Package: scoup (via r-universe)

November 25, 2026

Version 1.8.0

Date 2026-03-17

License GPL (>= 2)

Title Simulate Codons with Darwinian Selection Modelled as an OU Process

Description An elaborate molecular evolutionary framework that facilitates straightforward simulation of codon genetic sequences subjected to different degrees and/or patterns of Darwinian selection. The model is built upon the fitness landscape paradigm of Sewall Wright, as popularised by the mutation-selection model of Halpern and Bruno. This enables realistic evolutionary process of living organisms to be reproducible seamlessly. For example, an Ornstein-Uhlenbeck fitness update algorithm is incorporated herein. Consequently, otherwise complex biological processes, such as the effect of the interplay between genetic drift and fitness landscape fluctuations on the inference of diversifying selection, may now be investigated with minimal effort. Frequency-dependent and stochastic fitness landscape update techniques are available.

Depends R (>= 4.4), Matrix

Imports Biostrings, methods

Suggests BiocManager, BiocStyle, bookdown, htmltools, knitr, testthat (>= 3.0.0), yaml

VignetteBuilder knitr

Config/testthat/edition 3

biocViews Alignment, Classification, ComparativeGenomics, DataImport, Genetics, MathematicalBiology, ResearchField, Sequencing, SequenceMatching, Software, StatisticalMethod, WorkflowStep

Contact <hassan.t.sadiq@gmail.com>

URL https://github.com/thsadiq/scoup

BugReports https://github.com/thsadiq/scoup/issues

Config/pak/sysreqs zlib1g-dev

Repository Bioconductor 3.24

Date/Publication 2026-11-23 22:53:12 UTC

RemoteUrl https://github.com/bioc/scoup

RemoteRef RELEASE_3_24

RemoteSha afc0c5e669fbcc89657cb0d0a2451a83469730b4

Contents

aaGamma . . . 3
aaGauss . . . 4
alignsim . . . 5
aminoSC-class . . . 7
biTree . . . 8
codonCoeffs . . . 9
codonFreq . . . 10
codonvalues-class . . . 11
discrete-class . . . 12
discreteInput . . . 13
dndsCalculator . . . 15
fixMatrix . . . 16
hbInput . . . 17
hbParameters-class . . . 18
omega-class . . . 19
ou-class . . . 21
ouEvolve . . . 22
ouInput . . . 23
scoup . . . 24
scoup-class . . . 25
seqDetails . . . 26
seqParameters-class . . . 27
seqWriter . . . 29
subsMatrix . . . 30
wInput . . . 31

Index **33**

aaGamma *Obtain Gamma Distributed Amino Acid Selection Coefficients*

Description

Obtain a vector of values from the Gamma distribution that may be conveniently used as amino acid selection coefficients.

Usage

```
aaGamma(vNvS, nsynVar)
```

Arguments

vNvS	Ratio of the variance of the coefficients of the non-synonymous codons relative to the variance of the synonymous selection coefficients. Can be assigned a value equal to zero to eliminate synonymous selection.
nsynVar	A non-negative value that corresponds to the variance of the coefficients of the non-synonymous codons. That is, the the between-amino-acid variance.

Details

Twenty random observations from a $\texttt{Gamma}(1, 1/\sqrt{\texttt{nsynVar}})$ distribution are sampled as the amino acid selection coefficients. The associated variance of the synonymous coefficients (synVar) is calculated as nsynVar/vNvS. If $\texttt{nsynVar} < 10^{-12}$, all the amino acid coefficients will be equal to a single random draw from a $\texttt{Gamma}(1, 1/\texttt{synVar})$ distribution.

Value

Returns an object of class aminoSC, a vector that at least contains the following components.

coeffs A vector of 20 numeric elements that represent the sampled amino acid coefficients. The coefficients are ordered in terms of the 1-letter amino acid IUPAC labels. That is, (A, C, . . . , W, Y).

synVar Variance of the selection coefficients of the synonymous codons.

nsynVar Variance of the selection coefficients of the non-synonymous codons.

Author(s)

Hassan Sadiq

See Also

An alternative approach for generating amino acid selection coefficients from a normal distribution aaGauss. There is codonCoeffs as well, a function designed to convert amino acid to codon selection coefficients.

Examples

```
test1 <- aaGamma(0.50, 1e-04)
coeffs(test1)
synVar(test1)
nsynVar(test1)

test2 <- aaGamma(1e-02, 0)
coeffs(test2)
synVar(test2)
nsynVar(test2)
```

---

aaGauss *Obtain Gaussian Distributed Amino Acid Selection Coefficients*

Description

Obtain a vector of values from a Normal distribution that may be conveniently used as amino acid selection coefficients.

Usage

```
aaGauss(vNvS, nsynVar)
```

Arguments

vNvS	Ratio of the variance of the selection coefficients of the non-synonymous codons relative to the variance of the synonymous coefficients. It can be assigned a value equal to zero to eliminate synonymous selection.
nsynVar	A non-negative value that corresponds to the variance of the coefficients among the non-synonymous codons. That is, the between-amino-acid variance.

Details

An observation is sampled from a `Normal(0,nsynVar)` distribution independently for each of the 20 amino acid residues. The variance of the synonymous selection coefficients (`synVar`) is calculated as `nsynVar/vNvS`. If `nsynVar` is less than 10^{-12}, all the amino acid coefficients will be equal to a single random draw from a `Normal(0,synVar)` distribution.

Value

Returns an object of class `aminoSC`, a vector that at least contains the following component.

`coeffs` A vector of 20 numeric elements that represent the sampled amino acid coefficients. The coefficients are ordered in terms of the 1-letter amino acid IUPAC labels. That is, (A, C, . . . , W, Y).

`synVar` Variance of the selection coefficients of the synonymous codons.

`nsynVar` Variance of the selection coefficients of the non-synonymous codons.

Author(s)

Hassan Sadiq

See Also

An alternative sampling function, `aaGamma` is also available. The `codonCoeffs` function requires the output from this function (or from `aaGamma`).

Examples

```
case0 <- aaGauss(0.50, 1e-04)
nsynVar(case0)
synVar(case0)
coeffs(case0)

case1 <- aaGauss(1e-02, 0)
nsynVar(case1)
synVar(case1)
coeffs(case1)
```

`alignsim` *Simulate Codon Sequence Alignment*

Description

Obtain an alignment of codon sequences that have been artificially subjected to natural selection, imposed as perturbations on a static fitness landscape or seascape. Codon evolution may be executed as a semi-controlled (deterministic) frequency-dependent procedure or as a fully stochastic Ornstein-Uhlenbeck process.

Usage

```
alignsim(adaptIn, seqIn, ...)
```

Arguments

`adaptIn`	A list of class `discrete`, `omega` or `ou`.
`seqIn`	A list of class `seqParameters`.
`...`	Arguments to be passed to methods such as ‘`omega`‘ and ‘`ou`‘. See `modelIn` and `filename` in details below.

