Package ‘PCMBase’

March 15, 2024

Type Package

Title Simulation and Likelihood Calculation of Phylogenetic Comparative Models

Version 1.2.14

Maintainer Venelin Mitov <vmitov@gmail.com>

Description Phylogenetic comparative methods represent models of continuous trait data associated with the tips of a phylogenetic tree. Examples of such models are Gaussian continuous time branching stochastic processes such as Brownian motion (BM) and Ornstein-Uhlenbeck (OU) processes, which regard the data at the tips of the tree as an observed (final) state of a Markov process starting from an initial state at the root and evolving along the branches of the tree. The PCMBase R package provides a general framework for manipulating such models. This framework consists of an application programming interface for specifying data and model parameters, and efficient algorithms for simulating trait evolution under a model and calculating the likelihood of model parameters for an assumed model and trait data. The package implements a growing collection of models, which currently includes BM, OU, BM/OU with jumps, two-speed OU as well as mixed Gaussian models, in which different types of the above models can be associated with different branches of the tree. The PCMBase package is limited to trait-simulation and likelihood calculation of (mixed) Gaussian phylogenetic models. The PCMFt package provides functionality for inference of these models to tree and trait data. The package web-site <https://venelin.github.io/PCMBase/> provides access to the documentation and other resources.

Encoding UTF-8

License GPL (>= 3.0)

LazyData true

Depends R (>= 3.1.0)

Imports ape, abind, expm, mvtnorm, data.table, ggplot2, xtable

Suggests testthat, knitr, rmarkdown, ggtree, cowplot, covr, mvSLOUCH, BiocManager

RoxygenNote 7.1.2

ByteCompile yes
VignetteBuilder knitr, rmarkdown


BugReports https://github.com/venelin/PCMBase/issues

NeedsCompilation no

Repository CRAN

Author Venelin Mitov [aut, cre, cph] (<a href="https://venelin.github.io">venelin.github.io</a>),
Krzysztof Bartoszek [ctb],
Georgios Asimomitis [ctb],
Tanja Stadler [ths]

Date/Publication 2024-03-15 20:10:02 UTC

R topics documented:

- Args_MixedGaussian_MGPMDefaultModelTypes ........................................ 5
- Args_MixedGaussian_MGPMScalarOUType .............................................. 6
- Args_MixedGaussian_MGPMSurfaceOUType ............................................ 6
- as.MixedGaussian ................................................................. 7
- dataFig3 ................................................................. 8
- FormatCellAsLatex .......................................................... 8
- FormatTableAsLatex .......................................................... 9
- is.GaussianPCM ............................................................ 9
- is.MixedGaussian .......................................................... 10
- is.PCM ................................................................. 10
- is.PCMTree ............................................................... 11
- MatchListMembers ......................................................... 11
- MGPMScalarOUType ......................................................... 12
- MGPMSurfaceOUType .......................................................... 12
- MixedGaussian ............................................................. 13
- MixedGaussianTemplate .......................................................... 14
- PCM ................................................................. 15
- PCMAddToListAttribute ......................................................... 17
- PCMApplyTransformation .......................................................... 18
- PCMBaseIsADevRelease .......................................................... 19
- PCMBaseTestObjects .......................................................... 20
- PCMColorPalette ............................................................. 20
- PCMCombineListAttribute .......................................................... 22
- PCMCond ................................................................. 22
- PCMCond.GaussianPCM .......................................................... 23
- PCMCondVOU ................................................................. 24
- PCMCreateLikelihood .......................................................... 25
- PCMDefaultModelTypes .......................................................... 26
- PCMDefaultObject ........................................................... 27
- PCMDescrbe ................................................................. 28
- PCMBase ................................................................. 29
<table>
<thead>
<tr>
<th>Function Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCMDescibeParameters</td>
<td>29</td>
</tr>
<tr>
<td>PCMExtractDimensions</td>
<td>30</td>
</tr>
<tr>
<td>PCMExtractRegimes</td>
<td>30</td>
</tr>
<tr>
<td>PCMFindMethod</td>
<td>31</td>
</tr>
<tr>
<td>PCMFixParameter</td>
<td>32</td>
</tr>
<tr>
<td>PCMGenerateModelTypes</td>
<td>32</td>
</tr>
<tr>
<td>PCMGenerateParameterizations</td>
<td>33</td>
</tr>
<tr>
<td>PCMGetAttribute</td>
<td>34</td>
</tr>
<tr>
<td>PCMGetVecParamsRegimesAndModels</td>
<td>35</td>
</tr>
<tr>
<td>PCMInfo</td>
<td>35</td>
</tr>
<tr>
<td>PCMLik</td>
<td>37</td>
</tr>
<tr>
<td>PCMLikDmvNorm</td>
<td>39</td>
</tr>
<tr>
<td>PCMLikTrace</td>
<td>40</td>
</tr>
<tr>
<td>PCMListMembers</td>
<td>42</td>
</tr>
<tr>
<td>PCMListParameterizations</td>
<td>43</td>
</tr>
<tr>
<td>PCMLmr</td>
<td>44</td>
</tr>
<tr>
<td>PCMMapModelTypesToRegimes</td>
<td>45</td>
</tr>
<tr>
<td>PCMMean</td>
<td>46</td>
</tr>
<tr>
<td>PCMMeanAtTime</td>
<td>47</td>
</tr>
<tr>
<td>PCMModels</td>
<td>48</td>
</tr>
<tr>
<td>PCMModelTypes</td>
<td>49</td>
</tr>
<tr>
<td>PCNNumRegimes</td>
<td>49</td>
</tr>
<tr>
<td>PCNNumTraits</td>
<td>50</td>
</tr>
<tr>
<td>PCMOptions</td>
<td>50</td>
</tr>
<tr>
<td>PCMParam</td>
<td>52</td>
</tr>
<tr>
<td>PCMParamBindRegimeParams</td>
<td>53</td>
</tr>
<tr>
<td>PCMParamCount</td>
<td>55</td>
</tr>
<tr>
<td>PCMParamGetShortVector</td>
<td>56</td>
</tr>
<tr>
<td>PCMParamLoadOrStore</td>
<td>56</td>
</tr>
<tr>
<td>PCMParamLocateInShortVector</td>
<td>57</td>
</tr>
<tr>
<td>PCMParamLowerLimit</td>
<td>58</td>
</tr>
<tr>
<td>PCMParamRandomVecParams</td>
<td>58</td>
</tr>
<tr>
<td>PCMParamSetByName</td>
<td>59</td>
</tr>
<tr>
<td>PCMParamType</td>
<td>60</td>
</tr>
<tr>
<td>PCMParamUpperLimit</td>
<td>63</td>
</tr>
<tr>
<td>PCMParentClasses</td>
<td>63</td>
</tr>
<tr>
<td>PCMParseErrorMessage</td>
<td>64</td>
</tr>
<tr>
<td>PCMPExpMeanExp</td>
<td>64</td>
</tr>
<tr>
<td>PCMLambdaP_1</td>
<td>65</td>
</tr>
<tr>
<td>PCMPlotGaussianDensityGrid2D</td>
<td>66</td>
</tr>
<tr>
<td>PCMPlotGaussianSample2D</td>
<td>66</td>
</tr>
<tr>
<td>PCMPlotMath</td>
<td>67</td>
</tr>
<tr>
<td>PCMPlotTraitData2D</td>
<td>68</td>
</tr>
<tr>
<td>PCMPresentCoordinates</td>
<td>69</td>
</tr>
<tr>
<td>PCMRegimes</td>
<td>70</td>
</tr>
<tr>
<td>PCMSetAttribute</td>
<td>70</td>
</tr>
<tr>
<td>PCMSim</td>
<td>71</td>
</tr>
</tbody>
</table>
R topics documented:

- PCMSpecify .................................................. 73
- PCMTable .................................................... 73
- PCMTableParameterizations .............................. 74
- PCMTrajectory .............................................. 75
- PCMTree ..................................................... 77
- PCMTreeBackbonePartition .............................. 79
- PCMTreeDropClade ......................................... 81
- PCMTreeDtNodes ........................................... 82
- PCMTreeEdgeTimes ......................................... 83
- PCMTreeEvalNestedEDxOnTree ........................... 84
- PCMTreeExtractClade ...................................... 85
- PCMTreeGetBranchLength ................................. 86
- PCMTreeGetDaughters ..................................... 87
- PCMTreeGetLabels .......................................... 87
- PCMTreeGetParent ......................................... 88
- PCMTreeGetPartition ...................................... 89
- PCMTreeGetPartNames ..................................... 89
- PCMTreeGetPartRegimes .................................. 90
- PCMTreeGetPartsForNodes ............................... 90
- PCMTreeGetRegimesForEdges ............................ 91
- PCMTreeGetRegimesForNodes ............................ 91
- PCMTreeGetTipsInPart ..................................... 92
- PCMTreeGetTipsInRegime .................................. 93
- PCMTreeInsertSingletons .................................. 94
- PCMTreeJumps .............................................. 96
- PCMTreeListAllPartitions ............................... 97
- PCMTreeListCladePartitions ............................ 98
- PCMTreeListDescendants .................................. 99
- PCMTreeListRootPaths .................................... 100
- PCMTreeLocateEpochOnBranches ....................... 100
- PCMTreeLocateMidpointsOnBranches ................... 101
- PCMTreeMatchLabels ...................................... 101
- PCMTreeMatrixNodesInSamePart ....................... 102
- PCMTreeNearestNodesToEpoch ......................... 103
- PCMTreeNodeTimes ........................................ 104
- PCMTreeNumNodes ......................................... 104
- PCMTreeNumParts ......................................... 105
- PCMTreeNumTips ........................................... 105
- PCMTreePlot ................................................ 106
- PCMTreePostorder ......................................... 106
- PCMTreePreorder .......................................... 107
- PCMTreeSetLabels ......................................... 107
- PCMTreeSetPartition ...................................... 108
- PCMTreeSetPartRegimes .................................. 110
- PCMTreeSetRegimesForEdges ............................ 111
- PCMTreeSplitAtNode ....................................... 113
- PCMTreeTableAncestors .................................. 114
- PCMTreeToString .......................................... 115
Args_MixedGaussian_MGPMDefaultModelTypes

Arguments to be passed to the constructor MixedGaussian when constructing a MGPM model with some of the default MGPM model types.

Description

Arguments to be passed to the constructor MixedGaussian when constructing a MGPM model with some of the default MGPM model types.

Usage

Args_MixedGaussian_MGPMDefaultModelTypes(omitGlobalSigmae_x = TRUE)

Arguments

omitGlobalSigmae_x
logical, indicating if the returned list should specify the global Sigmae_x parameter as 'Omitted'. Default: TRUE.

Value

a list of named arguments. Currently only a named element Sigmae_x with specification depending on omitGlobalSigmae_x.

See Also

MGPMDefaultModelTypes
**Args_MixedGaussian_MGPMScalarOUType**

*Arguments for the MixedGaussian constructor for scalar OU MGPM models.*

---

**Description**

Arguments for the MixedGaussian constructor for scalar OU MGPM models.

**Usage**

```python
Args_MixedGaussian_MGPMScalarOUType()
```

**Value**

a list.

---

**Args_MixedGaussian_MGPMSurfaceOUType**

*Arguments for the MixedGaussian constructor for SURFACE OU MGPM models.*

---

**Description**

Arguments for the MixedGaussian constructor for SURFACE OU MGPM models.

**Usage**

```python
Args_MixedGaussian_MGPMSurfaceOUType()
```

**Value**

a list.
as.MixedGaussian

Convert a GaussianPCM model object to a MixedGaussian model object

Description

Convert a GaussianPCM model object to a MixedGaussian model object

Usage

as.MixedGaussian(o, modelTypes = NULL)

Arguments

- `o`: an R object: either a GaussianPCM or a MixedGaussian.
- `modelTypes`: NULL (the default) or a (possibly named) character string vector. Each such string denotes a mixed Gaussian regime model class, e.g. the result of calling MGPMDefaultModelTypes(). If specified, an attempt is made to match the deduced Gaussian regime model type from `o` with the elements of `modelTypes` and an error is raised if the match fails. If the match succeeds the converted MixedGaussian object will have the specified `modelTypes` parameter as an attribute "modelTypes".

Value

a MixedGaussian object.

Examples

```r
mg <- as.MixedGaussian(PCMBaseTestObjects$model.ab.123.bSigmae_x)
stopifnot(
  PCMLik(
    X = PCMBaseTestObjects$traits.ab.123,
    PCMBaseTestObjects$tree.ab,
    PCMBaseTestObjects$model.ab.123.bSigmae_x) ==
  PCMLik(
    X = PCMBaseTestObjects$traits.ab.123,
    PCMBaseTestObjects$tree.ab,
    mg))
```
dataFig3  
*Data for Fig 3 in the TPB manuscript*

**Description**

A list containing simulated tree, models and data used in Fig. 3

**Usage**

`dataFig3`

**Format**

This is a list containing the following named elements representing simulation parameters, a simulated tree and PCM objects, used in Fig. 3. For details on all these objects, read the file `data-raw/Fig3.Rmd`.

---

`FormatCellAsLatex`  
*Latex representation of a model parameter or other found in a data.table object*

**Description**

Latex representation of a model parameter or other found in a data.table object

**Usage**

`FormatCellAsLatex(x)`

**Arguments**

- `x` an R object. Currently, character vectors of length 1, numeric vectors and matrices are supported.

**Value**

a character string
**FormatTableAsLatex**

Latex representation of a data.table with matrix and vectors in its cells

Description

Latex representation of a data.table with matrix and vectors in its cells

Usage

FormatTableAsLatex(x, argsXtable = list(), ...)

Arguments

- **x**
  a data.table
- **argsXtable**
  a list (empty list by default) passed to xtable...
- **...**
  additional arguments passed to print.xtable.

Value

a character string representing a parseable latex text.

Examples

```r
dt <- data.table::data.table(
  A = list(
    matrix(c(2, 0, 1.2, 3), 2, 2),
    matrix(c(2.1, 0, 1.2, 3.2, 1.3, 3.4), 3, 2)),
  b = c(2.2, 3.1))
print(FormatTableAsLatex(dt))
```

---

**is.GaussianPCM**

Check if an object is a ‘GaussianPCM’

Description

Check if an object is a ‘GaussianPCM’

Usage

is.GaussianPCM(x)

Arguments

- **x**
  any object
**is.PCM**

Value

TRUE if x inherits from the S3 class ‘GaussianPCM’, FALSE otherwise.

---

**is.MixedGaussian**

*Check if an object is a ‘MixedGaussian’ PCM*

---

**Description**

Check if an object is a ‘MixedGaussian’ PCM

**Usage**

`is.MixedGaussian(x)`

**Arguments**

x  
any object

**Value**

TRUE if x inherits from the S3 class ‘MixedGaussian’, FALSE otherwise.

---

**is.PCM**

*Check if an object is a PCM.*

---

**Description**

Check if an object is a PCM.

**Usage**

`is.PCM(x)`

**Arguments**

x  
an object.

**Value**

TRUE if ‘x’ inherits from the S3 class "PCM".
is.PCMTree

**Check that a tree is a PCMTree**

**Description**
Check that a tree is a PCMTree

**Usage**
is.PCMTree(tree)

**Arguments**
- `tree`: a tree object.

**Value**
a logical TRUE if `inherits(tree, "PCMTree")` is TRUE.

---

MatchListMembers

**Find the members in a list matching a member expression**

**Description**
Find the members in a list matching a member expression

**Usage**
MatchListMembers(object, member, enclos = "?", q = "'", ...)  

**Arguments**
- `object`: a list containing named elements.
- `member`: a member expression. Member expressions are character strings denoting named elements in a list object (see examples).
- `enclos`: a character string containing the special symbol `?`. This symbol is to be replaced by matching expressions. The result of this substitution can be anything but, usually would be a valid R expression. Default: "?".
- `q`: a quote symbol, Default: "."
- `...`: additional arguments passed to `grep`. For example, these could be `ignore.case=TRUE` or `perl=TRUE`. 

**MGPMScalarOUType**

**Value**

A named character vector, with names corresponding to the matched member quoted expressions (using the argument `q` as a quote symbol), and values corresponding to the 'enclos-ed' expressions after substituting the '?'.

**See Also**

[PCMLListMembers](#)

**Examples**

```r
model <- PCMBaseTestObjects$model_MixedGaussian_ab
MatchListMembers(model, "Sigma_x", "diag(model?[,,1L])")
MatchListMembers(model, "S.*_x", "diag(model?[,,1L])")
MatchListMembers(model, "Sigma_x", "model?[,,1L][upper.tri(model?[,,1L])]")
MatchListMembers(model, "a$Sigma_x", "model?[,,1L][upper.tri(model?[,,1L])]")
```

---

**MGPMSurfaceOUType**

**Class name for the SURFACE OU MGPM model type**

**Description**

Class name for the SURFACE OU MGPM model type

**Usage**

```r
MGPMSurfaceOUType()
```

**Value**

A character vector of one named element (SURFACE)
MixedGaussian  

Create a multi-regime Gaussian model (MixedGaussian)

Description
Create a multi-regime Gaussian model (MixedGaussian)

Usage
MixedGaussian(
  k,
  modelTypes,
  mapping,
  className = paste0("MixedGaussian_", do.call(paste0, as.list(mapping))),
  X0 = structure(0, class = c("VectorParameter", "_Global"), description = "trait values at the root"),
  ...,  
  Sigmae_x = structure(0, class = c("MatrixParameter", "_UpperTriangularWithDiagonal", 
                       "_WithNonNegativeDiagonal", "_Global"), description = "Upper triangular factor of the non-phylogenetic variance-covariance")
)

Arguments
  k  integer specifying the number of traits.
  modelTypes, mapping  These two arguments define the mapping between the regimes in the model and actual types of models. For convenience, different combinations are possible as explained below:
    • modelTypes is a (possibly named) character string vector. Each such string denotes a mixed Gaussian regime model class, e.g. the result of calling MGPMDefaultModelTypes(). In that case mapping can be either an integer vector with values corresponding to indices in modelTypes or a character string vector. If mapping is a character string vector, first it is matched against names(modelTypes) and if the match fails either because of names(modelTypes) being NULL or because some of the entries in mapping are not present in names(modelTypes), then an attempt is made to match mapping against modelTypes, i.e. it is assumed that mapping contains actual class names.
    • modelTypes is a (possibly named) list of PCM models of k traits. In this case mapping can again be an integer vector denoting indices in modelTypes or a character string vector denoting names in modelTypes.
  As a final note, mapping can also be named. In this case the names are assumed to be the names of the different regimes in the model. If mapping is not named, the regimes are named automatically as as.character(seq_len(mapping)). For example, if modelTypes = c("BM", "OU") and mapping = c(a = 1, b = 1, c = 2, d = 1) defines an MixedGaussian with four different regimes named 'a',

mixed Gaussian template

'b', 'c', 'd', and model-types BM, BM, OU and BM, corresponding to each regime.

**className**
A character string defining a valid S3 class name for the resulting MixedGaussian object. If not specified, a className is generated using the expression `paste0("MixedGaussian_", do.call(paste0, as.list(mapping)))`.

**X0**
Specification for the global vector X0 to be used by all models in the MixedGaussian.

**...**
Specifications for other _Global parameters coming after X0.

**Sigmae_x**
Specification of a _Global Sigmae_x parameter. This is used by Submodels only if they have Sigmae_x _Omitted.

### Details
If X0 is not NULL it has no sense to use model-types including X0 as a parameter (e.g. use BM1 or BM3 instead of BM or BM2). Similarly if Sigmae_x is not NULL there is no meaning in using model-types including Sigmae_x as a parameter, (e.g. use OU2 or OU3 instead of OU or OU1).

### Value
An object of S3 class className inheriting from MixedGaussian, GaussianPCM and PCM.

### See Also
- `PCMTreeGetPartNames`
- `PCMModels()`

---

**MixedGaussianTemplate**  
*Create a template MixedGaussian object containing a regime for each model type*

### Description
Create a template MixedGaussian object containing a regime for each model type

### Usage
`MixedGaussianTemplate(mg, modelTypes = NULL)`

### Arguments
- **mg**
  
  A MixedGaussian object or an object that can be converted to such via `as.MixedGaussian`.  

- **modelTypes**
  
  A (possibly named) character string vector. Each such string denotes a mixed Gaussian regime model class, e.g. the result of calling `MGPMDefaultModelTypes()`.  
  
  If specified, an attempt is made to match `PCMModelTypes(as.MixedGaussian(mg))` with the elements of modelTypes and an error is raised if the match fails. If not named, the model types and regimes in the resulting MixedGaussian object are named by the capital latin letters A,B,C,...  
  
  Default: NULL, which is interpreted as `PCMModelTypes(as.MixedGaussian(mg, NULL))`.  

PCM

Value

A MixedGaussian with the same global parameter settings as for mg, the same modelTypes as
modelTypes, and with a regime for each model type. The function will stop with an error if mg
is not convertible to a MixedGaussian object or if there is a mismatch between the model types in
mg and modelTypes.

Examples

mg <- MixedGaussianTemplate(PCMBaseTestObjects$model.ab.123.bSigma_x)
mgTemplBMU <- MixedGaussianTemplate(PCMBaseTestObjects$model.OU.BM)

Description

This is the entry-point function for creating model objects within the PCMBase framework rep-
resenting a single model-type with one or several model-regimes of this type associated with the
branches of a tree. For mixed Gaussian phylogenetic models, which enable multiple model-types,
use the MixedGaussian function.

Usage

PCM(
  model,
  modelTypes = class(model)[1],
  k = 1L,
  regimes = 1L,
  params = NULL,
  vecParams = NULL,
  offset = 0L,
  spec = NULL,
  ...
)

Arguments

model This argument can take one of the following forms:
  • a character vector of the S3-classes of the model object to be created (one
    model object can have one or more S3-classes, with the class PCM at the
    origin of the hierarchy);
  • an S3 object which’s class inherits from the PCM S3 class.

The Details section explains how these two types of input are processed.

modelTypes a character string vector specifying a set (family) of model-classes, to which the
constructed model object belongs. These are used for model-selection.
k integer denoting the number of traits (defaults to 1).
regimes a character or integer vector denoting the regimes.
params NULL (default) or a list of parameter values (scalars, vectors, matrices, or arrays) or sub-models (S3 objects inheriting from the PCM class). See details.
vecParams NULL (default) or a numeric vector the vector representation of the variable parameters in the model. See details.
offset integer offset in vecParams; see Details.
spec NULL or a list specifying the model parameters (see PCMSpecify). If NULL (default), the generic PCMSpecify is called on the created object of class model.
... additional parameters intended for use by sub-classes of the PCM class.

Details
This is an S3 generic. The PCMBASE package defines three methods for it:

**PCM.PCM:** A default constructor for any object with a class inheriting from "PCM".
**PCM.character:** A default PCM constructor from a character string specifying the type of model.
**PCM.default:** A default constructor called when no other constructor is found. When called this constructor raises an error message.

Value
an object of S3 class as defined by the argument model.

See Also
MixedGaussian

Examples

```r
# a Brownian motion model with one regime
modelBM <- PCM(model = "BM", k = 2)
# print the model
modelBM

# a BM model with two regimes
modelBM.ab <- PCM("BM", k = 2, regimes = c("a", "b"))
modelBM.ab

# print a single parameter of the model (in this case, the root value)
modelBM.ab$X0

# assign a value to this parameter (note that the brackets [] are necessary # to preserve the parameter attributes):
modelBM.ab$X0[] <- c(5, 2)

PCMNumbTraits(modelBM)
PCMNumbRegimes(modelBM)
PCMNumbRegimes(modelBM.ab)
```
# number of numerical parameters in the model
PCParamCount(modelBM)

# Get a vector representation of all parameters in the model
PCParamGetShortVector(modelBM)

# Limits for the model parameters:
lowerLimit <- PCParamLowerLimit(modelBM)
upperLimit <- PCParamUpperLimit(modelBM)

# assign the model parameters at random: this will use uniform distribution
# with boundaries specified by PCParamLowerLimit and PCParamUpperLimit
# We do this in two steps:
# 1. First we generate a random vector. Note the length of the vector equals PCParamCount(modelBM)
randomParams <- PCParamRandomVecParams(modelBM, PCNumTraits(modelBM), PCNumRegimes(modelBM))
rnormParams
# 2. Then we load this random vector into the model.
PCParamLoadOrStore(modelBM, randomParams, 0, PCNumTraits(modelBM), PCNumRegimes(modelBM), TRUE)

print(modelBM)
PCParamGetShortVector(modelBM)

# generate a random phylogenetic tree of 10 tips
tree <- ape::rmtree(10)

# simulate the model on the tree
traitValues <- PCMSim(tree, modelBM, X0 = modelBM$X0)

# calculate the likelihood for the model parameters, given the tree and the trait values
PCMLik(traitValues, tree, modelBM)

# create a likelihood function for faster processing for this specific model.
# This function is convenient for calling in optim because it receives and parameter
# vector instead of a model object.
likFun <- PCMCreateLikelihood(traitValues, tree, modelBM)
likFun(randomParams)

---

**PCMAbCdEf**

*Quadratic polynomial parameters A, b, C, d, E, f for each node*

**Description**

An S3 generic function that has to be implemented for every model class. This function is called by **PCMLik**.

**Usage**

PCMAbCdEf(

  tree,
Add a value to a list-valued attribute of a member or members matching a pattern

Add a value to a list-valued attribute of a member or members matching a pattern
Arguments

name a character string denoting the attribute name.
value the value for the attribute.
object a PCM or a list object.
member a member expression. Member expressions are character strings denoting named elements in a list object (see examples). Default: "".
enclos a character string containing the special symbol '?'. This symbol is to be replaced by matching expressions. The result of this substitution can be anything but, usually would be a valid R expression. Default: "".
spec a logical (TRUE by default) indicating if the attribute should also be set in the corresponding member of the spec attribute (this is for PCM objects only).
inplace logical (TRUE by default) indicating if the attribute should be set to the object in the current environment, or a modified object should be returned.
... additional arguments passed to MatchListMembers.

Value

if inplace is TRUE no value is returned. Otherwise, a modified version of object is returned.

Description

This is an S3 generic that transforms the passed argument by applying the transformation rules for its S3 class.

This is an S3 generic. See ‘PCMAppearTransformation._CholeskyFactor’ for an example.

Usage

PCMAppearTransformation(o, ...)

Arguments

o a PCM object or a parameter
... additional arguments that can be used by implementing methods.
Details
This function returns the same object if it is not transformable.

Value
a transformed version of o.

See Also
is.Transformable

---

**PCMBaseIsADevRelease**  
*Check if the PCMBase version corresponds to a dev release*

**Description**
Check if the PCMBase version corresponds to a dev release

**Usage**
PCMBaseIsADevRelease()

**Value**
a logical

---

**PCMBaseTestObjects**  
*Test objects for the PCMBase package*

**Description**
A list containing simulated trees, trait-values and model objects for tests and examples of the PCMBase package

**Usage**
PCMBaseTestObjects
Format

This is a list containing the following named elements representing parameters of BM, OU and MixedGaussian models with up to three traits and up to two regimes, model objects, simulated trees with partition of their nodes in up to two parts (corresponding to the two regimes), and trait data simulated on these trees.

\[ a.H, b.H \] H matrices for OU models for regimes ‘a’ and ‘b’.

\[ a.Theta, b.Theta \] Theta vectors for OU models for regimes ‘a’ and ‘b’.

\[ a.Sigma_x, b.Sigma_x \] Sigma_x matrices for BM and OU models for regimes ‘a’ and ‘b’.

\[ a.Sigmae_x, b.Sigmae_x \] Sigmae_x matrices regimes ‘a’ and ‘b’.

\[ a.X0, b.X0 \] X0 vectors for regimes ‘a’ and ‘b’.

\[ H \] an array resulting from abind(a.H, b.H).

\[ Theta \] a matrix resulting from cbind(Theta.a, Theta.b).

\[ Sigma_x \] an array resulting from abind(a.Sigma_x, b.Sigma_x).

\[ Sigmae_x \] an array resulting from abind(a.Sigmae_x, b.Sigmae_x).

\[ model.a.1, model.a.2, model.a.3 \] univariate models with a single regime for each of 3 traits.

\[ model.a.1.Omitted_X0 \] same as model.a.1 but omitting X0; suitable for nesting in an MGPM model.

\[ model.a.123, model.b.123 \] single-regime 3-variate models.

\[ model.a.123.Omitted_X0 \] single-regime 3-variate model with omitted X0 (suitable for nesting in an MGPM.

\[ model.a.123.Omitted_X0__bSigmae_x \] same as model.a.123.Omitted_X0 but with the value of Sigmae_x copied from model.b.123.

\[ model.a.123.Omitted_X0__Omitted_Sigmae_x \] same as model.a.123 but omitting X0 and Sigmae_x.

\[ model.b.123.Omitted_X0, model.b.123.Omitted_X0__Omitted_Sigmae_x \] analogous to corresponding model.a.123... \[ model.ab.123 \] a two-regime 3-variate model.

\[ model.ab.123.bSigmae_x \] a two-regime 3-variate model having Sigmae_x from b.Sigmae_x.

\[ model_MixedGaussian_ab \] a two-regime MGPM model with a local Sigmae_x for each regime.

\[ model_MixedGaussian_ab_globalSigmae_x \] a two-regime MGPM model with a global Sigmae_x.

\[ N \] number of tips in simulated trees

\[ tree_15_tips \] a tree of 15 tips used for testing clade extraction.

\[ tree.a \] a tree with one part only (one regime)

\[ tree.ab \] a tree partitioned in two parts (two regimes)

\[ traits.a.1 \] trait values simulated with model.a.1.

\[ traits.a.123 \] trait values simulated with model.a.123.

\[ traits.a.2 \] trait values simulated with model.a.2.

\[ traits.a.3 \] trait values simulated with model.a.3.

\[ traits.ab.123 \] trait values simulated with model.ab.123 on tree.ab.
tree  a tree of 5 tips used for examples.

X  3-trait data for 5 tips used together with tree for examples.

model.OU.BM  a mixed Gaussian phylogenetic model for 3 traits and an OU and BM regime used in examples.

---

**PCMColorPalette**

*A fixed palette of n colors*

---

**Description**

A fixed palette of n colors

**Usage**

```r
PCMColorPalette(
  n, names,
  colors = structure(hcl(h = seq(15, 375, length = n + 1), l = 65, c =
           100)[seq_len(n)], names = names)
)
```

**Arguments**

- `n`  an integer defining the number of colors in the resulting palette.
- `names`  a character vector of length ‘n’.
- `colors`  a vector of n values convertible to colors. Default: `structure(hcl(h = seq(15, 375, length = n + 1), l = 65, c = 100)[seq_len(n)], names = names)`

**Value**

A vector of character strings which can be used as color specifications by R graphics functions.

---

**PCMCombineListAttribute**

*Combine all member attributes of a given name into a list*

---

**Description**

Combine all member attributes of a given name into a list

**Usage**

```r
PCMCombineListAttribute(object, name)
```
Arguments

- **object**: a named list object.
- **name**: a character string denoting the name of the attribute.

Value

- A list of attribute values

---

**Description**

An S3 generic function that has to be implemented for every model class.

**Usage**

```r
PCMCond(
  tree,
  model,
  r = 1,
  metaI = PCMInfo(NULL, tree, model, verbose = verbose),
  verbose = FALSE
)
```

**Arguments**

- **tree**: a phylo object with N tips.
- **model**: an S3 object specifying both, the model type (class, e.g. "OU") as well as the concrete model parameter values at which the likelihood is to be calculated (see also Details).
- **r**: an integer specifying a model regime
- **metaI**: a list returned from a call to `PCMInfo(X, tree, model, SE)`, containing metadata such as N, M and k. Alternatively, this can be a character string naming a function or a function object that returns such a list, e.g. the function `PCMInfo` or the function `PCMInfoCpp` from the `PCMBaseCpp` package.
- **verbose**: logical indicating if some debug-messages should printed.

**Value**

- An object of type specific to the type of model
**Description**

An S3 generic function that has to be implemented for every model class.

**Usage**

```r
## S3 method for class 'GaussianPCM'
PCMCond(
  tree,
  model,
  r = 1,
  metaI = PCMInfo(NULL, tree, model, verbose = verbose),
  verbose = FALSE
)
```

**Arguments**

- `tree` a phylo object with N tips.
- `model` an S3 object specifying both, the model type (class, e.g. "OU") as well as the concrete model parameter values at which the likelihood is to be calculated (see also Details).
- `r` an integer specifying a model regime
- `metaI` a list returned from a call to `PCMInfo(X, tree, model, SE)`, containing metadata such as N, M and k. Alternatively, this can be a character string naming a function or a function object that returns such a list, e.g. the function `PCMInfo` or the function `PCMInfoCpp` from the `PCMBaseCpp` package.
- `verbose` logical indicating if some debug-messages should printed.

**Value**

For GaussianPCM models, a named list with the following members:

- `omega d`
- `Phi`
- `V`
Variance-covariance matrix of an OU process with optional measurement error and jump at the start

Usage

PCMCondVOU(
  H,
  Sigma,
  Sigmae = NULL,
  Sigmaj = NULL,
  xi = NULL,
  e_Ht = NULL,
  threshold.Lambda_ij = getOption("PCMBase.Threshold.Lambda_ij", 1e-08)
)

Arguments

H
  a numerical k x k matrix - selection strength parameter.
Sigma
  a numerical k x k matrix - neutral drift unit-time variance-covariance matrix.
Sigmae
  a numerical k x k matrix - environmental variance-covariance matrix.
Sigmaj
  is the variance matrix of the normal jump distribution (default is NULL).
xi
  a vector of 0’s and 1’s corresponding to each branch in the tree. A value of 1 indicates that a jump takes place at the beginning of the branch. This argument is only used if Sigmaj is not NULL. Default is NULL.
e_Ht
  a numerical k x k matrix - the result of the matrix exponential expm(-t*H).
threshold.Lambda_ij
  a 0-threshold for abs(Lambda_i + Lambda_j), where Lambda_i and Lambda_j are eigenvalues of the parameter matrix H. This threshold-values is used as a condition to take the limit time of the expression ‘(1-exp(-Lambda_ij*time))/Lambda_ij’ as ‘(Lambda_i+Lambda_j) -> 0’. You can control this value by the global option "PCMBase.Threshold.Lambda_ij". The default value (1e-8) is suitable for branch lengths bigger than 1e-6. For smaller branch lengths, you may want to increase the threshold value using e.g. ‘options(PCMBase.Threshold.Lambda_ij=1e-6)’.

Value

a function of one numerical argument (time) and an integer indicating the branch-index that is used to check the corresponding element in xi.
PCMCreateLikelihood

*Create a likelihood function of a numerical vector parameter*

**Description**

Create a likelihood function of a numerical vector parameter

**Usage**

```r
PCMCreateLikelihood(
  X, 
  tree, 
  model, 
  SE = matrix(0, PCMNumTraits(model), PCMTreeNumTips(tree)), 
  metaI = PCMInfo(X, tree, model, SE), 
  positiveValueGuard = Inf
)
```

**Arguments**

- **X**: a $k \times N$ numerical matrix with possible NA and NaN entries. For $i=1,\ldots, N$, the column $i$ of $X$ contains the measured trait values for species $i$ (the tip with integer identifier equal to $i$ in `tree`). Missing values can be either not-available (NA) or not existing (NaN). These two values are treated differently when calculating likelihoods (see `PCMPresentCoordinates`).
- **tree**: a phylo object with $N$ tips.
- **model**: an S3 object specifying both, the model type (class, e.g. "OU") as well as the concrete model parameter values at which the likelihood is to be calculated (see also Details).
- **SE**: a $k \times N$ matrix specifying the standard error for each measurement in $X$. Alternatively, a $k \times k \times N$ cube specifying an upper triangular $k \times k$ factor of the variance covariance matrix for the measurement error for each tip $i=1,\ldots, N$. When $SE$ is a matrix, the $k \times k$ measurement error variance matrix for a tip $i$ is calculated as $VE[, , i] <- \text{diag}(SE[, i] \times SE[, i], nrow = k)$. When $SE$ is a cube, the way how the measurement variance matrix for a tip $i$ is calculated depends on the runtime option `PCMBase.Transpose.Sigma_x` as follows:
  - **if** `getOption("PCMBase.Transpose.Sigma_x", FALSE) == FALSE` (**default**):
    - $VE[, , i] <- SE[, , i] \times SE[, , i]$ 
  - **if** `getOption("PCMBase.Transpose.Sigma_x", FALSE) == TRUE`:
    - $VE[, , i] \leftarrow t(SE[, , i]) \times SE[, , i]$

  Note that the above behavior is consistent with the treatment of the model parameters $\Sigma_x$, $\Sigma e_x$ and $\Sigma maj_x$, which are also specified as upper triangular factors. Default: $\text{matrix}(0,0, \text{PCMNumTraits(model), PCMTreeNumTips(tree)})$. 

metaI

a list returned from a call to PCMInfo(X, tree, model, SE), containing metadata such as N, M and k. Alternatively, this can be a character string naming a function or a function object that returns such a list, e.g. the functionPCMInfo or the functionPCMInfoCpp from the PCMBaseCpp package.

positiveValueGuard

positive numerical value (default Inf), which serves as a guard for numerical error. Values exceeding this positiveGuard are most likely due to numerical error and PCMOptions()$PCMBase.Value.NA is returned instead.

Details

It is possible to specify a function for the argument metaI. This function should have three parameters (X, tree, model) and should return a metaInfo object. (see PCMInfo).

Value

a function of a numerical vector parameter called p returning the likelihood of X given the tree and the model with parameter values specified by p.

---

**PCMDefaultModelTypes**  

Class names for the the default PCM and MGPM model types

**Description**

Utility functions returning named character vector of the model class-names for the default model types used for PCM and MixedGaussian model construction.

**Usage**

PCMDefaultModelTypes()

MGPMDefaultModelTypes()

**Value**

Both, PCMDefaultModelTypes and MGPMDefaultModelTypes return a character string vector with named elements (A,B,C,D,E,F) defined as follows (Mitov et al. 2019a):

A. BM (H = 0, diagonal Σ): BM, uncorrelated traits.
B. BM (H = 0, symmetric Σ): BM, correlated traits.
C. OU (diagonal H, diagonal Σ): OU, uncorrelated traits.
D. OU (diagonal H, symmetric Σ): OU, correlated traits, but simple (diagonal) selection strength matrix.
E. OU (symmetric H, symmetric Σ): An OU with nondiagonal symmetric H and nondiagonal symmetric Σ.
F. OU (asymmetric H, symmetric Σ): An OU with nondiagonal asymmetric H and nondiagonal symmetric Σ.
The only difference between the two functions is that the model types returned by `PCMFDefaultModelTypes` have a global parameter $X_0$, while the model types returned by `MGPMFDefaultModelTypes` have an omitted parameter $X_0$.

References


See Also

`Args_MixedGaussian_MGPMDefaultModelTypes`

---

### PCMDefaultObject

**Generate a default object of a given PCM model type or parameter type**

#### Description

This is an S3 generic. See, e.g. ‘PCMDefaultObject.MatrixParameter’.

#### Usage

```r
PCMDefaultObject(spec, model, ...)
```

#### Arguments

- `spec` any object having a class attribute. The value of this object is not used, but its class is used for method-dispatch.
- `model` a PCM object used to extract attributes needed for creating a default object of class specified in `class(spec)`, such as the number of traits (k) or the regimes and the number of regimes;
- `...` additional arguments that can be used by methods.

#### Value

a parameter or a PCM object.
**PCMDescibe**  
*Human friendly description of a PCM*

**Description**  
Human friendly description of a PCM

**Usage**  
PCMDescibe(model, ...)

**Arguments**  
- **model**  
a PCM model object
- **...**  
additional arguments used by implementing methods.

**Details**  
This S3 generic function is intended to be specified for user models

**Value**  
a character string

---

PCMDescibeParameters  
*Describe the parameters of a PCM*

**Description**  
This is an S3 generic.

**Usage**  
PCMDescibeParameters(model, ...)

**Arguments**  
- **model**  
a PCM object.
- **...**  
additional arguments that can be used by implementing methods.

**Value**  
a named list with character elements corresponding to each parameter.
**PCMEextractDimensions**  
Given a PCM or a parameter object, extract an analogical object for a subset of the dimensions (traits) in the original object.

**Description**

Given a PCM or a parameter object, extract an analogical object for a subset of the dimensions (traits) in the original object.

**Usage**

PCMEextractDimensions(obj, dims = seq_len(PCMNumTraits(obj)), nRepBlocks = 1L)

**Arguments**

- **obj**: a PCM or a parameter object.
- **dims**: an integer vector; should be a subset or equal to seq_len(PCMNumTraits(obj)) (the default).

**nRepBlocks**: a positive integer specifying if the specified dimensions should be replicated to obtain a higher dimensional model, where the parameter matrices are block-diagonal with blocks corresponding to dims. Default: 1L.

**Details**

This is an S3 generic.

**Value**

an object of the same class as obj with a subset of obj’s dimensions multiplied nRepBlocks times.

---

**PCMEextractRegimes**  
Given a PCM or a parameter object, extract an analogical object for a subset of the regimes in the original object.

**Description**

Given a PCM or a parameter object, extract an analogical object for a subset of the regimes in the original object.

**Usage**

PCMEextractRegimes(obj, regimes = seq_len(PCMNumRegimes(obj)))
Arguments

- `obj`: a PCM or a parameter object.
- `regimes`: an integer vector; should be a subset or equal to `seq_len(PCMNumRegimes(obj))` (the default).

Details

This is an S3 generic

Value

- an object of the same class as `obj` with a subset of `obj`'s regimes

---

**Description**

Find the S3 method for a given PCM object or class-name and an S3 generic

**Usage**

```r
PCMFindMethod(x, method = "PCMCond")
```

**Arguments**

- `x`: a character string denoting a PCM S3 class name (e.g. "OU"), or a PCM object.
- `method`: a character string denoting the name of an S3 generic function. Default: "PCM-Cond".

**Value**

- a function object corresponding to the S3 method found or an error is raised if no such function is found for the specified object and method.
**PCMFixParameter**  
*Fix a parameter in a PCM model*

**Description**

Fix a parameter in a PCM model

**Usage**

```r
PCMFixParameter(model, name)
```

**Arguments**

- **model**  
a PCM object

- **name**  
a character string

**Value**

A copy of the model with added class '_Fixed' to the class of the parameter name

---

**PCMGenerateModelTypes**  
*Generate default model types for given PCM base-classes*

**Description**

This function calls 'PCMListParameterizations' or 'PCMListDefaultParameterizations' and generates the corresponding 'PCMParentClasses' and 'PCMSpecify' methods in the global environment.

**Usage**

```r
PCMGenerateModelTypes(
    baseTypes = list(BM = "default", OU = "default", White = "all"),
    sourceFile = NULL
)
```

**Arguments**

- **baseTypes**  
a named list with character string elements among c("default", "all") and names specifying base S3-class names for which the parametrizations (subclasses) will be generated. Defaults to list(BM="default", OU = "default", White = "all"). The element value specifies which one of 'PCMListParameterizations' or 'PCMListDefaultParameterizations' should be used:

  - "all" for calling 'PCMListParameterizations'
  - "default" for calling 'PCMListDefaultParameterizations'
Source file NULL or a character string indicating a .R filename, to which the automatically generated code will be saved. If NULL (the default), the generated source code is evaluated and the S3 methods are defined in the global environment. Default: NULL.

Value

This function has side effects only and does not return a value.

See Also

PCMListDefaultParameterizations

Description

A parameterization of a PCM of given type, e.g. OU, is a PCM-class inheriting from this type, which imposes some restrictions or transformations of the parameters in the base-type. This function generates the S3 methods responsible for creating such parameterizations, in particular it generates the definition of the methods for the two S3 generics ‘PCMParentClasses’ and ‘PCMSpecify’ for all parameterizations specified in the ‘tableParameterizations’ argument.

