Package ‘AnaCoDa’

September 14, 2018

Type Package
Title Analysis of Codon Data under Stationarity using a Bayesian Framework
Version 0.1.2
Date 2018-08-16
Maintainer Cedric Landerer <cedric.landerer@gmail.com>
URL https://github.com/clandere/AncDa
VignetteBuilder knitr
NeedsCompilation yes
Depends R (>= 3.3.0), Rcpp (>= 0.11.3), methods
Suggests knitr, Hmisc, VGAM, coda, testthat, lmodel2
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 In addition 'AnaCoDa' contains three currently unpublished models.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 6.0.1
Author Cedric Landerer [aut, cre],
            Gabriel Hanas [ctb],
            Jeremy Rogers [ctb],
            Alex Cope [ctb],
            Denizhan Pak [ctb]
Repository CRAN
Date/Publication 2018-09-14 16:10:11 UTC

R topics documented:

AAToCodon ....................................................... 3
acfCSP .......................................................... 4
acfMCMC .......................................................... 4
addObservedSynthesisRateSet .................................. 5
aminoAcids ....................................................... 6
calculateSCUO ................................................... 6
codons ............................................................. 7
codonToAA ........................................................ 7
convergence.test ................................................ 8
findOptimalCodon ................................................ 9
gemMean .......................................................... 10
getCAI ............................................................. 11
getCAIweights ................................................... 12
getCodonCounts .................................................. 13
getCodonCountsForAA ........................................... 13
getCSPEstimates ................................................ 14
getExpressionEstimates ........................................ 16
getMixtureAssignmentEstimate .................................. 17
getNames ......................................................... 18
getNc ............................................................... 19
getNcAA ............................................................ 20
getObservedSynthesisRateSet ................................... 20
getSelectionCoefficients ....................................... 21
getTrace ........................................................ 22
initializeCovarianceMatrices .................................. 23
initializeGenomeObject ........................................ 24
initializeMCMCOBJECT .......................................... 25
initializeModelObject .......................................... 26
initializeParameterObject ...................................... 27
length.Rcpp_Genome ............................................ 30
loadMCMCOBJECT ............................................... 31
loadParameterObject ............................................ 31
plot.Rcpp_FONSEModel .......................................... 32
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 first (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)

Arguments

- parameter: object of class Parameter
- csp: "Selection" or "Mutation", defaults to "Mutation"
- numMixtures: indicates the number of CSP mixtures used
- samples: number of samples at the end of the trace used to calculate the acf
- lag.max: Maximum amount of lag to calculate acf. Default is 10*log10(N), where N i the number of observations.
- plot: logical. If TRUE (default) a plot of the acf is created

See Also

acfMCMC

acfMCMC  

Autocorrelation function for the likelihood or posterior trace

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

- mcmc: object of class MCMC
- type: "LogPosterior" or "LogLikelihood", defaults to "LogPosterior"
- samples: number of samples at the end of the trace used to calculate the acf
- lag.max: Maximum amount of lag to calculate acf. Default is 10*log10(N), where N i the number of observations.
- plot: logical. If TRUE (default) a plot of the acf is created
addObservedSynthesisRateSet

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

geno me 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")
extpression_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)
aminoAcids  Amino acids

**Description**
Returns a vector of all amino acids

**Usage**
aminoAcids()

**Value**
Returns a vector of all amino acids

**See Also**
codons

calculateSCUO  calculates the synonymous codon usage order (SCUO)

**Description**
calculateSCUO calculates the SCUO value for each gene in genome

**Usage**
calculateSCUO(genome)

**Arguments**
genome  A genome object initialized with `initializeGenomeObject`.

**Value**
returns the SCUO value for each gene in genome

**Examples**

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

# reading genome
genome <- initializeGenomeObject(file = genome_file)
scurso <- calculateSCUO(genome)
```
**Description**

Returns a vector of all codons

**Usage**

codons()

**Value**

Returns a vector of all codons

**See Also**

aminoAcids

codonToAA (translates codon to amino acid)

**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

**Description**

Convergence Test

**Usage**

