Package ‘AnaCoDa’

September 15, 2020

Type Package

Title Analysis of Codon Data under Stationarity using a Bayesian Framework

Version 0.1.4.4

Date 2020-09-11

Author Authors@R

Maintainer Cedric Landerer <cedric.landerer@gmail.com>

URL https://github.com/clandere/AnaCoDa

VignetteBuilder knitr

NeedsCompilation yes

Depends R (>= 3.3.0), Rcpp (>= 0.11.3), VGAM, methods, mvtnorm

Suggests knitr, Hmisc, coda, testthat, lmodel2, markdown

RcppModules Test_mod, Trace_mod, CovarianceMatrix_mod,
          MCMCAlgorithm_mod, Model_mod, Parameter_mod, Genome_mod,
          Gene_mod, SequenceSummary_mod

Description Is a collection of models to analyze genome scale codon data using a Bayesian framework. Provides visualization routines and checkpointing for model fittings. Currently published models to analyze gene data for selection on codon usage based on Ribosome Overhead Cost (ROC) are: ROC (Gilchrist et al. (2015) <doi:10.1093/gbe/evv087>), and ROC with phi (Wallace & Drummond (2013) <doi:10.1093/molbev/mst051>). In addition 'AnaCoDa' contains three currently unpublished models. The FONSE (First order approximation On NonSense Error) model analyzes gene data for selection on codon usage against of nonsense error rates. The PA (PAusing time) and PANSE (PAusing time + NonSense Error) models use ribosome footprinting data to analyze estimate ribosome pausing times with and without nonsense error rate from ribosome footprinting data.

License GPL (>= 2)

Imports
LinkingTo: Rcpp
LazyLoad: yes
LazyData: yes
RoxygenNote: 7.1.1
Repository: CRAN
Date/Publication: 2020-09-15 09:40:19 UTC

R topics documented:

AAToCodon.................................................4
acfCSP....................................................4
acfMCMC..................................................5
addObservedSynthesisRateSet.......................6
aminoAcids..............................................7
calculateMarginalLogLikelihood.....................7
calculateSCUO..........................................8
codons.....................................................9
codonToAA...............................................9
convergence.test.......................................10
findOptimalCodon.......................................12
fixDEta..................................................13
fixDM....................................................13
fixSphi..................................................13
geomMean.............................................13
getAdaptiveWidth.................................14
getCAI..................................................15
getCAIweights.........................................15
getCodonCounts.......................................16
getCodonCountsForAA...............................17
getCodonSpecificPosteriorMeanForCodon..........18
getCodonSpecificPosteriorVarianceForCodon......18
getCodonSpecificQuantilesForCodon...............19
getCSPEstimates......................................20
getEstimatedMixtureAssignmentForGene............21
getEstimatedMixtureAssignmentProbabilitiesForGene22
getAddressEstimates...............................22
getGroupList..........................................24
getLogLikelihoodTrace..............................24
getLogPosteriorMean.................................24
getLogPosteriorTrace.................................25
getMixtureAssignmentEstimate......................25
getNames..............................................26
getNc...................................................27
getNcAA...............................................28
getNoiseOffsetPosteriorMean.......................28
getNoiseOffsetVariance..............................29
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>getObservedSynthesisRateSet</td>
<td>29</td>
</tr>
<tr>
<td>getSamples</td>
<td>30</td>
</tr>
<tr>
<td>getSelectionCoefficients</td>
<td>30</td>
</tr>
<tr>
<td>getStdDevSynthesisRatePosteriorMean</td>
<td>31</td>
</tr>
<tr>
<td>getStdDevSynthesisRateVariance</td>
<td>32</td>
</tr>
<tr>
<td>getStepsToAdapt</td>
<td>32</td>
</tr>
<tr>
<td>getSynthesisRate</td>
<td>33</td>
</tr>
<tr>
<td>getSynthesisRatePosteriorMeanForGene</td>
<td>33</td>
</tr>
<tr>
<td>getSynthesisRatePosteriorVarianceForGene</td>
<td>34</td>
</tr>
<tr>
<td>getThinning</td>
<td>34</td>
</tr>
<tr>
<td>getTrace</td>
<td>35</td>
</tr>
<tr>
<td>getTraceObject</td>
<td>35</td>
</tr>
<tr>
<td>initializeCovarianceMatrices</td>
<td>36</td>
</tr>
<tr>
<td>initializeGenomeObject</td>
<td>37</td>
</tr>
<tr>
<td>initializeMCMCOBJECT</td>
<td>38</td>
</tr>
<tr>
<td>initializeModelObject</td>
<td>39</td>
</tr>
<tr>
<td>initializeParameterObject</td>
<td>41</td>
</tr>
<tr>
<td>initializeSynthesisRateByGenome</td>
<td>44</td>
</tr>
<tr>
<td>initializeSynthesisRateByList</td>
<td>44</td>
</tr>
<tr>
<td>initializeSynthesisRateByRandom</td>
<td>45</td>
</tr>
<tr>
<td>initMutationCategories</td>
<td>45</td>
</tr>
<tr>
<td>initSelectionCategories</td>
<td>46</td>
</tr>
<tr>
<td>length.Rcpp_Genome</td>
<td>46</td>
</tr>
<tr>
<td>loadMCMCOBJECT</td>
<td>47</td>
</tr>
<tr>
<td>loadParameterObject</td>
<td>47</td>
</tr>
<tr>
<td>plot.Rcpp_FONSEModel</td>
<td>48</td>
</tr>
<tr>
<td>plot.Rcpp_FONSEParameter</td>
<td>49</td>
</tr>
<tr>
<td>plot.Rcpp_MCMCAlgorythm</td>
<td>50</td>
</tr>
<tr>
<td>plot.Rcpp_ROCModel</td>
<td>50</td>
</tr>
<tr>
<td>plot.Rcpp_ROCParameter</td>
<td>51</td>
</tr>
<tr>
<td>plot.Rcpp_Trace</td>
<td>52</td>
</tr>
<tr>
<td>plotAcceptanceRatios</td>
<td>53</td>
</tr>
<tr>
<td>plotCodonSpecificParameters</td>
<td>53</td>
</tr>
<tr>
<td>readPhiValue</td>
<td>54</td>
</tr>
<tr>
<td>runMCMC</td>
<td>54</td>
</tr>
<tr>
<td>setAdaptiveWidth</td>
<td>56</td>
</tr>
<tr>
<td>setGroupList</td>
<td>56</td>
</tr>
<tr>
<td>setRestartFileSettings</td>
<td>56</td>
</tr>
<tr>
<td>setRestartSettings</td>
<td>57</td>
</tr>
<tr>
<td>setSamples</td>
<td>58</td>
</tr>
<tr>
<td>setStepsToAdapt</td>
<td>58</td>
</tr>
<tr>
<td>setThinning</td>
<td>58</td>
</tr>
<tr>
<td>simulateGenome</td>
<td>59</td>
</tr>
<tr>
<td>summary.Rcpp_Genome</td>
<td>59</td>
</tr>
<tr>
<td>writeMCMCOBJECT</td>
<td>60</td>
</tr>
<tr>
<td>writeParameterObject</td>
<td>61</td>
</tr>
</tbody>
</table>

Index 62
AAToCodon  
*Amino Acid to codon set*

**Description**

Converts one character amino acid code to the set of codon encoding that amino acid

**Usage**

`AAToCodon(aa, focal = FALSE)`

**Arguments**

- `aa`  
  Amino acid in single character notation
- `focal`  
  logical, Include the alphabetically last (focal) codon

**Value**

Returns the names of the codon encoding the give amino acid

**See Also**

`codonToAA`

---

acfCSP  
*Plots ACF for codon specific parameter traces*

**Description**

The function calculates and by defaults plots the acf and estimates the autocorrelation in the trace

**Usage**

`acfCSP(
    parameter,
    csp = "Mutation",
    numMixtures = 1,
    samples = NULL,
    lag.max = 40,
    plot = TRUE
)`
acfMCMC

Arguments

<table>
<thead>
<tr>
<th>parameter</th>
<th>object of class Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>csp</td>
<td>indicates which parameter to calculate the autocorrelation. Must be Mutation (the default, ROC, FONSE), Selection (ROC, FONSE), Alpha (PA, PANSE), LambdaPrime (PA, PANSE), NSERate (PA, PANSE)</td>
</tr>
<tr>
<td>numMixtures</td>
<td>indicates the number of CSP mixtures used</td>
</tr>
<tr>
<td>samples</td>
<td>number of samples at the end of the trace used to calculate the acf</td>
</tr>
<tr>
<td>lag.max</td>
<td>Maximum amount of lag to calculate acf. Default is 10*log10(N), where N i the number of observations.</td>
</tr>
<tr>
<td>plot</td>
<td>logical. If TRUE (default) a plot of the acf is created</td>
</tr>
</tbody>
</table>

See Also

acfMCMC

Description

The function calculates and by defaults plots the acf and estimates the autocorrelation in the trace.

Usage

acfMCMC(mcmc, type = "LogPosterior", samples = NULL, lag.max = 40, plot = TRUE)

Arguments

<table>
<thead>
<tr>
<th>mcmc</th>
<th>object of class MCMC</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>&quot;LogPosterior&quot; or &quot;LogLikelihood&quot;, defaults to &quot;LogPosterior&quot;</td>
</tr>
<tr>
<td>samples</td>
<td>number of samples at the end of the trace used to calculate the acf</td>
</tr>
<tr>
<td>lag.max</td>
<td>Maximum amount of lag to calculate acf. Default is 10*log10(N), where N i the number of observations.</td>
</tr>
<tr>
<td>plot</td>
<td>logical. If TRUE (default) a plot of the acf is created</td>
</tr>
</tbody>
</table>

See Also

acfCSP
addObservedSynthesisRateSet

Add gene observed synthesis rates

Description

addObservedSynthesisRateSet returns the observed synthesis rates of the genes within the genome specified.

