new*)

Update density array of puma population instance.

Method updates entire density array of puma population instance based on densities of current populations living in a landscape. Only land squares in the density array are updated.

Parameters

- **`populations_old`** (*`list of Population type`*) – list of populations at current timestep
- **`populations_new`** (*`list of Population type`*) – list of populations with updated density array at t+dt

`update_density_ij`(*i*, *j*, *P*, *H*)

Return updated puma density at one (i,j) square.

Method implements discrete approximation of the following equation:

$$\frac{\partial P}{\partial t} = bHP - mP + l(\frac{\partial^2 P}{\partial x^2} + \frac{\partial^2 P}{\partial y^2})$$

where

- P = density of pumas
- H = density of hares
- b = birth rate of pumas
- m = death rate of pumas
- l = diffusion rate of pumas

Parameters

- **`i`** (*`int`*) – density array row number (first row is i=0)
- **`j`** (*`int`*) – density array column number (first column is j=0)
- **`P`** (*`numpy.ndarray of a float type`*) – density array of pumas
- **`H`** (*`numpy.ndarray of a float type`*) – density array of hares

Returns updated density i, j square

Return type float

6.1.4 pumha.sim module

Simulation module

Module contains only one class:

```
Simulation
```

and one function:

```
create_output_dir
```

The module is used to build new simulations using extended Population classes and create the output data.

To run a simulation, use run() method.

class pumha.sim.**Simulation**(**args*)

Bases: `object`

Simulate time and space evolution of populations

Only populations added to a populations list are simulated. If no populations are added, the simulation will run using density arrays of zeroes. If only one population is added but its update method requires existence of another population, the simulation will still run using zero density array for missing population.

This can be interpreted as follows:

Lets add only hare population; its update method requires puma population, if there are no pumas, since hare death rate is 0, they only increase in numbers until they rule a land!

Similarly, if there is no hares, pumas will all starve to death.

Example Create new simulation with two populations: puma, hare

Each population is of extended puma.pop.Population type

Run a population over 500 steps and save ppm output every 4th step

```
>>> from pumha.sim import Simulation
>>> sim = Simulation(puma, hare)
>>> sim.run(500, 4)
```

Variables populations (*list of pumha.pop.Population types*) – List of populations in a simulation

add_population(*pop*)

Add population object to a simulation

Parameters pop (`pumha.pop.Population`) – a population to be added to a simulation

remove_population(*pop*)

Remove population from a simulation

Parameters pop (`pumha.pop.Population`) – apopulation to be removed from a simulation

rescale_ppm_files(*max_density*)

Rescale all PPM files using common PPM color value (Maxval)

The method takes the highest recorded density from the entire simulation and uses it as common scaling factor for all PPM files. In this way the whole simulation is scaled properly.

Parameters max_density (*int, float*) – maximum density for a single square from the run

run (*num_steps*, *save_freq*)

Run a simulation over given number of steps and save an output to PPM

Instance population list is updated every second iteration. At the end of a simulation it is updated with the latest version. The method invokes save_density_grid_interface() every save_freq step in attempt to save output to a ppm file. At the end of the simulation rescale_ppm_files() method is invoked to rescale all ppm files using highest value of the density. Method also includes a simple timer for a loop which prints the total elapsed time at the end of a simulation to the standard output.

Parameters

- **num_steps** (*int*) – Number of steps for a simulation
- **save_freq** (*int*) – Number of time steps between outputs

save_average_density (*timestep*)

Calculate the average density of animals in the whole landscape

The average population is found by summing all the densities in the grid and dividing it by the numbers of squares in the grid. The density is saved to a file 'average_densities.txt', where the first column gives the timestep and second and third columns hare and puma densities at that time step respectively.

Parameters **timestep** (*int*) – timestep at which the averages are calculated.

save_density_grid (*timestep*)

Write the densities on each landscape square to a plain PPM file

The method writes the density of a population to a file in the output folder. The name of a file is in a format timestep.ppm.

The files are in a plain PPM format - each line is separated into a group of 3 values corresponding to a pixel, each value in those triplets represents either red, green or blue. The dimension of the landscape is given in the head of the file. The value after the dimensions is a scaling factor which is updated in every file in the end of the simulation, based on the highest maximum density encountered in the simulation. That is done with the rescale_ppm_files method.

Example:

```
P3
# some comment
4 4
255
0 0 255  0 0 255  0 0 255  0 0 255
0 0 255  34 56 255  28 60 255  0 0 255
0 0 255  30 50 255  30 57 225  0 0 255
0 0 255  0 0 255  0 0 255  0 0 255
```

This PPM file represents a small island surrounded by water. Since lines in a PPM file must be no longer than 70 characters, the function creates an array of strings, every string representing a pixel and then writes those strings to a file.

Parameters **timestep** (*int*) – the timestep to which the density matrix corresponds to

save_density_grid_interface (*i*)

Simple interface to save_density_grid method

Provides extendable interface to potential group of save_density_grid methods, each one to cover specific case for a simulation. This is mostly because of the limitation of the PPM file format. Currently only the case of simulation containing puma and hare populations is implemented.

Parameters **timestep** (*int*) – the timestep to which the density matrix corresponds to

update(*populations_old*, *populations_new*)
One step update for all populations in a simulation

Parameters

- **populations_old** – list of populations at time t
- **populations_new** – list of populations at time t+dt

pumha.sim.**create_output_dir**()
Create directory for output PPM and dat files

Directory is created using current date and time. All simulation output files are saved into this folder. The output directory is created in the directory where the script is running. The naming convention is as follows:

PumHa_out_%Y-%m-%d-%H-%M-%S

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX