Package ‘jaatha’

February 20, 2015

Version 2.7.0
Date 2014-11-26
License GPL (>= 3)
Title A Fast Parameter Estimation Method for Evolutionary Biology
BugReports https://github.com/paulstaab/jaatha
Description Jaatha is a composite likelihood method for inferring evolutionary parameters using genetic data. Given a model of the evolutionary history of two biological populations as well as SNP data from multiple individuals from each population, it estimates model parameters like the time of separation of both species.
URL http://evol.bio.lmu.de/_statgen/software/jaatha
Depends R (>= 3.0), methods
Imports phyclust (>= 0.1-14), Rcpp (>= 0.11.0), reshape2 (>= 1.4), parallel
Suggests ape (>= 2.7), testthat
LinkingTo Rcpp
Collate 'RcppExports.R' 'block.R' 'confidence_intervals.R'
  'estimate_log_likelihood.R' 'find_best_par_in_block.R'
  'fit_glm.R' 'helper_functions.R' 'initial_search.R' 'jaatha.R'
  'normalize.R' 'package_info.R' 'parallelization.R'
  'read_fasta.R' 'refined_search.R' 'run_simulations.R'
  'sim_likelyhood.R' 'sim_program_ms.R' 'sim_program_msms.R'
  'sim_program_seqgen.R' 'simulate_within_block.R'
  'sum_stat_fpc.R' 'sum_stat_generate.R' 'sum_stat_jsfs.R'
  'sum_stat_pmc.R'
Author Paul Staab [aut, cre, cph],
  Lisha Mathew [aut, cph],
  Dirk Metzler [aut, cph, ths]
Maintainer Paul Staab <develop@paulstaab.de>
NeedsCompilation yes
Repository  CRAN
Date/Publication  2014-11-26 18:09:02

R topics documented:

jaatha-package .................................................. 3
addFeature ...................................................... 3
calcalcJsfs .......................................................... 4
calculateJsfs ...................................................... 5
checkType .......................................................... 5
convertSimResultsToDataFrame .................................. 6
denormalize ....................................................... 7
dm.addGrowth ...................................................... 7
dm.addMigration ................................................... 8
dm.addMutation .................................................... 10
dm.addMutationRateHeterogenity ................................ 11
dm.addOutgroup ................................................... 12
dm.addParameter ................................................... 13
dm.addRecombination ............................................. 14
dm.addSampleSize .................................................. 15
dm.addSizeChange ................................................. 15
dm.addSpeciationEvent .......................................... 16
dm.addSummaryStatistic ......................................... 18
dm.addSymmetricMigration ...................................... 18
dm.createDemographicModel ..................................... 19
dm.createThetaTauModel ......................................... 20
dm.getGroups ...................................................... 21
dm.getLociLength .................................................. 21
dm.getLociNumber .................................................. 22
dm.setLociLength .................................................. 22
dm.setLociNumber .................................................. 23
dm.setMutationModel ............................................. 23
dm.simSumStats ..................................................... 24
estimateLogLikelihood ............................................ 25
findBestParInBlock ................................................. 25
fitGlm ............................................................. 26
fitGlmPoiTransformed ............................................ 26
fitPoiSmoothed ..................................................... 27
Jaatha-class ......................................................... 27
Jaatha.confidenceIntervals ...................................... 28
Jaatha.getClfsFromLogs .......................................... 29
Jaatha.getLikelihoods ............................................. 29
Jaatha.getStartingPoints ....................................... 30
Jaatha.initialize ................................................... 30
Jaatha.initialSearch .............................................. 31
Jaatha.refinedSearch ............................................. 32
Jaatha.setSeqgenExecutable .................................... 33
Description

Fast estimation of demographic parameters

Details

Jaatha is a composite maximum likelihood method to estimate parameters of a speciation model of closely related biological species out of SNP data.

Author(s)

Lisha Naduvilezhath <lisha (at) biologie.uni-muenchen.de>, Paul R. Staab <staab (at) biologie.uni-muenchen.de> and Dirk Metzler <metzler (at) biologie.uni-muenchen.de>

Maintainer: Paul R. Staab <staab (at) biologie.uni-muenchen.de>

addFeature

Low level function for adding a new feature to a demographic Model

Description

Use this function to add a feature to a dm if there is no higher level "dm.add*"-function available.

