Package 'selac'

March 1, 2019

Type Package

Title Selection Models for Amino Acid and/or Codon Evolution

Version 1.7.0

Date 2019-01-26

Maintainer Jeremy Beaulieu <jbeaulieu@nimbios.org>

Description Sets up and executes a SelAC model (Selection on Amino acids and codons) for testing the presence of selection in amino acid or codon among a set of genes on a fixed phylogeny. Beaulieu et al (2019) <doi:10.1093/molbev/msy222>.

Suggests testthat, igraph

Imports ape, deSolve, nloptr, nnet, data.table, expm, MASS, Matrix, methods, parallel, phangorn, seqinr, statmod, zoo, RColorBrewer

License GPL

ByteCompile TRUE

RoxygenNote 6.1.1

NeedsCompilation no

Author Jeremy Beaulieu [aut, cre],
JJ Chai [aut],
Mike Gilchrist [aut],
Cedric Landerer [aut],
Graham Derryberry [aut],
Brian O'Meara [aut]

Repository CRAN

Date/Publication 2019-02-28 23:00:03 UTC

R topics documented:

`GetAdequateManyReps` *Parallel model adequacy test*

Description

Performs model adequacy test using multiple cores

Usage

```
GetAdequateManyReps(nreps, n.cores, model.to.reconstruct.under = "selac",
  model.to.simulate.under = "gtr", selac.obj.to.reconstruct,
  selac.obj.to.simulate, aa.optim.input = NULL,
  fasta.rows.to.keep = NULL, taxon.to.drop = 2,
  partition.number = 17, numcode = 1, for.gtr.only = NULL)
```

Arguments

`nreps`	Specifies the number of repeated model adequact simulations.
`n.cores`	Specifies the number of cores you want to use.
`model.to.reconstruct.under`	Specifies the model that the internal nodes are to be reconstructed under assuming a single tip is pruned from the tree.
`model.to.simulate.under`	Specifies the model that the simulation will be conducted along the pruned tip.
`selac.obj.to.reconstruct`	The selac output object that contains the model parameters to be used in the reconstruction.
`selac.obj.to.simulate`	The selac output object that contains the model parameters to be used in the simulation.

`aa.optim.input` A list of optimal amino acids with each list element designating a character vector for each gene. The optimal amino acids be the MLE from a selac run (default) or a list of user defined optimal A.A.

`fasta.rows.to.keep`
Indicates which rows to remove in the input fasta files.

`taxon.to.drop` Specifies the tip based on the number in the phy object to be removed and simulated.

`partition.number`
Specifies the partition number to conduct the model adequacy test.

`numcode` The ncbi genetic code number for translation. By default the standard (numcode=1) genetic code is used.

`for.gtr.only` A selac object that can be used as the reference optimal AA for when the adequacy of a GTR+G model is tested only.

Details

Performs a parallelized analysis of the model adequacy test. The test prunes out a user-specified taxon from the tree, performs site data reconstruction for all nodes in the tree under a user-specified model, then simulates the expected data of the pruned taxon according to a user-specified model along uniformly sampled points along the branch. The functionality of the reconstructed sequence is also calculated along the way to see how functionality changes as the simulation reaches the end of the known branch length. The output is a list with elements equally the number of repititions. Each element contains the functionality of the simulated points along equally spaced sampling points along the known branch length (i.e., edge.length * seq(0, 1, by=0.05))

`GetAdequateSelac` *Model adequacy simulation*

Description

Performs a single model adequacy simulation

Usage

```
GetAdequateSelac(model.to.reconstruct.under, model.to.simulate.under,
  selac.obj.to.reconstruct, selac.obj.to.simulate, aa.optim.input = NULL,
  fasta.rows.to.keep = NULL, taxon.to.drop = 4,
  partition.number = 55, numcode = 1, for.gtr.only = NULL)
```

Arguments

`model.to.reconstruct.under`
Specifies the model that the internal nodes are to be reconstructed under assuming a single tip is pruned from the tree.

`model.to.simulate.under`
Specifies the model that the simulation will be conducted along the pruned tip.

`selac.obj.to.reconstruct`
: The selac output object that contains the model parameters to be used in the reconstruction.

`selac.obj.to.simulate`
: The selac output object that contains the model parameters to be used in the simulation.

`aa.optim.input`
: A list of optimal amino acids with each list element designating a character vector for each gene. The optimal amino acids be the MLE from a selac run (default) or a list of user defined optimal A.A.