Details

This is the primary function of the package. Codon sequence alignment be simulated in terms of the population genetics paradigm. Fitness landscape may be kept static or set to be updated during every substitution event along the branches of a balanced phylogeny based on any of the three available methods: Ornstein-Uhlenbeck, frequency-dependent or episodic (referred to as 'deterministic'). Other possible inputs include, `modelIn`: a "hbParameters" object. Only applicable when `adaptIn` is an "ou" object. `filename`: a string that specifies the full path of the file that will contain the simulated alignment in NEXUS format. Say it is given as `"seq.nex"`, a file with that name will be printed in the working directory. When set as `NA` (default), no file will be saved. When set as `NULL`, a `DNAStringSet` object will be returned.

Value

A NEXUS format file is saved in the specified (or working) directory. In addition, a `scoup` object that contains the following entries is returned.

`seqs` A matrix of integers between 1 and 61. The integers are the positions of the simulated codons within an ordered set of nucleotide triplets. The rows are the extant sequences and the columns are alignment sites.

`dNdS` A matrix of the corresponding site-wise (or codon-wise) dN/dS value for all the fitness landscapes utilised during the simulation.

`aInfo` A string of text that contains details of the parameter values that were used during simulation of the codon sequence alignment.

`cseq` A dataframe of the simulated codon sequence alignment.

`seqCOL` A DNAStringSet object with colorful sequences. Only applicable when `filename=NULL`.

Author(s)

Hassan Sadiq

References

Pages H, Aboyoun P, Gentleman R, DebRoy S (2024). *Biostrings: Efficient manipulation of biological strings. R package version 2.72.1*, https://bioconductor.org/packages/Biostrings.

See Also

Complementary functions that are useful for defining the simulation parameters needed to successfully utilise this function. These include, (a.) `discreteInput`, (b.) `hbInput`, (c.) `ouInput`, (d.) `seqDetails` and (d.) `wInput`. See also `DNAStringSet` in the `Biostrings` package.

Examples

```
alignEntry <- seqDetails(c(ntaxa=8,nsite=10))

dsim <- alignsim(discreteInput(), alignEntry)
aInfo(dsim)
cseq(dsim)
```

```
wsim <- alignsim(wInput(), alignEntry, filename=NULL)
seqCOL(wsim)
dNdS(wsim)

osim <- alignsim(ouInput(), alignEntry, modelIn=hbInput())
osim
```

---

aminoSC-class *Amino Acid Selection Coefficients*

Description

A numerical vector of values that are associated with the amino acid selection coefficients.

Objects from the Class

Objects of this class (aminoSC) can be created by calls of the form new("aminoSC", coeffs=..., synVar=..., nsynVar=...). The two amino acid sampling functions (that is, aaGamma and aaGauss) that are available in the scoup package return objects of this class.

Slots

coeffs: numeric vector returned by the coeffs method.

synVar: numeric value returned by the synVar method.

nsynVar: numeric value returned by the nsynVar method.

Methods

coeffs signature(x = "aminoSC"): vector of twenty values that correspond to the amino acid selection coefficients. The entries are ordered in increasing alphabetical order in terms of the one-letter IUPAC naming structure.

nsynVar signature(x = "aminoSC"): variance of the probability distribution where the returned amino acid selection coefficients were sampled.

show signature(object = "aminoSC"): summary of the contents of the aminoSC object including a snippet of the sampled coefficients as well as the values of the synVar (σ_s^2) and the nsynVar (σ_n^2) parameters.

synVar signature(x = "aminoSC"): variance of the uniform distribution where the synonymous selection coefficients should be sampled.

Author(s)

Hassan Sadiq

See Also

Functions that return objects of the the "aminoSC" class include the two amino acid selection coefficient generators. That is, aaGamma and aaGauss.

Examples

```
aasc1 <- aaGamma(1e-10, 1e-04)
coeffs(aasc1)
show(aasc1)
```

---

biTree *Generate a Balanced Bifurcating Evolutionary Tree*

Description

Obtain an evolutionary tree that is such that all its internal nodes have exactly two offspring and all the branches on the tree have equal lengths.

Usage

```
biTree(ntaxa, bLength, terModel=NA)
```

Arguments

ntaxa	Number of extant taxa. It must be an integer (t) that may be expressed as 2^m, where m is itself a positive integer.
bLength	Branch length. All the branches of the generated tree will have the same length that is equal to the specified value.
terModel	A text to be added as suffix to the extant taxa names. If set equal to NA (default), no suffix will be added. This is useful for assigning a model to the leaves in branch-specific analyses.

Value

tree A bifurcating tree in newick format.

Author(s)

Hassan Sadiq

Examples

```
biTree(16, 0.01, "{foreground}")

biTree(16, 0.01, " #1")

biTree(16, 0.01)
```

codonCoeffs *Transform Amino Acid to Codon Selection Coefficients*

Description

Convert a 20-element vector of amino acid selection coefficients to a 61-element vector of codon selection coefficients.

Usage

```
codonCoeffs(s01x22)
```

Arguments

s01x22	A 22-element vector of class aminoSC.

Details

Consider a vector of amino acid selection coefficients, (s_x: s_A, s_C, s_D, ..., s_W, s_Y) that are subset of s01x22. All the synonymous codons that encode each amino acid are assigned independently sampled values from Uniform(s_x - 3*synVar; s_x + 3*synVar) distribution, where synVar is the synonymous variance and it is also retrieved from s01x22. For amino acids M and W, the corresponding codon coefficient is simply set equal to s_M and s_W, respectively. The output from the function is of the order (s_(AAA), s_(AAC), s_(AAG), ..., s_(TTC), s_(TTG), s_(TTT)), excluding the stop codons.

Value

Returns a codonvalues object that will contain the following.

coeffs A 61-element vector of codon selection coefficients ordered alphabetically with respect to the IUPAC nucleotide triplets nomenclature.

Author(s)

Hassan Sadiq

See Also

aaGamma and aaGauss, functions useful for generating aminoSC objects.

Examples

```
# Example 1:
aasc1 <- aaGamma(1e-10, 1e-04)
ccfs0 <- codonCoeffs(aasc1)
coeffs(ccfs0)

# Example 2:
```

```
aasc2 <- aaGauss(1e-10, 1e-04)
ccfs1 <- codonCoeffs(aasc2)
coeffs(ccfs1)

# Example 3:
aasc3 <- aaGauss(1e-03, 0)
ccfs2 <- codonCoeffs(aasc3)
coeffs(ccfs2)
```

---

codonFreq *Generate Codon Frequencies From Selection Coefficients*

Description

Obtain codon frequencies from specified selection coefficients in a way that accounts for the magnitude of the coefficients on the real number line.

Usage

```
codonFreq(sc01x61)
```

Arguments

sc01x61	Vector of selection coefficients for the sense codons. They are expected to be ordered alphabetically in terms of the IUPAC nucleotide triplets nomenclature.