Usage

addObservedSynthesisRateSet(
  genome,
  observed.expression.file,
  match.expression.by.id = TRUE
)

Arguments

genome A genome object initialized with initializeGenomeObject to add observed expression data.

observed.expression.file A string containing the location of a file containing empirical expression rates (optional).

match.expression.by.id If TRUE (default) observed expression values will be assigned by matching sequence identifier. If FALSE observed expression values will be assigned by order

Value

Returns the genome after adding the new gene expression values

Examples

genome_file <- system.file("extdata", "genome.fasta", package = "AnaCoDa")
expression_file <- system.file("extdata", "expression.csv", package = "AnaCoDa")
## reading genome
genome <- initializeGenomeObject(file = genome_file)

## add expression values after the genome was initialized, ## or adding an additional set of expression values
genome <- addObservedSynthesisRateSet(genome = genome,
                                       observed.expression.file = expression_file)
description

returns a vector of all amino acids

Usage

aminoAcids()

value

returns a vector of all amino acids

See Also

codons

calculateMarginalLogLikelihood

Calculates the marginal log-likelihood for a set of parameters

Description

initializes the model object.

Usage

calculateMarginalLogLikelihood(
    parameter,
    mcmc,
    mixture,
    n.samples,
    divisor,
    warnings = TRUE
)

Arguments

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parameter</td>
<td>An object created with initializeParameterObject.</td>
</tr>
<tr>
<td>mcmc</td>
<td>An object created with initializeMCMCObject</td>
</tr>
<tr>
<td>mixture</td>
<td>determines for which mixture the marginal log-likelihood should be calculated</td>
</tr>
<tr>
<td>n.samples</td>
<td>How many samples should be used for the calculation</td>
</tr>
<tr>
<td>divisor</td>
<td>A value &gt; 1 in order to scale down the tails of the importance distribution</td>
</tr>
<tr>
<td>warnings</td>
<td>Print warnings such as when the variance of a parameter is 0, which might occur when parameter is fixed</td>
</tr>
</tbody>
</table>
Details

calculateMarginalLogLikelihood Calculate marginal log-likelihood for calculation of the Bayes factor using a generalized harmonix mean estimator of the marginal likelihood. See Gronau et al. (2017) for details

Value

This function returns the model object created.

Examples

```r
## Not run:
# Calculate the log-marginal likelihood
parameter <- loadParameterObject("parameter.Rda")
mcmc <- loadMCMCObject("mcmc.Rda")
calculate_marginal_likelihood(parameter, mcmc, mixture = 1, samples = 500, scaling = 1.5)

# Calculate the Bayes factor for two models
parameter1 <- loadParameterObject("parameter1.Rda")
parameter2 <- loadParameterObject("parameter2.Rda")
mcmc1 <- loadMCMCObject("mcmc1.Rda")
mcmc2 <- loadMCMCObject("mcmc2.Rda")
mll1 <- calculate_marginal_likelihood(parameter1, mcmc1, mixture = 1, samples = 500, scaling = 1.5)
mll2 <- calculate_marginal_likelihood(parameter2, mcmc2, mixture = 1, samples = 500, scaling = 1.5)
cat("Bayes factor: ", mll1 - mll2, "\n")

## End(Not run)
```

calculateSCUO calculates the synonymous codon usage order (SCUO)

Description

calculateSCUO calculates the SCUO value for each gene in genome. Note that if a codon is absent, this will be treated as NA and will be skipped in final calculation

Usage

calculateSCUO(genome)

Arguments

genome A genome object initialized with initializeGenomeObject.
Value
returns the SCUO value for each gene in genome

Examples

```r
gene_file <- system.file("extdata", "genome.fasta", package = "AnaCoDa")

## reading genome
genome <- initializeGenomeObject(file = gene_file)
scuo <- calculateSCUO(genome)
```

codons

<table>
<thead>
<tr>
<th>codons</th>
<th>Codons</th>
</tr>
</thead>
</table>

Description
Returns a vector of all codons

Usage
codons()

Value
Returns a vector of all codons

See Also
aminoAcids
codonToAA

<table>
<thead>
<tr>
<th>codonToAA</th>
<th>translates codon to amino acid</th>
</tr>
</thead>
</table>

Description
Translates a given codon into the amino acid encoded by it.

Usage
codonToAA(codon)

Arguments
codon character, codon to translate
Value

Returns the amino acid encoded by the given codon as character

See Also

AAToCodon

convergence.test

Convergence Test

Description

Convergence Test

Usage

convergence.test(
  object,
  samples = 10,
  frac1 = 0.1,
  frac2 = 0.5,
  thin = 1,
  plot = FALSE,
  what = "Mutation",
  mixture = 1
)

Arguments

object an object of either class Trace or MCMC
samples number of samples at the end of the trace used to determine convergence (< length of trace). Will use as starting point of convergence test. If the MCMC trace is of length x, then starting point for convergence test will be x - samples.
frac1 fraction to use from beginning of samples
frac2 fraction to use from end of samples
thin the thinning interval between consecutive observations, which is used in creating a coda::mcmc object (according to the Coda documentation, users should specify if a MCMC chain has already been thinned using a the thin parameter). This does not further thin the data.
plot (logical) plot result instead of returning an object
what (for Trace Object only) which parameter to calculate convergence.test – current options are Selection, Mutation, MixtureProbability, Sphi, Mphi, ExpectedPhi, and AcceptanceCSP
mixture (for Trace Object only) mixture for which to calculate convergence.test
convergence.test

Details

Be aware that convergence.test for Trace objects works primarily for Trace objects from the ROC parameter class. Future updates will adapt this function to work for parameters from other models and expression traces.

Value

Geweke score object evaluating whether means of two fractions (frac1 and frac2) differ. Convergence occurs when they don’t differ significantly, i.e. \( \text{pnorm}(\text{abs}(\text{convergence.test}(\text{mcmcObj})/2 > 0.05) \)

Examples

```r
## check for convergence after a run:

genome_file <- system.file("extdata", "genome.fasta", package = "AnaCoDa")

geno <- initializeGenomeObject(file = genome_file)
sphi_init <- c(1,1)
numMixtures <- 2
geneAssignment <- c(rep(1,floor(length(genome)/2)),rep(2,ceiling(length(genome)/2)))
parameter <- initializeParameterObject(genome = genome, sphi = sphi_init,
num.mixtures = numMixtures,
gene.assignment = geneAssignment,
mixture.definition = "allUnique")

samples <- 2500
thinning <- 50
adaptiveWidth <- 25
mcmc <- initializeMCMCObject(samples = samples, thinning = thinning,
adaptive.width=adaptiveWidth, est.expression=TRUE,
est.csp=TRUE, est.hyper=TRUE, est.mix = TRUE)
divergence.iteration <- 10

## Not run:
runMCMC(mcmc = mcmc, genome = genome, model = model,
ncores = 4, divergence.iteration = divergence.iteration)

# check if posterior trace has converged
convergence.test(object = mcmc, samples = 500, plot = TRUE)

trace <- getTrace(parameter)

# check if Mutation trace has converged
convergence.test(object = trace, samples = 500, plot = TRUE, what = "Mutation")

# check if Sphi trace has converged
convergence.test(object = trace, samples = 500, plot = TRUE, what = "Sphi")

# check if ExpectedPhi trace has converged
convergence.test(object = trace, samples = 500, plot = TRUE, what = "ExpectedPhi")

## End(Not run)
```
**findOptimalCodon**

*Find and return list of optimal codons*

**Description**

`findOptimalCodon` extracts the optimal codon for each amino acid.

**Usage**

```r
defineOptimalCodon(csp)
```

**Arguments**

- `csp`: a `data.frame` as returned by `getCSPEstimates`.

**Value**

A named list with optimal codons for each amino acid.

**Examples**

```r
gene_file <- system.file("extdata", "genome.fasta", package = "AnaCoDa")

genome <- initializeGenomeObject(file = gene_file)
sphi_init <- 1
numMixtures <- 1
geneAssignment <- rep(1, length(genome))
parameter <- initializeParameterObject(genome = genome, sphi = sphi_init,
num.mixtures = numMixtures,
gene.assignment = geneAssignment,
mixture.definition = "allUnique")

model <- initializeModelObject(parameter = parameter, model = "ROC")
samples <- 2500
thinning <- 50
adaptiveWidth <- 25
mcmc <- initializeMCMCObject(samples = samples, thinning = thinning,
adaptive.width=adaptiveWidth, est.expression=TRUE,
est.csp=TRUE, est.hyper=TRUE, est.mix = TRUE)

divergence.iteration <- 10
```
**fixDEta**

```r
fixDEta
```

**Description**

Method of Parameter class (access via parameter$<function name>, where parameter is an object initialized by initializeParameterObject). Fix the value of selection its current value

**fixDM**

```r
fixDM
```

**Description**

Method of Parameter class (access via parameter$<function name>, where parameter is an object initialized by initializeParameterObject). Fix the value of mutation its current value

**fixSphi**

```r
fixSphi
```

**Description**

Method of Parameter class (access via parameter$<function name>, where parameter is an object initialized by initializeParameterObject). Fix the value of s_phi (standard deviation of lognormal for synthesis rates) at its current value

**geomMean**

```r
geomMean
```

**Description**

Take the geometric mean of a vector

**Usage**

`geomMean(x, rm.invalid = TRUE, default = 1e-05)`
Arguments

x  A vector of numerical.

rm.invalid  Boolean value for handling 0, negative, or NA values in the vector. Default is TRUE and will not include these values in the calculation. If FALSE, these values will be replaced by the value given to default and will be included in the calculation.

default  Numerical value that serves as the value to replace 0, negative, or NA values in the calculation when rm.invalid is FALSE. Default is 1e-5.

Details

This function is a special version of the geometric mean specifically for AnaCoda. Most models in AnaCoda assume a log normal distribution for phi values, thus all values in x are expected to be positive. geomMean returns the geometric mean of a vector and can handle 0, negative, or NA values.

Value

Returns the geometric mean of a vector.

