User manual for pyvolve v1.0

Stephanie J. Spielman

Contents

1 Introduction

Pyvolve (pronouced "pie-volve") is an open-source python module for simulating genetic data along a phylogeny according to Markov models of sequence evolution, according to standard methods [13]. The module is available for download on github (and see here for API documentation). Note that pyvolve has several dependencies, including BioPython, NumPy, and SciPy. These modules must be properly installed and in your python path for pyvolve to work properly. Please file any and all bug reports on the github repository Issues section.

Pyvolve is written such that it can be seemlessly integrated into your python pipelines without having to interface with external software platforms. However, please note that for extremely large (e.g. >1000 taxa) and/or extremely heterogenous simulations (e.g. where each site evolves according to a unique evolutionary model), pyvolve may be quite slow and thus may take several minutes to run. Faster sequence simulators you may find useful include (but are certainly not limited to!) Indelible [1] and indel-Seq-Gen [10].

Pyvolve supports a variety of evolutionary models, including the following:

- Nucleotide Models
  - Generalized time-reversible model [11] and all nested variants
- Amino-acid exchangeability models

  - JTT [5], WAG [12], and LG [8]
- Codon models
  - Mechanistic (dN/dS) models (MG-style [9] and GY-style [2])
  - Empirical codon model [7]
- Mutation-selection models
  - Halpern-Bruno model [3], implemented for codons and nucleotides

Note that it is also possible to specify custom matrices (detailed in section 7 below). Both site-wise and temporal (branch) heterogeneity are supported. A detailed and highly-recommended overview of Markov process evolutionary models, for DNA, protein, and codons, is available in the book *Computational Molecular Evolution*, by Ziheng Yang [13].

2 Basic Usage

Similar to other simulation platforms, pyvolve evolves sequences in groups of **partitions**. Each partition has an associated size and model (or set of models, if branch heterogeneity is desired). All partitions will evolve according to the same phylogeny; if you wish to have each partition evolve according to a distinct phylogeny, I recommend performing several simulations and then merging the resulting alignments in the post-processing stage.

The general framework for a simple simulation is given below. In order to simulate sequences, you must define the phylogeny along which sequences evolve as well as any evolutionary model(s) you'd like to use. Each evolutionary model will requires its own particular parameters, as described in sections) require different parameters

```
######### General pyvolve framework #########
#############################################

# Import the pyvolve module
import pyvolve

# Read in phylogeny along which pyvolve should simulate
my_tree = pyvolve.read_tree(file = "file_with_tree_for_simulating.tre")

# Define and construct evolutionary models
my_model = pyvolve.Model(<model_type>, <custom_model_parameters>)
my_model.construct_model()

# Define partitions
my_partition = pyvolve.Partition(models = my_model, size = 100)

# Evolve partitions with the callable Evolver() class
pyvolve.Evolver(tree = my_tree, partitions = my_partition)()
```

By default, sequences will be output to a fasta-formatted file called "simulated_alignment.fasta". Two additional tab-delimited files, called "site_rates.txt" and "site_rates_info.txt" are also output. These files provide useful information when heterogeneity (either site or branch) is implemented. The former file indicates to which partition and rate category (if no rate heterogeneity specified, these values will all be 1) each site belongs, and the latter file provides more specific information about each rate category, in particular its associated partition, probability, and value.

To change the output file names for any of those files, provide the arguments **`ratefile`** ("site_rates.txt"), **`infofile`** ("site_rates_info.txt"), and/or **`seqfile`** ("simulated_alignment.fasta") when initializing an **`Evolver`** instance:

```
# Provide custom file names when initializing the Evolver instance
myevolver = pyvolve.Evolver(tree = my_tree, partitions = my_partition, ratefile = "custom_ratefile.txt",
     infofile = "custom_infofile.txt", seqfile = "custom_seqfile.fasta" )
myevolver() #evolve
```

To suppress the creation of any of these files, define the argument(s) as either **`None`** or **`False`**:

```
# Only output a sequence file (suppress the ratefile and infofile)
myevolver = pyvolve.Evolver(tree = my_tree, partitions = my_partition, ratefile = None, infofile = None)
myevolver() #evolve
```

The sequence file format can also be changed with the argument **`seqfmt`**:

```
# Save the sequence file as seqs.phy, in phylip format
myevolver = pyvolve.Evolver(tree = my_tree, partitions = my_partition, seqfile = "seqs.phy", seqfmt = "phylip")
myevolver() #evolve
```

pyvolve uses Biopython to write sequence files, so consult the Biopython AlignIO module documentation (or this nice wiki) for available formats.