Details

The conversion formula is based on a softmax identity such that the magnitude and the signs of the selection coefficients are accommodated. The frequency for the *i*th codon is estimated as:

$$\pi_i = \frac{e^{s_i - c}}{\sum_{j=1}^{61} e^{s_j - c}},$$

where $s_i \in$ sc01x61 is the selection coefficient of the *i*th codon and $c = \max_j s_j$.

Value

Returns a codonvalues object that contains the following.

freqs A vector of 61 fractional values that sum to one and represent the frequencies of sense codons. The output will be ordered alphabetically in terms of the IUPAC nucleotide triplets nomenclature.

Author(s)

Hassan Sadiq

See Also

codonCoeffs, a function that produces codon selection coefficients that may be used as an input.

Examples

```
aaEG1 <- aaGamma(1e-03, 0)
csc01 <- codonCoeffs(aaEG1)
cFq <- codonFreq(csc01)
freqs(cFq)
```

---

codonvalues-class *Codon Frequencies and Selection Coefficients*

Description

A numerical vector of values that correspond to the selection coefficients of the sense codons.

Objects from the Class

Objects of this class (codonvalues) can be created by calls of the form new("codonvalues", cdnums=...). Two codon-related transformation functions (that is, codonCoeffs and codonFreq) that are available in the scoup package return objects of this class.

Slots

cdnums: vector of 61 values that could correspond to the selection coefficients or the frequencies of the sense codons depending on the method called.

Methods

coeffs signature(x = "codonvalues"): vector of 61 values that correspond to the selection coefficients of the sense codons. The entries are ordered in increasing alphabetical order in terms of the IUPAC nucleotide triplets naming structure.

freqs signature(x = "codonvalues"): vector of 61 values that correspond to the frequencies of the sense codons. The entries are ordered in increasing alphabetical order in terms of the IUPAC nucleotide triplets naming structure.

show signature(object = "codonvalues"): prints the first six relevant (that is, coefficients or frequencies) codon related values.

Author(s)

Hassan Sadiq

See Also

The pair of functions that return objects of this class include, the generating function for codon selection coefficients codonCoeffs and the codonFreq function that converts codon coefficients to frequencies.

Examples

```
aasc1 <- aaGamma(1e-10, 1e-04)
ccfs0 <- codonCoeffs(aasc1)
cFq <- codonFreq(ccfs0)
coeffs(ccfs0)
freqs(cFq)
```

`discrete-class` *(Deterministic) Episodic Simulation Model Input*

Description

Creates an object suitable for use when interested in generating an alignment of genetic sequences following the episodic (referenced as, "deterministic") simulation technique available in the `scoup` package.

Objects from the Class

Objects can be created by calls of the form `new("discrete", lscape=..., sampler=..., nodeIndex=..., psize=..., t3mdl=...)`. Objects can also be created straightforwardly with the `discreteInput` function.

Slots

`lscape`**:** numeric matrix returned by the `lscape` method.

`sampler`**:** numeric value that can be set as 1 or 2. It indicates the probability distribution where the amino acid selection coefficients is sampled.

`nodeIndex`**:** numeric input only relevant for implicit execution of the simulation algorithm. It is of no practical utility to the end-user.

`psize`**:** numeric value returned by the `effpop` method.

`t3mdl`**:** character input that may be used to specify suffix for the leaves on the returned phylogeny. It is intended to facilitate inference analyses with external software such as `PAML` or `HyPhy`.

Methods

aaSCupdate `signature(parameters="discrete")`: background function that is not intended for end-use. It updates the amino acid selection coefficients intermittently during the sequence simulation process.

alignsim `signature(adaptIn="discrete", seqIn="seqParameters")`: primary simulation function available in the `scoup` package.

effpop `signature(x="discrete")`: effective population size.

lscape `signature(x="discrete")`: numerical matrix that contains parameters of the fitness landscape. The first row will contain the ratio of the variance of the non-synonymous to synonymous selection coefficients (`vNvS`) and the second row will contain the variance of the non-synonymous selection coefficients σ^2_n. The number of columns will be equal to the number of internal (bifurcating) stages. A phylogeny with 2^m leaves will have `m` internal stages.

sampler `signature(x="discrete")`: probability distribution where the amino acid selection coefficients should be obtained.

show `signature(object="discrete")`: prints characteristics of the corresponding genetic sequence, including the population size and the number of extant taxa.

sitesim `signature(parameters="discrete", nodeLength="numeric")`: background function that is not to be used by an end-user. It generates the DNA data at each site independently.

Author(s)

Hassan Sadiq

See Also

Objects of this class are returned by the `discreteInput` function. This class of objects are also required as inputs of the `alignsim` `function`.

Examples

```
dtest <- discreteInput()
effpop(dtest)
lscape(dtest)
sampler(dtest)
```

`discreteInput` *Populate (Deterministic) Episodic Seascape Model Parameters*

Description

Create an object that has a `discrete` class attribute. It is particularly useful for defining one of the possible inputs of the main simulation function `alignsim`, when interested in simulating codon sequences that undergo episodic evolution. That is, sequences that evolve such that fitness landscapes change at every internal node.

Usage

```
discreteInput(defList=list())
```

Arguments

`defList`	A list that may contain up to five named entries. See **Details** for further information.

Details

If fully specified, `defList` will be a list with five elements. The preferred list content include (a.) `p02xnodes`: a 2-row matrix with rows that are properly named as "`vNvS`" and "`nsynVar`". Entries in the "`vNvS`" row should be the ratio of the variance of the non-synonymous selection coefficients to the variance of the synonymous coefficients. Entries in the "`nsynVar`" row should be the variance of the non-synonymous selection coefficients. The number of the matrix columns will be used to determine the number of internal nodes to assume for the phylogeny to use for the simulation. Each column of the matrix will be used to determine the parameters of the sampling distribution where the coefficient updates will be sampled for every node. Default is a 2×4 matrix, wherein all the values in the "`vNvS`" row are equal to 1 and all the entries in the "`nsynVar`" row are equal to 10^{-5}. (b.) `technique`: a binary integer that could be `1` for `Gaussian` or `2` for `Gamma` (default) distribution. It informs about the probability distribution to use for updating the coefficients. (c.) `pSize`: (default = 1000) is the effective population size. (d.) `nodeIndex`: a nuisance input that is best left unspecified. It is updated within the `alignsim` operation. (e.) `leafModel`: a text that may be used to suffix the names of the terminal nodes (default = `NA`). Note that this function was not designed to be used in isolation. Its purpose is to complement the `alignsim` simulation function.

Value

A `discrete` object that contains the following.

`lscape` Matrix containing landscape parameters.

`sampler` Name of the sampling distribution used for selection coefficient updates.

`effpop` Effective population size.

Author(s)

Hassan Sadiq

See Also

The primary simulation function `alignsim` as well as the two in-built functions for generating amino acid selection coefficients, `aaGamma` and `aaGauss`. It may also be useful to check `biTree` to understand the tree generating process.

Examples

```
dtest <- discreteInput()
effpop(dtest)
lscape(dtest)
sampler(dtest)
```

`dndsCalculator` *Estimate dN/dS Value Analytically*

Description

Obtain an analytical estimate for the ratio of non-synonymous to synonymous rate (dN/dS) of codon substitution.