Usage

PCMGenerateParameterizations(
  model,
  listParameterizations = PCMListParameterizations(model),
  tableParameterizations = PCMTableParameterizations(model, listParameterizations),
  env = .GlobalEnv,
  useModelClassNameForFirstRow = FALSE,
  sourceFile = NULL
)

Arguments

model a PCM object.
listParameterizations a list or a sublist returned by ‘PCMListParameterizations’. Default: ‘PCMListParameterizations(model)’.

tableParameterizations a data.table containing the parameterizations to generate. By default this is generated from ‘listParameterizations’ using a call ‘PCMTableParameterizations(model, listParameterizations)’. If specified by the user, this parameter takes precedence over ‘listParameterizations’ and ‘listParameterizations’ is not used.
**Value**

This function does not return a value. It only has a side effect by defining S3 methods in 'env'.

**Description**

Value of an attribute of an object or values for an attribute found in its members

**Usage**

PCMGetAttribute(name, object, member = "", ...)

**Arguments**

- `name` attribute name.
- `object` a PCM model object or a PCMTree object.
- `member` a member expression. Member expressions are character strings denoting named elements in a list object (see examples). Default: "".
- ... additional arguments passed to MatchListMembers.

**Value**

if member is an empty string, attr(object, name). Otherwise, a named list containing the value for the attribute for each member in object matched by member.

**Examples**

PCMGetAttribute("class", PCMBaseTestObjects$model_MixedGaussian_ab)
PCMGetAttribute("dim", PCMBaseTestObjects$model_MixedGaussian_ab, member = "$Sigmae_x")
**PCMGetVecParamsRegimesAndModels**

*Get a vector of all parameters (real and discrete) describing a model on a tree including the numerical parameters of each model regime, the integer ids of the splitting nodes defining the regimes on the tree and the integer ids of the model types associated with each regime.*

---

**Description**

Get a vector of all parameters (real and discrete) describing a model on a tree including the numerical parameters of each model regime, the integer ids of the splitting nodes defining the regimes on the tree and the integer ids of the model types associated with each regime.

**Usage**

```r
PCMGetVecParamsRegimesAndModels(model, tree, ...)
```

**Arguments**

- `model`: a PCM model
- `tree`: a phylo object with an edge.part member.
- `...`: additional parameters passed to methods.

**Details**

This is an S3 generic. In the default implementation, the last entry in the returned vector is the number of numerical parameters. This is used to identify the starting positions in the vector of the first splitting node.

**Value**

A numeric vector concatenating the result

---

**PCMInfo**

*Meta-information about a tree and trait data associated with a PCM*

---

**Description**

This function pre-processes the given tree and data in order to create meta-information used during likelihood calculation.
Usage

PCMInfo(
  X,
  tree,
  model,
  SE = matrix(0, PCMNumTraits(model), PCMTreeNumTips(tree)),
  verbose = FALSE,
  preorder = NULL,
  ...
)

Arguments

X  a k x N numerical matrix with possible NA and NaN entries. For i=1, ..., N, the column i of X contains the measured trait values for species i (the tip with integer identifier equal to i in tree). Missing values can be either not-available (NA) or not existing (NaN). These two values are treated differently when calculating likelihoods (see PCMPresentCoordinates).

tree  a phylo object with N tips.

model  an S3 object specifying both, the model type (class, e.g. "OU") as well as the concrete model parameter values at which the likelihood is to be calculated (see also Details).

SE  a k x N matrix specifying the standard error for each measurement in X. Alternatively, a k x k x N cube specifying an upper triangular k x k factor of the variance covariance matrix for the measurement error for each tip i=1, ..., N. When SE is a matrix, the k x k measurement error variance matrix for a tip i is calculated as VE[, , i] <- diag(SE[, i] * SE[, i], nrow = k). When SE is a cube, the way how the measurement variance matrix for a tip i is calculated depends on the runtime option PCMBase.Transpose.Sigma_x as follows:

if getOption("PCMBase.Transpose.Sigma_x", FALSE) == FALSE (default):
  VE[, , i] <- SE[, i] %*% t(SE[, , i])

if getOption("PCMBase.Transpose.Sigma_x", FALSE) == TRUE:
  VE[, , i] <- t(SE[, , i]) %*% SE[, , i]

Note that the above behavior is consistent with the treatment of the model parameters Sigma_x, Sigmae_x and Sigmaj_x, which are also specified as upper triangular factors. Default: matrix(0, PCMNumTraits(model), PCMTreeNumTips(tree)).

verbose  logical indicating if some debug-messages should printed.

preorder  an integer vector of row-indices in tree$edge matrix as returned by PCMTreePreorder. This can be given for performance speed-up when several operations needing preorder are executed on the tree. Default : NULL.

...  additional arguments used by implementing methods.

Value

a named list with the following elements:
**PCMLik**

**Description**

The likelihood of a PCM represents the probability density function of observed trait values (data) at the tips of a tree given the tree and the model parameters. Seen as a function of the model parameters, the likelihood is used to fit the model to the observed trait data and the phylogenetic tree (which is typically inferred from another sort of data, such as an alignment of genetic sequences for the species at the tips of the tree). The `PCMLik` function provides a common interface for calculating the (log-)likelihood of different PCMs. Below we denote by N the number of tips, by M the total number of nodes in the tree including tips, internal and root node, and by k - the number of traits.

**Usage**

```r
PCMLik(
  X,
  tree,
  model,
  SE = matrix(0, PCMNumTraits(model), PCMTreeNumTips(tree)),
  metaI = PCMInfo(X = X, tree = tree, model = model, SE = SE, verbose = verbose),
  log = TRUE,
  verbose = FALSE
)
```
Arguments

- **X**: a \( k \times N \) numerical matrix with possible NA and NaN entries. For \( i = 1, \ldots, N \), the column \( i \) of \( X \) contains the measured trait values for species \( i \) (the tip with integer identifier equal to \( i \) in tree). Missing values can be either not-available (NA) or not existing (NaN). These two values are treated differently when calculating likelihoods (see `PCMPresentCoordinates`).

- **tree**: a phylo object with \( N \) tips.

- **model**: an S3 object specifying both, the model type (class, e.g. "OU") as well as the concrete model parameter values at which the likelihood is to be calculated (see also Details).

- **SE**: a \( k \times N \) matrix specifying the standard error for each measurement in \( X \). Alternatively, a \( k \times k \times N \) cube specifying an upper triangular \( k \times k \) factor of the variance covariance matrix for the measurement error for each tip \( i = 1, \ldots, N \). When \( SE \) is a matrix, the \( k \times k \) measurement error variance matrix for a tip \( i \) is calculated as \( VE[, , i] <- \text{diag}(SE[, i] \times SE[, i], nrow = k) \). When \( SE \) is a cube, the way how the measurement variance matrix for a tip \( i \) is calculated depends on the runtime option `PCMBase.Transpose.Sigma_x` as follows:

  - If `getOption("PCMBase.Transpose.Sigma_x", FALSE) == FALSE` (default):
    \[
    VE[, , i] <- SE[, , i] %*% t(SE[, , i])
    \]
  - If `getOption("PCMBase.Transpose.Sigma_x", FALSE) == TRUE`:
    \[
    VE[, , i] <- t(SE[, , i]) %*% SE[, , i]
    \]

  Note that the above behavior is consistent with the treatment of the model parameters \( \Sigma_x \), \( \Sigma_{ae} \) and \( \Sigma_{aj} \), which are also specified as upper triangular factors. Default: \( \text{matrix}(0, 0, \text{PCMNumTraits(model)}, \text{PCMTreeNumTips(tree)}) \).

- **metaI**: a list returned from a call to `PCMInfo(X, tree, model, SE)`, containing metadata such as \( N, M \) and \( k \). Alternatively, this can be a character string naming a function or a function object that returns such a list, e.g. the function `PCMInfo` or the function `PCMInfoCpp` from the `PCMBaseCpp` package.

- **log**: logical indicating whether a log-likelihood should be calculated. Default is `TRUE`.

- **verbose**: logical indicating if some debug-messages should printed.

Details

For efficiency, the argument `metaI` can be provided explicitly, because this is not supposed to change during a model inference procedure such as likelihood maximization.

Value

A numerical value with named attributes as follows:

- **X0**: A numerical vector of length \( k \) specifying the value at the root for which the likelihood value was calculated. If the model contains a member called \( X0 \), this vector is used; otherwise the value of \( X0 \) maximizing the likelihood for the given model parameters is calculated by maximizing the quadratic polynomial \( 'X0 * L_root * X0 + m_root * X0 + r_root' \);
**error** A character string with information if a numerical or other logical error occurred during likelihood calculation.

If an error occurred during likelihood calculation, the default behavior is to return NA with a non-NULL error attribute. This behavior can be changed in using global options:

"PCMBASE.VALUE.NA" Allows to specify a different NA value such as -Inf or -1e20 which can be used in combination with log = TRUE when using optim to maximize the log-likelihood;

"PCMBASE.ERRORS.AS.WARNINGS" Setting this option to FALSE will cause any error to result in calling the stop R-base function. If not caught in a tryCatch, this will cause the inference procedure to abort at the occurrence of a numerical error. By default, this option is set to TRUE, which means that getOption("PCMBASE.VALUE.NA", as.double(NA)) is returned with an error attribute and a warning is issued.

See Also

PCIMinfo PCIMABcDEf PCIMlrm PCIMSim PCIMCond

Examples

```r
N <- 10
to <- PCMTree(ape::rtree(N))

model <- PCMBaseTestObjects$model_MixedGaussian_ab

PCMTreeSetPartRegimes(to, c('11' = 'a'), setPartition = TRUE)

set.seed(1, kind = "Mersenne-Twister", normal.kind = "Inversion")
X <- PCMSim(to, model, X0 = rep(0, 3))

PCMLik(X, to, model)
```

**PCMLikDmvNorm**  
Calculate the likelihood of a model using the standard formula for multivariate pdf

Description

Calculate the likelihood of a model using the standard formula for multivariate pdf

Usage

```r
PCMLikDmvNorm(
  X,
  tree,
  model,
  SE = matrix(0, PCMNumTraits(model), PCMTreeNumTips(tree)),
  metaI = PCMInfo(X, tree, model, SE, verbose = verbose),
  log = TRUE,
  verbose = FALSE
)
```
Arguments

X  a k x N numerical matrix with possible NA and NaN entries. For i=1,..., N, the column i of X contains the measured trait values for species i (the tip with integer identifier equal to i in tree). Missing values can be either not-available (NA) or not existing (NaN). These two values are treated differently when calculating likelihoods (see PCMPresentCoordinates).

tree  a phylo object with N tips.

model  an S3 object specifying both, the model type (class, e.g. "OU") as well as the concrete model parameter values at which the likelihood is to be calculated (see also Details).

SE  a k x N matrix specifying the standard error for each measurement in X. Alternatively, a k x k x N cube specifying an upper triangular k x k factor of the variance covariance matrix for the measurement error for each tip i=1,..., N. When SE is a matrix, the k x k measurement error variance matrix for a tip i is calculated as VE[, , i] <- diag(SE[, i] * SE[, i], nrow = k). When SE is a cube, the way how the measurement variance matrix for a tip i is calculated depends on the runtime option PCMBase.Transpose.Sigma_x as follows:

if getOption("PCMBase.Transpose.Sigma_x", FALSE) == FALSE (default):
  VE[, , i] <- SE[, , i] %*% t(SE[, , i])
if getOption("PCMBase.Transpose.Sigma_x", FALSE) == TRUE:  VE[, , i]<- t(SE[, , i]) %*% SE[, , i]

Note that the above behavior is consistent with the treatment of the model parameters Sigma_x, Sigmae_x and Sigmaj_x, which are also specified as upper triangular factors. Default: matrix(0.0, PCMNumTraits(model), PCMTreeNumTips(tree)).

metaI  a list returned from a call to PCMInfo(X, tree, model, SE), containing metadata such as N, M and k. Alternatively, this can be a character string naming a function or a function object that returns such a list, e.g. the function PCMInfo or the function PCMInfoCpp from the PCMBaseCpp package.

log  logical indicating whether a log-likelehood should be calculated. Default is TRUE.

verbose  logical indicating if some debug-messages should printed.

Value

a numerical value with named attributes as follows:

PCMLikTrace  Tracing the log-likelihood calculation of a model over each node of the tree

Description

This is an S3 generic function providing tracing information for the likelihood calculation for a given tree, data and model parameters. Useful for illustration or for debugging purpose.
Usage

PCMLikTrace(
  X,
  tree,
  model,
  SE = matrix(0, PCMNumTraits(model), PCMTreeNumTips(tree)),
  metaI = PCMInfo(X = X, tree = tree, model = model, SE = SE, verbose = verbose),
  log = TRUE,
  verbose = FALSE
)

Arguments

X a k x N numerical matrix with possible NA and NaN entries. For i=1, ..., N, the column i of X contains the measured trait values for species i (the tip with integer identifier equal to i in tree). Missing values can be either not-available (NA) or not existing (NaN). These two values are treated differently when calculating likelihoods (see PCMPresentCoordinates).

tree a phylo object with N tips.

model an S3 object specifying both, the model type (class, e.g. "OU") as well as the concrete model parameter values at which the likelihood is to be calculated (see also Details).

SE a k x N matrix specifying the standard error for each measurement in X. Alternatively, a k x k x N cube specifying an upper triangular k x k factor of the variance covariance matrix for the measurement error for each tip i=1, ..., N. When SE is a matrix, the k x k measurement error variance matrix for a tip i is calculated as VE[, , i] <- diag(SE[, i] * SE[, i], nrow = k). When SE is a cube, the way how the measurement variance matrix for a tip i is calculated depends on the runtime option PCMBase.Transpose.Sigma_x as follows:

ifgetOption("PCMBase.Transpose.Sigma_x", FALSE) == FALSE (default): VE[, , i] <- SE[, , i] %*% t(SE[, , i])
ifgetOption("PCMBase.Transpose.Sigma_x", FALSE) == TRUE: VE[, , i] <- t(SE[, , i]) %*% SE[, , i]

Note that the above behavior is consistent with the treatment of the model parameters Sigma_x, Sigmae_x and Sigmaj_x, which are also specified as upper triangular factors. Default: matrix(0.0, PCMNumTraits(model), PCMTreeNumTips(tree)).

metaI a list returned from a call to PCMInfo(X, tree, model, SE), containing meta-data such as N, M and k. Alternatively, this can be a character string naming a function or a function object that returns such a list, e.g. the functionPCMInfo or the function PCMInfoCpp from the PCMBaseCpp package.

log logical indicating whether a log-likelihood should be calculated. Default is TRUE.

verbose logical indicating if some debug-messages should printed.
 PCMListMembers

Value
The returned object will, in general, depend on the type of model and the algorithm used for likelihood calculation. For a G_LInv model and pruning-wise likelihood calculation, the returned object will be a data.table with columns corresponding to the node-state variables, e.g. the quadratic polynomial coefficients associated with each node in the tree.

See Also
PCMInfo PCMAbCdEf PCMLmr PCMSim PCMCond PCMParseErrorMessage

---

PCMLListMembers A vector of access-code strings to all members of a named list

Description
A vector of access-code strings to all members of a named list

Usage
PCMLListMembers(
  l,
  recursive = TRUE,
  format = c("$", "$'", "$\"", "$\", "["", "[\[", "[\"")
)

Arguments
1 a named list object.
recursive logical indicating if list members should be gone through recursively. TRUE by default.
format a character string indicating the format for accessing a member. Acceptable values are c("$", "$'", "$\"", "$\", "["", "[\[", "[\") of which the first one is taken as default.

Value
a vector of character strings denoting each named member of the list.

Examples
PCMLListMembers(PCMBaseTestObjects$model_MixedGaussian_ab)
PCMLListMembers(PCMBaseTestObjects$model_MixedGaussian_ab, format = "$\")
PCMLListMembers(PCMBaseTestObjects$tree.ab, format = "$\")
Specify the parameterizations for each parameter of a model

Description

These are S3 generics. 'PCMListParameterizations' should return all possible parametrizations for the class of 'model'. 'PCMListDefaultParameterizations' is a handy way to specify a subset of all parametrizations. 'PCMListDefaultParameterizations' should be used to avoid generating too many model parametrizations which occupy space in the R-global environment while they are not used (see PCMGenerateParameterizations). It is mandatory to implement a specification for 'PCMListParameterizations' for each newly defined class of models. 'PCMListDefaultParameterizations' has a default implementation that calls 'PCMListParameterizations' and returns the first parametrization for each parameter. Hence, implementing a method for 'PCMListDefaultParameterizations' for a newly defined model type is optional.

Usage

PCMListParameterizations(model, ...)

PCMListDefaultParameterizations(model, ...)

Arguments

model a PCM.

... additional arguments used by implementing methods.

Value

a named list with list elements corresponding to each parameter in model. Each list element is a list of character vectors, specifying the possible S3 class attributes for the parameter in question. For an example, type 'PCMListParameterizations.BM' to see the possible parameterizations for the BM model.

See Also

PCMGenerateParameterizations
Description

Quadratic polynomial parameters L, m, r

Usage

```
PCMLmr(
  X,
  tree,
  model,
  SE = matrix(0, PCMNumTraits(model), PCMTreeNumTips(tree)),
  metaI = PCMInfo(X = X, tree = tree, model = model, SE = SE, verbose = verbose),
  root.only = TRUE,
  verbose = FALSE
)
```

Arguments

- `X`: a k x N numerical matrix with possible NA and NaN entries. For i=1, ..., N, the column i of X contains the measured trait values for species i (the tip with integer identifier equal to i in tree). Missing values can be either not-available (NA) or not existing (NaN). These two values are treated differently when calculating likelihoods (see `PCMPresentCoordinates`).
- `tree`: a phylo object with N tips.
- `model`: an S3 object specifying both, the model type (class, e.g. "OU") as well as the concrete model parameter values at which the likelihood is to be calculated (see also Details).
- `SE`: a k x N matrix specifying the standard error for each measurement in X. Alternatively, a k x k x N cube specifying an upper triangular k x k factor of the variance covariance matrix for the measurement error for each tip i=1, ..., N. When SE is a matrix, the k x k measurement error variance matrix for a tip i is calculated as VE[, , i] <- diag(SE[, i] * SE[, i], nrow = k). When SE is a cube, the way how the measurement variance matrix for a tip i is calculated depends on the runtime option `PCMBase.Transpose.Sigma_x` as follows:

  ```r
  if (getOption("PCMBase.Transpose.Sigma_x", FALSE) == FALSE) {
    VE[, , i] <- SE[, , i] %*% t(SE[, , i])
  } else {
    VE[, , i] <- t(SE[, , i]) %*% SE[, , i]
  }
  ```

  Note that the above behavior is consistent with the treatment of the model parameters Sigma_x, Sigmae_x and Sigma_m_x, which are also specified as upper triangular factors. Default: matrix(0,0,PCMNumTraits(model),PCMTreeNumTips(tree)).
metaI

A list returned from a call to PCMIfo(X, tree, model, SE), containing metadata such as N, M and k. Alternatively, this can be a character string naming a function or a function object that returns such a list, e.g. the function PCMIfo or the function PCMIfoCpp from the PCMBaseCpp package.

root.only

logical indicating whether to return the calculated values of L,m,r only for the root or for all nodes in the tree.

verbose

logical indicating if some debug-messages should printed.

Value

A list with the members A,b,C,d,E,f,L,m,r for all nodes in the tree or only for the root if root.only=TRUE.

Description

Integer vector giving the model type index for each regime

Usage

PCMMapModelTypesToRegimes(model, tree, ...)

Arguments

model

a PCM model

tree

a phylo object with an edge.part member

... additional parameters passed to methods

Details

This is a generic S3 method. The default implementation for the basic class PCM returns a vector of 1’s, because it assumes that a single model type is associated with each regime. The implementation for mixed Gaussian models returns the mapping attribute of the MixedGaussian object reordered to correspond to PCMTreeGetPartNames(tree).

Value

an integer vector with elements corresponding to the elements in PCMTreeGetPartNames(tree)
**Description**

Expected mean vector at each tip conditioned on a trait-value vector at the root.

**Usage**

```r
PCMMean(
  tree,
  model,
  X0 = model$X0,
  metaI = PCMInfo(NULL, tree, model, verbose = verbose),
  internal = FALSE,
  verbose = FALSE
)
```

**Arguments**

- `tree` : a phylo object with N tips.
- `model` : an S3 object specifying both, the model type (class, e.g. "OU") as well as the concrete model parameter values at which the likelihood is to be calculated (see also Details).
- `X0` : a k-vector denoting the root trait
- `metaI` : a list returned from a call to `PCMInfo(X, tree, model, SE)`, containing metadata such as N, M and k. Alternatively, this can be a character string naming a function or a function object that returns such a list, e.g. the function `PCMInfo` or the function `PCMInfoCpp` from the PCMBaseCpp package.
- `internal` : a logical indicating if the per-node mean vectors should be returned (see Value). Default FALSE.
- `verbose` : logical indicating if some debug-messages should printed.

**Value**

If `internal` is FALSE (default), then a k x N matrix Mu, such that Mu[, i] equals the expected mean k-vector at tip i, conditioned on X0 and the tree. Otherwise, a k x M matrix Mu containing the mean vector for each node.

**Examples**

```r
# a Brownian motion model with one regime
modelBM <- PCM(model = "BM", k = 2)
# print the model
modelBM
```
PCMMeanAtTime

# assign the model parameters at random: this will use uniform distribution
# with boundaries specified by PCParamLowerLimit and PCParamUpperLimit
# We do this in two steps:
# 1. First we generate a random vector. Note the length of the vector equals PCParamCount(modelBM)
   randomParams <- PCParamRandomVecParams(modelBM, PCNumTraits(modelBM), PCNumRegimes(modelBM))
   randomParams
# 2. Then we load this random vector into the model.
   PCParamLoadOrStore(modelBM, randomParams, 0, PCNumTraits(modelBM), PCNumRegimes(modelBM), TRUE)

# create a random tree of 10 tips
   tree <- ape::rtree(10)
   PCMMean(tree, modelBM)

PCMMeanAtTime(Calculate the mean at time t, given X0, under a PCM model)

Description

Calculate the mean at time t, given X0, under a PCM model

Usage

PCMMeanAtTime(
  t,
  model,
  X0 = model$X0,
  regime = PCMRegimes(model)[1L],
  verbose = FALSE
)

Arguments

  t              positive numeric denoting time
  model          a PCM model object
  X0             a numeric vector of length k, where k is the number of traits in the model (Defaults to model$X0).
  regime         an integer or a character denoting the regime in model for which to do the calculation; Defaults to PCMRegimes(model)[1L], meaning the first regime in the model.
  verbose        a logical indicating if (debug) messages should be written on the console (Defaults to FALSE).