`fasta.rows.to.keep`
: Indicates which rows to remove in the input fasta files.

`taxon.to.drop`
: Specifies the tip based on the number in the phy object to be removed and simulated.

`partition.number`
: Specifies the partition number to conduct the model adequacy test.

`numcode`
: The ncbi genetic code number for translation. By default the standard (numcode=1) genetic code is used.

`for.gtr.only`
: A selac object that can be used as the reference optimal AA for when the adequacy of a GTR+G model is tested only.

Details

Performs a single model adequacy simulation. The test prunes out a user-specified taxon from the tree, performs site data reconstruction for all nodes in the tree under a user-specified model, then simulates the expected data of the pruned taxon according to a user-specified model along uniformly sampled points along the branch. The functionality of the reconstructed sequence is also calculated along the way to see how functionality changes as the simulation reaches the end of the known branch length. The output is a vector with elements containing the functionality of the simulated points along equally spaced sampling points along the known branch length (i.e., edge.length * seq(0, 1, by=0.05))

`GetFunctionality` *Calculate functionality*

Description

Calculates the functionality of a single gene

Usage

```
GetFunctionality(gene.length, aa.data, optimal.aa, alpha, beta, gamma,
  gp = NULL, aa.properties = NULL)
```

Arguments

gene.length	Indicates the length of the gene used to calculate functionality.
aa.data	A matrix of amino acids
optimal.aa	A vector of inferred optimal amino acids.
alpha	The inferred Grantham composition paramter
beta	The inferred Grantham polarity parameter
gamma	The inferred Grantham molecular volume parameter
gp	A vector of gamma rates for calculating among site hetergeneity in functionality.
aa.properties	User-supplied amino acid distance properties. By default we assume Grantham (1974) properties.

Details

The purpose of this function is to provide the functionality of a gene based on the inferred parameters from SelAC. The functionality is often used to scale phi.

GetMarginalAllGenes *Get marginal reconstruction all genes*

Description

Calculates the marginal probability of each codon at all sites across all genes

Usage

```
GetMarginalAllGenes(selac.obj, aa.optim.input = NULL,
  fasta.rows.to.keep = NULL, taxon.to.drop, partition.number = NULL)
```

Arguments

selac.obj	An object of class SELAC.
aa.optim.input	A list of optimal amino acids with each list element designating a character vector for each gene. The optimal amino acids be the MLE from a selac run (default) or a list of user defined optimal A.A.
fasta.rows.to.keep	Indicates which rows to remove in the input fasta files.
taxon.to.drop	A single taxon (defined by number in phy object) to be removed from the reconstruction.
partition.number	If only a single gene is desired to be reconstructed, the input is the partition number in the selac object.

Details

Provides marginal probabilities for all nodes across all genes. The function is fairly simple to use as it only requires as input the selac output object and the working directory that the original analysis took place.

`GetPartitionOrder` *Get data partiion order*

Description

Provides the order of the partitions after the data is read into SELAC.

Usage

```
GetPartitionOrder(codon.data.path)
```

Arguments

`codon.data.path`
: Provides the path to the directory containing the gene specific fasta files of coding data. Must have a ".fasta" line ending.

Details

Provides the order of the partitions when the data is read into SELAC. This function is mainly useful for when users want to supply their own optimal amino acid list into SELAC.

`GetSelacPhiCat` *Phi rate category information under SELAC+gamma*

Description

Provides likelihood information and best rates across sites and across genes under SELAC+gamma

Usage

```
GetSelacPhiCat(selac.obj, codon.data.path, aa.optim.input = NULL,
  fasta.rows.to.keep = NULL, n.cores.by.gene.by.site = 1)
```

Arguments

selac.obj	An object of class SELAC.
codon.data.path	Provides the path to the directory containing the gene specific fasta files of coding data.
aa.optim.input	A list of optimal amino acids with each list element designating a character vector for each gene. The optimal amino acids be the MLE from a selac run (default) or a list of user defined optimal A.A.
fasta.rows.to.keep	Indicates which rows to remove in the input fasta files.
n.cores.by.gene.by.site	The number of cores to decidate to parallelize analyses by site WITHIN a gene. Note n.cores.by.gene*n.cores.by.gene.by.site is the total number of cores dedicated to the analysis.

Details

The purpose of this function is to determine which rate category best fits each site across genes. The output is a list object, with each list entry designating the optimal rate category across sites for that gene.