Examples

x <- c(1, 2, 3, 4)
geomMean(x)

y <- c(1, NA, 3, 4, 0, -1)
# Only take the mean of non-Na values greater than 0
geomMean(y)

# Replace values <= 0 or NAs with a default value 0.001 and then take the mean
geomMean(y, rm.invalid = FALSE, default = 0.001)

getAdaptiveWidth

Description

Return sample adaptiveWidth value, which is the number of samples (not iterations) between adapting parameter proposal widths.

Value

number of sample steps between adapting proposal widths
getCAI

*Calculate the Codon Adaptation Index*

**Description**

getCAI returns the Codon Adaptation Index for a genome based on a provided reference.

**Usage**

getCAI(referenceGenome, testGenome, default.weight = 0.5)

**Arguments**

- **referenceGenome**
  
  A genome object initialized with `initializeGenomeObject`. Serves as reference set to calculate the necessary codon weights.

- **testGenome**
  
  A genome object initialized with `initializeGenomeObject`. The genome for which the CAI is supposed to be calculated

- **default.weight**
  
  Default weight to use if codon is missing from referenceGenome

**Value**

Returns a named vector with the CAI for each gene

**Examples**

```r
genome_file1 <- system.file("extdata", "more_genes.fasta", package = "AnaCoDa")
genome_file2 <- system.file("extdata", "genome.fasta", package = "AnaCoDa")

# reading genome
referenceGenome <- initializeGenomeObject(file = genome_file1)
testGenome <- initializeGenomeObject(file = genome_file2)

cai <- getCAI(referenceGenome, testGenome)
```

getCAIweights

*Calculate the CAI codon weights for a reference genome*

**Description**

getCAIweights returns the weights for the Codon Adaptation Index based on a reference genome.

**Usage**

getCAIweights(referenceGenome, default.weight = 0.5)
getCodonCounts

Arguments

referenceGenome
A genome object initialized with initializeGenomeObject.
default.weight Set default weight for any codon not observed in the reference genome

Value
Returns a named vector with the CAI weights for each codon

Examples

```r
genome_file <- system.file("extdata", "genome.fasta", package = "AnaCoDa")

## reading genome
referenceGenome <- initializeGenomeObject(file = genome_file)

wi <- getCAIweights(referenceGenome)
```

getcodonCounts  Get Codon Counts For all Amino Acids

Description
provides the codon counts for a given amino acid across all genes

Usage
getcodonCounts(genome)

Arguments
genome A genome object from which the counts of each codon can be obtained.

Details
The returned matrix contains a row for each gene and a column for each synonymous codon of aa.

Value
Returns a data.frame storing the codon counts for each amino acid.
**getCodonCountsForAA**

**Get Codon Counts For a specific Amino Acid**

**Description**

provides the codon counts for a given amino acid across all genes

**Usage**

`getCodonCountsForAA(aa, genome)`

**Arguments**

- `aa` One letter code of the amino acid for which the codon counts should be returned
- `genome` A genome object from which the counts of each codon can be obtained.

**Details**

The returned matrix contains a row for each gene and a column for each synonymous codon of `aa`.

**Value**

Returns a data.frame storing the codon counts for the specified amino acid.

**Examples**

```r
genome_file <- system.file("extdata", "genome.fasta", package = "AnaCoDa")

## reading genome
genome <- initializeGenomeObject(file = genome_file)
counts <- getCodonCounts(genome)
```

```r
genome_file <- system.file("extdata", "genome.fasta", package = "AnaCoDa")

## reading genome
genome <- initializeGenomeObject(file = genome_file)
counts <- getCodonCountsForAA("A", genome)
```
getCodonSpecificPosteriorMeanForCodon

Description

Method of Parameter class (access via parameter$<function name>, where parameter is an object initialized by initializeParameterObject). Calculate codon-specific parameter (CSP) posterior mean

Arguments

- mixtureElement: mixture to calculate CSP posterior mean. Should be between 1 and n, where n is number of mixtures.
- samples: number of samples to use for calculating posterior mean
- codon: codon to calculate CSP
- paramType: CSP to calculate posterior mean for. 0: Mutation (ROC,FONSE) or Alpha (PA, PANSE). 1: Selection (ROC,FONSE), Lambda (PANSE), Lambda^prime (PA). 2: NSERate (PANSE)
- withoutReference: If model uses reference codon, then ignore this codon (fixed at 0). Should be TRUE for ROC and FONSE. Should be FALSE for PA and PANSE.
- log_scale: If true, calculate posterior mean on log scale. Should only be used for PA and PANSE.

Value

posterior mean value for CSP

getCodonSpecificPosteriorVarianceForCodon

Description

Method of Parameter class (access via parameter$<function name>, where parameter is an object initialized by initializeParameterObject). Calculate codon-specific parameter (CSP) variance
**Arguments**

- **mixtureElement**: mixture to calculate CSP variance. Should be between 1 and n, where n is number of mixtures.
- **samples**: number of samples to use for calculating variance
- **codon**: codon to calculate CSP
- **paramType**: CSP to calculate variance for. 0: Mutation (ROC,FONSE) or Alpha (PA, PANSE). 1: Selection (ROC,FONSE), Lambda (PANSE), Lambda^prime (PA). 2: NSerate (PANSE)
- **unbiased**: If TRUE, should calculate variance using unbiased (N-1). Otherwise, used biased (N) correction
- **withoutReference**: If model uses reference codon, then ignore this codon (fixed at 0). Should be TRUE for ROC and FONSE. Should be FALSE for PA and PANSE.
- **log_scale**: If true, calculate posterior mean on log scale. Should only be used for PA and PANSE.

**Value**

variance over trace for CSP

---

**Description**

Method of Parameter class (access via parameter$<function name>, where parameter is an object initialized by initializeParameterObject). Calculate quantiles of CSP traces

**Arguments**

- **mixtureElement**: mixture to calculate CSP variance. Should be between 1 and n, where n is number of mixtures.
- **samples**: number of samples to use for calculating variance
- **codon**: codon to calculate CSP
- **paramType**: CSP to calculate variance for. 0: Mutation (ROC,FONSE) or Alpha (PA, PANSE). 1: Selection (ROC,FONSE), Lambda (PANSE), Lambda^prime (PA). 2: NSerate (PANSE)
- **probs**: vector of two doubles between 0 and 1 indicating range over which to calculate quantiles. <0.0275, 0.975> would give 95% quantiles.
- **withoutReference**: If model uses reference codon, then ignore this codon (fixed at 0). Should be TRUE for ROC and FONSE. Should be FALSE for PA and PANSE.
- **log_scale**: If true, calculate posterior mean on log scale. Should only be used for PA and PANSE.
getCSPEstimates

**Value**

vector representing lower and upper bound of quantile

getCSPEstimates

Return Codon Specific Parameters (or write to csv) estimates as data.frame

**Description**

getCSPEstimates returns the codon specific parameter estimates for a given parameter and mixture or write it to a csv file.

**Usage**

getCSPEstimates(
  parameter,
  filename = NULL,
  mixture = 1,
  samples = 10,
  relative.to.optimal.codon = T,
  report.original.ref = T,
  log.scale = F
)

**Arguments**

- **parameter**: parameter an object created by initializeParameterObject.
- **filename**: Posterior estimates will be written to file (format: csv). Filename will be in the format `<parameter_name>_filename.csv`.
- **mixture**: estimates for which mixture should be returned.
- **samples**: The number of samples used for the posterior estimates.
- **relative.to/optimal.codon**: Boolean determining if parameters should be relative to the preferred codon or the alphabetically last codon (Default=TRUE). Only applies to ROC and FONSE models.
- **report.original.ref**: Include the original reference codon (Default = TRUE). Note this is only included for the purposes of simulations, which expect the input parameter file to be in a specific format. Later version of AnaCoDa will remove this.
- **log.scale**: Calculate posterior means, standard deviation, and posterior probability intervals on the natural log scale. Should be used for PA and PANSE models only.

**Value**

returns a list data.frame with the posterior estimates of the models codon specific parameters or writes it directly to a csv file if filename is specified.
getEstimatedMixtureAssignmentForGene

Examples

```r
genome_file <- system.file("extdata", "genome.fasta", package = "AnaCoDa")
genome <- initializeGenomeObject(file = genome_file)
sphi_init <- c(1,1)
numMixtures <- 2
geneAssignment <- c(rep(1,floor(length(genome)/2)),rep(2,ceiling(length(genome)/2)))
parameter <- initializeParameterObject(genome = genome, sphi = sphi_init,
  num.mixtures = numMixtures,
  gene.assignment = geneAssignment,
  mixture.definition = "allUnique")
model <- initializeModelObject(parameter = parameter, model = "ROC")
samples <- 2500
thinning <- 50
adaptiveWidth <- 25
mcmc <- initializeMCMCObject(samples = samples, thinning = thinning,
  adaptive.width=adaptiveWidth, est.expression=TRUE,
  est.csp=TRUE, est.hyper=TRUE, est.mix = TRUE)
divergence.iteration <- 10
## Not run:
runMCMC(mcmc = mcmc, genome = genome, model = model,
  ncores = 4, divergence.iteration = divergence.iteration)
## return estimates for codon specific parameters
csp_mat <- getCSPEstimates(parameter)
# write the result directly to the filesystem as a csv file. No values are returned
getCSPEstimates(parameter, filename=file.path(tempdir(), "test.csv"))
## End(Not run)
```

description

Method of Parameter class (access via parameter$<function name>, where parameter is an object initialized by initializeParameterObject). Get estimated mixture assignment for gene

Arguments

- **samples**: number of samples over which to calculate mixture assignment
- **geneIndex**: corresponding index of gene in genome. Should be a number between 1 and length(genome).
Value

returns value between 1 and n, where n is number of mixtures

---

getEstimatedMixtureAssignmentProbabilitiesForGene

**getEstimatedMixtureAssignmentProbabilitiesForGene**

Description

Method of Parameter class (access via parameter$<function name>, where parameter is an object initialized by initializeParameterObject). Get estimated mixture assignment probabilities for gene

Arguments

- `samples` number of samples over which to calculate mixture assignment probabilities
- `geneIndex` corresponding index of gene in genome. Should be a number between 1 and length(genome).