<model_type> is the type of model matrix. <custom_model_parameters> is a *dictionary* of parameters for your chosen model. See below for available model types and associated parameter keys.

2.1 Nucleotide Models

Rate matrices for nucleotide models are given by,

$$q_{ij} = \mu_{ij}\pi_j \tag{1}$$

where μ_{ij} describes the rate of change from nucleotide i to j, and π_j represents the equilibrium frequency of the target nucleotide j. Note that mutation rates are symmetric, e.g. $\mu_{ij} = \mu_{ji}$.

By default, nucleotide pyvolve models use equal mutation rates and equal equilibrium frequencies (corresponding to the Jukes-Cantor model [6]). A basic model can be constructed with,

```
# Simple nucleotide model
nuc_model = pyvolve.Model("nucleotide")
nuc_model.construct()
```

To customize a nucleotide model, include the second Model() argument **`params`** dictionary with optional keys **`"mu"`** for custom mutation rates and **`"state_freqs"`** for custom equilibrium frequencies.

```
# Define mutation rates in a dictionary with keys giving the nucleotide pair
# Below, the rate from A to C is 0.5, and similarly C to A is 0.5
custom_mu = {'AC':0.5, 'AG':0.25, 'AT':1.23, 'CG':0.55, 'CT':1.22, 'GT':0.47}

# Define custom frequencies, in order A C G T. This can be a list or numpy array.
freqs = [0.1, 0.45, 0.3, 0.15]

# Construct nucleotide model with custom mutation rates and frequencies.
nuc_model = pyvolve.Model( "nucleotide", {'mu':custom_mu, 'state_freqs':freqs} )
nuc_model.construct()
```

Note that any undefined mutation rates will be set to 1. Further, mutation rates are symmetric; if you provide a rate for $A \rightarrow T$, it will automatically be applied as the rate $T \rightarrow A$.

As an alternate to **"mu"**, you can provide the key **"kappa"**, which corresponds to the transition:transversion ratio (e.g. for an HKY85 model [4]), in the **"params"** dictionary. When specifying **"kappa"**, tranversion rates will be set to 1, and transition rates will be set to the provided value.

```
# Construct nucleotide model with transition-to-transversion bias, and default frequencies
nuc_model = pyvolve.Model( "nucleotide", {"kappa":2.75, "state_freqs":freqs} )
nuc_model.construct()
```

2.2 Amino-acid models

Amino-acid exchangeability models are given by,

$$q_{ij} = r_{ij}\pi_j \tag{2}$$

where r_{ij} is a symmetric exchangeability matrix which describes the probability of changing from amino acid i to j, and π_j is the equilibrium frequency of the target amino acid j. The r_{ij} matrix corresponds to an empirically determined model, such as WAG [12] or LG [8].

A basic model can be constructed with,

```
# Simple amino-acid model
aa_model = pyvolve.Model("WAG") # Here, WAG can be one of JTT, WAG, LG (case-insensitive)
aa_model.construct()
```

By default, pyvolve assign equal equilibrium frequencies. To customize an amino-acid model, include the second Model() argument **params** dictionary with the optional key **'state_freqs'** for custom equilibrium frequencies (see 5 for details). Note that amino-acid frequencies must be in the order A, C, D, E, ... Y. Further, to specify the *model's* default equilibrium frequencies, use the pyvolve EmpiricalModelFrequencies class:

```
# Define default wag state frequencies
f = EmpiricalModelFrequencies("WAG") # model name is case-insensitive
freqs = f.construct_frequencies()

# Construct amino-acid model with custom frequencies
aa_model = pyvolve.Model( "WAG", {"state_freqs":freqs} )
aa_model.construct()
```

2.3 Mechanistic codon models

GY-style matrix elements, for a the substitution from codon i to j, are generally given by

$$q_{ij} = \begin{cases} \mu_{o_i t_j}\pi_j\alpha & \text{synonymous change} \\ \mu_{o_i t_j}\pi_j\beta & \text{nonsynonymous change} \\ 0 & \text{multiple nucleotide changes} \end{cases}, \tag{3}$$

MG-style matrix elements, for a the substitution from codon i to j, are generally given by

$$q_{ij} = \begin{cases} \mu_{o_i t_j}\pi_{t_j}\alpha & \text{synonymous change} \\ \mu_{o_i t_j}\pi_{t_j}\beta & \text{nonsynonymous change} \\ 0 & \text{multiple nucleotide changes} \end{cases}, \tag{4}$$

Mutation parameters and either omega or alpha/beta.