```r
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)
- `frac1`: fraction to use from beginning of chain
- `frac2`: fraction to use from end of chain
- `thin`: the thinning interval between consecutive observations
- `plot`: (logical) plot result instead of returning an object
- `what`: Character describing which trace should be tested for convergence (only for Trace object). Valid options are Mutation, Selection, MixtureProbability, Sphi, Mphi, ExpectedPhi, or Expression
- `mixture`: Integer determining for which mixture distribution the convergence test should be applied (only for trace object).

**Value**

geweke score object

**Examples**

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

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

genome <- initializeGenomeObject(file = genome_file)
sphi_init <- c(1,1)
numMixtures <- 2
geneAssignment <- sample(1:2, length(genome), replace = TRUE) # random assignment to mixtures
parameter <- initializeParameterObject(genome = genome, sphi = sphi_init,
                                       numMixtures = numMixtures,
                                       gene.assignment = geneAssignment,
                                       mixture.definition = "allUnique")
```
findOptimalCodon

Find and return list of optimal codons

Description
findOptimalCodon extracts the optimal codon for each amino acid.

Usage
findOptimalCodon(csp)

Arguments
csp a data.frame as returned by getCSPEstimates.

Value
A named list with with optimal codons for each amino acid.

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

genoome <- initializeGenomeObject(file = genome_file)
sphi_init <- 1
numMixtures <- 1
geneAssignment <- rep(1, length(genome))
geomMean

Take the geometric mean of a vector

Description

geomMean will calculate the geometric mean of a list of numerical values.

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.
getCAI

Value

Returns the geometric mean of a vector.

Examples

```r
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)
```

getCAI  

_Calculate the Codon Adaptation Index_

Description

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

Usage

```r
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)
```
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)

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

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.

**Examples**

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

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

**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)
getCSPEstimates

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 <- getCodonCountsForAA("A", genome)
```

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

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

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.
getCSPEstimates

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.

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

Examples

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

genome <- initializeGenomeObject(file = genome_file)
sphi_init <- c(1,1)
numMixtures <- 2
geneAssignment <- sample(1:2, length(genome), replace = TRUE) # random assignment to mixtures
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)

## 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)
getExpressionEstimates

Returns the estimated phi posterior for a gene

Description

Posterior estimates for the phi value of specified genes

Usage

getExpressionEstimates(parameter, gene.index, samples, quantiles = c(0.025, 0.975))

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))

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 <- sample(1:2, length(genome), replace = TRUE) # random assignment to mixtures
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,
```

getMixtureAssignmentEstimate

```r
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)
```

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 <- sample(1:2, length(genome), replace = TRUE) # random assignment to mixtures
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 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

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.

to.return.all.gene.ids Returns a named vector with the Effective Number of Codons for each gene.

Examples

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

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

## return all gene ids for the genome
geneIDs <- getNames(genome, FALSE)

getNc <- getNc(genome)

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

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

## reading genome
genome <- initializeGenomeObject(file = gene_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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>genome</td>
<td>A genome object initialized with initializeGenomeObject.</td>
</tr>
</tbody>
</table>

**Value**

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

**Examples**

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

**getObserveSynthesisRateSet**

*Get gene observed synthesis rates*

**Description**

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

**Usage**

getObserveSynthesisRateSet(genome, simulated = FALSE)

**Arguments**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>genome</td>
<td>A genome object initialized with initializeGenomeObject.</td>
</tr>
<tr>
<td>simulated</td>
<td>A logical value denoting if the synthesis rates to be listed are simulated or not. The default value is FALSE.</td>
</tr>
</tbody>
</table>
Value

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

Examples

```r
# 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)
```

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