Value

returns vector of probabilities representing mixture probabilities for gene

---

gExpressionEstimates

**getExpressionEstimates**

Returns the estimated phi posterior for a gene

Description

Posterior estimates for the phi value of specified genes

Usage

```r
gExpressionEstimates(
  parameter,
  gene.index,
  samples,
  quantiles = c(0.025, 0.975),
  genome = NULL
)
```
getExpressionEstimates

Arguments

parameter on object created by initializeParameterObject.
gene.index a integer or vector of integers representing the gene(s) of interest.
samples number of samples for the posterior estimate
quantiles vector of quantiles, (default: c(0.025, 0.975))
genome if genome is given, then will include gene ids in output (default is NULL)

Details

The returned vector is unnamed as gene ids are only stored in the genome object, but the gene.index vector can be used to match the assignment to the genome.

Value

returns a vector with the mixture assignment of each gene corresponding to gene.index in the same order as the genome.

Examples

```
 genome_file <- system.file("extdata", "genome.fasta", package = "AnaCoDa")

genome <- initializeGenomeObject(file = genome_file)
sphi_init <- c(1,1)
numMixtures <- 2
geneAssignment <- c(rep(1,floor(length(genome)/2)),rep(2,ceiling(length(genome)/2)))
parameter <- initializeParameterObject(genome = genome, sphi = sphi_init,
                                        num.mixtures = numMixtures,
                                        gene.assignment = geneAssignment,
                                        mixture.definition = "allUnique")

model <- initializeModelObject(parameter = parameter, model = "ROC")
samples <- 2500
thinning <- 50
adaptiveWidth <- 25
mcmc <- initializeMCMCObject(samples = samples, thinning = thinning,
                            adaptive.width=adaptiveWidth, est.expression=TRUE,
                            est.csp=TRUE, est.hyper=TRUE, est.mix = TRUE)

divergence.iteration <- 10
## Not run:
runMCMC(mcmc = mcmc, genome = genome, model = model,
        ncores = 4, divergence.iteration = divergence.iteration)

# get the estimated expression values for all genes based on the mixture
# they are assigned to at each step
estimatedExpression <- getExpressionEstimates(parameter, 1:length(genome), 1000)

## End(Not run)
```
**getLogPosteriorMean**

**getGroupList**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method of Parameter class (access via parameter$&lt;function name&gt;$, where parameter is an object initialized by initializeParameterObject). Get amino acids (ROC, FONSE) or codons (PA, PANSE) for which parameters will be estimated</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>returns list of amino acids or codons</td>
</tr>
</tbody>
</table>

**getLogLikelihoodTrace**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method of MCMC class (access via mcmc$&lt;function name&gt;$, where mcmc is an object initialized by initializeMCMCOBJECT). Returns the logLikelihood trace</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>vector representing logLikelihood trace</td>
</tr>
</tbody>
</table>

**getLogPosteriorMean**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method of MCMC class (access via mcmc$&lt;function name&gt;$, where mcmc is an object initialized by initializeMCMCOBJECT). Calculate the mean log posterior probability over the last n samples</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>samples</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>mean logPosterior</td>
</tr>
</tbody>
</table>
getLogPosteriorTrace

Description

Method of MCMC class (access via mcmc$<function name>, where mcmc is an object initialized by initializeMCMCObject). Returns the logPosterior trace

Value

vector representing logPosterior trace

getMixtureAssignmentEstimate

Returns mixture assignment estimates for each gene

Description

Posterior estimates for the mixture assignment of specified genes

Usage

getMixtureAssignmentEstimate(parameter, gene.index, samples)

Arguments

parameter on object created by initializeParameterObject

gene.index a integer or vector of integers representing the gene(s) of interest.

samples number of samples for the posterior estimate

Details

The returned vector is unnamed as gene ids are only stored in the genome object, but the gene.index vector can be used to match the assignment to the genome.

Value

returns a vector with the mixture assignment of each gene corresponding to gene.index in the same order as the genome.
Examples

```r
genome_file <- system.file("extdata", "genome.fasta", package = "AnaCoDa")

genome <- initializeGenomeObject(file = genome_file)
sphi_init <- c(1,1)
numMixtures <- 2
geneAssignment <- c(rep(1,floor(length(genome)/2)),rep(2,ceiling(length(genome)/2)))
parameter <- initializeParameterObject(genome = genome, sphi = sphi_init,
                                        numMixtures = numMixtures,
                                        gene.assignment = geneAssignment,
                                        mixture.definition = "allUnique")

model <- initializeModelObject(parameter = parameter, model = "ROC")
samples <- 2500
thinning <- 50
adaptiveWidth <- 25
mcmc <- initializeMCMCObject(samples = samples, thinning = thinning, adaptive.width=adaptiveWidth,
                               est.expression=TRUE, est.csp=TRUE, est.hyper=TRUE, est.mix = TRUE)
divergence.iteration <- 10
## Not run:
runMCMC(mcmc = mcmc, genome = genome, model = model,
ncores = 4, divergence.iteration = divergence.iteration)

# get the mixture assignment for all genes
mixAssign <- getMixtureAssignmentEstimate(parameter = parameter,
                                            gene.index = 1:length(genome), samples = 1000)

# get the mixture assignment for a subsample
mixAssign <- getMixtureAssignmentEstimate(parameter = parameter,
                                            gene.index = 5:100, samples = 1000)
# or
mixAssign <- getMixtureAssignmentEstimate(parameter = parameter,
                                            gene.index = c(10, 30:50, 3, 90), samples = 1000)
## End(Not run)
```

---

getNames

**Gene Names of Genome**

**Description**

returns the identifiers of the genes within the genome specified.

**Usage**

```r
getNames(genome, simulated = FALSE)
```
getNc

Arguments

  genome A genome object initialized with initializeGenomeObject.
  simulated A logical value denoting if the gene names to be listed are simulated or not. The default value is FALSE.

Value

gene.names Returns the names of the genes as a vector of strings.

Examples

```r
genome_file <- system.file("extdata", "genome.fasta", package = "AnaCoDa")
## reading genome
genome <- initializeGenomeObject(file = genome_file)
## return all gene ids for the genome
geneIDs <- getNames(genome, FALSE)
```

---

getNc

*Calculate the Effective Number of Codons*

Description

getNc returns the Effective Number of Codons for a genome.

Usage

getNc(genome)

Arguments

  genome A genome object initialized with initializeGenomeObject.

Value

Returns a named vector with the Effective Number of Codons for each gene

Examples

```r
genome_file <- system.file("extdata", "more_genes.fasta", package = "AnaCoDa")
## reading genome
genome <- initializeGenomeObject(file = genome_file)
nc <- getNc(genome)
```
getNcAA

*Calculate the Effective Number of Codons for each Amino Acid*

Description

getNcAA returns the Effective Number of Codons for each Amino Acid.

Usage

getNcAA(genome)

Arguments

genome A genome object initialized with `initializeGenomeObject`.

Value

Returns an object of type `data.frame` with the Effective Number of Codons for each amino acid in each gene.

Examples

```r
genie_file <- system.file("extdata", "more_genes.fasta", package = "AnaCoDa")
## reading genome
genome <- initializeGenomeObject(file = genio_file)
nc <- getNcAA(genome)
```

getNoiseOffsetPosteriorMean

Method of Parameter class (access via parameter$<function name>, where parameter is an object initialized by `initializeParameterObject`). Calculate posterior mean of standard deviation parameter of lognormal describing distribution of synthesis rates.

Arguments

index mixture index to use. Should be number between 0 and n-1, where n is number of mixtures

samples number of samples over which to calculate posterior mean
**getNoiseOffsetVariance**

*Value*

returns posterior mean for standard deviation of lognormal distribution of synthesis rates

**Description**

Method of Parameter class (access via parameter$<function name>, where parameter is an object initialized by initializeParameterObject). Calculate variance of noise offset parameter used when fitting model with empirical estimates of synthesis rates (ie. withPhi fits)

**Arguments**

- **index**: mixture index to use. Should be number between 0 and n-1, where n is number of mixtures
- **samples**: number of samples over which to calculate variance
- **unbiased**: If TRUE, should calculate variance using unbiased (N-1). Otherwise, used biased (N) correction

*Value*

returns variance for noise offset

**getObservedSynthesisRateSet**

*Get gene observed synthesis rates*

**Description**

getObservedSynthesisRateSet returns the observed synthesis rates of the genes within the genome specified.

**Usage**

getObservedSynthesisRateSet(genome, simulated = FALSE)

**Arguments**

- **genome**: A genome object initialized with initializeGenomeObject.
- **simulated**: A logical value denoting if the synthesis rates to be listed are simulated or not. The default value is FALSE.
getSelectionCoefficients

Value

Returns a data.frame with the observed expression values in genome

Examples

```r
genome_file <- system.file("extdata", "genome.fasta", package = "AnaCoDa")
expression_file <- system.file("extdata", "expression.csv", package = "AnaCoDa")
# reading genome
genome <- initializeGenomeObject(file = genome_file)

# return expression values as a data.frame with gene ids in the first column.
expressionValues <- getObservedSynthesisRateSet(genome = genome)
```

getSamples

Description

Method of MCMC class (access via mcmc$<function name>, where mcmc is an object initialized by initializeMCMCObject). Return number of samples set for MCMCAlgorithm object

Value

number of samples used during MCMC

getSelectionCoefficients

Calculate Selection coefficients

Description

getSelectionCoefficients calculates the selection coefficient of each codon in each gene.

Usage

getSelectionCoefficients(genome, parameter, samples = 100)

Arguments

genome A genome object initialized with initializeGenomeObject to add observed expression data.
parameter an object created by initializeParameterObject.
samples The number of samples used for the posterior estimates.
Value

A matrix with selection coefficients.