GetSelacSiteLikelihoods

Calculate site likelihoods under SelAC

Description

Calculates the likelihoods across sites and across genes under SELAC

Usage

```
GetSelacSiteLikelihoods(selac.obj, codon.data.path,
  aa.optim.input = NULL, fasta.rows.to.keep = NULL)
```

Arguments

selac.obj	An object of class SELAC.
codon.data.path	Provides the path to the directory containing the gene specific fasta files of coding data.
aa.optim.input	A list of optimal amino acids with each list element designating a character vector for each gene. The optimal amino acids be the MLE from a selac run (default) or a list of user defined optimal A.A.
fasta.rows.to.keep	Indicates which rows to remove in the input fasta files.

Details

The purpose of this function is to provide the site likelihoods across genes. It is also flexible in that it allows different hypotheses about optimal acids across genes and/or site. The output is a list object, with each list entry designating 1) the tot.likelihood for that gene, and 2) the site likelihoods for that gene.

`NucSimulator` *Simulate DNA under General-Time Reversible model*

Description

Simulates nucleotide data based on parameters under the GTR+G model

Usage

```
NucSimulator(phy, pars, nsites, nuc.model, base.freqs, ncats)
```

Arguments

`phy`	The phylogenetic tree with branch lengths.
`pars`	A vector of parameters used for the simulation. They are ordered as follows: gamma shape and the rates for the nucleotide model.
`nsites`	The number of sites to simulate.
`nuc.model`	Indicates what type nucleotide model to use. There are three options: "JC", "GTR", or "UNREST".
`base.freqs`	The base frequencies for A C G T (in that order).
`ncats`	The number of discrete gamma categories.

Details

Simulates a nucleotide matrix using parameters under the GTR+G model. Note that the output can be written to a fasta file using the write.dna() function in the ape package.

Examples

```
pp <- SelacOptimize(codon.data.path="FOLDER_WITH_FASTA/", phy=phy,
data.type='nucleotide', optimal.aa='none', codon.model='none', nuc.model='GTR',
include.gamma=TRUE, gamma.type='quadrature')
sim.dat <- NucSimulator(phy=pp$phy, pars=pp$mle.pars, nsites=pp$nsites, nuc.model=pp$nuc.model)
```

PlotEquilbriumCodonDistribution

Function to plot a distribution of frequencies of codons given a 3d array of equilibrium frequency matrices

Description

Function to plot a distribution of frequencies of codons given a 3d array of equilibrium frequency matrices

Usage

```
PlotEquilbriumCodonDistribution(eq.freq.matrices, values,
  palette = "Set1", lwd = 2, ...)
```

Arguments

eq.freq.matrices	A 3d array of eq.freq.matrix returned from ComputeEquilibriumFrequencies
values	The vector of labels for each matrix (i.e., different Phi values)
palette	Color palette to use from RColorBrewer
lwd	Line width
...	Other paramters to pass to plot()

PlotExpectedFitness

Function to plot a distribution of fitnesses based on codon equilibrium freqs

Description

Function to plot a distribution of fitnesses based on codon equilibrium freqs

Usage

```
PlotExpectedFitness(codon.fitnesses.matrices, codon.eq.matrices, values,
  optimal.aa = NULL, palette = "Set1", lwd = 2,
  include.stop.codon = FALSE, type = "histogram", fitness = TRUE,
  numcode = 1, ...)
```

Arguments

codon.fitnesses.matrices	A 3d array of aa.fitness.matrix returned from ComputeEquilibriumAAFitness (first element in return)
codon.eq.matrices	A 3d array of codon equilibrium frequencies
values	The vector of labels for each matrix (i.e., different Phi values)
optimal.aa	Single letter code for the optimal aa. If NULL, integrates across aa.
palette	Color palette to use from RColorBrewer
lwd	Line width
include.stop.codon	Include stop codons
type	If "histogram", do a histogram plot; if "density", do a density plot
fitness	If TRUE, plot W; if FALSE, plot S (= 1 - W)
numcode	The genetic code
...	Other paramters to pass to plot()

PlotGeneSiteInfo *Function to plot info by site in a gene*

Description

Function to plot info by site in a gene

Usage

```
PlotGeneSiteInfo(all.info, aa.properties = NULL, mean.width = 10)
```

Arguments

all.info	The output of GetGeneSiteInfo
aa.properties	The aa.properties you want to use; if NULL, uses Grantham
mean.width	Sliding window width