Value

A numeric vector of length k
Examples

# a Brownian motion model with one regime
modelBM <- PCM(model = "BM", k = 2)
# print the model
modelBM
# assign the model parameters at random: this will use uniform distribution
# with boundaries specified by PCMParamLowerLimit and PCMParamUpperLimit
# We do this in two steps:
# 1. First we generate a random vector. Note the length of the vector equals PCMParamCount(modelBM)
randomParams <- PCMParamRandomVecParams(modelBM, PCMNumTraits(modelBM), PCMNumRegimes(modelBM))
randomParams
# 2. Then we load this random vector into the model.
PCMPParamLoadOrStore(modelBM, randomParams, 0, PCMNumTraits(modelBM), PCMNumRegimes(modelBM), TRUE)

# PCMMeanAtTime(1, modelBM)
# note that the variance at time 0 is not the 0 matrix because the model has a non-zero
# environmental deviation
PCMMeanAtTime(0, modelBM)

PCMMModels

Get a list of PCM models currently implemented

Description

Get a list of PCM models currently implemented

Usage

PCMMModels(pattern = NULL, parentClass = NULL, ...)

Arguments

pattern a character string specifying an optional for the model-names to search for.
parentClass a character string specifying an optional parent class of the models to look for.
... additional arguments used by implementing methods.

Details

The function is using the S3 api function methods looking for all registered implementations of the
class PCMSpecify.

Value

a character vector of the model classes found.

Examples

PCMMModels()
PCMMModels(""OU")
**PCMMModelTypes**

*Get the model type(s) of a model*

**Description**

For a regular PCM object, the model type is its S3 class. For a MixedGaussian each regime is mapped to one of several possible model types.

**Usage**

```
PCMMModelTypes(obj)
```

**Arguments**

- `obj` a PCM object

**Value**

a character vector

---

**PCMMNumRegimes**

*Number of regimes in a obj*

**Description**

Number of regimes in a obj

**Usage**

```
PCMMNumRegimes(obj)
```

**Arguments**

- `obj` a PCM object

**Value**

an integer
**PCMNumTraits**

*Number of traits modeled by a PCM*

**Description**

Number of traits modeled by a PCM

**Usage**

`PCMNumTraits(model)`

**Arguments**

- `model` a PCM object or a parameter object (the name of this argument could be misleading, because both, model and parameter objects are supported).

**Value**

an integer

---

**PCMOptions**

*Global options for the PCMBase package*

**Description**

Global options for the PCMBase package

**Usage**

`PCMOptions()`

**Value**

a named list with the currently set values of the following global options:

- `PCMBase.Value.NA` NA value for the likelihood; used in GaussianPCM to return this value in case of an error occurring during likelihood calculation. By default, this is set to `as.double(NA)`.

- `PCMBase.Errors.As.Warnings` a logical flag indicating if errors (occurring, e.g. during likelihood calculation) should be treated as warnings and added as an attribute "error" to attach to the likelihood values. Default TRUE.

- `PCMBase.Raise.Lik.Errors` Should numerical and other sort of errors occurring during likelihood calculation be raised either as errors or as warnings, depending on the option `PCMBase.Errors.As.Warnings`. Default TRUE. This option can be useful if too frequent warnings get raised during a model fit procedure.
PCMBase.Threshold.Lambda_ij a 0-threshold for abs(Lambda_i + Lambda_j), where Lambda_i and Lambda_j are eigenvalues of the parameter matrix H of an OU or other model. Default 1e-8. See PCMPExpxMeanExp.

PCMBase.Threshold.EV A 0-threshold for the eigenvalues of the matrix V for a given branch. The V matrix is considered singular if it has eigenvalues smaller than PCMBase.Threshold.EV or when the ratio min(svdV)/max(svdV) is below PCMBase.Threshold.SV. Default is 1e-5. Treatment of branches with singular V matrix is defined by the option PCMBase.Skip.Singular.

PCMBase.Threshold.SV A 0-threshold for min(svdV)/max(svdV), where svdV is the vector of singular values of the matrix V for a given branch. The V matrix is considered singular if it has eigenvalues smaller than PCMBase.Threshold.EV or when the ratio min(svdV)/max(svdV) is below PCMBase.Threshold.SV. Default is 1e-6. Treatment of branches with singular V matrix is defined by the option PCMBase.Skip.Singular.

PCMBase.Threshold.Skip.Singular A double indicating if an internal branch of shorter length with singular matrix V should be skipped during likelihood calculation. Setting this option to a higher value, together with a TRUE value for the option PCMBase.Skip.Singular will result in tolerating some parameter values resulting in singular variance covariance matrix of the transition distribution. Default 1e-4.

PCMBase.Skip.Singular A logical value indicating whether internal branches with singular matrix V and shorter than getOption("PCMBase.Threshold.Skip.Singular") should be skipped during likelihood calculation, adding their children L,m,r values to their parent node. Default TRUE. Note, that setting this option to FALSE may cause some models to stop working, e.g. the White model. Setting this option to FALSE will also cause errors or NA likelihood values in the case of trees with very short or 0-length branches.

PCMBase.Tolerance.Symmetric A double specifying the tolerance in tests for symmetric matrices. Default 1e-8; see also isSymmetric.

PCMBase.Lmr.mode An integer code specifying the parallel likelihood calculation mode.

PCMBase.ParamValue.LowerLimit Default lower limit value for parameters, default setting is -10.0.

PCMBase.ParamValue.LowerLimit.NonNegative Numeric (default: 0.0) indication the lower limit for parameters inheriting from class 'NonNegative's

PCMBase.ParamValue.LowerLimit.NonNegativeDiagonal Default lower limit value for parameters corresponding to non-negative diagonal elements of matrices, default setting is 0.0.

PCMBase.ParamValue.UpperLimit Default upper limit value for parameters, default setting is 10.0.

PCMBase.Transpose.Sigma_x Should upper diagonal factors for variance-covariance rate matrices be transposed, e.g. should Sigma = t(Sigma_x) Sigma_x or, rather Sigma = Sigma_x t(Sigma_x)? Note that the two variants are not equal. The default is FALSE, meaning Sigma = Sigma_x t(Sigma_x). In this case, Sigma_x is not the actual upper Cholesky factor of Sigma, i.e. chol(Sigma) != Sigma_x. See also chol and UpperTriFactor. This option applies to parameters Sigma_x, Sigma_e_x, Sigma_j_x and the measurement errors SE[,,i] for each measurement i when the argument SE is specified as a cube.

PCMBase.MaxLengthListCladePartitions Maximum number of tree partitions returned by PCMTreeListCladePartitions. This option has the goal to interrupt the recursive search for new partitions in the case of calling PCMTreeListCladePartitions on a big tree with a small value of the maxCladeSize argument. By default this is set to Inf.
PCMPairSums

**Description**

Sums of pairs of elements in a vector

**Usage**

`PCMPairSums(lambda)`

**Arguments**

- `lambda` a numeric vector

**Value**

A squared symmetric matrix with `elem_{ij} = lambda_i + lambda_j`. 

---

PCMPresentCoordinatesFun

A function with the same synopsis as `PCMPresentCoordinates` that can be specified in case of custom setting for the present coordinates for specific nodes of the tree. See `PCMPresentCoordinates` and `PCMInfo`.

PCMBase.Use1DClasses

Logical indicating if 1D arithmetic operations should be used instead of multi-dimensional ones. This can speed-up computations in the case of a single trait. Currently, this feature is implemented only in the PCMBaseCpp R-package and only for some model types, such as OU and BM. Default: FALSE

PCMBase.PrintSubscript_u

Logical indicating if a subscript 'u' should be printed instead of a subscript 'x'. Used in PCMTable. Default: FALSE.

PCMBase.MaxNForGuessSigma_x

A real fraction number in the interval (0, 1) or an integer bigger than 1 controlling the number of tips to use for analytical calculation of the evolutionary rate matrix under a BM assumption. This option is used in the suggested PCMFit R-package. Default: 0.25.

PCMBase.UsePCMVarForVCV

Logical (default: FALSE) indicating if the function `PCMTreeVCV` should use `PCMVar` instead of ape’s function `vcv` to calculate the phylogenetic variance covariance matrix under BM assumption. Note that setting this option to TRUE would slow down the function PCMTreeVCV considerably but may be more stable, particularly in the case of very big and deep trees, where previous ape’s versions of the `vcv` function have thrown stack-overflow errors.

**Examples**

`PCMOptions()`
Module PCParam

Description

Global and S3 generic functions for manipulating model parameters. The parameters in a PCM are named objects with a class attribute specifying the main type and optional properties (tags).

S3 generic functions:

- **PCParamCount()** Counting the number of actual numeric parameters (used, e.g. for calculating information scores, e.g. AIC);
- **PCParamLoadOrStore(), PCParamLoadOrStore()** Storing/loading a parameter to/from a numerical vector;
- **PCParamLowerLimit(), PCParamUpperLimit()** Specifying parameter upper and lower limits;
- **PCParamRandomVecParams()** Generating a random parameter vector;

For all the above properties, check-functions are defined, e.g. `is.Local(o)`, `is.Global(o)`, `is.ScalarParameter(o)`, `is.VectorParameter`, etc.

---

PCParamBindRegimeParams

*Bind named vectors or matrices into an array so that the names form the names of the last dimension.*

Description

Bind named vectors or matrices into an array so that the names form the names of the last dimension.

Usage

PCParamBindRegimeParams(...)

Arguments

... Any number of named vectors, or matrices. The dimensions of all the arrays must match. The names will be used for the names of the regimes.

Value

an array with dim attribute one longer than the number of dimensions of each argument, i.e. if there are 5 vector arguments of length 2, the returned value will be an array with dim c(2,5); if there are 5 matrix arguments of dim 2 x 2, the returned value will be an array with dim c(2,2,5).
Examples

# regimes

# in regime 'a' the three traits evolve according to three independent OU processes
a.X0 <- c(5, 2, 1)
a.H <- rbind(
  c(0, 0, 0),
  c(0, 2, 0),
  c(0, 0, 3))
a.Theta <- c(10, 6, 2)
a.Sigma_x <- rbind(
  c(1.6, 0.0, 0.0),
  c(0.0, 2.4, 0.0),
  c(0.0, 0.0, 2.0))
a.Sigmae_x <- rbind(
  c(0.0, 0.0, 0.0),
  c(0.0, 0.0, 0.0),
  c(0.0, 0.0, 0.0))
a.h_drift<-c(0, 0, 0)

# in regime 'b' there is correlation between the traits
b.X0 <- c(12, 4, 3)
b.H <- rbind(
  c(2.0, 0.1, 0.2),
  c(0.1, 0.6, 0.2),
  c(0.2, 0.2, 0.3))
b.Theta <- c(10, 6, 2)
b.Sigma_x <- rbind(
  c(1.6, 0.3, 0.3),
  c(0.0, 0.3, 0.4),
  c(0.0, 0.0, 2.0))
b.Sigmae_x <- rbind(
  c(0.2, 0.0, 0.0),
  c(0.0, 0.3, 0.0),
  c(0.0, 0.0, 0.4))
b.h_drift<-c(1, 2, 3)

H <- PCMParamBindRegimeParams(a = a.H, b = b.H)
Theta <- PCMParamBindRegimeParams(a = a.Theta, b = b.Theta)
Sigma_x <- PCMParamBindRegimeParams(a = a.Sigma_x, b = b.Sigma_x)
Sigmae_x <- PCMParamBindRegimeParams(a = a.Sigmae_x, b = b.Sigmae_x)
h_drift <- PCMParamBindRegimeParams(a = a.h_drift, b = b.h_drift)

model.a.BM_drift.123 <- PCM("BM_drift", k = 3, regimes = "a",
params = list(
  X0 = a.X0,
  h_drift = h_drift[, 1],
  Sigma_x = Sigma_x[, , 1],
  Sigmae_x = Sigmae_x[, , 1]))

# regimes 'a' and 'b', traits 1, 2 and 3
model.ab.123 <- PCM("OU", k = 3, regimes = c("a", "b"),
params = list(
  X0 = a.X0,
  H = H[,drop=FALSE],
  Theta = Theta[,drop=FALSE],
  Sigma.x = Sigma.x[,drop=FALSE],
  Sigma.e.x = Sigma.e.x[,drop=FALSE])

### PCMPParamCount

**Description**

Count the number of free parameters associated with a PCM or a PCM-parameter

**Usage**

```r
PCMPParamCount(
  o,
  countRegimeChanges = FALSE,
  countModelTypes = FALSE,
  offset = 0L,
  k = 1L,
  R = 1L,
  parentModel = NULL
)
```

**Arguments**

- `o` a PCM model object or a parameter of a PCM object
- `countRegimeChanges` logical indicating if regime changes should be counted. If TRUE, the default implementation would add `PCMNumRegimes(model) - 1`. Default FALSE.
- `countModelTypes` logical indicating whether the model type should be counted. If TRUE the default implementation will add +1 only if there are more than one modelTypes (length(attr(model, "modelTypes", exact = TRUE)) > 1), assuming that all regimes are regimes of the same model type (e.g. OU). The implementation for MRG models will add +1 for every regime if there are more than one modelTypes. Default FALSE.
- `offset` an integer denoting an offset count from which to start counting (internally used). Default: 0.
- `k` an integer denoting the number of modeled traits. Default: 1.
- `R` an integer denoting the number of regimes in the model. Default: 1.
- `parentModel` NULL or a PCM object. Default: NULL.
**PCMPParamLoadOrStore**

*Load (or store) a PCM parameter from (or to) a vector of the variable parameters in a model.*

**Description**

Load (or store) a PCM parameter from (or to) a vector of the variable parameters in a model.

**Usage**

PCMPParamLoadOrStore(o, vecParams, offset, k, R, load, parentModel = NULL)
**Arguments**

- **o**: a PCM model object or a parameter of a PCM object
- **vecParams**: a numeric vector.
- **offset**: an integer denoting an offset count from which to start counting (internally used). Default: 0.
- **k**: an integer denoting the number of modeled traits. Default: 1.
- **R**: an integer denoting the number of regimes in the model. Default: 1.
- **load**: logical indicating if parameters should be loaded from vecParams into o (TRUE) or stored to vecParams from o (FALSE).
- **parentModel**: NULL or a PCM object. Default: NULL.

**Details**

This S3 generic function has both, a returned value and side effects.

**Value**

an integer equaling the number of elements read from vecParams. In the case of type="custom", the number of indices bigger than offset returned by the function indices(offset, k).

---

**Description**

Locate a named parameter in the short vector representation of a model

**Usage**

`PCMParamLocateInShortVector(o, accessExpr, enclos = "?")`

**Arguments**

- **o**: a PCM model object.
- **accessExpr**: a character string used to access the parameter, e.g. "$Theta[,1]" or "[['Theta']][,1]".
- **enclos**: a character string containing the symbol '?', e.g. 'diag(?). The meaning of this symbol is to be replaced by the matching accessExpr (see examples). Default value: '?'.

**Value**

an integer vector of length `PCMParamCount(o)` with NAs everywhere except at the coordinates corresponding to the parameter in question.
Examples

```r
model <- PCM(PCMDefaultModelTypes()[, "D"], k = 3, regimes = c("a", "b"))
# The parameter H is a diagonal 3x3 matrix. If this matrix is considered as
# a vector the indices of its diagonal elements are 1, 5 and 9. These indices
# are indicated as the non-NA entries in the returned vector.
PCMPParamLocateInShortVector(model, "$H[,,1]"
PCMPParamLocateInShortVector(model, "$H[,,a]"
PCMPParamLocateInShortVector(model, "$H[,,b]"
PCMPParamLocateInShortVector(model, "$Sigma_x[,,b]", enclos = 'diag(?))
PCMPParamLocateInShortVector(model, "$Sigma_x[,,b]", enclos = '?[upper.tri(?)]')
```

**PCMPParamLowerLimit**  
*The lower limit for a given model or parameter type*

**Description**

This is an S3 generic function.

**Usage**

```r
PCMPParamLowerLimit(o, k, R, ...)
```

**Arguments**

- `o` an object such as a VectorParameter a MatrixParameter or a PCM.
- `k` integer denoting the number of traits
- `R` integer denoting the number of regimes in the model in which `o` belongs to.
- `...` additional arguments (optional or future use).

**Value**

an object of the same S3 class as `o` representing a lower limit for the class.

**PCMPParamRandomVecParams**

*Generate a random parameter vector for a model using uniform distribution between its lower and upper bounds.*

**Description**

Generate a random parameter vector for a model using uniform distribution between its lower and upper bounds.
Usage

\begin{verbatim}
PCMPParamRandomVecParams(
    o,
    k,
    R,
    n = 1L,
    argsPCMPParamLowerLimit = NULL,
    argsPCMPParamUpperLimit = NULL
)
\end{verbatim}

Arguments

- \textit{o} a PCM model object or a parameter
- \textit{k} integer denoting the number of traits.
- \textit{R} integer denoting the number of regimes.
- \textit{n} an integer specifying the number of random vectors to generate
- \text{argsPCMPParamLowerLimit, argsPCMPParamUpperLimit} named lists of arguments passed to PCMPParamLowerLimit and PCMPParamUpperLimit.

Value

a numeric matrix of dimension \( n \times \text{PCMPParamCount}(o) \).

See Also

PCMPParamLimits PCMPParamGetShortVector

---

**PCMPParamSetByName**

Set model parameters from a named list

Description

Set model parameters from a named list

Usage

\begin{verbatim}
PCMPParamSetByName(
    model,
    params,
    inplace = TRUE,
    replaceWholeParameters = FALSE,
    deepCopySubPCMs = FALSE,
    failIfNamesInParamsDontExist = TRUE,
    ...
)
\end{verbatim}
Arguments

- **model**: a PCM model object
- **params**: a named list with elements among the names found in model
- **inplace**: logical indicating if the parameters should be set "inplace" for the model object in the calling environment or a new model object with the parameters set as specified should be returned. Defaults to TRUE.
- **replaceWholeParameters**: logical, by default set to FALSE. If TRUE, the parameters will be completely replaced, meaning that their attributes (e.g. S3 class) will be replaced as well (dangerous).
- **deepCopySubPCMs**: a logical indicating whether nested PCMs should be 'deep'-copied, meaning element by element, eventually preserving the attributes as in model. By default this is set to FALSE, meaning that sub-PCMs found in params will completely overwrite the sub-PCMs with the same name in model.
- **failIfNamesInParamsDontExist**: logical indicating if an error should be raised if params contains elements not existing in model. Default: TRUE.

Value

If inplace is TRUE, the function only has a side effect of setting the parameters of the model object in the calling environment; otherwise the function returns a modified copy of the model object.

---

### PCMParamType

**Parameter types**

---

**Description**

The parameter types are divided in the following categories:

- **Main type**: These are the "ScalarParameter", "VectorParameter" and "MatrixParameter" classes. Each model parameter must have a main type.
- **Scope/Omission**: These are the "_Global" and "_Omitted" classes. Every parameter can be global for all regimes or local for a single regime. If not specified, local scope is assumed. In some special cases a parameter (e.g. Sigmae can be omitted from a model. This is done by adding "_Omitted" to its class attribute.
- **Constancy (optional)**: These are the "_Fixed", "_Ones", "_Identity" and "_Zeros" classes.
- **Transformation (optional)**: These are the "_Transformable", "_CholeskyFactor" and "_Schur" classes.
- **Other properties (optional)**: These are the "_NonNegative", "_WithNonNegativeDiagonal", "_LowerTriangular", "_AllEqual", "_ScalarDiagonal", "_Symmetric", "_UpperTriangular", "_LowerTriangularWithDiagonal" and "_UpperTriangularWithDiagonal" classes.
Usage

is.Local(o)

is.Global(o)

is.ScalarParameter(o)

is.VectorParameter(o)

is.MatrixParameter(o)

is.WithCustomVecParams(o)

is.Fixed(o)

is.Zeros(o)

is.Ones(o)

is.Identity(o)

is.AllEqual(o)

is.NonNegative(o)

is.Diagonal(o)

is.ScalarDiagonal(o)

is.Symmetric(o)

is.UpperTriangular(o)

is.UpperTriangularWithDiagonal(o)

is.WithNonNegativeDiagonal(o)

is.LowerTriangular(o)

is.LowerTriangularWithDiagonal(o)

is.Omitted(o)

is.CholeskyFactor(o)

is.Schur(o)

is.Transformable(o)
is.Transformed(o)

is.SemiPositiveDefinite(o)

**Arguments**

- **o** an object, i.e. a PCM or a parameter object.

**Value**

logical indicating if the object passed is from the type appearing in the function-name.

**Functions**

- is.Local:
- is.Global:
- is.ScalarParameter:
- is.VectorParameter:
- is.MatrixParameter:
- is.WithCustomVecParams:
- is.Fixed:
- is.Zeros:
- is.Ones:
- is.Identity:
- is.AllEqual:
- is.NonNegative:
- is.Diagonal:
- is.ScalarDiagonal:
- is.Symmetric:
- is.UpperTriangular:
- is.UpperTriangularWithDiagonal:
- is.WithNonNegativeDiagonal:
- is.LowerTriangular:
- is.LowerTriangularWithDiagonal:
- is.Omitted:
- is.CholeskyFactor:
- is.Schur:
- is.Transformable:
- is.Transformed:
- is.SemiPositiveDefinite:
**PCMPParamUpperLimit**

*The upper limit for a given model or parameter type*

**Description**

This is an S3 generic function.

**Usage**

PCMPParamUpperLimit(o, k, R, ...)