Usage

```
dndsCalculator(pi01x61, q61x61)
```

Arguments

`pi01x61`	Vector of sense codon frequencies that are ordered alphabetically with respect to nucleotide triplets according to IUPAC nomenclature.
`q61x61`	A 61×61 matrix of sense codon instantaneous substitution rates, where the rows and the columns are ordered in terms of IUPAC-lettered nucleotide triplets.

Details

The returned dN/dS estimate is obtained from the ratio of the following expressions.

$$dN = \frac{\sum_j \sum_{i \neq j} \pi_i \, \mathrm{A}_{ij} \, I_N}{\sum_j \sum_{i \neq j} \pi_i \, \mu_{ij} \, I_N}, \qquad dS = \frac{\sum_j \sum_{i \neq j} \pi_i \, \mathrm{A}_{ij} \, I_S}{\sum_j \sum_{i \neq j} \pi_i \, \mu_{ij} \, I_S},$$

where A and π are the input codon frequency vector (`pi01x61`) and the instantaneous substitution rate matrix (`q61x61`), respectively. The notation μ denotes codon mutation matrix (embedded as a function of the HKY85 nucleotide model) while I_S and I_N are Boolean matrices with ones at positions occupied by synonymous and non-synonymous codons, respectively.

Value

`dnds` An estimate for the corresponding dN/dS

Author(s)

Hassan Sadiq

References

Spielman, S. J. and Wilke, C. O. (2015). The Relationship between dN/dS and Scaled Selection Coefficients, *Molecular Biology and Evolution* **32**(4): 1097-1108.

See Also

Consider seeing the functions designed to facilitate generation of the two necessary inputs. That is, the frequency generator, `codonFreq` and the substitution matrix constructor, `subsMatrix`.

Examples

```
aasc <- aaGauss(0.5, 1e-03)
codonsc <- codonCoeffs(aasc)
piFreq <- codonFreq(codonsc)
smat <- subsMatrix(codonsc, 1000)
dndsCalculator(piFreq, smat)
```

`fixMatrix` *Construct Fixation Rate Matrix*

Description

Construct a 61 $\times$ 61 matrix that contains the rate at which a mutant (column) codon gets fixed at a position previously occupied by a resident (row) codon for all the pairwise combinations of the 61 sense codons.

Usage

```
fixMatrix(sc01x61, effpopsize)
```

Arguments

`sc01x61`	A vector of the selection coefficients for the sense codons, ordered alphabetically in terms of the IUPAC nucleotide triplets nomenclature.
`effpopsize`	Effective population size.

Details

If the additive fitness of a mutant codon (*j*) relative to a wild-type codon (*i*) is given by $s_{ij} = (s_j - s_i)$, then the rate at which codon *j* gets fixed in a codon position occupied by codon *i* can be expressed as follows.

$$f_{ij} = \begin{cases} \frac{1-e^{-2s_{ij}}}{1-e^{-2N_e s_{ij}}} & \text{if codons } i \text{ and } j \text{ differs only by one nucleotide,} \\ 0 & \text{otherwise,} \end{cases}$$

where s_i, $s_j \in$ `sc01x61` and N_e (`effpopsize`) is the effective population size. If $s_i - s_j = 0$, then $f_{ij} = 1$.

Value

`fixMtx` A 61 $\times$ 61 matrix of the fixation rates of the sense codons. The rows and columns are ordered alphabetically in terms of the IUPAC nucleotide triplets nomenclature. That is, `AAA`, `AAC`, `AAG`, ..., `TTG`, `TTT`.

Author(s)

Hassan Sadiq

References

Bershtein, S. and Serohijos, A. W. R. and Shakhnovich, E. I. (2017), Bridging the Physical Scales in Evolutionary Biology: From Protein Sequence Space to Fitness of Organisms and Populations, *Current Opinion in Structural Biology* **42**: 31-40.

Halpern, A. L. and Bruno, W. J. (1998). Evolutionary Distances for Protein-Coding Sequences: Modelling Site-Specific Residue Frequencies, *Molecular Biology and Evolution* **15**(7): 910-917.

McCandlish, D. M. (2011), Visualizing Fitness Landscapes, *Evolution* **65**(6): 1544-1558.

See Also

You may be interested in also consulting the `codonCoeffs` function designed to generate selection coefficients for the sense codons or similar functions, `aaGamma` and `aaGauss`, targeted at amino acid.

Examples

```
aasc <- aaGauss(0.5, 1e-03)
codonsc <- codonCoeffs(aasc)
fixMatrix(codonsc, 1000)
```

`hbInput` *Generate Halpern-Bruno Substitution Model Parameters*

Description

Create a `hbParameters` object that contains values necessary to construct a Halpern-Bruno mutation-selection codon substitution model.

Usage

```
hbInput(hbVector=0)
```

Arguments

`hbVector`	A named vector that provides values of the parameters necessary to successfully create a codon substitution model for simulation of genetic sequences. See **Details** for further information.

Details

When fully specified, `hbVector` will be a four-element named vector. That is, `hbVector=c(Ne, meth, vNvS, nsynVar)`. `Ne` is an integer that represents the effective population size (default = 1000). `meth` is a binary integer that indicates the probability distribution from where the initial amino acid selection coefficients must be sampled. It can be `1` for a truncated Gaussian or `2` for a Gamma (default) distribution. `vNvS` is the ratio of the variance of the non-synonymous to synonymous selection coefficients. Default value is `1`. The user can set the variance of the non-synonymous selection coefficients with `nsynVar` (default is 10^{-5}). This function was not intended for independent use. Rather as a complement to the `alignsim` simulation function.

Value

A `hbParameters` object that contains the following.

`psize` Effective population size.

`vNvS` Ratio of the variance of the non-synonymous to the synonymous selection coefficients.

`nsVar` Variance of the non-synonymous selection coefficients.

`sampler` Probability distribution where the initial amino acid selection coefficients are generated.

Author(s)

Hassan Sadiq

See Also

Selection coefficients sampling functions, `aaGamma` and `aaGauss` as well as the primary simulation function `alignsim`.

Examples

```
h0 <- hbInput()
vNvS(h0)
h0

h1 <- hbInput(c(Ne=100, meth=2, vNvS=1e-08, nsynVar=1e-08))
sampler(h1)
```

---

hbParameters-class *Halpern-Bruno Mutation-Selection Evolutionary Model Input*

Description

Creates an object of class `hbParameters` that contains the principal entries necessary to construct a Halpern-Bruno mutation-selection evolutionary model.

Objects from the Class

Objects of this class can be created by calls of the form `new("hbParameters", psize=??, vNvS=??, sampler=??, nsynVar=??, words=??)`. The object is an important input of the `alignsim` function when interested in simulating sequences with respect to the Ornstein-Uhlenbeck framework.

Slots

`psize:` numeric value returned by the `effpop` method.