Usage

addFeature(dm, type, parameter = NA, lower.range = NA, upper.range = NA, fixed.value = NA, par.new = T, pop.source = NA, pop.sink = NA, time.point = NA, group = 0, variance = 0, zero.inflation = 0)
Arguments

- **dm**: The demographic model to which the feature will be added
- **type**: The type of the feature coded as a character
- **parameter**: Either the name of the parameter (par.new=TRUE), or an R expression possibly containing one or more previously created parameter names.
- **lower.range**: The lower boundary for the value of the parameter
- **upper.range**: The upper boundary for the value of the parameter
- **fixed.value**: If given, the parameter will be set to a fixed value. This is equivalent to setting lower.range equal to upper.range.
- **par.new**: If ‘TRUE’ a new parameter will be created using the arguments ‘lower.range’ and ‘upper.range’ or ‘fixed.value’. It will be named ‘parameter’. If ‘FALSE’ the argument ‘parameter’ will be evaluated instead.
- **pop.source**: The source population if available (think e.g. of migration)
- **pop.sink**: The target or "sink" population if available (think e.g. of migration)
- **time.point**: Normally the point in backwards time where the feature starts.
- **group**: For genomic features, different groups can be created.
- **variance**: Set to a value different from 0 to introduce variation in the the parameter value for different loci. The variation follows a gamma distribution with mean equal to the value provided as parameter, and variance as given here. Can also be set to a previously created parameter, or an expression based on parameters.
- **zero.inflation**: If used, a zero inflated gamma distribution is used to model the variability between loci. This parameter should evaluate to the percent of loci for which the parameter is 0. The values for all other loci will be drawn from the discretized gamma distribution.

Value

The extended demographic model.

calcJsfs

*Calculate the JSFS from a list of segregating sites statistics*

Description

Calculate the JSFS from a list of segregating sites statistics

Usage

calcJsfs(seg_sites, sample_size)

Arguments

- **seg_sites**: List of segregating sites
- **sample_size**: A numeric vector of size 2, giving the sample sizes of the two population for which the JSFS is calculated
Value

The Joint Site Frequency Spectrum, as a matrix.

Description

The function 'read.dna' of package 'ape' allows you to import DNA data from various formats into R. This function calculates the JSFS of such imported data which can then be used as input for a Jaatha search. Please note that the data must be aligned.

Usage

calculateJsfs(ape.data, pop1.rows, pop2.rows, outgroup.rows = NA)

Arguments

data <- DNA sequence data of multiple individuals from two populations and an optional outgroup sequence. It should be in a format as returned by ape's 'read.dna' function. Please refer to ape's manual for further details.

pop1.rows <- A numeric vector indicating which individuals of your dataset belong to the first population. The individuals are reveres by their position in the dataset/their row number in 'ape.data'.

pop2.rows <- Same as 'pop1.rows', but for the individuals of the second population.

outgroup.rows <- Same as 'pop1.rows', but for the individuals of the outgroup (if any). The outgroup can consist of more than one individual to account for ancestral misidentification. In this case, only positions in which all outgroup sequences are identically are considered. If no outgroup is given a folded JSFS is calculated.

Value

The calculated JSFS, as matrix.

checkType

Checks if a variable is of a given type and calls stop() on type mismatch.

Description

See heading. Side-effect warning: Calls stop() on type mismatch.

Usage

checkType(variable, type, required = T, allow.na = T)
**convertSimResultsToDataFrame**

Converts simulation results into a data frame that is usable for fitting a glm.

### Arguments

- **variable**
  - the variable to check

- **type**
  - the name of the type the variable should have. Can be num/numeric, vec/vector mat/matrix or fun/function. Can also be s/single, in that case it must be a vector of length one. If a vector of type names is given, the variable must be of all types.

- **required**
  - A boolean that indicates whether the variable must be specified or can be missing. Value of NULL also counts as missing.

- **allow.na**
  - If FALSE, an error is returned if the variable contains NA values. Only works for vectors.

### Value

nothing

---

**convertSimResultsToDataFrame**

Converts simulation results into a data frame that is usable for fitting a glm.

### Description

Currently only works with nx2 matrix summary statistics.

### Usage

convertSimResultsToDataFrame(sim.data, sum.stat, mask = NULL)

### Arguments

- **sim.data**
  - Results from simulations

- **sum.stat**
  - Name of the summary statistics which should get converted

- **mask**
  - Boolean vector of positions to exclude in the data.frame

### Value

The summary statistics as data.frame
denormalize

Convert parameters from Jaatha’s internal scale into their natural scale

Description

Convert parameters from Jaatha’s internal scale into their natural scale

Usage

denormalize(value, jaatha)

Arguments

value A parameter combination in Jaatha’s internal scale
jaatha The current jaatha object

Value

The parameter from value, converted into their natural scale

dm.addGrowth Adds an growth or decline of the population size of one population to a model.