PlotMutationFitnessSpectra

Plot fitness of mutations, weighted by frequency of those mutations

Description

Plot fitness of mutations, weighted by frequency of those mutations

Usage

```
PlotMutationFitnessSpectra(mutation.fitness.object.list, values,
  optimal.aa = NULL, palette = "Set1", lwd = 2, ...)
```

Arguments

mutation.fitness.object.list	List that contains multiple objects from ComputeMutationFitnesses() calls
values	The vector of labels for each matrix (i.e., different Phi values)
optimal.aa	Single letter code for the optimal aa. If NULL, integrates across aa.
palette	Color palette to use from RColorBrewer
lwd	Line width
...	other arguments to pass to plot()

PlotPerAAFitness

Function to plot a distribution of fitnesses W or selection coefficients S for a given optimal aa and other terms.

Description

Function to plot a distribution of fitnesses W or selection coefficients S for a given optimal aa and other terms.

Usage

```
PlotPerAAFitness(aa.fitness.matrices, values, optimal.aa = NULL,
  palette = "Set1", lwd = 2, include.stop.codon = FALSE,
  type = "histogram", fitness = TRUE, scale.x.axis.by.Ne = FALSE,
  legend.title = NULL, Ne = 10^6, ...)
```

Arguments

aa.fitness.matrices,	A 3d array of aa.fitness.matrix returned from ComputeEquilibriumAAFitness (first element in return)
values	The vector of labels for each matrix (i.e., different Phi values)
optimal.aa	Single letter code for the optimal aa. If NULL, integrates across aa.
palette	Color palette to use from RColorBrewer
lwd	Line width
include.stop.codon	Include stop codons
type	If "histogram", do a histogram plot; if "density", do a density plot
fitness	If TRUE, plot fitness W; if FALSE, plot selection coefficient S (= W- 1)
scale.x.axis.by.Ne	if TRUE, x axis is transformed from S to S*Ne; if FALSE no scaling is done
legend.title	Sets the title of the figure legend.
Ne	used to scale x axis when scale.x.axis.by.Ne is TRUE
...	Other paramters to pass to plot()

---

selac example *Example yeast dataset*

Description

Example gene, tree, and model output file.

SelacOptimize *Efficient optimization of the SELAC model*

Description

Efficient optimization of model parameters under the SELAC model

Usage

```
SelacOptimize(codon.data.path, n.partitions = NULL, phy,
  data.type = "codon", codon.model = "selac",
  edge.length = "optimize", edge.linked = TRUE,
  optimal.aa = "optimize", nuc.model = "GTR", include.gamma = FALSE,
  gamma.type = "quadrature", ncats = 4, numcode = 1,
  diploid = TRUE, k.levels = 0, aa.properties = NULL,
  verbose = FALSE, n.cores.by.gene = 1, n.cores.by.gene.by.site = 1,
  max.tol = 0.001, max.tol.edges = 0.001, max.evals = 1e+06,
```

```
  max.restarts = 3, user.optimal.aa = NULL,
  fasta.rows.to.keep = NULL, recalculate.starting.brlen = TRUE,
  output.by.restart = TRUE, output.restart.filename = "restartResult",
  user.supplied.starting.param.vals = NULL, tol.step = 1,
  optimizer.algorithm = "NLOPT_LN_SBPLX", start.from.mle = FALSE,
  mle.matrix = NULL, partition.order = NULL, max.iterations = 6)
```

