Pylogeny

API Documentation

October 16, 2014

Contents

1 Package Pylogeny

Pylogeny is a Python library and code framework for phylogenetic tree reconstruction and scoring.

Allows one to perform the following tasks: (1) Generate and manage phylogenetic tree landscapes. (2) Build and rearrange phylogenetic trees using preset operators such as NNI, SPR, and TBR. (3) Score phylogenetic trees by Log-likelihood and Parsimony.

Dependencies: Pandas, P4 Phylogenetic Library. Suggested: FastTree, RAxML, PytBEAGLEhon.

1.1 Modules

- **alignment**: Handle input biological sequence alignment files for the purposes of phylogenetic inference.
  (Section 2, p. 3)
- **bipartition**: Using the term borrowed from nomenclature of a bipartite graph, a bipartition for a phylogenetic tree coincides with the definition of two disjoint sets U and V .
  (Section 3, p. 7)
- **fasttree**: Python interface for the FastTree executable.
  (Section 4, p. 10)
- **landscape**: Encapsulate a phylogenetic tree space.
  (Section 5, p. 11)
- **landscapeWriter**: Serialize a phylogenetic landscape into a Python-readable file (Pickle).
  (Section 6, p. 20)
- **model**: Phylogenetic tree scoring models; intended to be coupled with the use of pytbeaglehon (BEAGLE) high-performance library.
  (Section 7, p. 22)
- **newick**: Newick string parsing and object interaction.
  (Section 8, p. 25)
- **parsimony**: Toolkit for performance of Parsimonious Criterion (Parsimony) methods of optimization of a phylogenetic topology with a particular set of data.
  (Section 9, p. 32)
- **pll**: C Extension to wrap libpll library.
  (Section 10, p. 34)
- **raxml**: Python interface for RAxML executable.
  (Section 11, p. 36)
- **rearrangement**: Phylogenetic tree structure encapsulation; allow rearrangement of said structure.
  (Section 12, p. 37)
- **scoring**: Phylogenetic tree scoring.
  (Section 13, p. 43)

2 Module Pylogeny.alignment

Handle input biological sequence alignment files for the purposes of phylogenetic inference. Will read all types of alignment files by utilizing the P4 python phylogenetic library.

2.1 Variables

Name	Description
__package__	**Value:** `'Pylogeny'`

2.2 Class alignment

```
object ─┐
        │
  Pylogeny.alignment.alignment
```

Known Subclasses: Pylogeny.alignment.phylipFriendlyAlignment

Wrap a biological sequence alignment to enable functionality necessary for phylogenetic inference. Makes use of temporary files and therefore requires to be closed once no longer needed.

2.2.1 Methods

__init__(*self*, *inal*)

Instantiate an object intended to wrap an alignment for the purposes of running phylogenetic inference.

Overrides: object.__init__

__getitem__(*self*, *i*)

__str__(*self*)

str(x)

Overrides: object.__str__ extit(inherited documentation)

__len__(*self*)

close(*self*)

Delete FASTA temporary file, other temporary files.

toStrList(*self*)

Get all sequences as a list of strings.

getStateModel(*self*)

getSize(*self*)

Return the size of the alignment, or how many characters there are in each respective item in the alignment.

getNumSeqs(*self*)

Return the number of sequences that are present in the sequence alignment.

getDim(*self*)

Return the dimensionality of the sequence alignment (how many different types of characters).

getSequence(*self*, *i*)

Acquire the ith sequence.

getFASTA(*self*)

Get (and create if not already) a path to a temporary FASTA file. This will be deleted upon closure of the alignment instance.

getApproxMLNewick(*self*)

Get a tree in newick format via use of FastTree that serves as an approximation of the maximum likelihood tree for this data.

getApproxMLTree(*self*)

Get a tree object for an approximation of the maximum likelihood tree for this data using FastTree.

getTaxa(*self*)

Return taxa names.

getAlignment(*self*)

Acquire the alignment data structure (P4 module).

bootstrap(*self*)

Perform bootstrapping on the alignment data.

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __subclasshook__()

2.2.2 Properties

Name	Description
Inherited from object	
___class___	

2.3 Class phylipFriendlyAlignment

```
object ─┐
        │
Pylogeny.alignment.alignment ─┐
                              │
                              Pylogeny.alignment.phylipFriendlyAlignment
```

An alignment object that renames all comprising taxa in order to be able to be written as a Phylip file.

2.3.1 Methods

___init___(*self*, *inal*)

Instantiate an object intended to wrap an alignment for the purposes of running phylogenetic inference.