`vNvS:` numeric value returned by the `vNvS` method.

`sampler:` numeric value that can be set as 1 or 2. It indicates the probability distribution where the amino acid selection coefficients is sampled.

`nsynVar:` numeric value returned by the `nsynVar` method.

`words:` comments about the specified Halpern-Bruno model parameters. It is a `character` format string that is eventually added to the simulated sequence for reference.

Methods

effpop `signature(x = "hbParameters")`: effective population size.

nsynVar `signature(x = "hbParameters")`: variance of the non-synonymous selection coefficients, σ_n^2.

sampler `signature(x = "hbParameters")`: probability distribution where the amino acid selection coefficients should be randomly retrieved.

show `signature(object = "hbParameters")`: prints characteristics of the defined model including the population size, the vNvS and the σ_n^2.

vNvS `signature(x = "hbParameters")`: ratio of the variance of the non-synonymous to synonymous selection coefficients.

Author(s)

Hassan Sadiq

See Also

It may be useful to consult the `hbInput` function because it is the generator of objects of this class. The simulator function, `alignsim` that depends on this type of objects is also a good reference.

Examples

```
h1 <- hbInput(c(Ne=100, meth=2, vNvS=1e-08, nsynVar=1e-08))
sampler(h1)
h1
```

---

omega-class *Frequency-Dependent Evolutionary Model Object*

Description

Creates an object that contains the inputs that are necessary to define a frequency-dependent evolutionary algorithm and subsequently simulate a genetic sequence alignment based on the framework using the `alignsim` function in the `scoup` package.

Objects from the Class

Objects of this class can be created by calls of the form `new("omega", nsynVar=..., psize=..., sampler=..., aaPlus=..., vNvS=...)`. The object is an important input of the `alignsim` function when interested in simulating sequences with respect to the frequency-dependent framework. The `wInput` function in the `scoup` package returns this kind of object.

Slots

`nsynVar`: numeric value returned by the `nsynVar` method.

`psize`: numeric value returned by the `effpop` method.

`sampler`: numeric value that can be set as 1 or 2. It indicates the probability distribution where the amino acid selection coefficients are sampled.

`aaPlus`: indices of the amino acids (after the corresponding one-letter IUPAC names are arranged in increasing alphabetical order) that should be assigned non-zero `vNvS`. This input enables implementation of a directional evolutionary pattern.

`vNvS`: numeric value returned by the `vNvS` method.

Methods

alignsim `signature(adaptIn="omega", seqIn="seqParameters")`: primary simulation function availed in the `scoup` package.

effpop `signature(x="omega")`: effective population size.

lscape `signature(x="omega")`: IUPAC one-letter notations of the amino acids that were assigned non-zero `vNvS` values.

nsynVar `signature(x="omega")`: variance of the non-synonymous selection coefficients, σ_n^2.

sampler `signature(x="omega")`: probability distribution where the amino acid selection coefficients are randomly retrieved.

show `signature(object="omega")`: prints the `vNvS` and the count of amino acids that had positive `vNvS` values.

sitesim `signature(parameters="omega", nodeLength="numeric")`: background function that is not available to end-user. It generates the DNA data at each site independently.

vNvS `signature(x="omega")`: ratio of the variance of the non-synonymous to synonymous selection coefficients.

Author(s)

Hassan Sadiq

See Also

The primary simulation function, `alignsim`, accepts this class of object and they can be generated with the `wInput function`.

Examples

```
w1 <- wInput(list(aaPlus=c(4,2,11), nsynVar=10))
lscape(w1)
w1
```

ou-class *Ornstein-Uhlenbeck Stochastic Simulation Model Object*

Description

Contains the inputs that are necessary to implement the Ornstein-Uhlenbeck (OU) evolutionary algorithm.

Objects from the Class

Class ou objects can be created by calls of the form `new("ou", var=??, theta=??, mu=??, words=??)`. This type of object is returned by the `ouInput` function in the `scoup` package. It is an important input of the `alignsim` function when interested in codon sequences that evolved following the OU framework.

Slots

`var`**:** numeric value returned by the `asymVar` method.

`theta`**:** numeric value returned by the `reversion` method.

`mu`**:** numeric value returned by the `asymMean` method.

`words`**:** descriptive text that contains details of the input parameter values. Useful as reference comments to be included in the generated sequence alignment.

Methods

aaSCupdate `signature(parameters = "ou")`: background function that is not intended for end-use. It updates the amino acid selection coefficients intermittently during the sequence simulation process.

alignsim `signature(adaptIn="ou", seqIn="seqParameters")`: primary simulation function available in the `scoup` package.

asymMean `signature(x="ou")`: asymptotic mean, μ, of the OU evolutionary algorithm.

asymVar `signature(x="ou")`: asymptotic variance, Σ^2, of the OU evolutionary framework.

reversion `signature(x="ou")`: reversion parameter, θ, that acts as a selective pull in the OU process.

show `signature(object="ou")`: prints the input values of Σ^2, μ and θ.

sitesim `signature(parameters="ou", nodeLength="numeric")`: background function that is not to be used by an end-user. It generates the DNA data at each site independently.

Author(s)

Hassan Sadiq

See Also

Further help may be sought by consulting the help pages of the main simulation function, `alignsim`, and/or the page of the object generating function, `ouInput`.

Examples

```
o1 <- ouInput( c(eVar=1e-02, Theta=10))
asymMean(o1)
asymVar(o1)
```

`ouEvolve` *Simulate the Trend of an Ornstein-Uhlenbeck Process*

Description

Simulate the next state of an Ornstein-Uhlenbeck (OU) process for a given value.

Usage

```
ouEvolve(xInit, deltaT, sysTheta, asymptoteVar, asymptoteMew)
```

Arguments

`xInit`	Starting point of the OU process.
`deltaT`	Jump size.
`sysTheta`	Reversion rate.
`asymptoteVar`	Asymptotic variance.
`asymptoteMew`	Asymptotic mean.

Details

The state at time `k` (that is, x_{t_k}) of a process that evolves according to an OU algorithm can be expressed as an observation from a Gaussian distribution as follows.

$$x_{t_k} \sim \mathsf{Normal}\left(\mu + \left(x_{t_{k-1}} - \mu\right) e^{-\theta \Delta t};\ \frac{\sigma^2}{2\theta}\left(1 - e^{-2\theta \Delta t}\right)\right)$$

Observe that when time interval (`deltaT`) $\Delta t = t_k - t_{k-1}$ approaches infinity, the asymptotic mean (`asymptoteMew`) and the asymptotic variance (`asymptoteVar`) of the distribution are μ and $\Sigma^2 = \sigma^2/2\theta$ respectively, where θ is the reversion rate. The function may be used to simulate toy OU trends. See **Example** below.

Value

`xnew` A scalar that represents the updated state of the OU process.

Author(s)

Hassan Sadiq

References

Uhlenbeck, G. E. and Ornstein, L. S. (1930), On the Theory of the Brownian Motion, *Physical Review* **36**: 823-841.