Arguments

codon.data.path	Provides the path to the directory containing the gene specific fasta files of coding data. Must have a ".fasta" line ending.
n.partitions	The number of partitions to analyze. The order is based on the Unix order of the fasta files in the directory.
phy	The phylogenetic tree to optimize the model parameters.
data.type	The data type being tested. Options are "codon" or "nucleotide".
codon.model	The type of codon model to use. There are four options: "none", "GY94", "YN98", "FMutSel0", "FMutSel", "selac".
edge.length	Indicates whether or not edge lengths should be optimized. By default it is set to "optimize", other option is "fixed", which is the user-supplied branch lengths.
edge.linked	A logical indicating whether or not edge lengths should be optimized separately for each gene. By default, a single set of each lengths is optimized for all genes.
optimal.aa	Indicates what type of optimal.aa should be used. There are five options: "none", "majrule", "averaged, "optimize", or "user".
nuc.model	Indicates what type nucleotide model to use. There are three options: "JC", "GTR", or "UNREST".
include.gamma	A logical indicating whether or not to include a discrete gamma model.
gamma.type	Indicates what type of gamma distribution to use. Options are "quadrature" after the Laguerre quadrature approach of Felsenstein 2001 or median approach of Yang 1994 or "lognormal" after a lognormal quadrature approach.
ncats	The number of discrete categories.
numcode	The ncbi genetic code number for translation. By default the standard (numcode=1) genetic code is used.
diploid	A logical indicating whether or not the organism is diploid or not.
k.levels	Provides how many levels in the polynomial. By default we assume a single level (i.e., linear).
aa.properties	User-supplied amino acid distance properties. By default we assume Grantham (1974) properties.
verbose	Logical indicating whether each iteration be printed to the screen.
n.cores.by.gene	The number of cores to dedicate to parallelize analyses across gene.
n.cores.by.gene.by.site	The number of cores to decidate to parallelize analyses by site WITHIN a gene. Note n.cores.by.gene*n.cores.by.gene.by.site is the total number of cores dedicated to the analysis.

`max.tol` Supplies the relative optimization tolerance.

`max.tol.edges` Supplies the relative optimization tolerance for branch lengths only. Default is that is the same as the max.tol.

`max.evals` Supplies the max number of iterations tried during optimization.

`max.restarts` Supplies the number of random restarts.

`user.optimal.aa`
: If optimal.aa is set to "user", this option allows for the user-input optimal amino acids. Must be a list. To get the proper order of the partitions see "GetPartitionOrder" documentation.

`fasta.rows.to.keep`
: Indicates which rows to remove in the input fasta files.

`recalculate.starting.brlen`
: Whether to use given branch lengths in the starting tree or recalculate them.

`output.by.restart`
: Logical indicating whether or not each restart is saved to a file. Default is TRUE.

`output.restart.filename`
: Designates the file name for each random restart.

`user.supplied.starting.param.vals`
: Designates user-supplied starting values for C.q.phi.Ne, Grantham alpha, and Grantham beta. Default is NULL.

`tol.step` If > 1, makes for coarser tolerance at earlier iterations of the optimizer

`optimizer.algorithm`
: The optimizer used by nloptr.

`start.from.mle` If TRUE, will start optimization from the MLE. Default is FALSE.

`mle.matrix` The user-supplied matrix of parameter values for when start.from.mle is set to TRUE.

`partition.order`
: Allows for a specialized order of the partitions to be gathered from the working directory.

`max.iterations` Sets the number of cycles to optimize the different parts of the model.

Details

Here we optimize parameters across each gene separately while keeping the shared parameters, alpha, beta, edge lengths, and nucleotide substitution parameters constant across genes. We then optimize alpha, beta, gtr, and the edge lengths while keeping the rest of the parameters for each gene fixed. This approach is potentially more efficient than simply optimizing all parameters simultaneously, especially if fitting models across 100's of genes.

Examples

```
data(yeast)
system(paste("mkdir", "fastaSet", sep=" "))
sim.dat <- SelacSimulator(phy=phy, pars=model.fit$mle.pars[1,], aa.optim_array=
model.fit$aa.optim[[1]], codon.freq.by.aa=model.fit$codon.freq.by.aa[[1]],
```

```
codon.freq.by.gene=model.fit$codon.freq.by.gene[[1]], nuc.model=model.fit$nuc.model)
write.dna(sim.dat, file=paste("fastaSet", "/gene",  "TEST", ".fasta", sep=""),
format="fasta", colw=1000000)
pp <- SelacOptimize(codon.data.path="fastaSet/", phy=phy,
data.type="codon", nuc.model="GTR", include.gamma=TRUE, n.cores.by.gene=3)
```

`SelacSimulator` *Simulate DNA under the SELAC model*

Description

Simulates nucleotide data based on parameters under the SELAC model

Usage

```
SelacSimulator(phy, pars, aa.optim_array, codon.freq.by.aa = NULL,
  codon.freq.by.gene = NULL, numcode = 1, aa.properties = NULL,
  nuc.model, include.gamma = FALSE, gamma.type = "quadrature",
  ncats = 4, k.levels = 0, diploid = TRUE, site.cats.vector = NULL)
```