Overrides: object.___init___ extit(inherited documentation)

getPhylip(*self*)

Get a path to a temporary Phylip file. This will be deleted upon closure of the alignment instance.

writeProperNexus(*self*, *wri*)

Copy the temporary Nexus file with proper names.

reassignFromReinterpretedNewick(*self*, *tr*)

Replace all proper names with reassigned names in a Newick tree.

reinterpretNewick(*self*, *tr*)

Replaces all reassigned names to proper names in a Newick tree.

getProperName(*self*, *n*)

Return the actual name for an integer-based sequence name that was reassigned at initialization.

getTaxa(*self*)

Return current taxa names in the alignment.

Overrides: Pylogeny.alignment.alignment.getTaxa

recreateObject(*self*)

Reintializes the object.

Inherited from Pylogeny.alignment.alignment(Section 2.2)

___getitem___(), ___len___(), ___str___(), bootstrap(), close(), getAlignment(), getApproxMLNewick(), getApproxMLTree(), getDim(), getFASTA(), getNumSeqs(), getSequence(), getSize(), getStateModel(), toStrList()

Inherited from object

___delattr___(), ___format___(), ___getattribute___(), ___hash___(), ___new___(), ___reduce___(), ___reduce_ex___(), ___repr___(), ___setattr___(), ___sizeof___(), ___subclasshook___()

2.3.2 Properties

Name	Description
Inherited from object	
___class___	

3 Module Pylogeny.bipartition

Using the term borrowed from nomenclature of a bipartite graph, a bipartition for a phylogenetic tree coincides with the definition of two disjoint sets U and V . A branch in a phylogenetic tree defines a single bipartition that divides the tree into two disjoint sets U and V . The set U comprises all of the children leaf of the subtree associated with that branch. The set V contains the rest of the leaves or taxa in the tree. This package handle operations involving the representation of tree bipartitions.

3.1 Variables

Name	Description
__package__	**Value:** `'Pylogeny'`

3.2 Class bipartition

object
  └ **Pylogeny.bipartition.bipartition**

3.2.1 Methods

__init__(*self*, *topol*, *bra*=`None`)

x.__init__(...) initializes x; see help(type(x)) for signature

Overrides: object.__init__ extit(inherited documentation)

__hash__(*self*)

hash(x)

Overrides: object.__hash__ extit(inherited documentation)

__eq__(*self*, *o*)

__ne__(*self*, *o*)

fromStringRepresentation(*self*, *st*)

Acquire all component elements from a string representation of a bipartition.

getBranch(*self*)

Get branch corresponding to this bipartition.

getStringRepresentation(*self*)

Get the string representation corresponding to this bipartition.

getShortStringRepresentation(*self*)

Get the shorter string representation corresponding to this bipartition.

getShortStringMappings(*self*)

Get the mapping of symbols from taxa names for the shorter string representation.

getBranchListRepresentation(*self*)

Get the tuple of lists of branches that represent this bipartition.

getSPRRearrangements(*self*)

Return the set of all scores related to this bipartition.

getSPRScores(*self*, *ls*, *node*=`None`)

Given a landscape, return all possible scores, not actively performing scoring if not done.

getMedianSPRScore(*self*, *ls*, *node*=`None`)

Given a landscape, return the median SPR score.

getBestSPRScore(*self*, *ls*, *node*=`None`)

Given a landscape, return the best SPR score.

Inherited from object

___delattr___(), ___format___(), ___getattribute___(), ___new___(), ___reduce___(), ___reduce_ex___(), ___repr___(), ___setattr___(), ___sizeof___(), ___str___(), ___subclasshook___()

3.2.2 Properties

Name	**Description**
Inherited from object __class__	

4 Module Pylogeny.fasttree

Python interface for the FastTree executable. See http://www.microbesonline.org/fasttree/ for more information on FastTree. Requires FastTree to be installed.

4.1 Functions

exeExists(*cmd*)

4.2 Variables

Name	Description
FT_EXECUTABLE	**Value:** `'fasttree'`
__package__	**Value:** `'Pylogeny'`

4.3 Class fasttree

Denotes a single run of the FastTree executable.

4.3.1 Methods

__init__(*self*, *inp_align*, *out_file*=`None`, *isProtein*=`True`)

getInstructionString(*self*)

run(*self*)

Perform a run of the FastTree executable in order to acquire an approximate maximum likelihood tree for the input alignment.