Description

This function changes the growth factor of a population at given point in time (`at.time`). This factor than applies to the time interval farther into the past from this point.

Usage

dm.addGrowth(dm, min.growth.rate, max.growth.rate, fixed.growth.rate, par.new = T, new.par.name = “alpha”, parameter, population, at.time = “0”)
fixed.growth.rate
If specified, the growth rate will not be estimated, but assumed to have the given value.

par.new
If 'TRUE' a new parameter will be created using the arguments 'min.growth.rate' and 'max.growth.rate' or 'fixed.growth.rate'. It will be named 'new.par.name'
If 'FALSE' the argument 'parameter' will be evaluated instead.

new.par.name
Name for the new parameter.

parameter
Instead of creating a new parameter, you can also set the mutation rate to an expression based on existing parameters. For example setting this to "alpha" will use an parameter with name tau that you have previously created. You can also use R expression here, i.e. "2*alpha" or "5*M+2*alpha" (if M is another parameter) will also work (also the latter does not make much sense).

population
The number of the population in which the spilt occurs. See dm.addSpeciationEvent for more information.

at.time
The time point at which the size changes.

Details
The population size changes by factor exp(-alpha*t), where alpha is the growth parameter and t is the time since the growth has started. Hence, for positive alpha, the population will 'decline backwards in time' or grow forwards in time. Similar, will decline in forwards time for a negative value of alpha.

If you want to add an instantaneous change of the population size, then use the dm.addSizeChange function.

Value
The demographic model with a size change.

Examples

# A model with one smaller population
dm <- dm.createThetaTauModel(c(20,37), 88)
dm <- dm.addGrowth(dm, 0.1, 2, population=2, at.time="0")

dm.addMigration
Add migration/gene flow between two populations to a demographic model

Description
This function adds the assumption to the model that some individuals 'migrate' from one sub-population to another, i.e. they leave the one and become a member of the other. This is usually used to model ongoing gene flow through hybridisation after the populations separated.
**Usage**

```r
dm.addMigration(dm, lower.range, upper.range, fixed.value, pop.from, pop.to,
    time.start = "0", par.new = T, new.par.name = "M", parameter)
```

**Arguments**

- **dm**: The demographic model to which recombination events should be added.
- **lower.range**: If you want to estimate the migration parameter (see note
- **upper.range**: Same as lower.range, but the largest possible value.
- **fixed.value**: If specified, the migration rate will not be estimated, but assumed to have the given value.
- **pop.from**: The population from which the individuals leave.
- **pop.to**: The population to which the individuals move.
- **time.start**: The time point at which the migration with this rate starts.
- **par.new**: If ‘TRUE’ a new parameter will be created using the arguments ‘lower.range’ and ‘upper.range’ or ‘fixed.value’. It will be named ‘new.par.name’. If ‘FALSE’ the argument ‘parameter’ will be evaluated instead.
- **new.par.name**: The name for the new parameter.
- **parameter**: Instead of creating a new parameter, you can also set the mutation rate to an expression based on existing parameters. For example setting this to "M" will use an parameter with name M that you have previously created. You can also use R expression here, so "2*M" or "5*M+2*tau" (if tau is another parameter) will also work (also this does not make much sense).

**Details**

You can enter a time (‘time.start’) at which the migration is assumed to start (looking backwards in time). From that time on, a fixed number of migrants move from population 'pop.from' to population 'pop.to' each generation. This number is given via this feature’s parameter, which equals 4*N0*m, where m is the fraction of 'pop.to' that is replaced with migrants each generation. If 'pop.to' has also size Ne, than this is just the expected number of individuals that migrate each generation.

You can add different mutation rates at different times to your model. Then each rate will be used for the period from its time point to the next. Migration from and to an population always ends with the speciation event in which the population is created.

**Value**

The demographic model with migration

**Examples**

```r
# Constant asymmetric migration
dm <- dm.createThetaTauModel(c(25,25), 100)
dm <- dm.addMigration(dm, 0.01, 5, pop.from=1, pop.to=2, time.start="0")
```
# Stepwise decreasing mutation

```r
dm <- dm.createThetaTauModel(c(25,25), 100)
dm <- dm.addMigration(dm, 0.01, 5, pop.from=1, pop.to=2, new.par.name="M",
                      time.start="0")
dm <- dm.addMigration(dm, pop.from=1, pop.to=2, par.new=FALSE,
                      parameter="0.5*M", time.start="0.5*tau")
```

---

**dm.addMutation**  
*Adds mutations to a demographic model*

**Description**

This function adds the assumption to the model that neutral mutations occur in the genomes at a constant rate. The rate is quantified through a parameter usually named theta in population genetics. It equals $4N_0\mu$, where $N_0$ is the effective diploid population size of population one at the time of sampling and $\mu$ is the neutral mutation rate for an entire locus.