2.4 Empirical codon models

$$q_{ij} = \begin{cases} s^*_{ij}\pi_j\kappa(i,j)\alpha & \text{synonymous change} \\ s^*_{ij}\pi_j\kappa(i,j)\beta & \text{nonsynonymous change} \end{cases}, \tag{5}$$

Further, rest and unrest. The $\kappa(i,j)$ parameter is defined in their paper...

2.5 Mutation-selection models

$$q_{ij} = \begin{cases} \mu_{ij}\frac{S_{ij}}{1-1/S_{ij}} & \text{single nucleotide change} \\ 0 & \text{multiple nucleotide changes} \end{cases}, \tag{6}$$

where S_{ij} is the scaled selection coefficient. These models may be defined in 1 of 2 ways: with state frequencies or with fitness values. State frequencies should be specified with the key 'state_freqs' or with scaled fitness values ($F = 2Nf$) with the key 'fitness'. Either way you'll need mutation parameters. Your fitness vector should be a list of 4 (for nucleotide models) or 20/61 for codon models. If you give 20, they are assigned exactly to codons (such that all synonymous codons will have the same fitness). pyvolve will know if you want this in nucleotide or codon space, you just need to say MutSel when defining Model()

3 Site-wise heterogeneity

3.1 Nucleotide and amino-acid models

3.2 Codon models

3.3 Mutation-selection models

4 Temporal heterogeneity

5 Building a vector of stationary frequencies

6 Matrix scaling options

7 Using custom rate matrices

For this, I'll need to have a matrix argument in model definition, and some sanity checking on a provided matrix will have to be done.

This is my first python example:

```
from pyvolve import *

# Read in a newick tree
t = read_tree(file = "myfile.tre")

# Construct state frequency vector. Optional!
f = EqualFrequencies("amino")
freqs = f.construct_frequencies(type = "codon")

# Build the evolutionary model
m = Model("GY94", {'state_freqs':freqs, 'omega':1.5, kappa:3.4}
m.construct_model()

# Initialize partitions
p = Partition(models = m, size = 100)

# Evolve, and call.
Evolver(partitions = p, tree = t, seqfile = "sequences.phy", seqfmt = "phylip")()
```

References

[1] W Fletcher and Z Yang. INDELible: A Flexible Simulator of Biological Sequence Evolution. *Mol Biol Evol*, 26(8):1879–1888, 2009.

[2] N Goldman and Z Yang. A codon-based model of nucleotide substitution for protein-coding DNA sequences. *Mol Biol Evol*, 11:725–736, 1994.

[3] AL Halpern and WJ Bruno. Evolutionary distances for protein-coding sequences: modeling site-specific residue frequencies. *Mol Biol Evol*, 15:910–917, 1998.

[4] M Hasegawa, H Kishino, and T Yano. Dating of human-ape splitting by a molecular clock of mitochondrial DNA. *J Mol Evol*, 22(2):160–174, 1985.

[5] DT Jones, WR Taylor, and JM Thornton. The rapid generation of mutation data matrices from protein sequences. *CABIOS*, 8:275–282, 1992.

[6] TH Jukes and CR Cantor. Evolution of protein molecules. In HN Munro, editor, *Mammalian protein metabolism*. Academic Press, New York, 1969.

[7] C. Kosiol, I. Holmes, and N. Goldman. An empirical codon model for protein sequence evolution. *Mol Biol Evol*, 24:1464 – 1479, 2007.

[8] SQ Le and O Gascuel. An improved general amino acid replacement matrix. *Mol Biol Evol*, 25:1307–1320, 2008.

[9] SV Muse and BS Gaut. A likelihood approach for comparing synonymous and nonsynonymous nucleotide substitution rates, with application to the chloroplast genome. *Mol Biol Evol*, 11:715–724, 1994.

[10] Cory L Strope, Stephen D Scott, and Etsuko N Moriyama. indel-Seq-Gen: a new protein family simulator incorporating domains, motifs, and indels. *Mol Biol Evol*, 24(3):640–649, 2007.

[11] S Tavare. Lines of descent and genealogical processes, and their applications in population genetics models. *Theor Popul Biol*, 26:119–164, 1984.

[12] Simon Whelan and Nick Goldman. A general empirical model of protein evolution derived from multiple protein families using a maximum likelihood approach. *Mol Biol Evol*, 18:691–699, 2001.

[13] Z. Yang. *Computational Molecular Evolution*. Oxford University Press, 2006.