5 Module Pylogeny.landscape

Encapsulate a phylogenetic tree space. A phylogenetic landscape or tree space refers to the entire combinatorial space comprising all possible phylogenetic tree topologies for a set of n taxa. The landscape of n taxa can be defined as consisting of a finite set T of tree topologies. Tree topologies can be associated with a fitness function f (t i) describing their fit. This forms a discrete solution search space and finite graph (T, E) = G. E(G) refers to the neighborhood relation on T(G). Edges in this graph are bidirectional and represent transformation from one tree topology to another by a tree rearrangement operator. An edge between t i and t__j would be notated as e__ij in E(G).

5.1 Variables

Name	Description
___package___	**Value:** `'Pylogeny'`

5.2 Class graph

object
└─ **Pylogeny.landscape.graph**

Known Subclasses: Pylogeny.landscape.landscape

Define an empty graph object.

5.2.1 Methods

___init___(*self*, *gr*=`None`)

x.___init___(...) initializes x; see help(type(x)) for signature

Overrides: object.___init___ extit(inherited documentation)

___len___(*self*)

getSize(*self*)

getNodeNames(*self*)

getNodes(*self*)

getEdges(*self*)

getEdgesFor(*self*, *i*)

getNode(*self*, *i*)

getEdge(*self*, *i*, *j*)

__iter__(*self*)

getNeighborsFor(*self*, *i*)

getDegreeFor(*self*, *i*)

setDefaultWeight(*self*, *w*)

clearEdgeWeights(*self*)

getNumComponents(*self*)

Get the number of components of the graph.

getComponents(*self*)

Get the connected components in the graph.

getComponentOfNode(*self*, *i*)

Get the graph component of a given node.

getCliques(*self*)

Get the cliques present in the graph.

getCliqueNumber(*self*)

Get the clique number of the graph.

getNumCliques(*self*)

Get the number of cliques found in the graph.

getCliquesOfNode(*self*, *i*)

Get the clique that a node corresponds to.

getCenter(*self*)

Get the centre of the graph.

getDiameter(*self*)

Acquire the diameter of the graph.

getMST(*self*)

Acquire the minimum spanning tree for the graph.

hasPath(*self*, *nodA*, *nodB*)

See if a path exists between two nodes.

getShortestPath(*self*, *nodA*, *nodB*)

Get the shortest path between two nodes.

getShortestPathLength(*self*, *nodA*, *nodB*)

Get the shortest path length between two nodes.

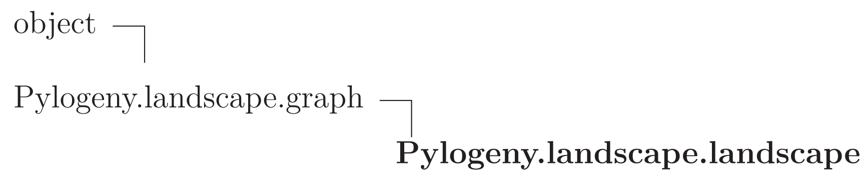
Inherited from object

___delattr___(), ___format___(), ___getattribute___(), ___hash___(), ___new___(), ___reduce___(), ___reduce_ex___(), ___repr___(), ___setattr___(), ___sizeof___(), ___str___(), ___subclasshook___()

5.2.2 Properties

Name	Description
Inherited from object ___class___	

5.3 Class landscape



Defines an entire phylogenetic tree space.

5.3.1 Methods

__init__(*self*, *ali*, *starting_tree*=`None`, *root*=`True`, *operator*=`'SPR'`)

Initialize the landscape.

Overrides: object.__init__

getAlignment(*self*)

getNumberTaxa(*self*)

getPossibleNumberTrees(*self*)

getRootTree(*self*)

setAlignment(*self*, *ali*)

Set the alignment present in this landscape. WARNING; will not modify existing scores.

getTree(*self*, *i*)

Get the tree object for a tree by its ID.

getVertex(*self*, *i*)

Acquire a vertex object from the landscape; this is a high-level representation of a tree in the landscape with additional functionality. Object created upon invocation of this function.

removeTree(*self*, *tree*)

Remove a tree by object.

addTree(*self*, *tree*)

Add a tree to the landscape.

addTreeByNewick(*self*, *newick*)

Add a tree to the landscape by Newick string.

exploreRandomTree(*self*, *i*)

Get only a single neighbor to a tree in the landscape.

exploreTree(*self*, *i*)

Get all neighbors to a tree in the landscape.

getLocks(*self*)

toggleLock(*self*, *lock*)