**Arguments**

- `o`: an object such as a VectorParameter a MatrixParameter or a PCM.
- `k`: integer denoting the number of traits
- `R`: integer denoting the number of regimes in the model in which `o` belongs to.
- `...`: additional arguments (optional or future use).

**Value**

an object of the same S3 class as `o` representing an upper limit for the class.

---

**PCMParentClasses**

*Parent S3 classes for a model class*

**Description**

Parent S3 classes for a model class

**Usage**

PCMParentClasses(model)

**Arguments**

- `model`: an S3 object.

**Details**

This S3 generic function is intended to be specified for user models. This function is called by the `PCM.character` method to determine the parent classes for a given model class.

**Value**

a vector of character string denoting the names of the parent classes
PCMParseErrorMessage  Extract error information from a formatted error message.

**Description**

Extract error information from a formatted error message.

**Usage**

```r
PCMParseErrorMessage(x)
```

**Arguments**

- `x` character string representing the error message.

**Value**

Currently the function returns `x` unchanged.

PCMPExpMeanExp  Create a function of time that calculates \((1 - \exp(-\lambda_{ij} \cdot time))/\lambda_{ij}\) for every element \(\lambda_{ij}\) of the input matrix \(\Lambda_{ij}\).

**Description**

Create a function of time that calculates \((1 - \exp(-\lambda_{ij} \cdot time))/\lambda_{ij}\) for every element \(\lambda_{ij}\) of the input matrix \(\Lambda_{ij}\).

**Usage**

```r
PCMPExpMeanExp(
    Lambda_ij, threshold.Lambda_ij = getOption("PCMBase.Threshold.Lambda_ij", 1e-08)
)
```

**Arguments**

- `Lambda_ij` a squared numerical matrix of dimension \(k \times k\)
- `threshold.Lambda_ij` a 0-threshold for \(\text{abs}(\Lambda_i + \Lambda_j)\), where \(\Lambda_i\) and \(\Lambda_j\) are eigenvalues of the parameter matrix \(H\). This threshold-value is used as a condition to take the limit time of the expression \((1 - \exp(-\lambda_{ij} \cdot time))/\lambda_{ij}\) as \(\lambda_{ij} \rightarrow 0\). You can control this value by the global option "PCMBase.Threshold.Lambda_ij". The default value (1e-8) is suitable for branch lengths bigger than 1e-6. For smaller branch lengths, you may want to increase the threshold value using, e.g., `options(PCMBase.Threshold.Lambda_ij=1e-6)`. 

Details

the function 
\(1 - \exp(-\lambda_{ij} \cdot \text{time})/\lambda_{ij}\)
corresponds to the product of the CDF of an exponential distribution with rate \(\Lambda_{ij}\) multiplied by its mean value (mean waiting time).

Value

a function of time returning a matrix with entries formed from the above function or the limit, time, if \(|\Lambda_{ij}| \leq \text{threshold0}\).

Description

Eigen-decomposition of a matrix \(H\)

Usage

```
PCMPLambdaP_1(H)
```

Arguments

- \(H\) a numeric matrix

Details

The function fails with an error message if \(H\) is defective, that is, if its matrix of eigenvectors is computationally singular. The test for singularity is based on the \(\text{rcond}\) function.

Value

a list with elements as follows:

- \(\lambda\) a vector of the eigenvalues of \(H\)
- \(P\) a squared matrix with column vectors, the eigenvectors of \(H\) corresponding to the eigenvalues in \(\lambda\)
- \(P^{-1}\) the inverse matrix of \(P\)
**Description**

A 2D Gaussian distribution density grid in the form of a ggplot object

**Usage**

```r
PCMPlotGaussianDensityGrid2D(
  mu,
  Sigma,
  xlim,
  ylim,
  xNumPoints = 100,
  yNumPoints = 100,
  ...
)
```

**Arguments**

- **mu**: numerical mean vector of length 2
- **Sigma**: numerical 2 x 2 covariance matrix
- **xlim, ylim**: numerical vectors of length 2
- **xNumPoints, yNumPoints**: integers denoting how many points should the grid contain for each axis.
- **...**: additional arguments passed to ggplot

**Value**

a ggplot object

**Description**

A 2D sample from Gaussian distribution

**Usage**

```r
PCMPlotGaussianSample2D(mu, Sigma, numPoints = 1000, ...)
```
Arguments

\( \mu \quad \text{numerical mean vector of length 2} \)
\( \Sigma \quad \text{numerical 2 x 2 covariance matrix} \)
\( \text{numPoints} \quad \text{an integer denoting how many points should be randomly sampled (see details).} \)
\( \ldots \quad \text{additional arguments passed to ggplot.} \)

Details

This function generates a random sample of numPoints 2d points using the function rmvnorm from the mvtnorm R-package. Then it produces a ggplot on the generated points.

Value

a ggplot object

Description

This is an S3 generic that produces a plotmath expression for its argument.

Usage

\[ \text{PCMPlotMath(o, roundDigits = 2, transformSigma_x = FALSE)} \]

Arguments

\( o \quad \text{a PCM or a parameter object.} \)
\( \text{roundDigits} \quad \text{an integer, default: 2.} \)
\( \text{transformSigma_x} \quad \text{a logical indicating if Cholesky transformation should be applied to Cholesky-factor parameters prior to generating the plotmath expression.} \)

Value

a character string.
**Description**

Scatter plot of 2-dimensional data

**Usage**

```r
PCMPlotTraitData2D(
  X,
  tree,
  sizePoints = 2,
  alphaPoints = 1,
  labeledTips = NULL,
  sizeLabels = 8,
  nudgeLabels = c(0, 0),
  palette = PCMColorPalette(PCMNumRegimes(tree), PCMRegimes(tree)),
  scaleSizeWithTime = !is.ultrametric(tree),
  numTimeFacets = if (is.ultrametric(tree) || scaleSizeWithTime) 1L else 3L,
  nrowTimeFacets = 1L,
  ncolTimeFacets = numTimeFacets
)
```

**Arguments**

- **X**
  - a k x N matrix
- **tree**
  - a phylo object
- **sizePoints, alphaPoints**
  - numeric parameters passed as arguments size and alpha to `geom_point`. Default: sizePoints = 2, alphaPoints = 1.
- **labeledTips**
  - a vector of tip-numbers to label (NULL by default)
- **sizeLabels**
  - passed to `geom_text` to specify the size of tip-labels for the trait-points.
- **nudgeLabels**
  - a numeric vector of two elements (default: c(0,0)), passed as arguments nudge_x and nudge_y of `geom_text`.
- **palette**
  - a named vector of colors
- **scaleSizeWithTime**
  - logical indicating if the size and the transparency of the points should reflect the distance from the present (points that are farther away in time with respect to the present moment, i.e. closer to the root of the tree, are displayed smaller and more transparent.). By default this is set to !is.ultrametric(tree).
- **numTimeFacets**
  - a number or a numeric vector controlling the creation of different facets corresponding to different time intervals when the tree is non-ultrametric. If a single number, it will be interpreted as an integer specifying the number of facets, each facets corresponding to an equal interval of time. If a numeric vector, it will be
PCMPresentCoordinates

Determine which traits are present (active) on each node of the tree

Description

For every node (root, internal or tip) in tree, build a logical vector of length k with TRUE values for every present coordinate. Non-present coordinates arize from NA-values in the trait data. These can occur in two cases:

**Missing measurements for some traits at some tips:** the present coordinates are FALSE for the corresponding tip and trait, but are full for all traits at all internal and root nodes.

**non-existent traits for some species:** the FALSE present coordinates propagate towards the parent nodes - an internal or root node will have a present coordinate set to FALSE for a given trait, if all of its descendants have this coordinate set to FALSE.

These two cases have different effect on the likelihood calculation: missing measurements (NA) are integrated out at the parent nodes; while non-existent traits (NaN) are treated as reduced dimensionality of the vector at the parent node.

Usage

PCMPresentCoordinates(X, tree, metaI)

Arguments

- **X**: numeric k x N matrix of observed values, with possible NA entries. The columns in X are in the order of tree$tip.label
- **tree**: a phylo object
- **metaI**: The result of calling PCMInfo.

Value

a k x M logical matrix. The function fails in case when all traits are NAs for some of the tips. In that case an error message is issued "PCMPresentCoordinates:: Some tips have 0 present coordinates. Consider removing these tips."

See Also

PCMLik
**PCMRegimes**

*Get the regimes (aka colors) of a PCM or of a PCMTree object*

---

**Description**

Get the regimes (aka colors) of a PCM or of a PCMTree object

**Usage**

`PCMRegimes(obj)`

**Arguments**

- `obj`: a PCM or a PCMTree object

**Value**

a character or an integer vector giving the regime names in the obj

---

**PCMSetAddress**

*Set an attribute of a named member in a PCM or other named list object*

---

**Description**

Set an attribute of a named member in a PCM or other named list object

**Usage**

```r
PCMSetAddress(
  name, 
  value, 
  object, 
  member = "", 
  spec = TRUE, 
  inplace = TRUE, 
  ... 
)
```
Arguments

- **name**: a character string denoting the attribute name.
- **value**: the value for the attribute.
- **object**: a PCM or a list object.
- **member**: a member expression. Member expressions are character strings denoting named elements in a list object (see examples). Default: "".
- **spec**: a logical (TRUE by default) indicating if the attribute should also be set in the corresponding member of the spec attribute (this is for PCM objects only).
- **inplace**: logical (TRUE by default) indicating if the attribute should be set to the object in the current environment, or a modified object should be returned.
- ... additional arguments passed to `MatchListMembers`.

Details

Calling this function can affect the attributes of multiple members matched by the `member` argument.

Value

if `inplace` is TRUE (default) nothing is returned. Otherwise, a modified version of `object` is returned.

Examples

```r
model <- PCMBaseTestObjects$model_MixedGaussian_ab
PCMSimAttribute("class", c("MatrixParameter", "_Fixed"), model, "H")
```

Description

Generate trait data on a tree according to a multivariate stochastic model with one or several regimes.

Usage

```r
PCMSim(
  tree,
  model,
  X0,
  SE = matrix(0, PCMNumTraits(model), PCMTreeNumTips(tree)),
  metaI = PCMInfo(X = NULL, tree = tree, model = model, SE = SE, verbose = verbose),
  verbose = FALSE
)
```
Arguments

- **tree**: A phylo object specifying a rooted tree.
- **model**: An S3 object specifying the model (see Details).
- **X0**: A numeric vector of length k (the number of traits) specifying the trait values at the root of the tree.
- **SE**: A k x N matrix specifying the standard error for each measurement in X. Alternatively, a k x k x N cube specifying an upper triangular k x k factor of the variance covariance matrix for the measurement error for each node i=1,...,N. Default: matrix(0.0, PCMNumTraits(model), PCMTreeNumTips(tree)).
- **metaI**: A named list containing meta-information about the data and the model.
- **verbose**: A logical indicating if informative messages should be written during execution.

Details

Internally, this function uses the **PCMCond** implementation for the given model class.

Value

A numeric M x k matrix of values at all nodes of the tree, i.e. root, internal and tip, where M is the number of nodes: $M=\dim(tree\$edge[,1]) + 1$, with indices from 1 to N=length(tree\$tip.label) corresponding to tips, N+1 corresponding to the root and bigger than N+1 corresponding to internal nodes. The function will fail in case that the length of the argument vector X0 differs from the number of traits specified in metaI$k. Error message: "PCMSim:: X0 must be of length ...".

See Also

- **PCMLik**
- **PCMInfo**
- **PCMCond**

Examples

```r
N <- 10
L <- 100.0
tr <- ape::stree(N)
tr$edge.length <- rep(L, N)
for(epoch in seq(1, L, by = 1.0)) {
  tr <- PCMTreeInsertSingletonsAtEpoch(tr, epoch)
}

model <- PCMBaseTestObjects$model_MixedGaussian_ab

PCMTreeSetPartRegimes(tr, c("\'1\'" = 'a'), setPartition = TRUE)

set.seed(1, kind = "Mersenne-Twister", normal.kind = "Inversion")
X <- PCMSim(tr, model, X0 = rep(0, 3))
```
**PCMSpecify**

*Parameter specification of PCM model*

**Description**

The parameter specification of a PCM model represents a named list with an entry for each parameter of the model. Each entry in the list is a structure defining the S3 class of the parameter and its verbal description. This is an S3 generic. See `PCMSpecify.OU` for an example method.

**Usage**

`PCMSpecify(model, ...)`

**Arguments**

- `model`: a PCM model object.
- `...`: additional arguments used by implementing methods.

**Value**

a list specifying the parameters of a PCM.

---

**PCMTable**

*A data.table representation of a PCM object*

**Description**

A data.table representation of a PCM object

**Usage**

`PCMTable(model, skipGlobalRegime = FALSE, addTransformed = TRUE, removeUntransformed = TRUE)`

**Arguments**

- `model`: a PCM object.
- `skipGlobalRegime`: logical indicating whether a raw in the returned table for the global-scope parameters should be omitted (this is mostly for internal use). Default (FALSE).
addTransformed  logical. If TRUE (the default), columns for the transformed version of the transformable parameters will be added.
removeUntransformed  logical. If TRUE (default), columns for the untransformed version of the transformable parameters will be omitted.

Details
This is an S3 generic.

Value
an object of S3 class PCMTable

Description
This function generates a data.table in which each column corresponds to one parameter of model and each row corresponds to one combination of parameterizations for the model parameters, such that the whole table corresponds to the Cartesian product of the lists found in ‘listParameterizations’. Usually, subsets of this table should be passed to ‘PCMGenerateParameterizations’

Usage
PCMTableParameterizations(
  model,
  listParameterizations = PCMListParameterizations(model, ...),
  ...
)

Arguments
model  a PCM object.
listParameterizations  a list returned by a method for ‘PCMListParameterizations’. Default: ‘PCMListParameterizations(model, ...)’.  
...  additional arguments passed to ‘PCMListParameterizations(model, ...)’.

Value
a data.table object.
PCMTrajectory

Generate a trajectory for the mean in one regime of a PCM

Description

Generate a trajectory for the mean in one regime of a PCM

Usage

PCMTrajectory(
  model,
  regime = PCMRegimes(model)[1],
  X0 = rep(0, PCMNumTraits(model)),
  W0 = matrix(0, nrow = PCMNumTraits(model), ncol = PCMNumTraits(model)),
  tX = seq(0, 100, by = 1),
  tVar = tX[seq(0, length(tX), length.out = 4)],
  dims = seq_len(PCMNumTraits(model)),
  sizeSamp = 100,
  doPlot2D = FALSE,
  plot = NULL
)

Arguments

- **model**: a PCM object.
- **regime**: a regime in ‘model’. Default is PCMRegimes(model)[1].
- **X0**: a numeric vector specifying an initial point in the trait space. Default is rep(0, PCMNumTraits(model))
- **W0**: a numeric k x k symmetric positive definite matrix or 0 matrix, specifying the initial variance covariance matrix at t0. By default, this is a k x k 0 matrix.
- **tX, tVar**: numeric vectors of positive points in time sorted in increasing order. tX specifies the points in time at which to calculate the mean (conditional on X0). tVar specifies a subset of the points in tX at which to generate random samples from the k-variate Gaussian distribution with mean equal to the mean value at the corresponding time conditional on X0 and variance equal to the variance at this time, conditional on W0. Default settings are ‘tX = seq(0, 100, by = 1)’ and ‘tVar = tX[seq(0, length(tX), length.out = 4)]’.
- **dims**: an integer vector specifying the traits for which samples at tVar should be generated (see tX, tVar above). Default: seq_len(PCMNumTraits(model)).
- **sizeSamp**: an integer specifying the number points in the random samples (see tX and tVar above). Default 100.
- **doPlot2D**: Should a ggplot object be produced and returned. This is possible only for two of the traits specified in dims. Default: FALSE.
- **plot**: a ggplot object. This can be specified when doPlot2D is TRUE and allows to add the plot of this trajectory as a layer in an existing ggplot. Default: NULL.
Value

if doPlot2D is TRUE, returns a ggplot. Otherwise a named list of two elements:

- **dt** a data.table with columns 'regime', 't', 'X', 'V' and 'samp'. For each row corresponding to time in tVar, the column samp represents a list of sizeSamp k-vectors.

- **dtPlot** a data.table with the same data as in dt, but with converted columns X and samp into 2 x k columns denoted xi, i=1,...,k and xsi (i=1...k) This is suitable for plotting with ggplot.

Examples

```r
set.seed(1, kind = "Mersenne-Twister", normal.kind = "Inversion")

# a Brownian motion model with one regime
modelOU <- PCM(model = PCMDefaultModelTypes()["F"], k = 2)

# assign the model parameters at random: this will use uniform distribution
# with boundaries specified by PCMParamLowerLimit and PCMParamUpperLimit
# We do this in two steps:
# 1. First we generate a random vector. Note the length of the vector equals
# PCMParamCount(modelBM).
randomParams <- PCMParamRandomVecParams(
  modelOU, PCMNumTraits(modelOU), PCMNumRegimes(modelOU))
# 2. Then we load this random vector into the model.
PCMParamLoadOrStore(
  modelOU,
  randomParams,
  0, PCMNumTraits(modelBM), PCMNumRegimes(modelBM), load = TRUE)

# let's plot the trajectory of the model starting from X0 = c(0,0)
PCMTrajectory(
  model = modelOU,
  X0 = c(0, 0),
  doPlot2D = TRUE)

# A faceted grid of plots for the two regimes in a mixed model:
pla <- PCMTrajectory(
  model = PCMBaseTestObjects$model_MixedGaussian_ab, regime = "a",
  X0 = c(0, 0),
  doPlot2D = TRUE) +
  ggplot2::scale_y_continuous(limits = c(0, 10)) +
  ggplot2::facet_grid(.~regime)

plb <- PCMTrajectory(
  model = PCMBaseTestObjects$model_MixedGaussian_ab, regime = "b",
  X0 = c(0, 0),
  doPlot2D = TRUE) +
  ggplot2::scale_y_continuous(limits = c(0, 10)) +
  ggplot2::facet_grid(.~regime) +
  ggplot2::theme(
    axis.title.y = ggplot2::element_blank(),
```
PCMTTree

axis.text.y = ggplot2::element_blank(),
axis.ticks.y = ggplot2::element_blank())

plot(pla)
plot/plb)

Create a PCMTTree object from a phylo object

Description

PCMTTree is a class that inherits from the class ‘phylo’ in the R-package ‘ape’. Thus, all the functions working on a phylo object would work in the same way if they receive as argument an object of class ‘PCMTTree’. A PCMTree object has the following members in addition to the regular members (‘tip.label’, ‘node.label’, ‘edge’, ‘edge.length’) found in a regular phylo object:

- **edge.part** a character vector having as many elements as there are branches in the tree (corresponding to the rows in ‘tree$edge’). Each element denotes the name of the part to which the corresponding branch belongs. A part in the tree represents a connected subset of its nodes and the branches leading to these nodes. A partition of the tree represents the splitting of the tree into a number of parts. Visually, a partition can be represented as a coloring of the tree, in which no color is assigned to more than one part. In other words, if two branches in the tree are connected by the same color, they either share a node, or all the branches on the path in the tree connecting these two branches have the same color. Formally, we define a partition of the tree as any set of nodes in the tree that includes the root. Each node in this set defines a part as the set of its descendant nodes that can be reached without traversing another partition node. We name each part by the label of its most ancestral node, that is, the node in it, which is closest to the root of the tree. The value of edge.part for an edge in the tree is the name of the part that contains the node to which the edge is pointing.

- **part.regime** a named vector of size the number of parts in the tree. The names correspond to part-names whereas the values denote the ids or character names of regimes in a PCM object.

The constructor PCMTree() returns an object of call

Usage

PCMTTree(tree)

Arguments

- **tree** a phylo object. If this is already a PCMTree object, a copy of this object will be returned.

Value

an object of class PCMTree. This is a copy of the passed phylo object which is guaranteed to have node.label, edge.part and a part.regime entries set.
Examples

tree <- ape::rtree(8)

# the following four are NULLs
tree$node.label
tree$edge.part
tree$part.regime
tree$edge.regime

# In previous version regimes were assigned directly to the edges via
# tree$edge.regime. This is supported but not recommended anymore:

tree$edge.regime <- sample(letters[1:3], size = PCMTreeNumNodes(tree) - 1, replace = TRUE)

tree.a <- PCMTree(tree)
PCMTreeGetLabels(tree.a)
tree.a$node.label
tree.a$edge.part
tree.a$part.regime

# this is set to NULL - starting from PCMBase 1.2.9 all of the information
# for the regimes is stored in tree$edge.part and tree$part.regime.
tree.a$edge.regime

PCMTreeGetPartition(tree.a)
PCMTreeGetPartNames(tree.a)
PCMTreeGetPartRegimes(tree.a)

# let's see how the tree looks like
if(requireNamespace("ggtree"))
    PCMTreePlot(tree.a) + ggtree::geom_nodelab() + ggtree::geom_tiplab()

# This is the recommended way to set a partition on the tree
PCMTreeSetPartition(tree.a, c(10, 12))

PCMTreeGetPartition(tree.a)
PCMTreeGetPartNames(tree.a)
PCMTreeGetPartRegimes(tree.a)

if(requireNamespace("ggtree"))
    PCMTreePlot(tree.a) + ggtree::geom_nodelab() + ggtree::geom_tiplab()

PCMTreeGetPartsForNodes(tree.a, c(11, 15, 12))
PCMTreeGetPartsForNodes(tree.a, c("11", "15", "12"))

PCMTreeSetPartRegimes(tree.a, c('9' = 'a', '10' = 'b', '12' = 'c'))

PCMTreeGetPartition(tree.a)
PCMTreeGetPartNames(tree.a)
Prune the tree leaving one tip for each or some of its parts

Description

Prune the tree leaving one tip for each or some of its parts

Usage

PCMTreeBackbonePartition(tree, partsToKeep = PCMTreeGetPartNames(tree))

Arguments

tree a PCMTre or a phylo object.
partsToKeep a character vector denoting part names in the tree to be kept. Defaults to `PCMTreeGetPartNames(tree)`.

Value

a PCMTre object representing a pruned version of tree.