Examples

```
x0 <- 0.015
xvec <- c()
xvec[1] <- x0
for(k in seq(2,100)){
    xstate <- ouEvolve(x0, 0.002, 10, 0.001, 0)
    xvec[k] <- xstate
    x0 <- xstate
}
plot(xvec, type="l")
```

`ouInput` *Populate Parameters of the Ornstein-Uhlenbeck Algorithm*

Description

Create an `ou` object that will contain the parameters necessary to simulate a codon sequence alignment that evolves according to an Ornstein-Uhlenbeck (OU) process.

Usage

```
ouInput(ouVector=0)
```

Arguments

`ouVector`	A vector that contains carefully named elements. Each element represents a parameter in an OU model. See **Details** for more information.

Details

In its full form, `ouVector` is a three-element vector. Its contents each represents part of the parameters required to implement an OU codon evolutionary process with respect to the `scoup` package. The vector contents include, `eMean`, `eVar` and `Theta`. Input `eMean` is the asymptotic mean (μ) and zero is its default value. `eVar` denotes the asymptotic variance (Σ^2). It has a 0.01 default value. `Theta` (default = 0.01) represents the reversion rate (θ). This function was aimed as a complement to `alignsim`.

Value

An `ou` object that contains the following.

`asymMean` Asymptotic mean of the OU process.

`asymVar` Asymptotic variance of the OU process.

`reversion` Reversion rate of the OU process.

Author(s)

Hassan Sadiq

References

Uhlenbeck, G. E. and Ornstein, L. S. (1930), On the Theory of the Brownian Motion, *Physical Review* **36**: 823-841.

See Also

The Ornstein-Uhlenbeck state generating function `ouEvolve` and the `alignsim` simulation function.

Examples

```
o0 <- ouInput()
reversion(o0)
o0

o1 <- ouInput( c(eVar=1e-02, Theta=10))
asymMean(o1)
asymVar(o1)
```

`scoup` *Simulate Codons with Darwinian Selection Imposed as an OU Process*

Description

The primary objective of this package is to facilitate more rigorous understanding of phylogenetic inferences of natural selection from codon sequences. Concepts from the Halpern-Bruno mutation-selection model and the Ornstein-Uhlenbeck stochastic process were creatively fused such that the end-product is a novelty with respect to computational genetic simulation. Users are able to seamlessly adjust the model parameters to mimic complex evolutionary procedures that may have been otherwise infeasible. For example, it is possible to explicitly interrogate the concepts of static and changing fitness landscapes with regards to Darwinian natural selection in the context of DNA sequences. The ratio of the variance of selection coefficients, vN/vS, is used to control the magnitude of Darwinian selection pressure. No such application previously existed. This package could be very useful for generating more appropriate test data sets for validation of likelihood-based (ω) codon models of evolution.

Details

Three simulation algorithms are available. (a.) The Ornstein-Uhlenbeck simulation technique. This technique was built around the stochastic Brownian motion evolutionary paradigm. Explicit parameters exist to control the extent of drift, mutation and selection that are acting on the biological system. (b.) The frequency-dependent approach where every substitution event causes a shift in the fitness landscape. (c.) The "deterministic" method that corresponds to an episodic evolutionary process, where the model parameters may be fixed for each internal node of the phylogeny.

Author(s)

Hassan Sadiq

References

Halpern, A. L. and Bruno, W. J. (1998). Evolutionary Distances for Protein-Coding Sequences: Modelling Site-Specific Residue Frequencies, *Molecular Biology and Evolution* **15**(7): 910-917.

Uhlenbeck, G. E. and Ornstein, L. S. (1930), On the Theory of the Brownian Motion, *Physical Review* **36**: 823-841.

`scoup-class` *Output from the* `scoup::alignsim` *Genetic Sequence Simulator*

Description

Stores the results from a successful implementation of any of the simulation algorithms available in the `scoup` package.

Objects from the Class

Objects can be created by calls of the form `new("scoup", seqs=..., DNDS=..., aInfo=..., cseq=..., seqCOL=...)`.

Slots

`seqs`**:** numerical matrix returned by the `seqs` method.

`DNDS`**:** numerical matrix returned by the `dNdS` method.

`aInfo`**:** character phrase returned by the `aInfo` method.

`cseq`**:** data frame returned by the `cseq` method.

`seqCOL`**:** `DNAStringSet` object returned by the `seqCOL` method.

Methods

aInfo `signature(x="scoup")`: details of the parameters used to execute the simulation process. This includes, the branch length of all the nodes of the balanced phylogeny, the name of the probability distribution where the amino acid selection coefficients is sampled as well as the (`vNvS` & non-synonymous) selection parameter set used at each internal node ("generation") stage.

cseq `signature(x="scoup")`: data frame that contains the simulated genetic sequence.

dNdS `signature(x="scoup")`: analytical estimates of the magnitude of the imposed selection effect. It is calculated node-wise or after every substitution event (depending on the simulation technique in use) as the ratio of the non-synonymous to synonymous substitutions.

seqCOL `signature(x="scoup")`: a `DNAStringSet` version of the simulated genetic sequence alignment.

seqs `signature(x="scoup")`: expression of the simulated sequence as a matrix of integers, where each entry corresponds to the position of the associated codon in an an alphabetically increasing ordered set of the DNA triplets of the 61 sense codons. This is intended to facilitate utility of R's concatenation function to generate site-partitioned sequence alignment.

show `signature(object="scoup")`: sentence that contains the number of codon sites and the number of extant taxa that make up the simulated genetic sequence alignment.

Author(s)

Hassan Sadiq

See Also

Simulation function `alignsim`. The `seqWriter` function is a good complement to facilitate the generation of site-partitioned sequences.

Examples

```
alignEntry <- seqDetails(c(ntaxa=8,nsite=10))
dsim <- alignsim(discreteInput(), alignEntry)
aInfo(dsim)
cseq(dsim)
```

`seqDetails` *Populate Sequence Alignment Information*

Description

Create a `seqParameters` object that contains features of the sequence that needs to be simulated.

Usage

```
seqDetails(seqVector=0)
```

Arguments

`seqVector`	A named vector that provides characteristics of the required sequence alignment. See **Details** for further information.

Details

If fully specified, `seqVector` should be a four-element named vector. That is, `seqVector = c(ntaxa, nsite, blength, terModel)`. `ntaxa` should be of the form 2^m, where `m` is an integer. It corresponds to the number of extant taxa, default is 64. `nsite`, also an integer (default = 250), is the number of codon sites. `blength` is the length of each branch on the balanced symmetric tree that will be used for the simulation (default = 0.10). `terModel` is a text that will be added as a suffix to the leaf names on the phylogeny (default = `NA`). It is meant to facilitate assignment of models to the terminal nodes for branch-wise selection analyses. The purpose of this function is to complement `alignsim`.

Value

A `seqParameters` object that contains the following.

`sites` Number of alignment sites.

`taxa` Number of extant taxa.

`nodes` Number of internal (bifurcating) stages on the evolutionary tree. A tree with 2^m leaves will have `m` internal stages.