Add a biparition to the list of locked bipartitions if not present; otherwise, remove it. Return status of lock.

lockBranchFoundInTree(*self*, *tr*, *br*)

Given a tree node and a branch object, add a given bipartition to the bipartition lock list. Returns true if locked.

getBipartitionFoundInTreeByIndex(*self*, *tr*, *brind*, *topol*=`None`)

Given a tree node and a branch index, return the associated bipartition.

lockBranchFoundInTreeByIndex(*self*, *tr*, *brind*)

Given a tree node and a branch index, add a given bipartition to the bipartition lock list. Returns true if locked.

isViolating(*self*, *i*)

Determine if a tree is violating any locks intrinsic to the landscape.

findTree(*self*, *newick*)

Find a tree by Newick string, taking into account branch lengths.

findTreeTopology(*self*, *newick*)

Find a tree by topology, not taking into account branch lengths.

findTreeTopologyByStructure(*self*, *struct*)

Find a tree by topology, not taking into account branch lengths, given the topology.

getBestImprovement(*self*, *i*)

For a tree in the landscape, investigate neighbors to find a tree that leads to the best improvement of fitness function score on the basis of likelihood.

getPathOfBestImprovement(*self*, *i*)

For a tree in the landscape, investigate neighbors iteratively until a best path of score improvement is found on basis of likelihood.

getAllPathsOfBestImprovement(*self*)

Return all paths of best improvement as a dictionary.

iterAllPathsOfBestImprovement(*self*)

Return an iterator for all paths of best improvement.

isLocalOptimum(*self*, *i*)

Determine if a tree is, without any doubt, a local optimum.

getLocalOptima(*self*)

Get all trees in the landscape that can be labelled as a local optimum.

getGlobalOptimum(*self*)

Get the global optimum of the current space.

toTreeFile(*self*, *fout*, *proper*=`True`)

Output this landscape as a series of trees, separated by newlines, as a text file saved at the given path.

Inherited from Pylogeny.landscape.graph(Section 5.2)

___iter___(), ___len___(), clearEdgeWeights(), getCenter(), getCliqueNumber(), getCliques(), getCliquesOfNode(), getComponentOfNode(), getComponents(), getDegreeFor(), getDiameter(), getEdge(), getEdges(), getEdgesFor(), getMST(), getNeighborsFor(), getNode(), getNodeNames(), getNodes(), getNumCliques(), getNumComponents(), getShortestPath(), getShortestPathLength(), getSize(), hasPath(), setDefaultWeight()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

5.3.2 Properties

Name	Description
Inherited from object	
__class__	

5.4 Class vertex

object
  Pylogeny.landscape.vertex

Encapsulate a single vertex in the landscape and add convenient functionality to alias parent landscape functions.

5.4.1 Methods

__init__(*self*, *i*, *obj*, *ls*)

x.__init__(...) initializes x; see help(type(x)) for signature

Overrides: object.__init__ extit(inherited documentation)

getIndex(*self*)

getDict(*self*)

getObject(*self*)

getTree(*self*)

getNewick(*self*)

getScore(*self*)

getOrigin(*self*)

getNeighbors(*self*)

getDegree(*self*)

isLocalOptimum(*self*)

isExplored(*self*)

isFailed(*self*)

approximatePossibleNumNeighbors(*self*)

Approximate the possible number of neighbors to this vertex by considering the type of tree rearrangement operator.

scoreLikelihood(*self*)

getBestImprovement(*self*)

Alias function for function of same name in parent landscape.

getPathOfBestImprovement(*self*)

Alias function for function of same name in parent landscape.

isBestImprovement(*self*)

Check to see if this vertex is a best move for another node.

isViolating(*self*)

Alias function for function of same name in parent landscape.

getProperNewick(*self*)

Get the proper Newick string for a tree.

getBipartitions(*self*)

Get all bipartitions for this vertex.

getBipartitionScores(*self*)

Get all corresponding bipartition vectors of SPR scores.

getNeighborsOfBipartition(*self*, *bi*)

Get corresponding neighbors of a bipartition in this vertex's tree.

getNeighborsOfBranch(*self*, *br*)

Get corresponding neighbors of a branch in this vertex's tree.

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

5.4.2 Properties

Name	Description
Inherited from object	
__class__	

6 Module Pylogeny.landscapeWriter

Serialize a phylogenetic landscape into a Python-readable file (Pickle).