**Usage**

```r
dm.addMutation(dm, lower.range, upper.range, fixed.value, par.new = T,
               new.par.name = "theta", parameter, group = 0, variance = 0)
```

**Arguments**

- **dm**: The demographic model to which mutations should be added
- **lower.range**: If you want to estimate the mutation rate, this will be used as the smallest possible value.
- **upper.range**: If you want to estimate the mutation rate, this will be used as the largest possible value.
- **fixed.value**: If specified, the mutation rate will not be estimated, but assumed to be fixed at the given value.
- **par.new**: If 'TRUE' a new parameter will be created using the arguments 'lower.range' and 'upper.range' or 'fixed.value'. It will be named 'new.par.name'. If 'FALSE' the argument 'parameter' will be evaluated instead.
- **new.par.name**: The name for the new parameter.
- **parameter**: Instead of creating a new parameter, you can also set the mutation rate to an expression based on existing parameters. For example setting this to "theta" will use an parameter with name theta that you have previously created. You can also use R expression here, so "2*theta" or "5*theta+2*tau" (if tau is another parameter) will also work (also it does not make much sense).
- **group**: Group of loci for with this feature is added. 0 means that the feature applies to all groups, and 1 is the default group. Set to 1 or an greater integer to set this feature only for the corresponding group of loci.
- **variance**: Set to a value different from 0 to introduce variation in the the parameter value for different loci. The variation follows a gamma distribution with mean equal to the value provided as parameter, and variance as given here. Can also be set to a previously created parameter, or an expression based on parameters.
**dm.addMutationRateHeterogenity**

**Value**

The demographic model with mutation.

**Examples**

```r
# Create a new parameter
dm <- dm.createDemographicModel(c(25, 25), 100)
dm <- dm.addSpeciationEvent(dm, 0.01, 5)
dm <- dm.addMutation(dm, 1, 20)

# Create a new fixed parameter
dm <- dm.createDemographicModel(c(25, 25), 100)
dm <- dm.addSpeciationEvent(dm, 0.01, 5)
dm <- dm.addMutation(dm, fixed.value=7)

# Use an existing parameter
dm <- dm.createDemographicModel(c(25, 25), 100)
dm <- dm.addParameter(dm, "theta", 0.01, 5)
dm <- dm.addMutation(dm, par.new=FALSE, parameter="2*\log(\theta)+1")
```

**Description**

This function adds a Gamma distributed rate heterogeneity as implemented in 'seq-gen' to the model.

**Usage**

```r
dm.addMutationRateHeterogenity(dm, min.alpha, max.alpha, fixed.alpha, par.new = T, new.par.name = "alpha", parameter, categories.number)
```

**Arguments**

- `dm` The demographic model to which the rate heterogeneity should be added.
- `min.alpha` If you want to estimate the rate heterogeneity, this will be used as the smallest possible value.
- `max.alpha` Same as min.growth.rate, but the largest possible value.
- `fixed.alpha` If specified, the growth rate will not be estimated, but assumed to have the given value.
- `par.new` If ‘TRUE’ a new parameter will be created using the arguments ‘min.alpha’ and ‘max.alpha’ or ‘fixed.alpha’. It will be named ‘new.par.name’ If ‘FALSE’ the argument ‘parameter’ will be evaluated instead.
### dm.addOutgroup

**new.par.name**  
Name for the new parameter. occurs. See dm.addSpeciationEvent for more information.

**parameter**  
Instead of creating a new parameter, you can also set the mutation rate to an expression based on existing parameters. For example setting this to "alpha" will use a parameter with name alpha that you have previously created. You can also use R expression here, i.e. "2*alpha" or "5*M+2*alpha" (if M is another parameter) will also work (also the latter does not make much sense).

**categories.number**  
If this is set, a fixed number of categories will be used to model the gamma distribution instead of drawing every parameter separately (see text).

### Details

"The [...] model of rate heterogeneity assigns different rates to different sites according to a gamma distribution (Yang, 1993). The distribution is scaled such that the mean rate for all the sites is 1 but the user must supply a parameter which describes its shape. A low value for this parameter (<1.0) simulates a large degree of site-specific rate heterogeneity and as this value increases the simulated data becomes more rate-homogeneous. This can be performed as a continuous model, i.e. every site has a different rate sampled from the gamma distribution of the given shape, or as a discrete model, i.e. each site falls into one of N rate categories approximating the gamma distribution. For a review of site-specific rate heterogeneity and its implications for phylogenetic analyses, see Yang (1996)." [From the seq-gen homepage http://bioweb2.pasteur.fr/docs/seq-gen ]

The Parameter in this text will be referred to as 'alpha'. Simulation a model with rate heterogeneity requires that 'seq-gen' is installed on your system.