Arguments

`phy`	The phylogenetic tree with branch lengths.
`pars`	A vector of parameters used for the simulation. They are ordered as follows: C.q.phi, alpha, beta, Ne, base.freqs for A C G, and the rates for the nucleotide model.
`aa.optim_array`	A vector of optimal amino acids for each site to be simulated.
`codon.freq.by.aa`	A matrix of codon frequencies for each possible optimal amino acid. Rows are aa (including stop codon), cols are codons.
`codon.freq.by.gene`	A matrix of codon frequencies for each gene.
`numcode`	The ncbi genetic code number for translation. By default the standard (numcode=1) genetic code is used.
`aa.properties`	User-supplied amino acid distance properties. By default we assume Grantham (1974) properties.
`nuc.model`	Indicates what type nucleotide model to use. There are three options: "JC", "GTR", or "UNREST".
`include.gamma`	A logical indicating whether or not to include a discrete gamma model.
`gamma.type`	Indicates what type of gamma distribution to use. Options are "quadrature" after the Laguerre quadrature approach of Felsenstein 2001 or median approach of Yang 1994.
`ncats`	The number of discrete categories.

k.levels Provides how many levels in the polynomial. By default we assume a single level (i.e., linear).

diploid A logical indicating whether or not the organism is diploid or not.

site.cats.vector
A vector designating the rate category for phi when include.gamma=TRUE.

Details

Simulates a nucleotide matrix using parameters under the SELAC model. Note that the output can be written to a fasta file using the write.dna() function in the ape package.

Examples

```
data(yeast)
system(paste("mkdir", "fastaSet", sep=" "))
sim.dat <- SelacSimulator(phy=phy, pars=model.fit$mle.pars[1,], aa.optim_array=
model.fit$aa.optim[[1]], codon.freq.by.aa=model.fit$codon.freq.by.aa[[1]],
codon.freq.by.gene=model.fit$codon.freq.by.gene[[1]], nuc.model=model.fit$nuc.model)
write.dna(sim.dat, file=paste("fastaSet", "/gene",  "TEST", ".fasta", sep=""),
format="fasta", colw=1000000)
```

SelacSimulatorEvolvingRates

Simulate DNA under the SELAC model and evolving rates

Description

Simulates nucleotide data based on parameters under the SELAC model but assumes either Phi or Ne evolves along the tree.

Usage

```
SelacSimulatorEvolvingRates(phy, pars, aa.optim_array,
  root.codon.frequencies, numcode = 1, aa.properties = NULL, nuc.model,
  k.levels = 0, diploid = TRUE, pars.to.evolve = "phi",
  evolve.type = "BM", evolve.pars = c(1, 0), Ne.vals.evolved = NULL)
```

Arguments

phy The phylogenetic tree with branch lengths.

pars A vector of parameters used for the simulation. They are ordered as follows: C.q.phi, alpha, beta, and Ne.

aa.optim_array A vector of optimal amino acids for each site to be simulated.

root.codon.frequencies
A vector of codon frequencies for each possible optimal amino acid. Thus, the vector is of length 64x64.

numcode	The The ncbi genetic code number for translation. By default the standard (numcode=1) genetic code is used.
aa.properties	User-supplied amino acid distance properties. By default we assume Grantham (1974) properties.
nuc.model	Indicates what type nucleotide model to use. There are three options: "JC", "GTR", or "UNREST".
k.levels	Provides how many levels in the polynomial. By default we assume a single level (i.e., linear).
diploid	A logical indicating whether or not the organism is diploid or not.
pars.to.evolve	Indicates which parameters to assume evolve along the tree. Only two options: "phi" or "Ne".
evolve.type	The process by which the focal parameter evovles. There are two options: Brownian motion ("BM") or Ornstein-Uhlenbeck ("OU").
evolve.pars	The process parameters used to simulate focal parameter evolution. Under "BM", the order is root.state, rate; under "OU", the order is alpha, sigma.sq, and the mean.
Ne.vals.evolved	Under selac we assume a global Ne for all genes. Thus, when the focal parameter to evolve is "Ne", then a user specified vector of simulated Ne values are provided here.

Details

Simulates a nucleotide matrix using parameters under the SELAC model, but allows either Phi or Ne to evolve along the tree. Note that the output can be written to a fasta file using the write.dna() function in the ape package.

`selon example` *Example archosaur dataset*

Description

Example tree and model output file.