6.1 Variables

Name	Description
__package__	**Value:** `'Pylogeny'`

6.2 Class landscapeWriter

object ─┐
Pylogeny.landscapeWriter.landscapeWriter

Encapsulate the writing of a landscape to a file format.

6.2.1 Methods

__init__(*self*, *landscape*, *name*)

x.__init__(...) initializes x; see help(type(x)) for signature

Overrides: object.__init__ extit(inherited documentation)

writeFile(*self*, *path*=`'.'`)

Write the landscape serialized file to given path.

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

6.2.2 Properties

Name	Description
Inherited from object	
__class__	

6.3 Class landscapeParser

Encapsulates the construction of a landscape object from a pickle file.

6.3.1 Methods

__init__(*self*, *path*)

parse(*self*)

Parse the file.

7 Module Pylogeny.model

Phylogenetic tree scoring models; intended to be coupled with the use of pytbeaglehon (BEAGLE) high-performance library.

7.1 Variables

Name	Description
pytbeaglehonEnabled	**Value:** `True`
__package__	**Value:** `'Pylogeny'`

7.2 Class PhyloModelError

object
exceptions.BaseException
exceptions.Exception
Pylogeny.model.PhyloModelError

7.2.1 Methods

__init__(*self*, *v*)

x.__init__(...) initializes x; see help(type(x)) for signature

Overrides: object.__init__ extit(inherited documentation)

__str__(*self*)

str(x)

Overrides: object.__str__ extit(inherited documentation)

Inherited from exceptions.Exception

__new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __unicode__()

Inherited from object

___format___(), ___hash___(), ___reduce_ex___(), ___sizeof___(), ___subclasshook___()

7.2.2 Properties

Name	Description
Inherited from exceptions.BaseException args, message	
Inherited from object ___class___	

7.3 Class DiscreteStateModel

```
object ─┐
        Pylogeny.model.DiscreteStateModel
```

Initialize a discrete state model for phylogenetic data. State frequencies and character time are determined from the given alignment object.

7.3.1 Methods

___init___(*self*, *alignment*)

x.___init___(...) initializes x; see help(type(x)) for signature

Overrides: object.___init___ extit(inherited documentation)

getAlignment(*self*)

getAlignmentAsStateList(*self*)

getSequenceMatrix(*self*)

getCharType(*self*)

getStateFreqs(*self*)

getRawStateFreqs(*self*)

getRawStateFreqsAsList(*self*)

getRawStateFreqsAsDict(*self*)

getFrequencyOfState(*self*, *i*)

getRawFrequencyOfState(*self*, *i*)

Inherited from object

___delattr___(), ___format___(), ___getattribute___(), ___hash___(), ___new___(), ___reduce___(), ___reduce_ex___(), ___repr___(), ___setattr___(), ___sizeof___(), ___str___(), ___subclasshook___()

7.3.2 Properties

Name	Description
Inherited from object	
___class___	

8 Module Pylogeny.newick

Newick string parsing and object interaction. A Newick string can represent a phylogenetic tree.

8.1 Functions

numberUnrootedTrees(*t*)

assignParents(*top*)

Should be a one-time use function. Goes through and assigns parents to the parsed newick tree structure nodes and branches to allow for up-traversal.

removeBranchLengths(*top*)

Goes through and removes any stored branch lengths.

removeUnaryInternalNodes(*top*)

Goes through and ensures any degree-2 internal nodes are smoothed into a single degree-3 internal node.

invertAlongPathToNode(*target*, *top*)

DANGEROUS: Reverses all directionality to a given node from a top-level node. Intended as a low-level function for rerooting a tree.

isLeaf(*n*)

Given a node, see if a leaf.

isInternalNode(*n*)

Given a node, see if is an internal node.

shuffleLeaves(*top*)

DANGEROUS: Given a top-level node, shuffle all leaves in this tree.

getAllLeaves(*top*)

Given a top-level node, find all leaves.

getAllInternalNodes(*top*)

Given a top-level node, find all internal nodes.

getAllNodes(*top*)

Given a node, traverse all nodes and return as a list in pre-order.

postOrderTraversal(*top*)

Given a node, traverse all nodes and return as a list in post-order.

getAllBranches(*br*)

Given a branch, traverse subtree and return comprising branches as a list.

isSibling(*br*, *other*)

Given a branch, determine if that branch is adjacent to another branch.

getBalancingBracket(*newick*, *i*)

Given a position of an opening bracket in a newick string, i, output the closing bracket's position that corresponds to this opening bracket.

getBranchLength(*newick*, *i*)

Given a position of a colon symbol (indicating a branch length), return the branch length.

getLeafName(*newick*, *i*)

Given the position of a leaf, find its complete name.

parseNewick(*newick*, *i*, *j*, *top*)

Parse a newick string into a topological newick structure given a top-level node.