Examples

gene_file <- system.file("extdata", "genome.fasta", package = "AnaCoDa")

genome <- initializeGenomeObject(file = gene_file)
sphi_init <- 1
numMixtures <- 1
geneAssignment <- rep(1, length(genome))
param <- initializeParameterObject(genome = genome, sphi = sphi_init,  
  numMixtures = numMixtures,  
  geneAssignment = geneAssignment,  
  mixture.definition = "allUnique")
model <- initializeModelObject(parameter = param, model = "ROC")
samples <- 2500
thinning <- 50
adaptiveWidth <- 25
mcmc <- initializeMCMCObject(samples = samples, thinning = thinning,  
  adaptive.width=adaptiveWidth, est.expression=TRUE,  
  est.csp=TRUE, est.hyper=TRUE, est.mix = TRUE)
divergence.iteration <- 10
## Not run:
runMCMC(mcmc = mcmc, genome = genome, model = model,  
  ncores = 4, divergence.iteration = divergence.iteration)

## return estimates for selection coefficients s for each codon in each gene
selection.coefficients <- getSelectionCoefficients(genome = genome,  
  parameter = parameter, samples = 1000)

## End(Not run)

getStdDevSynthesisRatePosteriorMean

getStdDevSynthesisRatePosteriorMean

Description

Method of Parameter class (access via parameter$<function name>, where parameter is an object initialized by initializeParameterObject). Calculate posterior mean of standard deviation parameter of lognormal describing distribution of synthesis rates

Arguments

- **samples**: number of samples over which to calculate posterior mean
- **mixture**: mixture index to use. Should be number between 0 and n-1, where n is number of mixtures
Value

returns posterior mean for standard deviation of lognormal distribution of synthesis rates

---

`getStdDevSynthesisRateVariance`

Description

Method of Parameter class (access via parameter$<function name>, where parameter is an object initialized by initializeParameterObject). Calculate variance of standard deviation parameter of lognormal describing distribution of synthesis rates

Arguments

- `samples`: number of samples over which to calculate variance
- `mixture`: mixture index to use. Should be number between 0 and n-1, where n is number of mixtures
- `unbiased`: If TRUE, should calculate variance using unbiased (N-1). Otherwise, used biased (N) correction

Value

returns variance for standard deviation of lognormal distribution of synthesis rates

---

`getStepsToAdapt`

Description

Method of MCMC class (access via mcmc$<function name>, where mcmc is an object initialized by initializeMCMCOobject). Return number of iterations (total iterations = samples * thinning) to allow proposal widths to adapt

Value

number of sample steps to adapt
**getSynthesisRate**

**Description**

Method of Parameter class (access via parameter$<function name>$, where parameter is an object initialized by initializeParameterObject). Get current synthesis rates for all genes and all mixtures.

**Value**

2 by 2 vector of numeric values

---

**getSynthesisRatePosteriorMeanForGene**

**Description**

Method of Parameter class (access via parameter$<function name>$, where parameter is an object initialized by initializeParameterObject). Get posterior mean synthesis rate value for a gene.

**Arguments**

- **samples**: number of samples over which to calculate mean
- **geneIndex**: corresponding index of gene in genome for which posterior mean synthesis rate will be calculated. Should be a number between 1 and length(genome)
- **log_scale**: Calculate posterior mean on log scale

**Value**

posterior mean synthesis rate for gene
getSynthesisRatePosteriorVarianceForGene

Description

Method of Parameter class (access via parameter$<function name>, where parameter is an object initialized by initializeParameterObject). Get synthesis rate variance for a gene.

Arguments

- **samples**: number of samples over which to calculate variance.
- **geneIndex**: corresponding index of gene in genome for which synthesis rate variance will be calculated. Should be a number between 1 and length(genome).
- **unbiased**: Should calculate variance using unbiased (N-1) or biased (N) correction.
- **log_scale**: Calculate variance on log scale.

Value

posterior mean synthesis rate for gene.

getThinning

Description

Method of MCMC class (access via mcmc$<function name>, where mcmc is an object initialized by initializeMCMCObject). Return thinning value, which is the number of iterations (total iterations = samples * thinning) not being kept.

Value

thinning value used during MCMC.
getTrace

extracts an object of traces from a parameter object.

Description
extracts an object of traces from a parameter object.

Usage
getTrace(parameter)

Arguments
parameter A Parameter object that corresponds to one of the model types.

Value
trace Returns an object of type Trace extracted from the given parameter object

Examples

```
genome_file <- system.file("extdata", "genome.fasta", package = "AnaCoDa")

genome <- initializeGenomeObject(file = genome_file)
sphi_init <- c(1,1)
numMixtures <- 2
geneAssignment <- c(rep(1,floor(length(genome)/2)),rep(2,ceiling(length(genome)/2)))
parameter <- initializeParameterObject(genome = genome, sphi = sphi_init,
                                        num.mixtures = numMixtures,
                                        gene.assignment = geneAssignment,
                                        mixture.definition = "allUnique")

trace <- getTrace(parameter) # empty trace object since no MCMC was performed
```

getTraceObject

getTraceObject

Description
Method of Parameter class (access via parameter$<function name>, where parameter is an object initialized by initializeParameterObject). Get Trace object stored by a Parameter object. Useful for plotting certain parameter traces.

Value
Trace object
Description

Initialize Covariance Matrices

Usage

initializeCovarianceMatrices(
  parameter,
  genome,
  numMixtures,
  geneAssignment,
  init.csp.variance = 0.0025
)

Arguments

parameter A Parameter object that corresponds to one of the model types. Valid values are "ROC", "PA", and "FONSE".

genome An object of type Genome necessary for the initialization of the Parameter object.

numMixtures The number of mixture elements for the underlying mixture distribution (numMixtures > 0).

geneAssignment A vector holding the initial mixture assignment for each gene. The vector length has to equal the number of genes in the genome. Valid values for the vector range from 1 to numMixtures. It is possible but not advised to leave a mixture element empty.

init.csp.variance initial proposal variance for codon specific parameter, default is 0.0025.

Value

parameter Returns the Parameter argument, now modified with initialized mutation, selection, and covariance matrices.
**initializeGenomeObject**

*Genome Initialization*

**Description**

`initializeGenomeObject` initializes the Rcpp Genome object

**Usage**

```r
initializeGenomeObject(
  file,
  genome = NULL,
  observed.expression.file = NULL,
  fasta = TRUE,
  positional = FALSE,
  match.expression.by.id = TRUE,
  append = FALSE
)
```

**Arguments**

- **file**: A file of coding sequences in fasta or RFPData format
- **genome**: A genome object can be passed in to concatenate the input file to it (optional).
- **observed.expression.file**: String containing the location of a file containing empirical expression rates (optional). Default value is NULL.
- **fasta**: Boolean value indicating whether `file` argument is a fasta file (TRUE) or an RFPData file (FALSE). Default value is TRUE.
- **positional**: Boolean indicating if the positional information in the RFPData file is necessary. Default value is FALSE.
- **match.expression.by.id**: If TRUE, observed expression values will be assigned by matching sequence identifier. If FALSE, observed expression values will be assigned by order. Default value is TRUE.
- **append**: If TRUE, function will read in additional genome data to append to an existing genome. If FALSE, genome data is cleared before reading in data (no preexisting data). Default value is FALSE.

**Value**

This function returns the initialized Genome object.
initializeMCMCOBJ ect

Examples

```r
geno me_file <- system.file("extdata", "genome.fasta", package = "AnaCoDa")
genes_file <- system.file("extdata", "more_genes.fasta", package = "AnaCoDa")
expression_file <- system.file("extdata", "expression.csv", package = "AnaCoDa")

## reading genome
genome <- initializeGenomeObject(file = genome_file)

## reading genome and observed expression data
genome <- initializeGenomeObject(file = genome_file, observed.expression.file = expression_file)

## add additional genes to existing genome
genome <- initializeGenomeObject(file = genome_file)
genome <- initializeGenomeObject(file = genes_file, genome = genome, append = TRUE)
```

---

**initializeMCMCOBJ ect**  
*Initialize MCMC*

**Description**

*initializeMCMCOBJ ect* initializes a MCMC object to perform a model fitting for a parameter and model object.

**Usage**

```r
initializeMCMCOBJ ect(
  samples,
  thinning = 1,
  adaptive.width = 100,
  est.expression = TRUE,
  est.csp = TRUE,
  est.hyper = TRUE,
  est.mix = TRUE
)
```

**Arguments**

- `samples` Number of samples to be produced when running the MCMC algorithm. No default value.
- `thinning` The thinning interval between consecutive observations. If set to 1, every step will be saved as a sample. Default value is 1.
- `adaptive.width` Number that determines how often the acceptance/rejection window should be altered. Default value is 100. Proportion of MCMC steps where the proposal distribution is adaptive can be set using `mcmc$setStepsToAdapt`. The default parameter passed in as -1 uses the full iterations.
initializeModelObject

est.expression Boolean that tells whether or not synthesis rate values should be estimated in the MCMC algorithm run. Default value is TRUE.
est.csp Boolean that tells whether or not codon specific values should be estimated in the MCMC algorithm run. Default value is TRUE.
est.hyper Boolean that tells whether or not hyper parameters should be estimated in the MCMC algorithm run. Default value is TRUE. Setting for expression noise parameter sepsilon can be overridden by setting fix.observation.noise in initializeModelObject()
est.mix Boolean that tells whether or not the genes’ mixture element should be estimated in the MCMC algorithm run. Default value is TRUE.

Details

initializeMCMCObject sets up the MCMC object (monte carlo markov chain) and returns the object so a model fitting can be done. It is important to note that est.expression and est.hyper will affect one another negatively if their values differ.

Value

mcmc Returns an initialized MCMC object.

