gpmap Documentation

Release 0.1

Zach Sailer

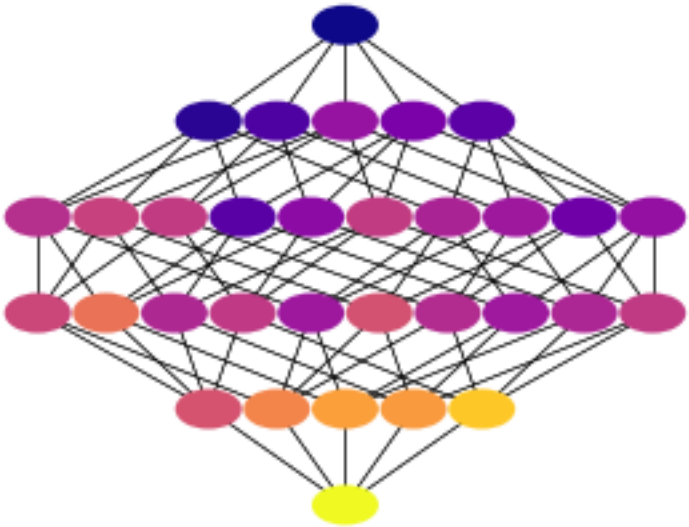
Dec 07, 2017

Contents

The **gpmap** package standardizes a data structure for genotype-phenotype (GP) maps. Subset, manipulate, extend, etc. genotype-phenotype maps easily. Calculate statistics, model evolutionary trajectories, predict phenotypes. Efficient memory usage and manipulation, using Pandas Dataframe/Series.

This package includes modules for simulating computational genotype-phenotype maps using methods described in the literature. See the Simulating page.

The GenotypePhenotypeMap object can be easily ported to network graphs (via NetworkX and GPGraph).



CHAPTER 1

GenotypePhenotypeMap

The GenotypePhenotypeMap class is main entry-point to the gpmap package. It offers intuitive and useful methods and attributes for analyzing genotype-phenotype data. We've also created a number of other packages that easily interact with the GenotypePhenotypeMap.

1.1 Example

```
from gpmap import GenotypePhenotypeMap

# Create list of genotypes and phenotypes
wildtype = "AA"
genotypes = ["AA", "AV", "AM", "VA", "VV", "VM"]
phenotypes = [1.0, 1.1, 1.4, 1.5, 2.0, 3.0]

# Create GenotypePhenotypeMap object
gpm = GenotypePhenotypeMap(wildtype, genotypes, phenotypes)
```

1.2 BinaryMap

All GenotypePhenotypeMap objects append a BinaryMap instance to a the binary attribute. The BinaryMap class creates a binary representation of all genotypes and maps them to a genotype-phenotype map. Most attributes in the GenotypePhenotypeMap also exist in the binary object, updated with the binary genotype representations.

CHAPTER 2

Simulating

The GPMap package comes with a suite of objects to simulate genotype-phenotype maps following models in the literature. They are found in the `gpmap.simulate` module.

All Simulation objects inherit the `GenotypePhenotypeMap` object as their base class. Thus, anything you can do with a GenotypePhenotypeMap, you can do with the simulation objects.

2.1 NK model

Construct a genotype-phenotype map using Kauffman's NK Model.[1] The NK fitness landscape is created using a table with binary, length-K, sub-sequences mapped to random values. All genotypes are binary with length N. The fitness of a genotype is constructed by summing the values of all sub-sequences that make up the genotype using a sliding window across the full genotypes.

For example, imagine an NK simulation with $N = 5$ and $K = 2$. To construct the fitness for the 01011 genotype, select the following sub-sequences from an NK table: "01", "10", "01", "11", "10". Sum their values together.

```
# import the NKSimulation class
from gpmap.simulate import NKSimulation

# Create an instance of the model. Using `from_length` makes this easy.
gpm = NKSimulation.from_length(6, K=3)
```

2.2 House of Cards model

Construct a 'House of Cards' fitness landscape. This is a limit of the NK model where $K = N$. It represents a fitness landscape with maximum roughness.