### Value

The demographic model with a size change.

### Examples

```r
# A model with one smaller population
dm <- dm.createThetaTauModel(c(20,37), 88)
dm <- dm.setMutationModel(dm, "HKY")
dm <- dm.addMutationRateHeterogenity(dm, 0.1, 5, new.par.name="alpha")
```

---

**dm.addOutgroup**  
*Adds an outgroup to a demographic model*

### Description

This function adds an outgroup consisting of one individual to a demographic model. The outgroup consists of one individual. An outgroup is required for a finite sites analysis.

### Usage

```r
dm.addOutgroup(dm, separation.time)
```
**dm.addParameter**

**Arguments**
- **dm**: The demographic model to which we add the outgroup
- **separation.time**: The time point at which the outgroup splits from the ancestral population. This can be an absolute value (e.g. 10) or relative to another time points (e.g. `5*t_split_1`).

**Value**
The extended demographic model

---

**dm.addParameter** *Create a parameter that can be used for one or more features*

**Description**
The function creates a new model parameter. It can either have a fixed value or you can enter a range of possible values if you want to estimate it.

**Usage**
```
dm.addParameter(dm, par.name, lower.boundary, upper.boundary, fixed.value)
```

**Arguments**
- **dm**: The demographic model to which the parameter will be added
- **par.name**: The name of the parameter. You can use this name later to access the parameter from a model feature
- **lower.boundary**: The lower boundary of the range within which the parameter value will be estimated. Don’t specify ‘fixed.value’ if you want to do so.
- **upper.boundary**: Like ‘lower.boundary’, but the upper end of the parameter range.
- **fixed.value**: If this argument is given, than rather than being estimated a fixed value will be used.

**Value**
The original model extended with the new parameter.

**Examples**
```
dm <- dm.createThetaTauModel(c(15,23), 100)
dm <- dm.addParameter(dm, "mig", 0.1, 5)
dm <- dm.addMigration(dm, par.new=FALSE, parameter="mig", pop.from=1, pop.to=2)
dm <- dm.addMigration(dm, par.new=FALSE, parameter="2*mig", pop.from=2, pop.to=1)
```
dm.addRecombination

Adds recombination events to a demographic model

Description

This function adds the assumption to the model that recombination events may occur within each locus. The corresponding parameter - usually named rho - equals 4*N0*r, where r is the probability that a recombination event within the locus will occur in one generation. Even when using an infinite sites mutation model, this assumes an finite locus length which is given by the 'seq.length' parameter of the demographic model.

Usage

dm.addRecombination(dm, lower.range, upper.range, fixed.value, par.new = T,
new.par.name = "rho", parameter, group = 0, variance = 0)

Arguments

dm
The demographic model to which recombination events should be added.

lower.range
If you want to estimate the recombination rate (see note

upper.range
Same as lower.range, but the largest possible value.

fixed.value
If specified, the mutation rate will not be estimated, but assumed to have the given value.

par.new
If 'TRUE' a new parameter will be created using the arguments 'lower.range' and 'upper.range' or 'fixed.value'. It will be named 'new.par.name'. If 'FALSE' the argument 'parameter' will be evaluated instead.

new.par.name
The name for the new parameter.

parameter
Instead of creating a new parameter, you can also set the mutation rate to an expression based on existing parameters. For example setting this to "rho" will use an parameter with name theta that you have previously created. You can also use R expression here, so '2*rho' or '5*rho+2*tau' (if tau is another parameter) will also work (also it does not make much sense).

group
Group of loci for with this feature is added. 0 means that the feature applies to all groups, and 1 is the default group. Set to 1 or an greater integer to set this feature only for the corresponding group of loci.

variance
Set to a value different from 0 to introduce variation in the the parameter value for different loci. The variation follows a gamma distribution with mean equal to the value provided as parameter, and variance as given here. Can also be set to a previously created parameter, or an expression based on parameters.

Details

Please note that it does not make sense to estimate recombination rates with Jaatha because it assumes unlinked loci.
**dm.addSampleSize**

*Value*

The demographic model with recombination

*Examples*

```r
dm <- dm.createDemographicModel(c(25,25), 100)
dm <- dm.addSpeciationEvent(dm, 0.01, 5)
dm <- dm.addRecombination(dm, fixed=20)
dm <- dm.addMutation(dm, 1, 20)
```

**Description**

Sets how many individuals from each population are sampled at time 0.