8.2 Variables

Name	Description
__package__	**Value:** `'Pylogeny'`

8.3 Class ParsingError

object

exceptions.BaseException

exceptions.Exception

Pylogeny.newick.ParsingError

8.3.1 Methods

__init__(*self*, *val*)

x.__init__(...) initializes x; see help(type(x)) for signature

Overrides: object.__init__ extit(inherited documentation)

__str__(*self*)

str(x)

Overrides: object.__str__ extit(inherited documentation)

Inherited from exceptions.Exception

__new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

8.3.2 Properties

Name	Description
Inherited from exceptions.BaseException args, message	
Inherited from object __class__	

8.4 Class tree

```
object ─┐
        Pylogeny.newick.tree
```

Defines a single phylogenetic tree by newick string; can possess other metadata.

8.4.1 Methods

__init__(*self*, *newi*='', *check*=False)

x.__init__(...) initializes x; see help(type(x)) for signature

Overrides: object.__init__ extit(inherited documentation)

__eq__(*self*, *o*)

__ne__(*self*, *o*)

__str__(*self*)

str(x)

Overrides: object.__str__ extit(inherited documentation)

setName(*self*, *n*)

setOrigin(*self*, *o*)

setScore(*self*, *s*)

getName(*self*)

getScore(*self*)

getOrigin(*self*)

getNewick(*self*)

getStructure(*self*)

getSimpleNewick(*self*)

Return a Newick string with all Taxa name replaced with successive integers.

toTopology(*self*)

Return a topology instance for this tree.

Inherited from object

___delattr___(), ___format___(), ___getattribute___(), ___hash___(), ___new___(), ___reduce___(), ___reduce_ex___(), ___repr___(), ___setattr___(), ___sizeof___(), ___subclasshook___()

8.4.2 Properties

Name	Description
Inherited from object	
___class___	

8.5 Class node

object ─┐
Pylogeny.newick.node

Newick node.

8.5.1 Methods

___init___(*self*, *lbl*=`''`, *strees*=`None`, *parent*=`None`)

x.___init___(...) initializes x; see help(type(x)) for signature

Overrides: object.___init___ extit(inherited documentation)

___str___(*self*)

str(x)

Overrides: object.___str___ extit(inherited documentation)

Inherited from object

___delattr___(), ___format___(), ___getattribute___(), ___hash___(), ___new___(), ___reduce___(), ___reduce_ex___(), ___repr___(), ___setattr___(), ___sizeof___(), ___subclasshook___()

8.5.2 Properties

Name	Description
Inherited from object	
___class___	

8.6 Class branch

```
object ─┐
        Pylogeny.newick.branch
```

Newick branch.

8.6.1 Methods

___init___(*self*, *chi*, *l*, *parent*=`None`, *s*=`None`)

x.___init___(...) initializes x; see help(type(x)) for signature

Overrides: object.___init___ extit(inherited documentation)

___str___(*self*)

str(x)

Overrides: object.___str___ extit(inherited documentation)

Inherited from object

___delattr___(), ___format___(), ___getattribute___(), ___hash___(), ___new___(), ___reduce___(), ___reduce_ex___(), ___repr___(), ___setattr___(), ___sizeof___(), ___subclasshook___()

8.6.2 Properties

Name	Description
Inherited from object	
___class___	

8.7 Class parser

Parsing object for Newick strings representing a phylogenetic tree.

8.7.1 Methods

__init__(*self*, *newick*)

parse(*self*)

Parse the stored newick string into a topological structure.

__str__(*self*)

9 Module Pylogeny.parsimony

Toolkit for performance of Parsimonious Criterion (Parsimony) methods of optimization of a phylogenetic topology with a particular set of data.

9.1 Functions

fitch_cost(*topology*, *profiles*)

Calculate the cost using Fitch algorithm on profile set and alignment. Deprecated: Python implementation of the Fitch algorithm; see fitch C++ module for a C++ implementation that is roughly four times faster.

fitch(*topology*, *alignment*)

Perform the Fitch algorithm on a given tree topology and associated alignment. Deprecated: Python implementation of the Fitch algorithm; see fitch C++ module for a C++ implementation that is roughly four times faster.