See Also

PCMTreeSetPartition
PCMTree

Examples

```r
set.seed(1, kind = "Mersenne-Twister", normal.kind = "Inversion")
tree <- PCMTree(ape::rtree(25))

if(requireNamespace("ggtree"))
  PCMTreePlot(tree) + ggtree::geom_nodelab() + ggtree::geom_tiplab()

backb <- PCMTreeBackbonePartition(tree)

if(requireNamespace("ggtree"))
  PCMTreePlot(backb) + ggtree::geom_nodelab(angle = 45) +
  ggtree::geom_tiplab(angle = 45)

tree2 <- PCMTreeSetPartRegimes(
```
tree, c('26' = "a", '28' = "b"), setPartition = TRUE, inplace = FALSE)

if(requireNamespace("ggtree"))
    PCMTreePlot(tree2) + ggtree::geom_nodelab(angle = 45) +
    ggtree::geom_tiplab(angle = 45)

backb <- PCMTreeBackbonePartition(tree2)

if(requireNamespace("ggtree"))
    PCMTreePlot(backb) + ggtree::geom_nodelab(angle = 45) +
    ggtree::geom_tiplab(angle = 45)

tree3 <- PCMTreeSetPartRegimes(
    tree, c('26' = "a", '28' = "b", '41' = "c"), setPartition = TRUE, inplace = FALSE)

if(requireNamespace("ggtree"))
    PCMTreePlot(tree3) + ggtree::geom_nodelab(angle = 45) +
    ggtree::geom_tiplab(angle = 45)

backb <- PCMTreeBackbonePartition(tree3)

if(requireNamespace("ggtree"))
    PCMTreePlot(backb) + ggtree::geom_nodelab(angle = 45) +
    ggtree::geom_tiplab(angle = 45)

backb41 <- PCMTreeBackbonePartition(tree3, partsToKeep = "41")

if(requireNamespace("ggtree"))
    PCMTreePlot(backb41) + ggtree::geom_nodelab(angle = 45) +
    ggtree::geom_tiplab(angle = 45)

backbMoreNodes <- PCMTreeInsertSingletonsAtEpoch(
    backb, epoch = 3.7, minLength = 0.001)
PCMTreeGetPartRegimes(backbMoreNodes)

if(requireNamespace("ggtree"))
    PCMTreePlot(backbMoreNodes) + ggtree::geom_nodelab(angle=45) +
    ggtree::geom_tiplab(angle=45)

backbMoreNodes <- PCMTreeInsertSingletonsAtEpoch(
    backbMoreNodes, epoch = 0.2, minLength = 0.001)
PCMTreeGetPartRegimes(backbMoreNodes)

if(requireNamespace("ggtree"))
    PCMTreePlot(backbMoreNodes) + ggtree::geom_nodelab(angle=45) +
    ggtree::geom_tiplab(angle=45)

backbMoreNodes <- PCMTreeInsertSingletonsAtEpoch(
    backbMoreNodes, epoch = 1.2, minLength = 0.001)
PCMTreeGetPartRegimes(backbMoreNodes)
if (requireNamespace("ggtree"))
PCMTreePlot(backbMoreNodes) + ggtree::geom_nodelab(angle=45) +
ggtree::geom_tiplab(angle=45)

---

**PCMTreeDropClade**

*Drop a clade from a phylogenetic tree*

**Description**

Drop a clade from a phylogenetic tree

**Usage**

```r
PCMTreeDropClade(
  tree,  
  cladeRootNode,  
  tableAncestors = NULL,  
  X = NULL,  
  returnList = !is.null(X),  
  errorOnMissing = FALSE
)
```

**Arguments**

- `tree` a phylo object
- `cladeRootNode` a character string denoting the label or an integer denoting a node in the tree
- `tableAncestors` an integer matrix returned by a previous call to `PCMTreeTableAncestors(tree)` or NULL.
- `X` an optional k x N matrix with trait value vectors for each tip in tree.
- `returnList` logical indicating if a list of the phylo object associated with the tree after dropping the clade and the corresponding entries in X should be returned. Defaults to `!is.null(X)`
- `errorOnMissing` logical indicating if an error should be raised if `cladeRootNode` is not among the nodes in tree. Default FALSE, meaning that if `cladeRootNode` is not a node in tree the tree (and X if `returnList` is TRUE) is/are returned unchanged.

**Value**

If `returnList` is FALSE, a phylo object associated with the remaining tree after dropping the clade, otherwise, a list with two named members:

- **tree** the phylo object associated with the remaining tree after dropping the clade
- **X** the submatrix of X with columns corresponding to the tips in the remaining tree
**See Also**

PCMTreeSpliAtNode PCMTreeExtractClade

**Examples**

```r
set.seed(1, kind = "Mersenne-Twister", normal.kind = "Inversion")
tree <- PCMTree(ape:::mtree(25))
PCMTreeSetPartRegimes(
  tree, c("26"="a", "28"="b", "45"="c"), setPartition = TRUE)

if(requireNamespace("ggtree"))
  PCMTreePlot(tree, palette=c(a = "red", b = "green", c = "blue")) +
    ggplot::geom_nodelab(angle = 45) + ggplot::geom_tiplab(angle = 45)

redGreenTree <- PCMTreeDropClade(tree, 45)
PCMTreeGetPartRegimes(redGreenTree)

if(requireNamespace("ggtree"))
  PCMTreePlot(redGreenTree, palette=c(a = "red", b = "green", c = "blue")) +
    ggplot::geom_nodelab(angle = 45) + ggplot::geom_tiplab(angle = 45)

# we need to use the label here, because the node 29 in tree is not the same
# id in redGreenTree:
redGreenTree2 <- PCMTreeDropClade(redGreenTree, "29")

if(requireNamespace("ggtree"))
  PCMTreePlot(redGreenTree2, palette=c(a = "red", b = "green", c = "blue")) +
    ggplot::geom_nodelab(angle = 45) + ggplot::geom_tiplab(angle = 45)
```

---

**Description**

A data.table with time, part and regime information for the nodes in a tree

**Usage**

PCMTreeDtNodes(tree)

**Arguments**

- **tree** a phylo object with node-labels and parts
Value

A data table with a row for each node in tree and columns as follows:

- **startNode** the starting node of each edge or NA_integer_ for the root
- **endNode** the end node of each edge or the root id for the root
- **startNodeLab** the character label for the startNode
- **endNodeLab** the character label for endNode
- **startTime** the time (distance from the root node) for the startNode or NA_real_ for the root node
- **endTime** the time (distance from the root node) for the endNode or NA_real_ for the root node
- **part** the part to which the edge belongs, i.e. the part of the endNode
- **regime** the regime to which the edge belongs, i.e. the regime of the part of the endNode

Examples

```r
PCMTreeDtNodes(PCMBaseTestObjects$tree.ab)
```

---

**PCMTreeEdgeTimes**

A matrix with the begin and end time from the root for each edge in tree

Description

A matrix with the begin and end time from the root for each edge in tree

Usage

```r
PCMTreeEdgeTimes(tree)
```

Arguments

- **tree** a phylo
PCMTeeEvalNestedEDxOnTree

Perform nested extractions or drops of clades from a tree

Description

Perform nested extractions or drops of clades from a tree

Usage

PCMTeeEvalNestedEDxOnTree(expr, tree)

Arguments

expr
  a character string representing an R expression of nested calls of functions
  E(x, node) denoting extracting the clade rooted at node from the tree x, or
  D(x, node), denoting dropping the clade rooted at node from the tree x. These
  calls can be nested, i.e. x can be either the symbol x (corresponding to the
  original tree passed as argument) or a nested call to d or e.

tree
  a phylo object with named tips and internal nodes

Value

the resulting phylo object from evaluating expr on tree.

Examples

set.seed(1, kind = "Mersenne-Twister", normal.kind = "Inversion")
tree <- PCMTee(ap:::rtee(25))
PCMTeeSetPartRegimes(
  tree, c("a", "b", "c", "d"), setPartition = TRUE)

if(requireNamespace("ggtree"))
  PCMTeePlot(
    tree, palette=c(a = "red", b = "green", c = "blue", d = "magenta") +
    ggtree::geom_nodelab(angle = 45) + ggtree::geom_tiplab(angle = 45)
  )

bluePart <- PCMTeeEvalNestedEDxOnTree("D(E(tree,45),47)", tree)
PCMTeeGetPartRegimes(bluePart)

if(requireNamespace("ggtree"))
  PCMTeePlot(
    bluePart, palette=c(a = "red", b = "green", c = "blue", d = "magenta") +
    ggtree::geom_nodelab(angle = 45) + ggtree::geom_tiplab(angle = 45)
  )

# Swapping the D and E calls has the same result:
bluePart2 <- PCMTeeEvalNestedEDxOnTree("E(D(tree,47),45)", tree)
PCMTeeGetPartRegimes(bluePart2)
if(requireNamespace("ggtree"))
    PCMTreePlot(
        bluePart2, palette=c(a = "red", b = "green", c = "blue", d = "magenta")) +
        ggtree::geom_nodelab(angle = 45) + ggtree::geom_tiplab(angle = 45)

greenPart <- PCMTreeEvalNestedEDxOnTree("E(tree,28)", tree)
bgParts <- bluePart+greenPart

if(requireNamespace("ggtree")) {
    PCMTreePlot(
        greenPart, palette=c(a = "red", b = "green", c = "blue", d = "magenta")) +
        ggtree::geom_nodelab(angle = 45) + ggtree::geom_tiplab(angle = 45)
    PCMTreePlot(
        bluePart + greenPart, palette=c(a = "red", b = "green", c = "blue", d = "magenta")) +
        ggtree::geom_nodelab(angle = 45) + ggtree::geom_tiplab(angle = 45)
    PCMTreePlot(
        greenPart + bluePart, palette=c(a = "red", b = "green", c = "blue", d = "magenta")) +
        ggtree::geom_nodelab(angle = 45) + ggtree::geom_tiplab(angle = 45)
    PCMTreePlot(
        bgParts, palette=c(a = "red", b = "green", c = "blue", d = "magenta")) +
        ggtree::geom_nodelab(angle = 45) + ggtree::geom_tiplab(angle = 45)
}

PCMTreerExtractClade

---

**PCMTreerExtractClade**  
*Extract a clade from phylogenetic tree*

---

**Description**  
Extract a clade from phylogenetic tree

**Usage**

```r
PCMTreerExtractClade(
    tree,  
    cladeRootNode,  
    tableAncestors = NULL,  
    X = NULL,  
    returnList = !is.null(X)
)
```

**Arguments**

- `tree`  
a PCMTree object.
- `cladeRootNode`  
a character string denoting the label or an integer denoting a node in the tree.
- `tableAncestors`  
an integer matrix returned by a previous call to PCMTreeTableAncestors(tree) or NULL.
X an optional k x N matrix with trait value vectors for each tip in tree.

returnList logical indicating if only the phylo object associated with the clade should be returned. Defaults to !is.null(X)

Value

If returnList is FALSE, a phylo object associated with the clade, otherwise, a list with two named members:

- tree the phylo object associated with the clade
- X the submatrix of X with columns corresponding to the tips in the clade

See Also

PCMTreeSplitAtNode PCMTreeDropClade

Examples

```r
set.seed(1, kind = "Mersenne-Twister", normal.kind = "Inversion")
tree <- PCMTree(ape::rtree(25))
PCMTreeSetPartRegimes(
  tree, c("26"="a", "28"="b", "45"="c"), setPartition = TRUE)

if(requireNamespace("ggtree"))
  PCMTreePlot(tree, palette=c(a = "red", b = "green", c = "blue")) +
  ggtree::geom_nodelab(angle = 45) + ggtree::geom_tiplab(angle = 45)

blueTree <- PCMTreeExtractClade(tree, 45)
PCMTreeGetPartRegimes(blueTree)

if(requireNamespace("ggtree"))
  PCMTreePlot(blueTree, palette=c(a = "red", b = "green", c = "blue")) +
  ggtree::geom_nodelab(angle = 45) + ggtree::geom_tiplab(angle = 45)

# we need to use the label here, because the node 29 in tree is not the same
# id in redGreenTree:
blueTree2 <- PCMTreeDropClade(blueTree, "48")

if(requireNamespace("ggtree"))
  PCMTreePlot(blueTree2, palette=c(a = "red", b = "green", c = "blue")) +
  ggtree::geom_nodelab(angle = 45) + ggtree::geom_tiplab(angle = 45)
```

---

**PCMTreeGetBranchLength**

*The length of the branch leading to a node*

**Description**

The length of the branch leading to a node
Usage

PCMTreeGetBranchLength(tree, daughterId)

Arguments

tree a phylo object.
daughterId an integer denoting the id of a daughter node

Value

a double denoting the length of the branch leading to daughterId

Description

A vector of the daughter nodes for a given parent node id in a tree

Usage

PCMTreeGetDaughters(tree, parentId)

Arguments

tree a phylo object.
parentId an integer denoting the id of the parent node

Value

an integer vector of the direct descendants of parentId

Description

Node labels of a tree

Usage

PCMTreeGetLabels(tree)

PCMTreeGetRootLabel(tree)

PCMTreeGetNodeLabels(tree)

PCMTreeGetTipLabels(tree)
Arguments

- tree: a phylo object

Value

- a character vector

Functions

- PCMTreeGetLabels: Get all labels in the order (tips, root, internal).
- PCMTreeGetRootLabel: Get the root label
- PCMTreeGetNodeLabels: Get the internal node labels
- PCMTreeGetTipLabels: Get the tip labels

Description

The parent node id of a daughter node in a tree

Usage

PCMT TreeGetParent(tree, daughterId)

Arguments

- tree: a phylo object.
- daughterId: an integer denoting the id of the daughter node

Value

- an integer denoting the parent of daughterId
**PCMTreeGetPartition**  
*Get the starting branch’ nodes for each part on a tree*

**Description**
Get the starting branch’ nodes for each part on a tree

**Usage**
PCMTreeGetPartition(tree)

**Arguments**
tree  
A phylo object with an edge.part member denoting parts. The function assumes that each part covers a linked set of branches on the tree.

**Details**
We call a starting branch the first branch from the root to the tips of a given part. A starting node is the node at which a starting branch ends.

**Value**
a named integer vector with elements equal to the starting nodes for each part in tree and names equal to the labels of these nodes.

**See Also**
PCMTreeSetPartition

**Examples**
set.seed(1, kind = "Mersenne-Twister", normal.kind = "Inversion")
PCMTreeGetPartition(PCMTree(ape::rtree(20)))

---

**PCMTreeGetPartNames**  
*Unique parts on a tree in the order of occurrence from the root to the tips (preorder)*

**Description**
Unique parts on a tree in the order of occurrence from the root to the tips (preorder)

**Usage**
PCMTreeGetPartNames(tree)
Arguments

tree  a phylo object with an additional member edge.part which should be a character or an integer vector of length equal to the number of branches.

Value

a character vector.

PCMTreGetPartRegimes  Regime mapping for the parts in a tree

Description

Regime mapping for the parts in a tree

Usage

PCMTreGetPartRegimes(tree)

Arguments

tree  a PCMTre or a phylo object.

Value

a named vector with names corresponding to the part names in tree and values corresponding to regime names or ids.

PCMTreGetPartsForNodes

Get the parts of the branches leading to a set of nodes or tips

Description

Get the parts of the branches leading to a set of nodes or tips

Usage

PCMTreGetPartsForNodes(tree, nodes = seq_len(PCMTreNumNodes(tree)))

Arguments

tree  a phylo object with an edge.part member denoting parts.

nodes  an integer vector denoting the nodes. Default is seq_len(PCMTreNumNodes(tree)).

Value

a character vector denoting the parts of the branches leading to the nodes, according to tree$edge.part.
PCMTreeGetRegimesForEdges

Model regimes (i.e. colors) associated with the branches in a tree

Description
Model regimes (i.e. colors) associated with the branches in a tree

Usage
PCMTreeGetRegimesForEdges(tree)

Arguments
- tree: a PCMTree or a phylo object.

Value
A vector with entries corresponding to the rows in tree$edge denoting the regime associated with each branch in the tree. The type of the vector element is defined by the type of tree$part.regime.

PCMTreeGetRegimesForNodes

Get the regimes of the branches leading to a set of nodes or tips

Description
Get the regimes of the branches leading to a set of nodes or tips

Usage
PCMTreeGetRegimesForNodes(tree, nodes = seq_len(PCMTreeNumNodes(tree)))

Arguments
- tree: a phylo object with an edge.part member denoting parts.
- nodes: an integer vector denoting the nodes. Default is seq_len(PCMTreeNumNodes(tree)).

Value
A character vector denoting the parts of the branches leading to the nodes, according to tree$edge.part.
PCMTreeGetTipsInPart

Get the tips belonging to a part in a tree

Description

Get the tips belonging to a part in a tree

Usage

PCMTreeGetTipsInPart(tree, part)

Arguments

tree a phylo object with an edge.regime member or a PCMTree object

part a character or integer denoting a part name in the tree.

Value

an integer vector with the ids of the tips belonging to part

See Also

PCMTreeGetTipsInRegime, PCMTreeGetPartNames, PCMRegimes, PCMTreeGetPartRegimes, PCMTreeSetPartRegimes

Examples

set.seed(1, kind = "Mersenne-Twister", normal.kind = "Inversion")
tree <- ape::rtree(10)
regimes <- sample(letters[1:3], nrow(tree$edge), replace = TRUE)
PCMTreeSetRegimesForEdges(tree, regimes)

if(requireNamespace("ggtree"))
PCMTreePlot(tree) + ggtree::geom_nodelab() + ggtree::geom_tiplab()

part <- PCMTreeGetPartNames(tree)[1]
PCMTreeGetTipsInPart(tree, part)
print(part)
PCMTreeGetTipsInRegime

*Get the tips belonging to a regime in a tree*

**Description**

Get the tips belonging to a regime in a tree

**Usage**

`PCMTreeGetTipsInRegime(tree, regime)`

**Arguments**

- `tree`: a phylo object with an edge.regime member or a PCMTree object
- `regime`: a character or integer denoting a regime in the tree.

**Value**

an integer vector with the ids of the tips belonging to regime.

**See Also**

`PCMTreeGetTipsInPart`, `PCMTreeGetPartNames`, `PCMRegimes`, `PCMTreeGetPartRegimes`, `PCMTreeSetPartRegimes`, `PCMTreeGetPartition`

**Examples**

```r
set.seed(1, kind = "Mersenne-Twister", normal.kind = "Inversion")
tree <- ape::rtree(10)
regimes <- sample(letters[1:3], nrow(tree$edge), replace = TRUE)
PCMTreeSetRegimesForEdges(tree, regimes)
if(requireNamespace("ggtree"))
PCMTreePlot(tree) + ggtree::geom_nodelab() + ggtree::geom_tiplab()

regime <- PCMRegimes(tree)[1]
PCMTreeGetTipsInRegime(tree, regime)
print(regime)
```
PCMTreeInsertSingletons

Insert tips or singleton nodes on chosen edges

Description
Insert tips or singleton nodes on chosen edges

Usage
PCMTreeInsertSingletons(tree, nodes, positions)

PCMTreeInsertSingletonsAtEpoch(tree, epoch, minLength = 0.1)

PCMTreeInsertTipsOrSingletons(
  tree,
  nodes,
  positions = rep(0, length(nodes)),
  singleton = FALSE,
  tipBranchLengths = 0.01,
  nodeLabels = NULL,
  tipLabels = NULL
)

Arguments

- **tree**: a phylo object
- **nodes**: an integer vector denoting the terminating nodes of the edges on which a singleton node is to be inserted. This vector should not have duplicated nodes - if there is a need to insert two or more singleton nodes at distinct positions of the same edge, this should be done by calling the function several times with the longest position first and so on.
- **positions**: a positive numeric vector of the same length as nodes denoting the root-ward distances from nodes at which the singleton nodes should be inserted. For PCMTreeInsertTipsOrSingletons this can contain 0’s and is set by default to rep(0, length(nodes)).
- **epoch**: a numeric indicating a distance from the root at which a singleton node should be inserted in all lineages that are alive at that time.
- **minLength**: a numeric indicating the minimum allowed branch-length after dividing a branch by insertion of a singleton nodes. No singleton node is inserted if this would result in a branch shorter than ‘minLength’. Note that this condition is checked only in ‘PCMTreeInsertSingletonsAtEpoch’.
- **singleton** (PCMTreeInsertTipsOrSingletons only): a logical indicating if a singleton node should be inserted and no tip node should be inserted.
tipBranchLengths
(PCMTreeInsertTipsOrSingletons only) positive numeric vector of the length of nodes, denoting the lengths of the new edges leading to tips.

nodeLabels
(PCMTreeInsertSingletons and PCMTreeInsertTipsOrSingletons) a character vector of the same length as nodes, indicating the names of the newly inserted nodes. These names are ignored where positions is 0. This argument is optional and default node labels will be assigned if this is not specified or set to NULL. If specified, then it should not contain node-labels already present in the tree.

tipLabels
(PCMTreeInsertTipsOrSingletons only) a character vector of the same length as nodes of the new tip-labels. This argument is optional and default tip labels will be assigned if this is not specified or set to NULL. If specified, then it should not contain tip-labels already present in the tree.

Value
a modified copy of tree.