`branchL` Length of the branches on the phylogeny.

`phylogeny` Evolutionary tree in newick format.

`details` Text that describes the evolutionary tree.

Author(s)

Hassan Sadiq

See Also

The codon sequence simulator `alignsim` and `biTree`, the balanced evolutionary tree generator.

Examples

```
t0 <- seqDetails()
sites(t0)

t1 <- seqDetails(c(ntaxa=16, nsite=10, blength=0.20, terModel=" #1"))
details(t1)
```

`seqParameters-class` *Codon Sequence Structure*

Description

A S4 object that contains information about the structure (that is, size, length, etc) of the simulated genetic sequence.

Objects from the Class

This is the object class of the output from the `seqDetails` function. It is a core input of the `alignsim` function. Objects can be created by calls of the form `new("seqParameters", sites=??, taxa=??, nodes=??, branchL=??, phylogeny=??, details=??)`.

Slots

`sites`: numeric value returned by the `sites` method.

`taxa`: numeric value returned by the `taxa` method.

`nodes`: numeric value returned by the `nodes` method.

`branchL`: numeric value returned by the `branchL` method.

`phylogeny`: character returned by the `phylogeny` method.

`details`: character returned by the `details` method.

Methods

alignsim `signature(adaptIn="discrete", seqIn="seqParameters")`: an option of the primary simulation function in the `scoup` package. This setting activates the episodic (branch-wise) framework.

alignsim `signature(adaptIn="omega", seqIn="seqParameters")`: an option of the main function in the `scoup` package. This setting calls the frequency-dependent framework.

alignsim `signature(adaptIn="ou", seqIn="seqParameters")`: an option of the simulation function in the `scoup` package. This setting executes the Ornstein-Uhlenbeck algorithm.

branchL `signature(xo="seqParameters")`: branch length. Only balanced evolutionary trees are permitted. Therefore, all tree nodes have the same length.

details `signature(xo="seqParameters")`: a text string that contains the parameter inputs for initiating the simulation process. It is included as comments to saved sequence output.

nodes `signature(xo="seqParameters")`: number of internal (bifurcating) stages of the balanced phylogeny. An evolutionary tree with 2^m extant taxa will have `m` nodes.

phylogeny `signature(xo="seqParameters")`: a newick string of the phylogeny for the codon sequence simulation.

show `signature(object="seqParameters")`: a summary descriptive details with respect to the sequence alignment.

sites `signature(xo="seqParameters")`: the number of codon sites that make up the sequence.

taxa `signature(xo="seqParameters")`: the number of leaves on the phylogeny.

Author(s)

Hassan Sadiq

See Also

Codon sequence simulator `alignsim` and the sequence preparatory function `seqDetails`.

Examples

```
t0 <- seqDetails()
sites(t0)
```

seqWriter *Write Numeric Codon Alignment to a NEXUS File*

Description

Save numeric codon alignment matrix to a file in NEXUS format. It is particularly useful when data with site partitions is required.

Usage

```
seqWriter(alignmentMatrix, treeInfo=NA, addText="", fileTag=NULL)
```

Arguments

`alignmentMatrix`	A numerical matrix of codon sequence alignment that is similar to the `seqs` matrix from the output of `alignsim`. The rows of the matrix should each correspond to an extant taxa and the columns should be the alignments sites. The entries of the matrix should be integers between 1 and 61 and they will be decoded in terms of the ordered IUPAC sense codon triplets. That is, 1=`AAA`, 2=`AAC`, 3=`AAG`, 4=`AAT`, 5=`ACA`, ..., 57=`TGT`, 58=`TTA`, 59=`TTC`, 60=`TTG`, `TTT`.
`treeInfo`	Phylogeny to be printed with the sequence. If unspecified (default = `NA`) a balanced phylogeny with branch length = 0.10 and number of extant taxa set as the number of rows of the input `alignmentMatrix` will be used.
`addText`	A string of comments to be printed with the alignment (default = "").
`fileTag`	Full path to where the output file should be printed. It should be a string (default = `NULL`). If not provided, the NEXUS file returned will be saved as `cranrSeqs.nex` in a temporary directory.

Value

`NULL` A NEXUS file with codon alignment printed therein will be saved in a temporary (or specified) directory.

Author(s)

Hassan Sadiq

See Also

Simulation function `alignsim`.

Examples

```
sqAlign <- alignsim(ouInput(), seqDetails(), hbInput(), NA)
seqWriter(seqs(sqAlign))
```

subsMatrix *Build Mutation-Selection Codon Substitution Matrix*

Description

Construct an instantaneous codon substitution matrix based on the mutation-selection framework.

Usage

```
subsMatrix(sc01x61, effpopsize)
```

Arguments

sc01x61	Vector of selection coefficients associated with the 61 sense codons, ordered alphabetically according to the nucleotide triplets and the IUPAC naming structure.
effpopsize	Effective population size.

Details

Given an arbitrary scaling constant (k), codon fixation rates (f_{ij}) and mutation rates (m_{ij}), the instantaneous rate by which a wild-type codon i is substituted by a mutant codon j may be expressed as follows.

$$q_{ij} = \begin{cases} k \times \mathrm{m}_{ij} \times \mathrm{f}_{ij} & \text{if } i \text{ and } j \text{ differs by only one nucleotide,} \\ 0 & \text{if } i \text{ and } j \text{ differs by more than one nucleotide,} \end{cases}$$

and $q_{ii} = -\sum_j q_{ij}$. The HKY85 nucleotide mutation matrix was embedded into the alignsim function (with transition to transversion rate, κ = 4, average rate, μ = 0.25 and equal equilibrium frequencies).

Value

mainMatrix Instantaneous codon substitution matrix such that the rows and the columns are arranged with respect to the IUPAC naming structure of nucleotide triplets in alphabetical order.

Author(s)

Hassan Sadiq

References

Halpern, A. L. and Bruno, W. J. (1998). Evolutionary Distances for Protein-Coding Sequences: Modelling Site-Specific Residue Frequencies, *Molecular Biology and Evolution* **15**(7): 910-917.

Hasegawa, M., Kishino, H. and Yano, T. (1985). Dating of the Human-Ape Splitting by a Molecular Clock of Mitochondria DNA, *Journal of Molecular Evolution* **22**: 160-174.

See Also

Selection coefficients conversion function codonCoeffs and fixation matrix generating function fixMatrix.

Examples

```
aacoeffs <- aaGauss(1e-03, 0)
codonsc <- codonCoeffs(aacoeffs)
subsMatrix(codonsc, 1000)
```

---

wInput *Populate Frequency-Dependent Simulation Model Parameters*

Description

Create an omega object. The utility is for defining the parameters that are necessary to simulate codon sequences that mimic the evolutionary process described by the frequency-dependent models.

Usage

```
wInput(wList=list())
```

Arguments

wList	A list that may contain up to five named entries. See **Details** for further information.