[1] Kauffman, Stuart A., and Edward D. Weinberger. "The NK model of rugged fitness landscapes and its application to maturation of the immune response." Journal of theoretical biology 141.2 (1989): 211-245.

```
# import the HouseOfCardsSimulation class
from gpmap.simulate import HouseOfCardsSimulation

# Create an instance of the model. Using `from_length` makes this easy.
gpm = HouseOfCardsSimulation.from_length(6)
```

2.3 Mount Fuji model

Construct a genotype-phenotype map from a Mount Fuji model.[2]

A Mount Fuji sets a "global" fitness peak (max) on a single genotype in the space. The fitness goes down as a function of hamming distance away from this genotype, called a "fitness field". The strength (or scale) of this field is linear and depends on the parameters *field_strength*.

Roughness can be added to the Mount Fuji model using a random *roughness* parameter. This assigns a random roughness value to each genotype.

$$f(g) = \nu(g) - c \cdot d(g_0, g)$$

where ν is the roughness parameter, c is the field strength, and d is the hamming distance between genotype g and the reference genotype.

```
# import the HouseOfCardsSimulation class
from gpmap.simulate import MountFujiSimulation

# Create an instance of the model. Using `from_length` makes this easy.
gpm = MountFujiSimulation.from_length(6)

# add roughness, sampling from a range of values.
gpm.set_roughness(range=(-1,1))
```

2.4 References

[2] Szendro, Ivan G., et al. "Quantitative analyses of empirical fitness landscapes." Journal of Statistical Mechanics: Theory and Experiment 2013.01 (2013): P01005.

CHAPTER 3

Read/Write

The `GenotypePhenotypeMap` object is really just a container of Pandas Series that can be easily read/written as a DataFrame. Any tabular format (i.e. Excel files, csv, tsv, ...) can be loaded directly into the object. It requires two columns for genotypes and phenotypes, and optionally takes stdeviations and n_replicates as input.

3.1 read_excel

Excel files are supported through the `read_excel` method. This method requires *genotypes* and *phenotypes* columns, and can include *n_replicates* and *stdeviations* as optional columns. All other columns are ignored.

Example: Excel spreadsheet file ("data.xlsx")

Read the spreadsheet directly into the GenotypePhenotypeMap.

```
from gpmap import GenotypePhenotypeMap

gpm = GenotypePhenotypeMap.read_excel(wildtype="PTEE", filename="data.xlsx")
```

3.2 read_csv

CSV files are supported through the `read_excel` method. This method requires *genotypes* and *phenotypes* columns, and can include *n_replicates* and *stdeviations* as optional columns. All other columns are ignored.

Example: CSV File

Read the csv directly into the GenotypePhenotypeMap.

```
from gpmap import GenotypePhenotypeMap

gpm = GenotypePhenotypeMap.read_csv(wildtype="PTEE", filename="data.csv")
```

3.3 read_json

The only keys recognized by the json reader are:

1. *genotypes*
2. *phenotypes*
3. *stdeviations*
4. *mutations*
5. *n_replicates*
6. *log_transform*

All other keys are ignored in the epistasis models. You can keep other metadata stored in the JSON, but it won't be appended to the epistasis model object.

```
{
    "genotypes" : [
        '000',
        '001',
        '010',
        '011',
        '100',
        '101',
        '110',
        '111'
    ],
    "phenotypes" : [
        0.62344582,
        0.87943151,
        -0.11075798,
        -0.59754471,
        1.4314798,
        1.12551439,
        1.04859722,
        -0.27145593
    ],
    "stdeviations" : [
        0.01,
        0.01,
        0.01,
        0.01,
        0.01,
        0.01,
        0.01,
        0.01,
    ],
    "mutations" : {
        0 : ["0", "1"],
        1 : ["0", "1"],
        2 : ["0", "1"],
    }
    "n_replicates" : 12,
    "log_transform" : false,
    "title" : "my data",
    "description" : "a really hard experiment"
}
```

CHAPTER 4

API Documentation

The `GenotypePhenotypeMap` is the main entry point to the gpmap package. Load in your data using the `read` methods attached to this object. The following subpackages include various objects to analyze this object.

4.1 Subpackages

4.1.1 gpmap.binary module

4.1.2 gpmap.errors module

4.1.3 gpmap.mapping module

4.1.4 gpmap.sample module

4.1.5 gpmap.stats module

4.1.6 gpmap.utils module

4.1.7 gpmap.simulate

gpmap.simulate.base module

gpmap.simulate.fuji module

gpmap.simulate.hoc module

gpmap.simulate.nk module

Module contents

4.2 GenotypePhenotypeMap

CHAPTER 5

Indices and tables

- genindex
- modindex
- search