`SelonOptimize` *Optimize parameters under the SELON model*

Description

Optimizes model parameters under the SELON model

Usage

```
SelonOptimize(nuc.data.path, n.partitions = NULL, phy,
  edge.length = "optimize", edge.linked = TRUE,
  optimal.nuc = "majrule", nuc.model = "GTR",
  global.nucleotide.model = TRUE, diploid = TRUE, verbose = FALSE,
  n.cores = 1, max.tol = .Machine$double.eps^0.25, max.evals = 1e+06,
  cycle.stage = 12, max.restarts = 3, output.by.restart = TRUE,
  output.restart.filename = "restartResult", fasta.rows.to.keep = NULL)
```

Arguments

`nuc.data.path`	Provides the path to the directory containing the gene specific fasta files that contains the nucleotide data.
`n.partitions`	The number of partitions to analyze. The order is based on the Unix order of the fasta files in the directory.
`phy`	The phylogenetic tree to optimize the model parameters.
`edge.length`	Indicates whether or not edge lengths should be optimized. By default it is set to "optimize", other option is "fixed", which user-supplied branch lengths.
`edge.linked`	A logical indicating whether or not edge lengths should be optimized separately for each gene. By default, a single set of each lengths is optimized for all genes.
`optimal.nuc`	Indicates what type of optimal.nuc should be used. At the moment there is only a single option: "majrule".
`nuc.model`	Indicates what type nucleotide model to use. There are three options: "JC", "GTR", or "UNREST".
`global.nucleotide.model`	assumes nucleotide model is shared among all partitions
`diploid`	A logical indicating whether or not the organism is diploid or not.
`verbose`	Logical indicating whether each iteration be printed to the screen.
`n.cores`	The number of cores to run the analyses over.
`max.tol`	Supplies the relative optimization tolerance.
`max.evals`	Supplies the max number of iterations tried during optimization.
`cycle.stage`	Specifies the number of cycles per restart. Default is 12.
`max.restarts`	Supplies the number of random restarts.
`output.by.restart`	Logical indicating whether or not each restart is saved to a file. Default is TRUE.
`output.restart.filename`	Designates the file name for each random restart.
`fasta.rows.to.keep`	Indicates which rows to remove in the input fasta files.

Details

SELON stands for SELection On Nucleotides. This function takes a user supplied topology and a set of fasta formatted sequences and optimizes the parameters in the SELON model. Selection is based on selection towards an optimal nucleotide at each site, which is based simply on the majority rule of the observed data. The strength of selection is then varied along sites based on a Taylor series, which scales the substitution rates. Still a work in development, but so far, seems very promising.

Examples

```
system(paste("mkdir", "uceSet", sep=" "))
sim.dat <- SelonSimulator(phy=phy, pars=model.fit$pars.sims[1,],
nuc.optim_array=model.fit$optimal.nuc.list[[1]], nuc.model=model.fit$nuc.model,
diploid=TRUE)
write.dna(sim.dat, file=paste("uceSet", "/gene",  "TEST", ".fasta", sep=""),
format="fasta", colw=1000000)
pp <- SelonOptimize(nuc.data.path="uceSet/", phy=phy, nuc.model="GTR", n.cores=3)
```

`SelonSimulator` *Simulate DNA under the SELON model*

Description

Simulates nucleotide data based on parameters under the SELAC model

Usage

```
SelonSimulator(phy, pars, nuc.optim_array, nuc.model, diploid = TRUE)
```

Arguments

`phy`	The phylogenetic tree with branch lengths.
`pars`	A vector of parameters used for the simulation. They are ordered as follows: a0, a1, a2, Ne, base.freqs for A C G, and the nucleotide rates.
`nuc.optim_array`	A vector of optimal nucleotide for each site to be simulated.
`nuc.model`	Indicates what type nucleotide model to use. There are three options: "JC", "GTR", or "UNREST".
`diploid`	A logical indicating whether or not the organism is diploid or not.

Details

Simulates a nucleotide matrix using parameters under the SELON model. Note that the output can be written to a fasta file using the write.dna() function in the ape package.

Examples

```
system(paste("mkdir", "uceSet", sep=" "))
sim.dat <- SelonSimulator(phy=phy, pars=model.fit$pars.sims[1,],
nuc.optim_array=model.fit$optimal.nuc.list[[1]], nuc.model=model.fit$nuc.model,
diploid=TRUE)
write.dna(sim.dat, file=paste("uceSet", "/gene",  "TEST", ".fasta", sep=""),
format="fasta", colw=1000000)
```

Index