```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.
extpressionValues <- getObservedSynthesisRateSet(genome = genome)

```

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 <- sample(1:2, length(genome), replace = TRUE) # random assignment to mixtures
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

initializeCovarianceMatrices

Initialize Covariance Matrices

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

initializeGenomeObject(file, genome = NULL, observed.expression.file = NULL, 
fasta = TRUE, simulated = 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 A string containing the location of a file containing empirical expression rates (optional).

fasta A boolean value which decides whether to initialize with a fasta file or an RFP-Data file. (TRUE for fasta, FALSE for RFPData)

simulated boolean to determine if the data should be treated as a simulated data set (Default = FALSE).

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.

append If TRUE (FALSE is default), 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).

Value

This function returns the initialized Genome object.

Examples

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

```r
# 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)
```

### initializeMCMCObject

**Initialize MCMC**

**Description**

initializeMCMCObject initializes a MCMC object to perform a model fitting for a parameter and model object.

**Usage**

```r
initializeMCMCObject(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 samples.
- `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.
- `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.

Usage

```r
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.
- **fix.observation.noise**: (ROC only) Allows to fix the noise in the observed expression dataset to the initial condition. The initial condition for the observed expression noise can be set in the parameter object.
initializeParameterObject

rfp.count.column

(PA and PANSE only) A number representing the RFP count column to use.

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 <- sample(1:2, length(genome), replace = TRUE) # random assignment to mixtures
parameter <- initializeParameterObject(genome = genome, sphi = sphi_init,
                                       num.mixtures = numMixtures,
                                       gene.assignment = geneAssignment,
                                       mixture.definition = "allUnique")

# 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.
initializeParameterObject

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.sd = 0.35, init.csp.variance = 0.0025,
init.sepsilon = 0.1, init.w.obs.phi = FALSE)

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 intial phi values. The length of the vector has to equal the number of genes in the Genome object. 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.sd
Controlling the standard deviation of the normal prior on the mutation parameters

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
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.

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

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

gene <- initializeGenomeObject(file = genome_file)

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

## re-initialize a parameter object from a restart file. Useful for checkpointing
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 <- sample(1:2, length(genome), replace = TRUE) # random assignment to mixtures
parameter <- initializeParameterObject(genome = genome, sphi = c(0.5, 2), num.mixtures = 2,
                                     gene.assignment = geneAssignment,
                                     mixture.definition.matrix = def.matrix)

---

### length.Rcpp_Genome

#### Length of 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  

Load MCMC Object

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  

Load Parameter Object

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 check pointing, 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, ...)
```
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.
...

Value

This function has no return value.

Description

Plot Parameter

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
...

Details

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

Value

This function has no return value.
plot.Rcpp_MCMCAlgorithm

Plot MCMC algorithm

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

Plot Model Object

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_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**

```r
## 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.
- **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.

**Details**

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

**Value**

This function has no return value.
plot.Rcpp_Trace  
**Plot Trace Object**

**Description**

Plots different traces, specified with the what parameter.

**Usage**

```r
## S3 method for class 'Rcpp_Trace'
```

**Arguments**

- `x` An Rcpp trace object initialized with `initializeTraceObject`.
- `what` A string containing one of the following to graph: Mutation, Selection, Alpha, LambdaPrime, Mean, MixtureProbability, Sphi, Mphi, Aphi, Sepsilon, ExpectedPhi, Expression.
- `geneIndex` When plotting expression, the index of the gene to be plotted.
- `mixture` The mixture for which to plot values.
- `...` Optional, additional arguments. For this function, may be a logical value determining if the trace is ROC-based or not.

**Value**

This function has no return value.

---

plotCodonSpecificHyperParameters  
**Plot Codon Specific Hyper Parameter**

**Description**

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

**Usage**