Functions
• PCMTreeInsertSingletonsAtEpoch:
• PCMTreeInsertTipsOrSingletons:

See Also
PCMTreeEdgeTimes PCMTreeLocateEpochOnBranches PCMTreeLocateMidpointsOnBranches

Examples
set.seed(1, kind = "Mersenne-Twister", normal.kind = "Inversion")
tree <- PCMTree(ape::rtree(25))
PCMTreeSetPartRegimes(
  tree, c('26'="a", '28'="b", '45'="c", '47'="d"), setPartition = TRUE)

if(requireNamespace("ggtree"))
  PCMTreePlot(
    tree,
    palette=c(a = "red", b = "green", c = "blue", d = "magenta") +
    ggtree::geom_nodelab(angle = 45) + ggtree::geom_tiplab(angle = 45)
  )
  cbind(tree$edge, PCMTreeEdgeTimes(tree))

id47 <- PCMTreeMatchLabels(tree, "47")
length47 <- PCMTreeGetBranchLength(tree, id47)

# insert a singleton at 0.55 (root-ward) from node 47
  tree <- PCMTreeInsertSingletons(tree, nodes = "47", positions = (length47/2))

if(requireNamespace("ggtree"))
  PCMTreePlot(
Jumps in modeled traits associated with branches in a tree

Description

Jumps in modeled traits associated with branches in a tree
Usage

PCMTreeJumps(tree)

Arguments

tree  a phylo object

Value

an integer vector of 0’s and 1’s with entries corresponding to the denoting if a jump took place at
the beginning of a branch.

pcmTreeListAllPartitions

A list of all possible (including recursive) partitions of a tree

Description

A list of all possible (including recursive) partitions of a tree

Usage

PCMTreeListAllPartitions(
  tree,
  minCladeSize,
  skipNodes = character(),
  tableAncestors = NULL,
  verbose = FALSE
)

Arguments

tree  a phylo object with set labels for the internal nodes
minCladeSize  integer indicating the minimum number of tips allowed in one part.
skipNodes  an integer or character vector indicating the ids or labels of nodes that should not be used as partition nodes. By default, this is an empty character vector.
tableAncestors  NULL (default) or an integer matrix returned by a previous call to PCMTreeTableAncestors(tree).
verbose  a logical indicating if informative messages should be printed to the console.

Value

a list of integer vectors.
Examples

```r
set.seed(1, kind = "Mersenne-Twister", normal.kind = "Inversion")
tree <- PCMTree(ape::rtree(10))

if(requireNamespace("ggtree"))
  PCMTreePlot(tree) + ggtree::geom_nodelab() + ggtree::geom_tiplab()

# list of all partitions into parts of at least 4 tips
PCMTreeListAllPartitions(tree, 4)

# list of all partitions into parts of at least 3 tips
PCMTreeListAllPartitions(tree, 3)

# list all partitions into parts of at least 3 tips, excluding the partitions
# where node 16 is one of the partition nodes:
PCMTreeListAllPartitions(tree, minCladeSize = 3, skipNodes = "16")
```

---

**PCMTreeListCladePartitions**

*A list of all possible clade partitions of a tree with a number of splitting nodes*

**Description**

Each subset of $n$ nodes distinct internal or tip nodes defines a partition of the branches of the tree into $n+1$ blocks called parts. This function generates partitions where each part has $n$ splitting nodes and contains at least $\text{minCladeSize}$ tips.

**Usage**

```r
PCMTreeListCladePartitions(
  tree, 
  nNodes, 
  minCladeSize = 0, 
  skipNodes = character(0), 
  tableAncestors = NULL, 
  countOnly = FALSE, 
  verbose = FALSE
)
```

**Arguments**

- `tree`: a phylo object
- `nNodes`: an integer giving the number of partitioning nodes. There would be $n+1$ blocks in each partition (see details).
- `minCladeSize`: integer indicating the minimum number of tips allowed in a clade.
skipNodes an integer or character vector indicating the ids or labels of nodes that should not be used as partition nodes. By default, this is an empty character vector.
tableAncestors NULL (default) or an integer matrix returned by a previous call to PCMTreeTableAncestors(tree).
countOnly logical indicating if the only the number of partitions should be returned.
verbose a logical indicating if informative messages should be printed to the console.

Value

a list of integer nNodes-vectors. By default a full traversal of all partitions is done. It is possible to truncate the search to a limited number of partitions by setting the option PCMBase.MaxLengthListCladePartitions to a finite positive integer.

See Also

PCMOptions

Description

A list of the descendants for each node in a tree

Usage

PCMTreeListDescendants(tree, tableAncestors = PCMTreeTableAncestors(tree))

Arguments

tree a phylo object
tableAncestors an integer matrix resulting from a call to PCMTreeTableAncestors(tree).

Details

This function has time and memory complexity $O(M^2)$, where $M$ is the number of nodes in the tree. It can take several minutes and gigabytes of memory on trees of more than 10000 tips.

Value

a list with unnamed elements in the order of nodes in the tree. Each element is an integer vector containing the descendant nodes (in increasing order) of the node identified by its index-number in the list.
**PCMTreeListRootPaths**  
*A list of the path to the root from each node in a tree*

**Description**  
A list of the path to the root from each node in a tree

**Usage**  
PCMTreeListRootPaths(tree, tableAncestors = PCMTreeTableAncestors(tree))

**Arguments**

- **tree**  
a phylo object

- **tableAncestors**  
an integer matrix resulting from a call to PCMTreeTableAncestors(tree).

**Details**

This function has time and memory complexity $O(M^2)$, where $M$ is the number of nodes in the tree. It can take several minutes and gigabytes of memory on trees of more than 10000 tips.

**Value**

a list with unnamed elements in the order of nodes in the tree. Each element is an integer vector containing the ancestors nodes on the path from the node $(i)$ to the root of the tree in that order (the first element in the vector is the parent node of $i$ and so on).

**PCMTreeLocateEpochOnBranches**  
*Find the crossing points of an epoch-time with each lineage of a tree*

**Description**  
Find the crossing points of an epoch-time with each lineage of a tree

**Usage**  
PCMTreeLocateEpochOnBranches(tree, epoch)

**Arguments**

- **tree**  
a phylo

- **epoch**  
a positive numeric indicating tip-ward distance from the root
**PCMTreeLocateMidpointsOnBranches**

Find the middle point of each branch longer than a threshold

**Usage**

```r
PCMTreeLocateMidpointsOnBranches(tree, threshold = 0)
```

**Arguments**

- `tree`: a phylo
- `threshold`: a positive numeric; only branches longer than threshold will be returned; Default 0.

**Value**

a named list with an integer vector element "nodes" denoting the ending nodes for each branch crossing epoch and numeric vector element "positions" denoting the root-ward offset from each node in nodes.

**PCMTreeMatchLabels**

Get the node numbers associated with tip- or node-labels in a tree

**Description**

Get the node numbers associated with tip- or node-labels in a tree

**Usage**

```r
PCMTreeMatchLabels(tree, labels, stopIfNotFound = TRUE)
```

**Arguments**

- `tree`: a phylo object
- `labels`: a character vector with valid tip or node labels from tree
- `stopIfNotFound`: logical indicating if an error should be raised in case a label has not been found in the tree labels. Default: TRUE
Value

an integer vector giving the tip- or node- integer indices corresponding to labels. If stopIfNotFound is set to FALSE, this vector may contain NAs for the labels that were not found.

Examples

set.seed(1, kind = "Mersenne-Twister", normal.kind = "Inversion")
PCMTreeMatchLabels(PCMTree(ape::rtree(20)), c("t1", "t15", "21", "39"))
PCMTreeMatchLabels(PCMTree(ape::rtree(20)), c("t1", "45"), stopIfNotFound = FALSE)

Description

Which couples from a given set of nodes in a tree belong to the same part?

Usage

PCMTreeMatrixNodesInSamePart(
  tree,
  nodes = seq_len(PCMTreeNumNodes(tree)),
  upperTriangle = TRUE,
  returnVector = TRUE
)

PCMTreeMatrixNodesInSameRegime(
  tree,
  nodes = seq_len(PCMTreeNumNodes(tree)),
  upperTriangle = TRUE,
  returnVector = TRUE
)

Arguments

tree a PCMTreec object or a phylo object.

nodes an integer vector of length L >= 2 denoting a set of nodes in the tree.

upperTriangle logical indicating if all duplicated entries and diagonal entries should be set to NA (by default TRUE).

returnVector logical indicating if a vector instead of a matrix should be returned (corresponding to calling as.vector on the resulting matrix and removing NAs). Default: TRUE
Value

A `L x L` logical matrix with TRUE on the diagonal and for each couple of tips that belong to the same part or regime. If `returnVector` is TRUE (default) only a vector of the non-NA entries will be returned.

Examples

```r
set.seed(1, kind = "Mersenne-Twister", normal.kind = "Inversion")
Tree <- PCMTree(ape::tree(8))
PCMTreeMatrixNodesInSamePart(tree, returnVector = FALSE)

PCMTreeSetPartition(tree, c(10, 12))
PCMTreeMatrixNodesInSamePart(tree, returnVector = FALSE)

PCMTreeMatrixNodesInSamePart(tree)
PCMTreeMatrixNodesInSamePart(tree, seq_len(PCMTreeNumTips(tree)))
PCMTreeMatrixNodesInSamePart(
  tree, seq_len(PCMTreeNumTips(tree)), returnVector = FALSE)

set.seed(1, kind = "Mersenne-Twister", normal.kind = "Inversion")
Tree <- PCMTree(ape::tree(8))
PCMTreeMatrixNodesInSamePart(tree, returnVector = FALSE)

PCMTreeSetPartition(tree, c(10, 12))
PCMTreeMatrixNodesInSamePart(tree, returnVector = FALSE)

PCMTreeMatrixNodesInSamePart(tree)
PCMTreeMatrixNodesInSamePart(tree, seq_len(PCMTreeNumTips(tree)))
PCMTreeMatrixNodesInSamePart(
  tree, seq_len(PCMTreeNumTips(tree)), returnVector = FALSE)
```

Description

Find the nearest node to a given time from the root (epoch) on each lineage crossing this epoch.

Usage

```r
PCMTreeNearestNodesToEpoch(tree, epoch)
```
Arguments

- **tree**: a phylo
- **epoch**: a non-negative numeric

Value

an integer vector

**PCMTreeNodeTimes**  
*Calculate the time from the root to each node of the tree*

Description

Calculate the time from the root to each node of the tree

Usage

`PCMTreeNodeTimes(tree, tipsOnly = FALSE)`

Arguments

- **tree**: an object of class phylo
- **tipsOnly**: Logical indicating whether the returned results should be truncated only to the tips of the tree.

Value

A vector of size the number of nodes in the tree (tips, root, internal) containing the time from the root to the corresponding node in the tree.

**PCMTTreeNumNodes**  
*Number of all nodes in a tree*

Description

Number of all nodes in a tree

Usage

`PCMTTreeNumNodes(tree)`

Arguments

- **tree**: a phylo object
**PCMTreeNumParts**

**Details**
Wrapper for `nrow(tree$edge) + 1`

**Value**
the number of nodes in tree including root, internal and tips.

---

**Description**
Number of unique parts on a tree

**Usage**
```
PCMTreeNumParts(tree)
```

**Arguments**
- `tree` a phylo object

**Value**
the number of different parts encountered on the tree branches

---

**PCMTreeNumTips**

**Description**
Wrapper for `length(tree$tip.label)`

**Usage**
```
PCMTreeNumTips(tree)
```

**Arguments**
- `tree` a phylo object

**Value**
the number of tips in tree
**PCMTTreePlot**  
*Plot a tree with parts and regimes assigned to these parts*

**Description**  
Plot a tree with parts and regimes assigned to these parts

**Usage**  
```r  
PCMTTreePlot(  
  tree,  
  palette = PCMCColorPalette(PCMNumRegimes(tree), PCMRegimes(tree)),  
  ...  
)  
```

**Arguments**  
- **tree**  
  a PCMTree or a phylo object.
- **palette**  
  a named vector of colors corresponding to the regimes in tree
- **...**  
  Arguments passed to ggtree, e.g. layout = 'fan', open.angle = 8, size=.25.

**Note**  
This function requires that the ggtree package is installed. At the time of releasing this version the ggtree package is not available on CRAN. Check the ggtree homepage for instruction on how to install this package: .

**PCMTTreePostorder**  
*Post-order tree traversal*

**Description**  
Post-order tree traversal

**Usage**  
```r  
PCMTTreePostorder(tree)  
```

**Arguments**  
- **tree**  
  a phylo object with possible singleton nodes (i.e. internal nodes with one daughter node)

**Value**  
a vector of indices of edges in tree$edge in post-order.
Pre-order tree traversal

Description

Pre-order tree traversal

Usage

PCMTreePreorder(tree)

Arguments

tree a phylo object with possible singleton nodes (i.e. internal nodes with one daughter node)

Value

a vector of indices of edges in tree$edge in pre-order.

Set tip and internal node labels in a tree

Description

Set tip and internal node labels in a tree

Usage

PCMTreeSetLabels(
  tree,
  labels = as.character(1:PCMTreeNumNodes(tree)),
  inplace = TRUE
)

Arguments

tree a phylo object or a PCMTree object. If this is a PCMTree object, the internal edge.part and part.regime members will be updated accordingly.

labels a character vector in the order 1:PCMTreeNumNodes(tree) as denoted in the tree$edge matrix.

inplace a logical indicating if the change should be done in place on the object in the calling environment (in this case tree must not be a temporary object, e.g. returned by another function call). Default is TRUE.
Value

if inplace is FALSE, a copy of tree with set or modified tree$tip.label and tree$node.label. If the original tree has a member edge.part, the returned tree has tree$edge.part and tree$part.regime updated. If inplace is TRUE (the default), nothing is returned and the above changes are made directly on the input tree.

See Also

PCMTree

Examples

tree <- ape::rtree(5)
tree$tip.label
# the following three are NULLs
tree$node.label
tree$edge.part
tree$part.regime

tree <- PCMTree(tree)
PCMTreeSetPartition(tree, c(6, 8))
tree$tip.label
tree$node.label
tree$edge.part
tree$part.regime

PCMTreeSetLabels(
  tree, labels = paste0(c(rep("t", 5), rep("n", 4)), PCMTreeGetLabels(tree)))
PCMTreeGetLabels(tree)
tree$tip.label
tree$node.label
tree$edge.part
tree$part.regime
**Arguments**

- **tree**: a PCMTTree object.
- **nodes**: a character vector containing tip or node labels or an integer vector denoting tip or internal nodes in tree - the parts change at the start of the branches leading to these nodes. Default: `c(PCMTTreeNumTips(tree) + 1L)`.
- **inplace**: a logical indicating if the change should be done to the tree in the calling environment (TRUE) or a copy of the tree with set edge.part member should be returned (FALSE). Default is TRUE.

**Value**

If inplace is TRUE nothing, otherwise a copy of the tree with set edge.part member.

**See Also**

- PCMTTreeGetPartition
- PCMTTree

**Examples**

```r
set.seed(1, kind = "Mersenne-Twister", normal.kind = "Inversion")
tree <- PCMTree(ape::rtree(8))
PCMTreeSetLabels(tree, paste0("x", PCMTreeGetLabels(tree)))
PCMTreeGetPartition(tree)
PCMTreeGetPartNames(tree)
PCMTreeGetPartRegimes(tree)

if(requireNamespace("ggtree"))
  PCMTreePlot(tree) + ggtree::geom_nodelab() + ggtree::geom_tiplab()

tree <- PCMTreeSetPartition(tree, c(12, 14), inplace = FALSE)
PCMTreeGetPartition(tree)
PCMTreeGetPartNames(tree)
PCMTreeGetPartRegimes(tree)

if(requireNamespace("ggtree"))
  PCMTreePlot(tree) + ggtree::geom_nodelab() + ggtree::geom_tiplab()

# reset the partition to a default one, where there is only one part:
PCMTreeSetPartition(tree)
PCMTreeGetPartition(tree)
PCMTreeGetPartNames(tree)
PCMTreeGetPartRegimes(tree)

if(requireNamespace("ggtree"))
  PCMTreePlot(tree) + ggtree::geom_nodelab() + ggtree::geom_tiplab()

# reset the labels to the default labels which are character representations
# of the node ids
```
PCMTREESETPARTREGIMES

Description

Set regimes for the parts in a tree

Usage

PCMTREESETPARTREGIMES(tree, part.regime, setPartition = FALSE, inplace = TRUE)

Arguments

- **tree**: a PCMTREE object.
- **part.regime**: a named vector containing regimes to be assigned to some of or to each of the parts in the tree.
- **setPartition**: a logical indicating if the partition of the tree should be set as well. If this argument is set to TRUE, the names of part.regime are passed as the nodes argument in a call to PCMTREESETPARTITION. Default: FALSE.
- **inplace**: a logical indicating if the change should be done to the tree in the calling environment (TRUE) or a copy of the tree with set edge.part member should be returned (FALSE). Default is TRUE.

Value

If inplace is TRUE nothing, otherwise a copy of the tree with set edge.part and part.regime members.

See Also

PCMTREE

Examples

```r
tree <- PCMTREE(ape::rtree(25))
PCMTREEGetPartition(tree)
PCMTREEGetPartRegimes(tree)
PCMTREEGetPartNames(tree)

PCMTREESETPARTREGIMES(tree, c(26^2 = 2))
PCMTREEGetPartition(tree)
PCMTREEGetPartRegimes(tree)
PCMTREEGetPartNames(tree)
```
PCMTreesSetRegimesForEdges

PCMTreesPartRegimes(tree, c("26" = "global-regime"))
PCMTreesPartition(tree)
PCMTreesPartRegimes(tree)
PCMTreesPartNames(tree)

# This should fail because no partition with nodes 26, 28 and 41 has been
# done.
ggplot2::should_stop(
  PCMTreesPartRegimes(tree, c("26" = "a", "28" = "b", "41" = "c")))

# This should succeed and change the partition as well as regime assignment
PCMTreesPartRegimes(
  tree, c("26" = "a", "28" = "b", "41" = "c"), setPartition = TRUE)
PCMTreesPartition(tree)
PCMTreesPartRegimes(tree)
PCMTreesPartNames(tree)

set.seed(1, kind = "Mersenne-Twister", normal.kind = "Inversion")

# number of tips
N <- 40

# tree with one regime
tree.a <- ape::rtree(N)
tree.a <- PCMTrees(tree.a)

PCMTreesPartRegimes(
  tree.a,
  part.regime = structure("a", names = as.character(N+1L)),
  setPartition = TRUE,
  inplace = TRUE)

if(requireNamespace("ggtree"))
PCMTreesPlot(tree.a) + ggtree::geom_nodelab() + ggtree::geom_tiplab()

tree.ab <- tree.a
PCMTreesPartRegimes(
  tree.ab,
  part.regime = structure(c("a", "b"), names = as.character(c(N+1L, N+31L))),
  setPartition = TRUE,
  inplace = TRUE)

if(requireNamespace("ggtree"))
PCMTreesPlot(tree.ab) + ggtree::geom_nodelab() + ggtree::geom_tiplab()
Set the regime for each individual edge in a tree explicitly

Description

Set the regime for each individual edge in a tree explicitly

Usage

```r
PCMTTreeSetRegimesForEdges(tree, regimes, inplace = TRUE)
```

Arguments

- `tree`: a PCMTree or a phylo object.
- `regimes`: a vector of the length equal to `nrow(tree$edge)`. 
- `inplace`: a logical indicating if the change should be done within the tree in the calling environment or a copy of the tree with modified regime assignment should be returned.

Value

- if `inplace` is `TRUE`, nothing, otherwise a modified copy of tree.

Note

Calling this function overwrites the current partitioning of the tree.

Examples

```r
set.seed(1, kind = "Mersenne-Twister", normal.kind = "Inversion")

tree <- ape::rte(10)
regimes <- sample(letters[1:3], nrow(tree$edge), replace = TRUE)
PCMTTreeSetRegimesForEdges(tree, regimes)

if(requireNamespace("ggtree"))
PCMTTreePlot(tree)
```
**PCMTreeSplitAtNode**

Slit a tree at a given internal node into a clade rooted at this node and the remaining tree after dropping this clade

**Description**

Slit a tree at a given internal node into a clade rooted at this node and the remaining tree after dropping this clade.

**Usage**

\[
\text{PCMTreeSplitAtNode}(\text{tree}, \text{node}, \text{tableAncestors} = \text{PCMTreeTableAncestors(tree)}, X = \text{NULL})
\]

**Arguments**

- **tree**: a PCMTTree object.
- **node**: an integer or character indicating a root, internal or tip node.
- **tableAncestors**: an integer matrix returned by a previous call to PCMTreeTableAncestors(tree) or NULL.
- **X**: an optional k x N matrix with trait value vectors for each tip in tree.

**Details**

In the current implementation, the edge.jump and edge.part members of the tree will be discarded and not present in the clade.

**Value**

A list containing two named phylo objects:

- **clade**: The subtree (clade) starting at node.
- **Xclade**: The portion of X attributable to the tips in clade; NULL if X is NULL.
- **rest**: The tree resulting after dropping all tips in the clade.
- **Xrest**: The portion of X attributable to the tips in rest; NULL if X is NULL.
Examples

```r
set.seed(1, kind = "Mersenne-Twister", normal.kind = "Inversion")
tree <- PCMTree(ape::rtree(25))

if(requireNamespace("ggtree"))
  PCMTreePlot(tree) + ggtree::geom_nodelab(angle = 45) +
  ggtree::geom_tiplab(angle = 45)

spl <- PCMTreeSplitAtNode(tree, 28)

if(requireNamespace("ggtree"))
  PCMTreePlot(PCMTree(spl$clade)) + ggtree::geom_nodelab(angle = 45) +
  ggtree::geom_tiplab(angle = 45)

if(requireNamespace("ggtree"))
  PCMTreePlot(PCMTree(spl$rest)) + ggtree::geom_nodelab(angle = 45) +
  ggtree::geom_tiplab(angle = 45)
```

PCMTreeTableAncestors

A matrix (table) of ancestors/descendants for each node in a tree

Description

A matrix (table) of ancestors/descendants for each node in a tree

Usage

`PCMTreeTableAncestors(tree, preorder = PCMTreePreorder(tree))`

Arguments

- `tree`: a phylo object
- `preorder`: an integer vector returned by a previous call to `PCMTreePreorder(tree)`. Default `PCMTreePreorder(tree)`.

Details

This function has time and memory complexity $O(M^2)$, where $M$ is the number of nodes in the tree. It can take several minutes and gigabytes of memory on trees of more than 10000 tips.

Value

an integer square matrix of size $M \times M$ where $M$ is the number of nodes in the tree. Element $j$ on row $i$ is 0 if $j$ is not an ancestor of $i$ or a positive integer equal to the position of $j$ on the path from the root to $i$ if $j$ is an ancestor of $i$. 
**PCMTreeToString**

A character representation of a phylo object.

**Description**

A character representation of a phylo object.