Examples

```r
## initializing an object of type mcmc

samples <- 2500
thinning <- 50
adaptiveWidth <- 25

## estimate all parameter types
mcmc <- initializeMCMCObject(samples = samples, thinning = thinning, adaptive.width=adaptiveWidth,
est.expression=TRUE, est.csp=TRUE, est.hyper=TRUE, est.mix = TRUE)

## do not estimate expression values, initial conditions will remain constant
mcmc <- initializeMCMCObject(samples = samples, thinning = thinning, adaptive.width=adaptiveWidth,
est.expression=FALSE, est.csp=TRUE, est.hyper=TRUE, est.mix = TRUE)

## do not estimate hyper parameters, initial conditions will remain constant
mcmc <- initializeMCMCObject(samples = samples, thinning = thinning, adaptive.width=adaptiveWidth,
est.expression=TRUE, est.csp=TRUE, est.hyper=FALSE, est.mix = TRUE)
```

initializeModelObject  Model Initialization

Description

initializes the model object.
initializeModelObject

Usage

initializeModelObject(
  parameter,  
  model = "ROC",  
  with.phi = FALSE,  
  fix.observation.noise = FALSE,  
  rfp.count.column = 1
)

Arguments

parameter An object created with initializeParameterObject.
model A string containing the model to run (ROC, FONSE, or PA), has to match parameter object.
with.phi (ROC only) A boolean that determines whether or not to include empirical phi values (expression rates) for the calculations. Default value is FALSE.
fix.observation.noise (ROC only) Allows fixing the noise term epsilont in the observed expression dataset to its initial condition. This value should override the est.hyper=TRUE setting in initializeMCMCObject() The initial condition for the observed expression noise is set in the parameter object. Default value is FALSE.
rfp.count.column (PA and PANSE only) A number representing the RFP count column to use. Default value is 1.

Details

initializeModelObject initializes a model. The type of model is determined based on the string passed to the model argument. The Parameter object has to match the model that is initialized. E.g. to initialize a ROC model, it is required that a ROC parameter object is passed to the function.

Value

This function returns the model object created.

Examples

#initializing a model object

genome_file <- system.file("extdata", "genome.fasta", package = "AnaCoDa")
expression_file <- system.file("extdata", "expression.csv", package = "AnaCoDa")
genome <- initializeGenomeObject(file = genome_file, observed.expression.file = expression_file)
sphi_init <- c(1,1)
numMixtures <- 2
geneAssignment <- c(rep(1,floor(length(genome)/2)),rep(2,ceiling(length(genome)/2)))
parameter <- initializeParameterObject(genome = genome, sphi = sphi_init,
# initializing a model object assuming we have observed expression (\(\phi\)) values stored in the genome object.
initializeModelObject(parameter = parameter, model = "ROC", with\(\phi\) = TRUE)

# initializing a model object ignoring observed expression (\(\phi\)) values stored in the genome object.
initializeModelObject(parameter = parameter, model = "ROC", with\(\phi\) = FALSE)

---

**initializeParameterObject**

*Initialize Parameter*

**Description**

initializeParameterObject initializes a new parameter object or reconstructs one from a restart file.

**Usage**

```
initializeParameterObject(
    genome = NULL,
    sphi = NULL,
    num.mixtures = 1,
    gene.assignment = NULL,
    initial.expression.values = NULL,
    model = "ROC",
    split.serine = TRUE,
    mixture.definition = "allUnique",
    mixture.definition.matrix = NULL,
    init.with.restart.file = NULL,
    mutation.prior.mean = 0,
    mutation.prior.sd = 0.35,
    propose.by.prior = FALSE,
    init.csp.variance = 0.0025,
    init.sepsilon = 0.1,
    init.w.obs.phi = FALSE,
    init.initiation.cost = 4,
    init.partition.function = 1
)
```

**Arguments**

- **genome**: An object of type Genome necessary for the initialization of the Parameter object. The default value is NULL.
sphi

Initial values for sphi. Expected is a vector of length numMixtures. The default value is NULL.

num.mixtures

The number of mixtures elements for the underlying mixture distribution (numMixtures > 0). The default value is 1.

gene.assignment

A vector holding the initial mixture assignment for each gene. The vector length has to equal the number of genes in the genome. Valid values for the vector range from 1 to numMixtures. It is possible but not advised to leave a mixture element empty. The default Value is NULL.

initial.expression.values

(Optional) A vector with initial phi values. The length of the vector has to equal the number of genes in the Genome object and the order of the genes should match the order of the genes in the Genome. The default value is NULL.

model

Specifies the model used. Valid options are "ROC", "PA", "PANSE", or "FONSE". The default model is "ROC". ROC is described in Gilchrist et al. 2015. PA, PANSE and FONSE are currently unpublished.

split.serine

Whether serine should be considered as one or two amino acids when running the model. TRUE and FALSE are the only valid values. The default value for split.serine is TRUE.

mixture.definition

A string describing how each mixture should be treated with respect to mutation and selection. Valid values consist of "allUnique", "mutationShared", and "selectionShared". The default value for mixture.definition is "allUnique". See details for more information.

mixture.definition.matrix

A matrix representation of how the mutation and selection categories correspond to the mixtures. The default value for mixture.definition.matrix is NULL. If provided, the model will use the matrix to initialize the mutation and selection categories instead of the definition listed directly above. See details for more information.

init.with.restart.file

File name containing information to reinitialize a previous Parameter object. If given, all other arguments will be ignored. The default value for init.with.restart.file is NULL.

mutation.prior.mean

Controlling the mean of the normal prior on mutation parameters. If passed in as single number (default is 0), this will be the mean value for all categories, for all codons. User may also supply a vector with n * 40 values, where n is the number of mutation categories. Future versions will check the number of rows matches the number of mutation categories definded by user.

mutation.prior.sd

Controlling the standard deviation of the normal prior on the mutation parameters. If passed in as single number (default is 0.35), this will be the standard deviation value for all categories, for all codons. User may also supply a vector with n * 40 values, where n is the number of mutation categories. Future versions will check the number of rows matches the number of mutation categories definded by user.
propose.by.prior
Mutation bias parameters will be proposed based on the means and standard
deviations set in mutation.prior.mean and mutation.prior.sd

init.csp.variance
specifies the initial proposal width for codon specific parameter (default is 0.0025).
The proposal width adapts during the runtime to reach a target acceptance rate of
~0.25

init.sepsilon
specifies the initial value for sepsilon. default is 0.1

init.w.obs.phi
If TRUE, initialize phi values with observed phi values (data from RNAseq, mass spectrometry, ribosome footprinting) Default is FALSE. If multiple observed phi values exist for a gene, the geometric mean of these values is used as initial phi. When using this function, one should remove any genes with missing phi values, as these genes will not have an initial phi value.

init.initiation.cost
FOR FONSE ONLY. Initializes the initiation cost a_1 at this value.

init.partition.function
FOR PANSE ONLY. initializers the partition function Z.

Details
initializeParameterObject checks the values of the arguments given to insure the values are valid.

The mixture definition and mixture definition matrix describes how the mutation and selection categories are set up with respect to the number of mixtures. For example, if mixture.definition = "allUnique" and numMixtures = 3, a matrix representation would be matrix(c(1,2,3,1,2,3),ncol=2) where each row represents a mixture, the first column represents the mutation category, and the second column represents the selection category. Another example would be mixture.definition = "selectionShared" and numMixtures = 4 (matrix(c(1,2,3,4,1,1,1,1),ncol=2)). In this case, the selection category is the same for every mixture. If a matrix is given, and it is valid, then the mutation/selection relationship will be defined by the given matrix and the keyword will be ignored. A matrix should only be given in cases where the keywords would not create the desired matrix.

Value
parameter Returns an initialized Parameter object.

Examples

```
geome_file <- system.file("extdata", "genome.fasta", package = "AnaCoDa")
restart_file <- system.file("extdata", "restart_file.rst", package = "AnaCoDa")

genome <- initializeGenomeObject(file = genome_file)

## initialize a new parameter object
sphi_init <- 1
numMixtures <- 1
geneAssignment <- rep(1, length(genome))
parameter <- initializeParameterObject(genome = genome, sphi = sphi_init,
```
num.mixtures = numMixtures,
gene.assignment = geneAssignment,
mixture.definition = "allUnique")

## re-initialize a parameter object from a restart file. Useful for checkpoin
ting
parameter <- initializeParameterObject(init.with.restart.file = restart_file)

## initialize a parameter object with a custom mixture definition matrix
def.matrix <- matrix(c(1,1,1,2), ncol=2)
geneAssignment <- c(rep(1,floor(length(genome)/2)),rep(2,ceiling(length(genome)/2)))
parameter <- initializeParameterObject(genome = genome, sphi = c(0.5, 2), num.mixtures = 2,
gene.assignment = geneAssignment,
mixture.definition.matrix = def.matrix)

initializeSynthesisRateByGenome

initializeSynthesisRateByGenome

Description

Method of Parameter class (access via parameter$<function name>, where parameter is an object initialized by initializeParameterObject). Initialize synthesis rates using SCUO values calculated from the genome

Arguments

geno me a Genome object

initializeSynthesisRateByList

initializeSynthesisRateByList

Description

Method of Parameter class (access via parameter$<function name>, where parameter is an object initialized by initializeParameterObject). Initialize synthesis rates with values passed in as a list

Arguments

expression a list of values to use as initial synthesis rate values. Should be same size as number of genes in genome.
**initializeSynthesisRateByRandom**

**Description**

Method of Parameter class (access via parameter$<function name>$, where parameter is an object initialized by initializeParameterObject). Initialize synthesis rates by drawing a from a lognormal distribution with mean = -(sd_phi)^2/2 and sd = sd_phi

**Arguments**

sd_phi a positive value which will be the standard deviation of the lognormal distribution

---

**initMutationCategories**

**Description**

Initialize values for mutation CSP. File should be of comma-separated with header. Three columns should be of order Amino_acid,Codon,Value

**Arguments**

files list of files containing starting values. Number of files should equal the number of categories.

numCategories number of mutation categories (should be less than or equal to number of mixtures)

fix Can use this parameter to fix mutation at current values (won’t change over course of MCMC run)
initSelectionCategories

Description
Method of Parameter class (access via parameter$<function name>, where parameter is an object initialized by initializeParameterObject). Initialize values for selection CSP. File should be of comma-separated with header. Three columns should be of order Amino_acid,Codon,Value

Arguments
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>files</td>
<td>list of files containing starting values. Number of files should equal the number of categories.</td>
</tr>
<tr>
<td>numCategories</td>
<td>number of mutation categories (should be less than or equal to number of mixtures)</td>
</tr>
<tr>
<td>fix</td>
<td>Can use this parameter to fix selection at current values (won’t change over course of MCMC run)</td>
</tr>
</tbody>
</table>

length.Rcpp_Genome

Description
length gives the length of a genome

Usage
```r
## S3 method for class 'Rcpp_Genome'
length(x)
```

Arguments
- x A genome object initialized with initializeGenomeObject.