```r
plotCodonSpecificHyperParameters(trace, mixture, type = "RandomNumber", main = "Random Number Parameter Traces", PA = TRUE)
```
plotCodonSpecificParameters

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>trace</td>
<td>An Rcpp trace object initialized with initializeTraceObject.</td>
</tr>
<tr>
<td>mixture</td>
<td>The mixture for which to plot values.</td>
</tr>
<tr>
<td>type</td>
<td>A string containing one of the following to graph: Random, LoglikelihoodRatio, currLoglikelihoodL, propLoglikelihoodL, currLoglikelihoodadjustedL, propLoglikelihoodadjustedL.</td>
</tr>
<tr>
<td>main</td>
<td>The title of the plot.</td>
</tr>
<tr>
<td>PA</td>
<td>A logical value determining if the Parameter was PA or not.</td>
</tr>
</tbody>
</table>

Value

This function has no return value.

---

plotCodonSpecificParameters

*Plot Codon Specific Parameter*

Description

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

Usage

```r
plotCodonSpecificParameters(trace, mixture, type = "Mutation", main = "Mutation Parameter Traces", ROC = TRUE)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>trace</td>
<td>An Rcpp trace object initialized with initializeTraceObject.</td>
</tr>
<tr>
<td>mixture</td>
<td>The mixture for which to plot values.</td>
</tr>
<tr>
<td>type</td>
<td>A string containing one of the following to graph: Mutation, Selection, Alpha, LambdaPrime, Mean.</td>
</tr>
<tr>
<td>main</td>
<td>The title of the plot.</td>
</tr>
<tr>
<td>ROC</td>
<td>A logical value determining if the Parameter was ROC or not.</td>
</tr>
</tbody>
</table>

Value

This function has no return value.
runMCMC

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.
- genome: 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

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

genome <- initializeGenomeObject(file = gene_file)
sphi_init <- c(1,1)
numMixtures <- 2
geneAssignment <- sample(1:2, length(genome), replace = TRUE) # random assignment to mixtures
parameter <- initializeParameterObject(genome = genome, sphi = sphi_init, numMixtures = numMixtures, gene.assignment = geneAssignment, mixture.definition = "allUnique")
setRestartSettings

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)

---

setRestartSettings  Set Restart Settings

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

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

---

**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 <- sample(1:2, length(genome), replace = TRUE) # random assignment to mixtures
parameter <- initializeParameterObject(genome = genome, sphi = sphi_init,
    numMixtures = 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"))
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

## Not run:

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

genome <- initializeGenomeObject(file = genome_file)
sphi_init <- c(1,1)
numMixtures <- 2
geneAssignment <- sample(1:2, length(genome), replace = TRUE) # random assignment to mixtures
parameter <- initializeParameterObject(genome = genome, sphi = sphi_init,
    numMixtures = 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, 3, 7
acfCSP, 4, 5
acfMCMC, 4, 4
addObservedSynthesisRateSet, 5
aminoAcids, 6, 7
calculateSCUO, 6
codons, 6, 7
codonToAA, 3, 7
convergence.test, 8
findOptimalCodon, 9
gemMean, 10
getCAI, 11
getCAIWeights, 12
getCodonCounts, 13
getCodonCountsForAA, 13
getCSPEstimates, 14
getExpressionEstimates, 16
getMixtureAssignmentEstimate, 17
getNames, 18
getNc, 19
getNCAA, 20
getObservedSynthesisRateSet, 20
getSelectionCoefficients, 21
getTrace, 22
initializeCovarianceMatrices, 23
initializeGenomeObject, 5, 6, 11, 12, 19–21, 24, 30, 40
initializeMCMCObject, 25
initializeModelObject, 26
initializeParameterObject, 27
length.Rcpp_Genome, 30
loadMCMCObject, 31
loadParameterObject, 31, 42
plot.Rcpp_MCMCAlogorithm, 34
plot.Rcpp_ROCModel, 34
plot.Rcpp_ROCParameter, 35
plot.Rcpp_Trace, 36
plotCodonSpecificHyperParameters, 36
plotCodonSpecificParameters, 37
runMCMC, 38
setRestartSettings, 39
summary.Rcpp_Genome, 40
writeMCMCObject, 41
writeParameterObject, 32, 42

43