**Usage**

```r
dm.addSampleSize(dm, sample.size, group = 0)
```

**Arguments**

- `dm`: The demographic model to which recombination events should be added.
- `sample.size`: A vector with sample sizes for each population.
- `group`: The group of loci with this sample size.

*Value*

The demographic model with the sample

**dm.addSizeChange**

*Description*

Adds an instantaneous change of the population size of one population to a model.

This function changes the effective population size of one population. The change is performed at a given time point (`at.time`) and applies to the time interval farther into the past from this point. The population size is set to a fraction of `N0`, the present day size of population one.

**Usage**

```r
dm.addSizeChange(dm, min.size.factor, max.size.factor, fixed.size.factor,
                 par.new = T, new.par.name = "q", parameter, population, at.time = "0")
```
dm.addSpeciationEvent

Arguments

- **dm**: The demographic model to which the size change should be added.
- **min.size.factor**: If you want to estimate the size factor, this will be used as the smallest possible value.
- **max.size.factor**: Same as min.size.factor, but the largest possible value.
- **fixed.size.factor**: If specified, the size factor not be estimated, but assumed to have the given value.
- **par.new**: If 'TRUE' a new parameter will be created using the arguments 'min.size.factor' and 'max.size.factor' or 'fixed.size.factor'. It will be named 'new.time.point.name'. If 'FALSE' the argument 'parameter' will be evaluated instead.
- **new.par.name**: Name for the new parameter.
- **parameter**: Instead of creating a new parameter, you can also set the mutation rate to an expression based on existing parameters. For example setting this to "tau" will use an parameter with name tau that you have previously created. You can also use R expression here, i.e. "2*tau" or "5*M+2*tau" (if M is another parameter) will also work (also this does not make much sense).
- **population**: The number of the population in which the spilt occurs. See dm.addSpeciationEvent for more information.
- **at.time**: The time point at which the size changes.

Details

If you want to add a slow, continuous change over some time, then use the dm.addGrowth function.

Value

The demographic model with a size change.

Examples

```r
# A model with one smaller population
dm <- dm.createThetaTauModel(c(20,37), 88)
dm <- dm.addSizeChange(dm, 0.1, 1, population=2, at.time="0")
```

---

dm.addSpeciationEvent  Adds a speciation event to a demographic model

Description

You can use this function the create a new population that splits of from an existing population at a given time in the past. The time can be given as parameter or as an expression based on previously generated time points.
dm.addSpeciationEvent

Usage

dm.addSpeciationEvent(dm, min.time, max.time, fixed.time, in.population = 1,
                       new.time.point = T, new.time.point.name = NA, time.point = NA)

Arguments

dm          The demographic model to which the split should be added.
min.time    If you want to estimate the time point, this will be used as the smallest possible
            value.
max.time    Same as min.time, but the largest possible value.
fixed.time  If specified, the time.point will not be estimated, but assumed to have the given
            value.
in.population The number of the population in which the split occurs. See above for more
            information.
new.time.point If 'TRUE' a new parameter will be created using the arguments 'min.time' and
                'max.time' or 'fixed.time'. It will be named 'new.time.point.name'. If 'FALSE'
je the argument 'time.point' will be evaluated instead.
new.time.point.name The name for the new time point.
time.point   Instead of creating a new parameter, you can also set the mutation rate to an
            expression based on existing parameters. For example setting this to "tau" will
            use an parameter with name tau that you have previously created. You can also
            use R expression here, i.e. "2*tau" or "5*M+2*tau" (if M is another parameter)
            will also work (also this does not make much sense).

Details

Time in measured in Number of 4N0 generations in the past, where N0 is the size of population 1
at time 0.

The command will print the number of the new population, which will be the number of previously
existing populations plus one. The first population has number "1".

Value

The demographic model with a split.

Examples

dm <- dm.createDemographicModel(c(25,25), 100)
dm <- dm.addSpeciationEvent(dm,0.01,5)
dm <- dm.addMutation(dm,1,20)
**dm.addSymmetricMigration**

*Adds symmetric migration between all populations*

**Description**

This adds migration between all subpopulation, all with the same rate. Please look at the documentation for `dm.addMigration` for detailed information about migration.

**dm.addSummaryStatistic**

*Adds a summary statistic to the model.*

**Description**

This summary statistic will be calculated for each simulation and returned by the `simSumStats` function.