Value
returns the number of genes in a genome

Examples
```r
genome_file <- system.file("extdata", "genome.fasta", package = "AnaCoDa")

## reading genome
genome <- initializeGenomeObject(file = genome_file)
length(genome) # 10
```
loadMCMCObject

Description

loadMCMCObject creates a new MCMC object and fills it with the information in the file given.

Usage

loadMCMCObject(files)

Arguments

files The filenames where the data will be stored.

Details

This MCMC object is not intended to be used to do another model fitting, only to graph the stored results.

Value

This function has no return value.

Examples

## loading mcmc objects from the filesystem
## Not run:
# load one mcmc object
mcmc <- loadMCMCObject(files = "mcmc.Rda")

# load and combine multiple mcmc objects. Useful when using checkpointing
mcmc <- loadMCMCObject(files = c("mcmc1.Rda", "mcmc2.Rda"))

## End(Not run)

loadParameterObject

Description

loadParameterObject will load a parameter object from the filesystem

Usage

loadParameterObject(files)
Arguments

files A list of parameter filenames to be loaded. If multiple files are given, the parameter objects will be concatenated in the order provided.

Details

The function loads one or multiple files. In the case of multiple file, e.g. due to the use of checkpointing, the files will be concatenated to one parameter object. See `writeParameterObject` for the writing of parameter objects.

Value

Returns an initialized Parameter object.

Examples

```r
## Not run:
# load a single parameter object
parameter <- loadParameterObject("parameter.Rda")

# load and concatenate multiple parameter object
parameter <- loadParameterObject(c("parameter1.Rda", "parameter2.Rda"))

## End(Not run)
```

Description

Plots traces from the model object such as synthesis rates for each gene. Will work regardless of whether or not expression/synthesis rate levels are being estimated. If you wish to plot observed/empirical values, these values MUST be set using the initial.expression.values parameter found in initializeParameterObject. Otherwise, the expression values plotted will just be SCUO values estimated upon initialization of the Parameter object.

Usage

```r
## S3 method for class 'Rcpp_FONSEModel'
plot(
  x, genome, samples = 100, mixture = 1, simulated = FALSE, codon.window = NULL,
  ...
)
```
plot.Rcpp_FONSEParameter

Arguments

- **x**: An Rcpp model object initialized with `initializeModelObject`.
- **genome**: An Rcpp genome object initialized with `initializeGenomeObject`.
- **samples**: The number of samples in the trace.
- **mixture**: The mixture for which to graph values.
- **simulated**: A boolean value that determines whether to use the simulated genome.
- **codon.window**: A boolean value that determines the codon window to use for calculating codon frequencies. If NULL (the default), use complete sequences.
- **...**: Optional, additional arguments. For this function, a possible title for the plot in the form of a list if set with "main".

Value

This function has no return value.

Description

plot graphs the mutation or selection parameter for a ROC or FONSE parameter object for each mixture element.

Usage

```r
## S3 method for class 'Rcpp_FONSEParameter'
plot(
x, what = "Mutation", samples = 100, mixture.name = NULL, with.ci = TRUE, ...
)
```

Arguments

- **x**: A parameter object
- **what**: Which aspect of the parameter to plot. Default value is "Mutation".
- **samples**: Number of samples to plot using the posterior mean. Default value is 100.
- **mixture.name**: A vector with names/descriptions of the mixture distributions in the parameter object.
- **with.ci**: Plot with or without confidence intervals. Default value is TRUE.
- **...**: Arguments to be passed to methods, such as graphical parameters.
plot.Rcpp_ROCModel

Details

Graphs are based off the last # samples for the posterior mean.

Value

This function has no return value.

plot.Rcpp_MCMCAlgorithm

Description

This function will plot the logLikelihood trace, and if the Hmisc package is installed, it will plot a subplot of the logLikelihood trace with the first few samples removed.

Usage

```r
## S3 method for class 'Rcpp_MCMCAlgorithm'
plot(x, what = "LogPosterior", zoom.window = NULL, ...)
```

Arguments

- `x`: An Rcpp_MCMC object initialized with `initializeMCMCObject`.
- `what`: character defining if log(Posterior) (Default) or log(Likelihood) options are: LogPosterior or logLikelihood
- `zoom.window`: A vector describing the start and end of the zoom window.
- `...`: Arguments to be passed to methods, such as graphical parameters.

Value

This function has no return value.

plot.Rcpp_ROCModel

Description

Plots traces from the model object such as synthesis rates for each gene. Will work regardless of whether or not expression/synthesis rate levels are being estimated. If you wish to plot observed/empirical values, these values MUST be set using the initial.expression.values parameter found in initializeParameterObject. Otherwise, the expression values plotted will just be SCUO values estimated upon initialization of the Parameter object.
plot.Rcpp_ROCParameter

Usage

## S3 method for class 'Rcpp_ROCModel'
plot(x, genome = NULL, samples = 100, mixture = 1, simulated = FALSE, ...)

Arguments

x An Rcpp model object initialized with initializeModelObject.

genome An Rcpp genome object initialized with initializeGenomeObject.

samples The number of samples in the trace

mixture The mixture for which to graph values.

simulated A boolean value that determines whether to use the simulated genome.

... Optional, additional arguments. For this function, a possible title for the plot in the form of a list if set with "main".

Value

This function has no return value.

plot.Rcpp_ROCParameter

Plot Parameter

Description

plot graphs the mutation or selection parameter for a ROC or FONSE parameter object for each mixture element.

Usage

## S3 method for class 'Rcpp_ROCParameter'
plot(
  x,
  what = "Mutation",
  samples = 100,
  mixture.name = NULL,
  with.ci = TRUE,
  ...
)

Arguments

x A parameter object

what Which aspect of the parameter to plot. Default value is "Mutation".

samples Number of samples to plot using the posterior mean. Default value is 100.
Plot Trace Object

Description
Plots different traces, specified with the what parameter.

Usage
```r
## S3 method for class 'Rcpp_Trace'
plot(
  x,
  geneIndex = 1,
  mixture = 1,
  log.10.scale = F,
  ...
)
```

Arguments
- `x` An Rcpp trace object initialized with initializeTraceObject.
- `what` A string containing one of the following to graph: Mutation, Selection, Alpha, LambdaPrime, MeanWaitingTime, MixtureProbability, Sphi, Mphi, Aphi, Spesilon, ExpectedPhi, Expression.
- `geneIndex` When plotting expression, the index of the gene to be plotted.
- `mixture` The mixture for which to plot values.
- `log.10.scale` A logical value determining if figures should be plotted on the log.10.scale (default=F). Should not be applied to mutation and selection parameters estimated by ROC/FONSE.
- `...` Optional, additional arguments. For this function, may be a logical value determining if the trace is ROC-based or not.
plotAcceptanceRatios

Description
Plots acceptance ratios for codon-specific parameters. Will be by amino acid for ROC and FONSE models, but will be by codon for PA and PANSE models. Note assumes estimating parameters for all codons.

Usage
plotAcceptanceRatios(trace, main = "CSP Acceptance Ratio Traces")

Arguments
- trace: An Rcpp trace object initialized with initializeTraceObject.
- main: The title of the plot.

Value
This function has no return value.

plotCodonSpecificParameters

Description
Plots a codon-specific set of traces, specified with the type parameter.

Usage
plotCodonSpecificParameters(
  trace,
  mixture,
  type = "Mutation",
  main = "Mutation Parameter Traces",
  ROC.or.FONSE = TRUE,
  log.10.scale = F
)

Value
This function has no return value.
Arguments

- **trace**: An Rcpp trace object initialized with `initializeTraceObject`.
- **mixture**: The mixture for which to plot values.
- **type**: A string containing one of the following to graph: `Mutation`, `Selection`, `Alpha`, `LambdaPrime`, `MeanWaitingTime`, `VarWaitingTime`.
- **main**: The title of the plot.
- **ROC.or.FONSE**: A logical value determining if the Parameter was ROC/FONSE or not.
- **log.10.scale**: A logical value determining if figures should be plotted on the log.10.scale (default=F). Should not be applied to mutation and selection parameters estimated by ROC/FONSE.

Value

This function has no return value.

---

**readPhiValue**

**Description**

Method of Parameter class (access via `parameter$<function name>`, where `parameter` is an object initialized by `initializeParameterObject`). Read synthesis rate values from file. File should be two column file `<gene_id,phi>` and is expected to have a header row.

**Arguments**

- **filename**: name of file to be read

---

**runMCMC**

**Run MCMC**

**Description**

`runMCMC` will run a monte carlo markov chain algorithm for the given mcmc, genome, and model objects to perform a model fitting.

**Usage**

`runMCMC(mcmc, genome, model, ncores = 1, divergence.iteration = 0)`
Arguments

mcmc  MCMC object that will run the model fitting algorithm.
geno me  Genome that the model fitting will run on. Should be the same genome associated with the parameter and model objects.
model  Model to run the fitting on. Should be associated with the given genome.
ncores  Number of cores to perform the model fitting with. Default value is 1.
divergence.iteration  Number of steps that the initial conditions can diverge from the original conditions given. Default value is 0.

Details

runMCMC will run for the number of samples times the number thinning given when the mcmc object is initialized. Updates are provided every 100 steps, and the state of the chain is saved every thinning steps.