Details

In its full form, wList will contain five named elements. The elements include (a.) pSize: an integer that represents the effective population size (default = 1000). (b.) vNvS: a numerical value that corresponds to the ratio of the variance of the non-synonymous to the synonymous selection coefficients (default = 1). (c.) aaPlus: its default is a vector of integers between 1 and 20, inclusive. It gives the indices, if the one-letter IUPAC amino acid notations were ordered alphabetically, of the residues that should be assigned non-zero coefficient variances. It is useful for executing directional evolution. (d.) technique: it informs of the preferred probability distribution where the selection coefficients should be sampled. It could be set as 1 for Gaussian or 2 for Gamma (default) distribution. (e.) nsynVar: variance of the non-synonymous selection coefficients. This is a complementary function to alignsim.

Value

An omega object that contains the following.

nsynVar Variance of the non-synonymous selection coefficients.

technique Probability density function for sampling the amino acid selection coefficients.

aaPlus Indices of the amino acids to be assigned non-zero coefficient variance values.

vNvS Ratio of the variance of the non-synonymous to the synonymous selection coefficients.

psize Effective population size.

Author(s)

Hassan Sadiq

References

Goldman, N. and Yang, Z. (1994), A Codon-Based Model of Nucleotide Substitution for Protein-Coding DNA Sequences, *Molecular Biology and Evolution* **11**(5): 725-736.

Muse, S. V. and Gaut, B. S. (1994), A A Likelihood Approach for Comparing Synonymous and Nonsynonymous Nucleotide Substitution Rates, with Application to the Chloroplast Genome, *Molecular Biology and Evolution* **11**(5): 715-724.

See Also

Sequence simulation function `alignsim` as well as selection coefficient conversion functions `aaGamma` and `aaGauss`.

Examples

```
w0 <- wInput()
vNvS(w0)
w0

w1 <- wInput(list(aaPlus=c(4,2,11), nsynVar=10))
lscape(w1)
w1
```

Index

∗ **classes**
aminoSC-class, 7
codonvalues-class, 11
discrete-class, 12
hbParameters-class, 18
omega-class, 19
ou-class, 21
scoup-class, 25
seqParameters-class, 27

aaGamma, 3, *5*, *7*, *9*, *14*, *17, 18*, *32*
aaGauss, *3*, 4, *7*, *9*, *14*, *17, 18*, *32*
aaSCupdate *(*alignsim*)*, 5
aaSCupdate,discrete-method *(*discrete-class*)*, 12
aaSCupdate,ou-method *(*ou-class*)*, 21
aInfo *(*scoup-class*)*, 25
aInfo,scoup-method *(*scoup-class*)*, 25
alignsim, 5, *13, 14*, *18–21*, *24*, *26–29*, *32*
alignsim,discrete,seqParameters-method *(*discrete-class*)*, 12
alignsim,omega,seqParameters-method *(*omega-class*)*, 19
alignsim,ou,seqParameters-method *(*ou-class*)*, 21
aminoSC-class, 7
asymMean *(*ou-class*)*, 21
asymMean,ou-method *(*ou-class*)*, 21
asymVar *(*ou-class*)*, 21
asymVar,ou-method *(*ou-class*)*, 21

biTree, 8, *14*, *27*
branchL *(*seqParameters-class*)*, 27
branchL,seqParameters-method *(*seqParameters-class*)*, 27

codonCoeffs, *3*, *5*, 9, *11*, *17*, *31*
codonFreq, 10, *11*, *15*
codonvalues-class, 11
coeffs *(*aminoSC-class*)*, 7
coeffs,aminoSC-method *(*aminoSC-class*)*, 7
coeffs,codonvalues-method *(*codonvalues-class*)*, 11
cseq *(*scoup-class*)*, 25
cseq,scoup-method *(*scoup-class*)*, 25

details *(*seqParameters-class*)*, 27
details,seqParameters-method *(*seqParameters-class*)*, 27
discrete-class, 12
discreteInput, *6*, *12*, 13, *13*
dNdS *(*scoup-class*)*, 25
dNdS,scoup-method *(*scoup-class*)*, 25
dndsCalculator, 15

effpop *(*alignsim*)*, 5
effpop,discrete-method *(*discrete-class*)*, 12
effpop,hbParameters-method *(*hbParameters-class*)*, 18
effpop,omega-method *(*omega-class*)*, 19

fixMatrix, 16, *31*
freqs *(*codonvalues-class*)*, 11
freqs,codonvalues-method *(*codonvalues-class*)*, 11

hbInput, *6*, 17, *19*
hbParameters-class, 18

lscape *(*alignsim*)*, 5
lscape,discrete-method *(*discrete-class*)*, 12
lscape,omega-method *(*omega-class*)*, 19

nodes *(*seqParameters-class*)*, 27
nodes,seqParameters-method *(*seqParameters-class*)*, 27
nsynVar *(*alignsim*)*, 5
nsynVar,aminoSC-method *(*aminoSC-class*)*, 7

nsynVar,hbParameters-method
(hbParameters-class), 18
nsynVar,omega-method (omega-class), 19

omega-class, 19
ou-class, 21
ouEvolve, 22, *24*
ouInput, *6*, *21*, 23

phylogeny (seqParameters-class), 27
phylogeny,seqParameters-method
(seqParameters-class), 27

reversion (ou-class), 21
reversion,ou-method (ou-class), 21

sampler (alignsim), 5
sampler,discrete-method
(discrete-class), 12
sampler,hbParameters-method
(hbParameters-class), 18
sampler,omega-method (omega-class), 19
scoup, *7*, *11*, *12*, *19–21*, *23*, 24, *25*, *28*
scoup-class, 25
scoup-package (scoup), 24
seqCOL (scoup-class), 25
seqCOL,scoup-method (scoup-class), 25
seqDetails, *6*, 26, *27*, *28*
seqParameters-class, 27
seqs (scoup-class), 25
seqs,scoup-method (scoup-class), 25
seqWriter, *26*, 29
show,aminoSC-method (aminoSC-class), 7
show,codonvalues-method
(codonvalues-class), 11
show,discrete-method (discrete-class),
12
show,hbParameters-method
(hbParameters-class), 18
show,omega-method (omega-class), 19
show,ou-method (ou-class), 21
show,scoup-method (scoup-class), 25
show,seqParameters-method
(seqParameters-class), 27
sites (seqParameters-class), 27
sites,seqParameters-method
(seqParameters-class), 27
sitesim (alignsim), 5
sitesim,discrete,numeric-method
(discrete-class), 12
sitesim,omega,numeric-method
(omega-class), 19
sitesim,ou,numeric-method (ou-class), 21
subsMatrix, *15*, 30
synVar (aminoSC-class), 7
synVar,aminoSC-method (aminoSC-class), 7

taxa (seqParameters-class), 27
taxa,seqParameters-method
(seqParameters-class), 27
tree (biTree), 8

vNvS (alignsim), 5
vNvS,hbParameters-method
(hbParameters-class), 18
vNvS,omega-method (omega-class), 19

wInput, *6*, *19*, *20*, 31