**Usage**

```r
dm.addSummaryStatistic(dm, sum.stat, group = 0)
```

**Arguments**

- `dm` The demographic model to which a summary statistic should be added.
- `sum.stat` The summary statistic to add. Use the names mentioned above.
- `group` If given, the summary statistic is only calculated for a given group of loci.

**Details**

Available summary statistics are:
- `'jsfs'` - calculates the Joint Site Frequency Spectrum
- `'seg.sites'` - returns the simulated segregating sites as matrix
- `'file'` - returns a file in which the simulation output is written
- `'fpc'` - calculates the Four-Gamete-Condition based statistic

**Value**

The model with a summary statistic added.

**Examples**

```r
dm <- dm.createThetaTauModel(c(15, 20), 100)
dm <- dm.addSummaryStatistic(dm, 'seg.sites')
str(dm.simSumStats(dm, c(1, 5)))
```
**Usage**

```r
dm.addSymmetricMigration(dm, lower.range, upper.range, fixed.value,
par.new = T, new.par.name = "M", parameter, time.start = "0")
```

**Arguments**

- `dm` The demographic model to which migration events should be added.
- `lower.range` If you want to estimate the migration parameter (see note
- `upper.range` Same as lower.range, but the largest possible value.
- `fixed.value` If specified, the migration rate will not be estimated, but assumed to have the
given value.
- `par.new` If 'TRUE' a new parameter will be created using the arguments 'lower.range'
and 'upper.range' or 'fixed.value'. It will be named 'new.par.name'. If 'FALSE'
the argument 'parameter' will be evaluated instead.
- `new.par.name` The name for the new parameter.
- `parameter` Instead of creating a new parameter, you can also set the mutation rate to an
expression based on existing parameters. For example setting this to "M" will
use an parameter with name M that you have previously created. You can also
use R expression here, so "2*M" or "5*M+2*tau" (if tau is another parameter)
will also work (also this does not make much sense).
- `time.start` The time point at which the migration with this rate starts.

**Value**

The demographic model with migration

**Examples**

```r
dm <- dm.createThetaTauModel(c(25,25), 100)
dm <- dm.addSymmetricMigration(dm, 0.01, 5, time.start="0.5*tau")
```

---

**Description**

This function creates a basic empty demographic model, which is returned. Features like mutation,
pop.source splits and migration can be added afterwards.

**Usage**

```r
dm.createDemographicModel(sample.sizes, loci.num, seq.length = 1000)
```
Arguments

- sample.sizes: Number of haploid individuals/chromosomes that are sampled. If your model consists of multiple populations, this needs to be a vector containing the sample sizes from each population.
- loci.num: Number of loci that will be simulated
- seq.length: (Average) number of bases for each locus

Value

The demographic model

Examples

```r
dm <- dm.createThetaTauModel(sample.sizes=c(25,25), loci.num=100)
```

Description

Creates a standard "Theta/Tau" demographic model.

Usage

```r
dm.createThetaTauModel(sample.sizes, loci.num, seq.length = 1000)
```

Arguments

- sample.sizes: A numeric vector with the sample sizes of the two pop.sources.
- loci.num: The number of Loci to simulate.
- seq.length: For recombination, each locus is assumed to be of this length

Value

A Theta/Tau Model

Examples

```r
dm <- dm.createThetaTauModel(c(20,25), 100)
dm
```
dm.getGroups

Returns the groups currently in the model

Description

Returns the groups currently in the model

Usage

dm.getGroups(dm)

Arguments

dm The demographic model

Value

The groups in the model.

dm.getLociLength

Gets how long the loci in a group are

Description

Gets how long the loci in a group are

Usage

dm.getLociLength(dm, group = 1)

Arguments

dm The Demographic Model

group The group for which we get the length of loci

Value

The length of the loci in the group
**dm.getLociNumber**

*Gets how many loci belong to a group of loci*

**Description**

Gets how many loci belong to a group of loci

**Usage**

```python
dm.getLociNumber(dm, group = 1)
```

**Arguments**

- `dm`: The Demographic Model
- `group`: The group for which we get the number of loci. Defaults to the first group.