Value

This function has no return value.

Examples

# fitting a model to a genome using the runMCMC function
genome_file <- system.file("extdata", "genome.fasta", package = "AnaCoDa")
genome <- initializeGenomeObject(file = genome_file)
sphi_init <- c(1,1)
numMixtures <- 2
geneAssignment <- c(rep(1,floor(length(genome)/2)),rep(2,ceiling(length(genome)/2)))
parameter <- initializeParameterObject(genome = genome, sphi = sphi_init,
num.mixtures = numMixtures,
gene.assignment = geneAssignment,
mixture.definition = "allUnique")
model <- initializeModelObject(parameter = parameter, model = "ROC")
samples <- 2500
thinning <- 50
adaptiveWidth <- 25
mcmc <- initializeMCMCObject(samples = samples, thinning = thinning, adaptive.width=adaptiveWidth, est.expression=TRUE, est.csp=TRUE, est.hyper=TRUE, est.mix = TRUE)
divergence.iteration <- 10

## Not run:
runMCMC(mcmc = mcmc, genome = genome, model = model,
ncores = 4, divergence.iteration = divergence.iteration)

## End(Not run)
setRestartFileSettings

Description
Method of MCMC class (access via mcmc$<function name>, where mcmc is an object initialized by initializeMCMCObject). Set restart file output name and frequency prior to running MCMC.

Arguments
- filename: name of restart file
- interval: number of samples (i.e. iterations * thinning) between writing new restart file
- multiple: if true, will output a new restart file at each interval (file name will include sample it was written at)
setRestartSettings

**Description**

setRestartSettings sets the needed information (what the file is called, how often the file should be written) to write information to restart the MCMC algorithm from a given point.

**Usage**

```r
setRestartSettings(mcmc, filename, samples, write.multiple = TRUE)
```

**Arguments**

- `mcmc` MCMC object that will run the model fitting algorithm.
- `filename` Filename for the restart files to be written.
- `samples` Number of samples that should occur before a file is written.
- `write.multiple` Boolean that determines if multiple restart files are written. Default value is TRUE.

**Details**

setRestartSettings writes a restart file every set amount of samples that occur. Also, if write.multiple is true, instead of overwriting the previous restart file, the sample number is prepended onto the file name and multiple restart files are generated for a run.

**Value**

This function has no return value.

**Examples**

```r
## set restart settings for checkpointing

samples <- 2500
thinning <- 50
adaptiveWidth <- 25

## estimate all parameter types
mcmc <- initializeMCMCObject(samples = samples, thinning = thinning,
    adaptive.width=adaptiveWidth, est.expression=TRUE,
    est.csp=TRUE, est.hyper=TRUE, est.mix = TRUE)

# prompts the mcmc to write a restart file every 100 samples during the run.
setRestartSettings(mcmc = mcmc, filename = "test_restart", samples = 100)

# prompts the mcmc to write a restart file every 100 samples during the run,
```
Description

Method of MCMC class (access via `mcmc$<function name>`, where `mcmc` is an object initialized by `initializeMCMCObject`). Set number of samples set for MCMCAlgorithm object.

Arguments

- `samples`: positive value

Description

Method of MCMC class (access via `mcmc$<function name>`, where `mcmc` is an object initialized by `initializeMCMCObject`). Set number of iterations (total iterations = `samples * thinning`) to allow proposal widths to adapt.

Arguments

- `steps`: a positive value

Description

Set thinning value, which is the number of iterations (total iterations = `samples * thinning`) not being kept.

Arguments

- `thinning`: positive value

```
# but will overwrite it each time.
setRestartSettings(mcmc = mcmc, filename = "test_restart", samples = 100,
                    write.multiple = FALSE)
```
simulateGenome

**Description**

Method of Model class (access via model$<function name>, where model is an object initialized by initializeModelObject). Will simulate a version of the given genome using the current set of parameters stored in the Parameter object. This can be written to a FASTA file using genome$writeFasta(<filename>, simulated = TRUE).

**Arguments**

- **genome**
  - a Genome object initialized by initializeGenomeObject

summary.Rcpp_Genome  

**Summary of Genome**

**Description**

summary summarizes the description of a genome, such as number of genes and average gene length.

**Usage**

```r
## S3 method for class 'Rcpp_Genome'
summary(object, ...)
```

**Arguments**

- **object**
  - A genome object initialized with initializeGenomeObject.
- **...**
  - Optional, additional arguments to be passed to the main summary function that affect the summary produced.

**Value**

This function returns by default an object of class c("summaryDefault", table).
writeMCMCObject  Write MCMC Object

Description

writeMCMCObject stores the MCMC information from the model fitting run in a file.

Usage

writeMCMCObject(mcmc, file)

Arguments

mcmc MCMC object that has run the model fitting algorithm.
file A filename where the data will be stored.

Value

This function has no return value.

Examples

## saving the MCMC object after model fitting
genome_file <- system.file("extdata", "genome.fasta", package = "AnaCoDa")

genome <- initializeGenomeObject(file = genome_file)
sphi_init <- c(1,1)
numMixtures <- 2
geneAssignment <- c(rep(1,floor(length(genome)/2)),rep(2,ceiling(length(genome)/2)))
parameter <- initializeParameterObject(genome = genome, sphi = sphi_init,
    num.mixtures = numMixtures,
    gene.assignment = geneAssignment,
    mixture.definition = "allUnique")
samples <- 2500
thinning <- 50
adaptiveWidth <- 25
mcmc <- initializeMCMCObject(samples = samples, thinning = thinning,
    adaptive.width=adaptiveWidth, est.expression=TRUE,
    est.csp=TRUE, est.hyper=TRUE, est.mix = TRUE)
divergence.iteration <- 10
## Not run:
runMCMC(mcmc = mcmc, genome = genome, model = model,
    ncores = 4, divergence.iteration = divergence.iteration)
writeMCMCObject(mcmc = mcmc, file = file.path(tempdir(), "file.Rda"))

## End(Not run)
writeParameterObject Write Parameter Object to a File

Description

writeParameterObject will write the parameter object as binary to the filesystem

Usage

writeParameterObject(parameter, file)

Arguments

- parameter: parameter on object created by initializeParameterObject.
- file: A filename that where the data will be stored.

Details

As Rcpp object are not serializable with the default R save function, therefore this custom save function is provided (see loadParameterObject).

Value

This function has no return value.

Examples

```r
## Not run:

genome_file <- system.file("extdata", "genome.fasta", package = "AnaCoDa")

geno <- initializeGenomeObject(file = genome_file)
sphi_init <- c(1,1)
numMixtures <- 2
geneAssignment <- c(rep(1,floor(length(genome)/2)),rep(2,ceiling(length(genome)/2)))
parameter <- initializeParameterObject(genome = genome, sphi = sphi_init,
  num.mixtures = numMixtures,
  gene.assignment = geneAssignment,
  mixture.definition = "allUnique")

## writing an empty parameter object as the runMCMC routine was not called yet
writeParameterObject(parameter = parameter, file = file.path(tempdir(), "file.Rda"))

## End(Not run)
```
Index

AAToCodon, 4, 10
acfCSP, 4, 5
acfMCMC, 5, 5
addObservedSynthesisRateSet, 6
aminoAcids, 7, 9

calculateMarginalLogLikelihood, 7
calculateSCUO, 8
codons, 7, 9
codonToAA, 4, 9
convergence.test, 10

findOptimalCodon, 12
fixDEta, 13
fixDM, 13
fixSphi, 13

geomMean, 13
getAdaptiveWidth, 14
getCAI, 15
getCAIWeights, 15
getCodonCounts, 16
getCodonCountsForAA, 17
getCodonSpecificPosteriorMeanForCodon, 18
getCodonSpecificPosteriorVarianceForCodon, 18
getCodonSpecificQuantilesForCodon, 19
getCSPEstimates, 20
getEstimatedMixtureAssignmentForGene, 21
getEstimatedMixtureAssignmentProbabilitiesForGene, 22
getExpressionEstimates, 22
getGroupList, 24
getLogLikelihoodTrace, 24
getLogPosteriorMean, 24
getLogPosteriorTrace, 25
getMixtureAssignmentEstimate, 25
getNames, 26
getNc, 27
getNcAA, 28
getNoiseOffsetPosteriorMean, 28
getNoiseOffsetVariance, 29
getSelectedObservedSynthesisRateSet, 29
getSamples, 30
getSelectionCoefficients, 30
getStdDevSynthesisRatePosteriorMean, 31
getStdDevSynthesisRateVariance, 32
getStepsToAdapt, 32
getSynthesisRate, 33
getSynthesisRatePosteriorMeanForGene, 33
getSynthesisRatePosteriorVarianceForGene, 34
getThinning, 34
getTrace, 35
getTraceObject, 35

initializeCovarianceMatrices, 36
initializeGenomeObject, 6, 8, 15, 16,
27–30, 37, 46, 59
initializeMCMCObject, 38
initializeModelObject, 39
initializeParameterObject, 41
initializeSynthesisRateByGenome, 44
initializeSynthesisRateByList, 44
initializeSynthesisRateByRandom, 45
initMutationCategories, 45
initSelectionCategories, 46

length.Rcpp_Genome, 46
loadMCMCObject, 47
loadParameterObject, 47, 61

plot.Rcpp_FONSEModel, 48
plot.Rcpp_FONSEParameter, 49
plot.Rcpp_MCMCAlgorithm, 50
plot.Rcpp_ROCModel, 50
INDEX

plot.Rcpp_ROCParameter, 51
plot.Rcpp_Trace, 52
plotAcceptanceRatios, 53
plotCodonSpecificParameters, 53
readPhiValue, 54
runMCMC, 54

setAdaptiveWidth, 56
setGroupList, 56
setRestartFileSettings, 56
setRestartSettings, 57
setSamples, 58
setStepsToAdapt, 58
setThinning, 58
simulateGenome, 59
summary.Rcpp_Genome, 59
writeMCMCObject, 60
writeParameterObject, 48, 61