9.2 Variables

Name	Description
__package__	**Value:** `'Pylogeny'`

9.3 Class profile_set

Hold a set of site_profile profiles for an entire alignment.

9.3.1 Methods

__init__(*self*, *alignment*)

__len__(*self*)

weight(*self*, *val*)

get(*self*, *val*)

getForTaxa(*self*, *val*, *tax*)

9.4 Class site_profile

Consolidate the single-column alignment at a region into a set of components on the basis of similarity alone.

9.4.1 Methods

__init__(*self*, *alignment*, *site*)

__eq__(*self*, *o*)

__ne__(*self*, *o*)

__str__(*self*)

10 Module Pylogeny.pll

C Extension to wrap libpll library.

10.1 Variables

Name	Description
__package__	**Value:** `'Pylogeny'`

10.2 Class dataModel

Encapsulating a phylogenetic tree (as topology) + corresponding alignment into a libpll-associated data structure. Allows for log-likelihood scoring of this model. MUST BE CLOSED AFTER USE.

10.2.1 Methods

__init__(*self*, *topo*, *alignm*, *model*=`None`)

Initialize all structures.

getLogLikelihood(*self*)

Calculates log-likelihood using libpll.

close(*self*)

If done with this particular problem.

10.3 Class partitionModel

A partition model intended for libpll.

10.3.1 Methods

__init__(*self*, *ali*)

getFileName(*self*)

Get the file name of the model file.

createSimpleModel(*self*, *protein*)

Establish a simple model (e.g., one type).

createModel(*self*, *models*, *partnames*, *ranges*)

Establish a more complex model.

close(*self*)

Delete file.

11 Module Pylogeny.raxml

Python interface for RAxML executable.

11.1 Variables

Name	Description
RX_EXECUTABLE	**Value:** `'raxmlHPC'`
__package__	**Value:** `'Pylogeny'`

11.2 Class raxml

11.2.1 Methods

__init__(*self*, *inp_align*, *out_file*, *model*=`None`, *is_Protein*=`True`, *interTrees*=`False`, *alg*=`None`, *startingTree*=`None`, *rapid*=`False`, *slow*=`False`, *optimizeBootstrap*=`False`, *numboot*=`100`, *log*=`None`, *wdir*=`None`)

getInstructionString(*self*)

runFunction(*self*, *alg*)

run(*self*)

12 Module Pylogeny.rearrangement

Phylogenetic tree structure encapsulation; allow rearrangement of said structure. Tree rearrangements inducing other topologies include Nearest Neighbor Interchange (NNI), Subtree Pruning and Regrafting (SPR), and Tree Bisection and Reconstruction (TBR). Each of these describe a transfer of one node in phylogenetic trees from one parent of a tree to a new parent. Respectively, these operators describe transformations that are subsets of those possible by the successive operator. For example, an NNI operator can perform transformations that are a subset of the transformations possible by the SPR operator.

12.1 Functions

dup(*topo*, *where*=`None`)

12.2 Variables

Name	Description
TYPE_NNI	**Value:** `2`
TYPE_SPR	**Value:** `1`
TYPE_TBR	**Value:** `3`
__package__	**Value:** `'Pylogeny'`

12.3 Class RearrangementError

```
object ─┐
exceptions.BaseException ─┐
          exceptions.Exception ─┐
                                Pylogeny.rearrangement.RearrangementError
```

12.3.1 Methods

__init__(*self*, *val*)

x.__init__(...) initializes x; see help(type(x)) for signature

Overrides: object.__init__ extit(inherited documentation)

___str___(*self*)

str(x)

Overrides: object.___str___ extit(inherited documentation)

Inherited from exceptions.Exception

___new___()

Inherited from exceptions.BaseException

___delattr___(), ___getattribute___(), ___getitem___(), ___getslice___(), ___reduce___(), ___repr___(), ___setattr___(), ___setstate___(), ___unicode___()

Inherited from object

___format___(), ___hash___(), ___reduce_ex___(), ___sizeof___(), ___subclasshook___()

12.3.2 Properties

Name	Description
Inherited from exceptions.BaseException args, message	
Inherited from object ___class___	

12.4 Class rearrangement

Encapsulates a single rearrangement move of type SPR, NNI, ...