**Usage**

```r
PCMTreeToString(tree, includeLengths = FALSE, includePartition = FALSE)
```

**Arguments**

- **tree**
  - a phylo object.
- **includeLengths**
  - logical. Default: FALSE.
- **includePartition**
  - logical. Default: FALSE.

**Value**

- a character string.

---

**PCMTreeVCV**

Phylogenetic Variance-covariance matrix

**Description**

This is a simplified wrapper for ape’s `vcv` function. Setting the runtime option `PCMBase.UsePCMVarForVCV` to TRUE will switch to a computation of the matrix using the function `PCMVar`.

**Usage**

```r
PCMTreeVCV(tree)
```

**Arguments**

- **tree**
  - a phylo object

**Value**

- a N x N matrix. Assuming a BM model of evolution, this is a matrix in which element (i,j) is equal to the shared root-distance of the nodes i and j.

**See Also**

`vcv`, `PCMVar`, `PCMOptions`
**PCMUnfixParameter**

*Unfix a parameter in a PCM model*

**Description**

Unfix a parameter in a PCM model

**Usage**

`PCMUnfixParameter(model, name)`

**Arguments**

- model: a PCM object
- name: a character string

**Value**

A copy of the model with removed class `_Fixed` from the class of the parameter name

---

**PCMVar**

*Expected variance-covariance matrix for each couple of tips (i,j)*

**Description**

Expected variance-covariance matrix for each couple of tips (i,j)

**Usage**

```r
PCMVar(
  tree,
  model,
  W0 = matrix(0, PCMNumTraits(model), PCMNumTraits(model)),
  SE = matrix(0, PCMNumTraits(model), PCMTreeNumTips(tree)),
  metaI = PCMInfo(NULL, tree, model, verbose = verbose),
  internal = FALSE,
  diagOnly = FALSE,
  verbose = FALSE
)
```
Arguments

tree
a phylo object with N tips.

model
an S3 object specifying both, the model type (class, e.g. "OU") as well as the concrete model parameter values at which the likelihood is to be calculated (see also Details).

W0
a numeric matrix denoting the initial k x k variance covariance matrix at the root (default is the k x k zero matrix).

SE
a k x N matrix specifying the standard error for each measurement in X. Alternatively, a k x k x N cube specifying an upper triangular k x k factor of the variance covariance matrix for the measurement error for each tip i=1, ..., N. When SE is a matrix, the k x k measurement error variance matrix for a tip i is calculated as VE[, , i] <- diag(SE[, i] * SE[, i], nrow = k). When SE is a cube, the way how the measurement variance matrix for a tip i is calculated depends on the runtime option PCMBase.Transpose.Sigma_x as follows:

if getOption("PCMBase.Transpose.Sigma_x", FALSE) == FALSE (default):
  VE[, , i] <- SE[, , i] %*% t(SE[, , i])
if getOption("PCMBase.Transpose.Sigma_x", FALSE) == TRUE:
  VE[, , i] <- t(SE[, , i]) %*% SE[, , i]

Note that the above behavior is consistent with the treatment of the model parameters Sigma_x, Sigmae_x and Sigmaj_x, which are also specified as upper triangular factors. Default: matrix(0.0, PCMNumTraits(model), PCMTreeNumTips(tree)).

metaI
a list returned from a call to PCMInfo(X, tree, model, SE), containing metadata such as N, M and k. Alternatively, this can be a character string naming a function or a function object that returns such a list, e.g. the function PCMInfo or the function PCMInfoCpp from the PCMBaseCpp package.

internal
a logical indicating if the per-node variance-covariances matrices for the internal nodes should be returned (see Value). Default FALSE.

diagOnly
a logical indicating if only the variance blocks for the nodes should be calculated. By default this is set to FALSE, meaning that the co-variances are calculated for all couples of nodes.

verbose
logical indicating if some debug-messages should printed.

Value

If internal is FALSE, a (k x N) x (k x N) matrix W, such that k x k block W[((i-1)*k+1:k), ((j-1)*k+1:k)] equals the expected covariance matrix between tips i and j. Otherwise, a list with an element 'W' as described above and a k x M matrix element 'Wii' containing the per-node variance covariance matrix for each node: The k x k block W[i, (i-1)*k + (1:k)] represents the variance covariance matrix for node i.

Examples

# a Brownian motion model with one regime
modelBM <- PCM(model = "BM", k = 2)
# print the model
modelBM
# assign the model parameters at random: this will use uniform distribution
# with boundaries specified by PCMParamLowerLimit and PCMParamUpperLimit
# We do this in two steps:
# 1. First we generate a random vector. Note the length of the vector equals PCParamCount(modelBM)
randomParams <- PCParamRandomVecParams(modelBM, PCNumTraits(modelBM), PCNumRegimes(modelBM))
randomParams
# 2. Then we load this random vector into the model.
PCParamLoadOrStore(modelBM, randomParams, 0, PCNumTraits(modelBM), PCNumRegimes(modelBM), TRUE)

# create a random tree of 10 tips
tree <- ape::rtree(10)
covMat <- PCMVar(tree, modelBM)

---

**PCMVarAtTime**

*Calculate the variance covariance k x k matrix at time t, under a PCM model*

### Description

Calculate the variance covariance k x k matrix at time t, under a PCM model

### Usage

```
PCMVarAtTime(
  t,
  model,
  W0 = matrix(0, PCNumTraits(model), PCNumTraits(model)),
  SE = matrix(0, PCNumTraits(model), PCNumTraits(model)),
  regime = PCRegimes(model)[1L],
  verbose = FALSE
)
```

### Arguments

- **t**
  - positive numeric denoting time
- **model**
  - a PCM model object
- **W0**
  - a numeric matrix denoting the initial k x k variance covariance matrix at the root (default is the k x k zero matrix).
- **SE**
  - a k x k matrix specifying the upper triangular factor of the measurement error variance-covariance matrix. The product SE Default: SE = matrix(0.0, PCNumTraits(model), PCNumTraits(model)).
- **regime**
  - an integer or a character denoting the regime in model for which to do the calculation; Defaults to PCRegimes(model)[1L], meaning the first regime in the model.
- **verbose**
  - a logical indicating if (debug) messages should be written on the console (Defaults to FALSE).
A numeric k x k matrix

# a Brownian motion model with one regime
modelBM <- PCM(model = "BM", k = 2)
# print the model
modelBM
# assign the model parameters at random: this will use uniform distribution
# with boundaries specified by PCMParamLowerLimit and PCMParamUpperLimit
# We do this in two steps:
# 1. First we generate a random vector. Note the length of the vector equals PCMParamCount(modelBM)
randomParams <- PCMParamRandomVecParams(modelBM, PCMNumTraits(modelBM), PCMNumRegimes(modelBM))
randomParams
# 2. Then we load this random vector into the model.
PCMParamLoadOrStore(modelBM, randomParams, 0, PCMNumTraits(modelBM), PCMNumRegimes(modelBM), TRUE)

# PCMVarAtTime(1, modelBM)

# note that the variance at time 0 is not the 0 matrix because the model has a non-zero
# environmental deviation
PCMVarAtTime(0, modelBM)

Check if all packages listed in Suggests are available

Check if all packages listed in Suggests are available

Check if all packages listed in Suggests are available

RequireSuggestedPackages()

logical TRUE if suggested packages are installed and can be loaded; FALSE otherwise
**TruePositiveRate**

*True positive rate of a set of binary predictions against their trues*

**Description**

Let the set of predictions be described by a logical vector ‘pred’, and let the corresponding trues by described in a logical vector ‘true’ of the same length. Then, the true positive rate is given by the expression: \( \frac{\text{sum}(\text{pred} \& \text{true})}{\text{sum}(\text{true})} \). The false positive rate is given by the expression: \( \frac{\text{sum}(\text{pred} \& \neg \text{true})}{\text{sum}(\neg \text{true})} \). If these expressions do not give a finite number, \( \text{NA}\_\text{real} \) is returned.

**Usage**

TruePositiveRate(pred, true)

FalsePositiveRate(pred, true)

**Arguments**

pred, true  
 vectors of the same positive length that can be converted to logical.

**Value**

a double between 0 and 1 or \( \text{NA}\_\text{real} \) if the result is not a finite number.

**Examples**

TruePositiveRate(c(1,0,1,1), c(1,1,0,1))
TruePositiveRate(c(0,0,0,0), c(1,1,0,1))
TruePositiveRate(c(1,1,1,1), c(1,1,0,1))
FalsePositiveRate(c(1,0,1,1), c(1,1,0,1))
FalsePositiveRate(c(0,0,0,0), c(1,1,0,1))
FalsePositiveRate(c(1,1,1,1), c(1,1,0,1))
TruePositiveRate(c(1,0,1,1), c(0,0,0,0))
FalsePositiveRate(c(1,0,1,1), c(1,1,1,1))

**UpperTriFactor**

*Upper triangular factor of a symmetric positive definite matrix*

**Description**

This function is an analog to the Cholesky decomposition.

**Usage**

UpperTriFactor(Sigma)
Arguments

Sigma  A symmetric positive definite k x k matrix that can be passed as argument to chol.

Value

an upper triangular matrix Sigma_x, such that Sigma = Sigma_x %*% t(Sigma_x)

See Also

chol;
the option PCMBase.Transpose.Sigma_x in PCMOptions.

Examples

# S is a symmetric positive definite matrix
M<-matrix(rexp(9),3,3); S <- M %*% t(M)

# This should return a zero matrix:
UpperTriFactor(S) %*% t(UpperTriFactor(S)) - S

# This should return a zero matrix too:
t(chol(S)) %*% chol(S) - S

# Unless S is diagonal, in the general case, this will return a
# non-zero matrix:
chol(S) %*% t(chol(S)) - S

White  White Gaussian PCM ignoring phylogenetic history

Description

White model ignoring phylogenetic history, treating trait values as independent samples from a k-variate Gaussian.

Details

Calculating likelihoods for this model does not work if the global option PCMBase.Singular.Skip is set to FALSE.
Index

* datasets
  - dataFig3, 8
  - PCMBaseTestObjects, 20
  - Args_MixedGaussian_MGPMDefaultModelTypes, 5
  - Args_MixedGaussian_MGPMScalarOUType, 6
  - Args_MixedGaussian_MGPMSurfaceOUType, 6
  - as.MixedGaussian, 7, 14
  - chol, 51, 121
  - dataFig3, 8

- FalsePositiveRate (TruePositiveRate), 120
- FormatCellAsLatex, 8
- FormatTableAsLatex, 9

- grep, 11

- is.AllEqual (PCParamType), 60
- is.CholeskyFactor (PCParamType), 60
- is.Diagonal (PCParamType), 60
- is.Fixed (PCParamType), 60
- is.GaussianPCM, 9
- is.Global (PCParamType), 60
- is.Identity (PCParamType), 60
- is.Local (PCParamType), 60
- is.LowerTriangular (PCParamType), 60
- is.LowerTriangularWithDiagonal (PCParamType), 60
- is.MatrixParameter (PCParamType), 60
- is.MixedGaussian, 10
- is.NonNegative (PCParamType), 60
- is.Omitted (PCParamType), 60
- is.Ones (PCParamType), 60
- is.PCM, 10
- is.PCMTree, 11
- is.ScalarDiagonal (PCParamType), 60
- is.ScalarParameter (PCParamType), 60
- is.Schur (PCParamType), 60
- is.SemiPositiveDefinite (PCParamType), 60
- is.Symmetric (PCParamType), 60
- is.Transformable, 20
- is.Transformable (PCParamType), 60
- is.Transformed (PCParamType), 60
- is.UpperTriangular (PCParamType), 60
- is.UpperTriangularWithDiagonal (PCParamType), 60
- is.VectorParameter (PCParamType), 60
- is.WithCustomVecParams (PCParamType), 60
- is.WithNonNegativeDiagonal (PCParamType), 60
- is.Zeros (PCParamType), 60
- isSymmetric, 51

- MatchListMembers, 11, 19, 34, 71

- methods, 48

- MGPMDefaultModelTypes
  - (PCMDefaultModelTypes), 27
- MGPMScalarOUType, 12
- MGPMSurfaceOUType, 12
- MixedGaussian, 13, 15, 16
- MixedGaussianTemplate, 14

- PCM, 15
- PCMAddEf, 17, 39, 42
- PCMAddToListAttribute, 18
- PCMApplyTransformation, 19
- PCMBaseIsADevRelease, 20
- PCMBaseTestObjects, 20
- PCMClaColorPalette, 22
- PCMCondMatchListAttribute, 22
- PCMCond, 23, 39, 42, 72
- PCMCond.GaussianPCM, 24
- PCMCondVOU, 25
- PCMCreateLikelihood, 26

122
<table>
<thead>
<tr>
<th>Function/Method</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCMDefaultModelTypes</td>
<td>27</td>
</tr>
<tr>
<td>PCMDefaultObject</td>
<td>28</td>
</tr>
<tr>
<td>PCMDescribe</td>
<td>29</td>
</tr>
<tr>
<td>PCMDescribeParameters</td>
<td>29</td>
</tr>
<tr>
<td>PCMExtractDimensions</td>
<td>30</td>
</tr>
<tr>
<td>PCMExtractRegimes</td>
<td>30</td>
</tr>
<tr>
<td>PCMFindMethod</td>
<td>31</td>
</tr>
<tr>
<td>PCMFixParameter</td>
<td>32</td>
</tr>
<tr>
<td>PCMGenerateModelTypes</td>
<td>32</td>
</tr>
<tr>
<td>PCMGenerateParameterizations</td>
<td>33, 43</td>
</tr>
<tr>
<td>PCMGetAttribute</td>
<td>34</td>
</tr>
<tr>
<td>PCMGetVecParamsRegimesAndModels</td>
<td>35</td>
</tr>
<tr>
<td>PCMInfo</td>
<td>27, 35, 39, 42, 52, 72</td>
</tr>
<tr>
<td>PCMLik</td>
<td>17, 37, 37, 69, 72</td>
</tr>
<tr>
<td>PCMLikDmvNorm</td>
<td>39</td>
</tr>
<tr>
<td>PCMLikTrace</td>
<td>40</td>
</tr>
<tr>
<td>PCMListDefaultParameterizations (PCMListParameterizations), 43</td>
<td></td>
</tr>
<tr>
<td>PCMListMembers</td>
<td>12, 42</td>
</tr>
<tr>
<td>PCMListParameterizations</td>
<td>43</td>
</tr>
<tr>
<td>PCMLmr</td>
<td>39, 42, 44</td>
</tr>
<tr>
<td>PCMMapModelTypesToRegimes</td>
<td>45</td>
</tr>
<tr>
<td>PCMMean</td>
<td>46</td>
</tr>
<tr>
<td>PCMMeanAtTime</td>
<td>47</td>
</tr>
<tr>
<td>PCMMModels</td>
<td>14, 48</td>
</tr>
<tr>
<td>PCMMModelTypes</td>
<td>49</td>
</tr>
<tr>
<td>PCMNumRegimes</td>
<td>49</td>
</tr>
<tr>
<td>PCMNumTraits</td>
<td>50</td>
</tr>
<tr>
<td>PCMOpts</td>
<td>37, 50, 99, 115, 121</td>
</tr>
<tr>
<td>PCMPairSums</td>
<td>52</td>
</tr>
<tr>
<td>PCMParParam</td>
<td>53</td>
</tr>
<tr>
<td>PCMParParamBindRegimeParams</td>
<td>53</td>
</tr>
<tr>
<td>PCMParParamCount</td>
<td>55</td>
</tr>
<tr>
<td>PCMParParamGetShortVector</td>
<td>56</td>
</tr>
<tr>
<td>PCMParParamLoadOrStore</td>
<td>56</td>
</tr>
<tr>
<td>PCMParParamLocateInShortVector</td>
<td>57</td>
</tr>
<tr>
<td>PCMParParamLowerLimit</td>
<td>58</td>
</tr>
<tr>
<td>PCMParParamRandomVecParams</td>
<td>58</td>
</tr>
<tr>
<td>PCMParParamSetByName</td>
<td>59</td>
</tr>
<tr>
<td>PCMParParamType</td>
<td>60</td>
</tr>
<tr>
<td>PCMParParamUpperLimit</td>
<td>63</td>
</tr>
<tr>
<td>PCMParentClasses</td>
<td>63</td>
</tr>
<tr>
<td>PCMParseErrorMessage</td>
<td>42, 64</td>
</tr>
<tr>
<td>PCMExpMeanExp</td>
<td>51, 64</td>
</tr>
<tr>
<td>PCMLambdaP_1</td>
<td>65</td>
</tr>
<tr>
<td>PCMPPlotGaussianDensityGrid2D</td>
<td>66</td>
</tr>
<tr>
<td>PCMPPlotGaussianSample2D</td>
<td>66</td>
</tr>
<tr>
<td>PCMPPlotMath</td>
<td>67</td>
</tr>
<tr>
<td>PCMPPlotTraitData2D</td>
<td>68</td>
</tr>
<tr>
<td>PCMPresentCoordinates</td>
<td>26, 36-38, 40, 41, 44, 52, 69</td>
</tr>
<tr>
<td>PCMRregimes</td>
<td>70, 92, 93</td>
</tr>
<tr>
<td>PCMSetAttribute</td>
<td>70</td>
</tr>
<tr>
<td>PCMSim</td>
<td>39, 42, 71</td>
</tr>
<tr>
<td>PCMSpecify</td>
<td>16, 48, 73</td>
</tr>
<tr>
<td>PCMTable</td>
<td>73</td>
</tr>
<tr>
<td>PCMTableParameterizations</td>
<td>74</td>
</tr>
<tr>
<td>PCMTrajectory</td>
<td>75</td>
</tr>
<tr>
<td>PCMTree</td>
<td>77, 108-110</td>
</tr>
<tr>
<td>PCMTreeBackbonePartition</td>
<td>79</td>
</tr>
<tr>
<td>PCMTreeDropClade</td>
<td>81</td>
</tr>
<tr>
<td>PCMTreeDtnodes</td>
<td>82</td>
</tr>
<tr>
<td>PCMTreeEdgeTimes</td>
<td>83, 95</td>
</tr>
<tr>
<td>PCMTreeEvalNestedEDxOnTree</td>
<td>84</td>
</tr>
<tr>
<td>PCMTreeExtractClade</td>
<td>85</td>
</tr>
<tr>
<td>PCMTreeGetBranchLength</td>
<td>86</td>
</tr>
<tr>
<td>PCMTreeGetDaughters</td>
<td>87</td>
</tr>
<tr>
<td>PCMTreeGetLabels</td>
<td>87</td>
</tr>
<tr>
<td>PCMTreeGetNodeLabels</td>
<td>87</td>
</tr>
<tr>
<td>PCMTreeGetParent</td>
<td>88</td>
</tr>
<tr>
<td>PCMTreeGetPartition</td>
<td>89, 93, 109</td>
</tr>
<tr>
<td>PCMTreeGetPartNames</td>
<td>14, 89, 92, 93</td>
</tr>
<tr>
<td>PCMTreeGetPartRegimes</td>
<td>90, 92, 93</td>
</tr>
<tr>
<td>PCMTreeGetPartsForNodes</td>
<td>90</td>
</tr>
<tr>
<td>PCMTreeGetRegimesForEdges</td>
<td>91</td>
</tr>
<tr>
<td>PCMTreeGetRegimesForNodes</td>
<td>91</td>
</tr>
<tr>
<td>PCMTreeGetRootLabel (PCMTreeGetLabels), 87</td>
<td></td>
</tr>
<tr>
<td>PCMTreeGetTipLabels (PCMTreeGetLabels), 87</td>
<td></td>
</tr>
<tr>
<td>PCMTreeGetTipsInPart</td>
<td>92, 93</td>
</tr>
<tr>
<td>PCMTreeGetTipsInRegime</td>
<td>92, 93</td>
</tr>
<tr>
<td>PCMTreeInsertSingletons</td>
<td>94</td>
</tr>
<tr>
<td>PCMTreeInsertSingletonsAtEpoch</td>
<td>94</td>
</tr>
<tr>
<td>PCMTreeGetTipLabels (PCMTreeInsertSingletons), 94</td>
<td></td>
</tr>
<tr>
<td>PCMTreeInsertTipsOrSingletons</td>
<td>94</td>
</tr>
<tr>
<td>PCMTreeInsertSingletonsAtEpoch</td>
<td>94</td>
</tr>
<tr>
<td>PCMTreeJumps</td>
<td>96</td>
</tr>
<tr>
<td>PCMTreeListAllPartitions</td>
<td>97</td>
</tr>
<tr>
<td>PCMTreeListCladePartitions</td>
<td>51, 98</td>
</tr>
<tr>
<td>PCMTreeListDescendants</td>
<td>99</td>
</tr>
<tr>
<td>PCMTreeListRootPaths</td>
<td>100</td>
</tr>
<tr>
<td>PCMTreeLocateEpochOnBranches</td>
<td>95, 100</td>
</tr>
<tr>
<td>PCMTreeLocateMidpointsOnBranches</td>
<td>95, 101</td>
</tr>
</tbody>
</table>
PCMTreeMatchLabels, 101
PCMTreeMatrixNodesInSamePart, 102
PCMTreeMatrixNodesInSameRegime
  (PCMTreeMatrixNodesInSamePart), 102
PCMTreeNearestNodesToEpoch, 103
PCMTreeNodeTimes, 104
PCMTreeNumNodes, 104
PCMTreeNumParts, 105
PCMTreeNumTips, 105
PCMTreePlot, 106
PCMTreePostorder, 106
PCMTreePreorder, 107
PCMTreeSetLabels, 107
PCMTreeSetPartition, 89, 108
PCMTreeSetPartRegimes, 92, 93, 110
PCMTreeSetRegimesForEdges, 111
PCMTreeSplitAtNode, 113
PCMTreeTableAncestors, 114
PCMTreeToString, 115
PCMTreeVCV, 52, 115
PCMUnfixParameter, 116
PCMVar, 52, 115
PCMVarAtTime, 118
rcond, 65
RequireSuggestedPackages, 119
stop, 39
TruePositiveRate, 120
tryCatch, 39
UpperTriFactor, 51, 120
vcv, 52, 115
White, 121