12.4.1 Methods

___init___(*self*, *struct*, *type*, *targ*, *dest*)

Initialize by providing a pointer to a base topology, a target branch to be moved, and its destination.

getType(*self*)

Get the type of movement.

isNNI(*self*)

isSPR(*self*)

isTBR(*self*)

toTopology(*self*)

Commit the actual move and return the topology.

toNewick(*self*)

Commit the move but do not create a new structure. Only retrieve resultant Newick string; will be more efficient.

toTree(*self*)

Commit the move and transform to tree object.

doMove(*self*)

__str__(*self*)

12.5 Class topology

Encapsulate a tree topology, wrapping the newick tree structure. Is immutable.

12.5.1 Methods

__init__(*self*, *t*=`None`, *rerootToLeaf*=`True`, *toLeaf*=`None`)

Initialize structure with a top-level internal node OR nothing.

rerootToLeaf(*self*, *toleaf*=`None`)

PRIVATE: Reroots the given tree structure such that it is rooted nearest the lowest-order leaf.

getRoot(*self*)

Return the top-level, root, node of the tree.

getInternalNodes(*self*)

getBranches(*self*)

getLeaves(*self*)

getBipartitions(*self*)

Get all bipartitions.

getStrBipartitionFromBranch(*self*, *br*)

Given a branch, return corresponding bipartition.

getBranchFromStrBipartition(*self*, *bip*)

Given a bipartition of taxa, return a branch that creates that partition of tree taxa.

getBranchFromBipartition(*self*, *bip*)

Given a bipartition object, return a branch that creates that partition of taxa.

lockBranch(*self*, *branch*)

Given a branch, lock it such that no transitions can ever occur across it.

move(*self*, *branch*, *destination*, *returnStruct*=`True`)

Move a branch and attach to a destination branch. Return new structure, or return merely the resultant Newick string.

SPR(*self*, *branch*, *destination*)

Perform an SPR move of a branch to a destination branch, creating a new node there. Returns a rearrangement structure (not the actual new structure) that can then be polled for the actual move; this is in order to save memory.

NNI(*self*, *branch*, *destination*)

Perform an NNI move of a branch to a destination, only if that destination branch is a parent's parent or a parent's sibling. Returns a rearrangement structure (not the actual new structure) that can then be polled for the actual move; this is in order to save memory.

iterSPRForBranch(*self*, *br*, *flip*=`True`)

Consider all valid SPR moves for a given branch in the topology and yield all possible rearrangements as a generator.

allSPRForBranch(*self*, *br*, *flip*=`True`)

Consider all valid SPR moves for a given branch in the topology and return all possible rearrangements.

allSPR(*self*)

Consider all valid SPR moves for a given topology and return all possible rearrangements.

iterNNIForBranch(*self*, *br*, *flip*=`True`)

Consider all valid NNI moves for a given branch in the topology and and yield all possible rearrangements as a generator.

allNNIForBranch(*self*, *br*, *flip*=`True`)

Consider all valid NNI moves for a given branch in the topology and return all possible rearrangements.

allNNI(*self*)

Consider all valid NNI moves for a given topology and return all possible rearrangements.

fromNewick(*self*, *newickstr*)

Alias for parse().

parse(*self*, *newickstr*)

Parse a newick string and assign the tree to this object. Cannot already be initialized with a tree.

toNewick(*self*)

Return the newick string of the tree.

toUnrootedNewick(*self*)

Return the newick string of the tree as an unrooted topology with a multifurcating top-level node.

toTree(*self*)

Return the tree object for this topology.

toUnrootedTree(*self*)

Return the tree object of the unrooted version of this topology.

__str__(*self*)

Return the newick string of the tree.

13 Module Pylogeny.scoring

Phylogenetic tree scoring.

13.1 Functions

getLogLikelihoodForTopology(*topo*, *alignment*)

Acquire log-likelihood via C library libpll. Parameters: rearrangement.topology object and alignment object.

getLogLikelihood(*tree*, *alignment*)

Acquire log-likelihood via C library libpll. Parameters: newick.tree object and alignment object.

getParsimony(*newick*, *alignment*)

Acquire parsimony via a C++ implementation. Parameters: newick string and alignment object.

getParsimonyForTopology(*topology*, *alignment*)

Acquire parsimony via a C++ implementation. Parameters: rearrangement.topology and alignment object.

getParsimonyFromProfiles(*newick*, *profiles*)

Acquire parsimony via a C++ implementation. Parameters: newick string and parsimony.profile_set object.

getParsimonyFromProfilesForTopology(*topology*, *profiles*)

Acquire parsimony via a C++ implementation. Parameters: rearrangement.topology and parsimony.profile_set object.

13.2 Variables

Name	Description
__package__	**Value:** `'Pylogeny'`

Index