**Value**

The number of loci in the group

---

**dm.setLociLength**

*Defines the sequence length of each loci in a group of loci*

**Description**

Defines the sequence length of each loci in a group of loci

**Usage**

```python
dm.setLociLength(dm, loci.length, group = 0)
```

**Arguments**

- `dm`: The Demographic Model
- `loci.length`: The length each loci in the given loci group
- `group`: The group for which we set the loci number

**Value**

The changed Demographic Model
dm.setLociNumber  

Defines how many identical loci belong to a group of loci

Description

Defines how many identical loci belong to a group of loci

Usage

\[ \text{dm.setLociNumber}(\text{dm}, \text{loci.number}, \text{group} = 0) \]

Arguments

- **dm**: The Demographic Model
- **loci.number**: The number of loci in the group
- **group**: The group for which we set the loci number

Value

The changed Demographic Model

---

dm.setMutationModel  

Defines what mutation model is used for simulations

Description

As default, we simulate mutation using the Infinite Sites Model. Using the function, you can change it either to the Hasegawa, Kishino and Yano (HKY), to the Felsenstein and Churchill 96 (F84) or to the Generalised time reversible (GTR) model. This requires that seq-gen is installed on our system.

Usage

\[ \text{dm.setMutationModel}(\text{dm}, \text{mutation.model}, \text{base.frequencies}, \text{tstv.ratio}, \text{gtr.rates}) \]

Arguments

- **dm**: The demographic model for which the mutation model will be set.
- **mutation.model**: The mutation model you want to use. Can be HKY, F84 or GTR.
- **base.frequencies**: The equilibrium frequencies of the four bases. Must be a numeric vector of length four. Order is A, C, G, T.
- **tstv.ratio**: The ratio of transitions to transversions. The default is 0.5, which means that all amino acid substitutions are equally likely. In this case, the HKY model is identical to the Felsenstein 81 model.
Details

The HKY and F84 models use the arguments 'base.frequencies' and 'tstv.ratio'. The GTR model uses 'gtr.rates'.

Value

The demographic model with the new mutation model.

Examples

dm <- dm.createThetaTauModel(10:11, 10, 100)
dm <- dm.addOutgroup(dm, "2*tau")
dm.hky <- dm.setMutationModel(dm, "HKY", c(0.2, 0.2, 0.3, 0.3), 2)
dm.f81 <- dm.setMutationModel(dm, "F84", c(0.3, 0.2, 0.3, 0.2), 2)
dm.gtr <- dm.setMutationModel(dm, "GTR", gtr.rates=c(0.2, 0.2, 0.1, 0.1, 0.1, 0.2))

dm.simSumStats(dm, parameters, sum.stats = c("all"))

Description

Simulates data according to a demographic model

Usage

dm.simSumStats(dm, parameters, sum.stats = c("all"))

Arguments

dm The demographic model according to which the simulations should be done

parameters A vector of parameters which should be used for the simulations. If a matrix is given, a simulation for each row of the matrix will be performed

sum.stats A vector with names of the summary statistics to simulate, or "all" for simulating all possible statistics.

Value

A matrix where each row is the vector of summary statistics for the parameters in the same row of the "parameter" matrix

Examples

dm <- dm.createDemographicModel(c(25,25), 100)
dm <- dm.addSpeciationEvent(dm,0.01,5)
dm <- dm.addMutation(dm,1,20)
dm.simSumStats(dm,c(1,10))
estimateLogLikelihood  Estimates the likelihood for a parameter combination

Description
Estimates the likelihood for a parameter combination.

Usage
estimateLogLikelihood(param, glm.fitted, sum.stats)

Arguments
- **param**: The parameter combination. Must be a named vector where the names are the name of the parameters.
- **glm.fitted**: A list of the fitted GLMs, as produced by fitGLM().
- **sum.stats**: The summary statistic description, as in jaatha’s sum.stats slot.

Value
The estimated log-likelihood

findBestParInBlock  Functions that uses the fitted GLMs to estimate the parameters that have the highest likelihood.

Description
Functions that uses the fitted GLMs to estimate the parameters that have the highest likelihood.

Usage
findBestParInBlock(block, glm.fitted, sum.stats)

Arguments
- **block**: The block we are in.
- **glm.fitted**: The fitted GLMs.
- **sum.stats**: The observed summary statistics

Value
A list with maximum likelihood parameter (est) and log-likelihood (score)
fitGlm

Fits a Generalized Linear Model within a block.

Description
The model describes how the summary statistics depend on the parameters.

Usage
fitGlm(sim.data, jaatha, weighting = NULL)

Arguments
sim.data	Simulation data in this block.
jaatha	A Jaatha Object.
weighting	Potentially weights for the simulations.

Value
A list containing a list of fitted GLMs for each summary statistic.

fitGlmPoiTransformed

Fits a GLM for a summary statistics of type "poisson.transformed" and "poisson.independent"

Description
Fits a GLM for a summary statistics of type "poisson.transformed" and "poisson.independent"

Usage
fitGlmPoiTransformed(sim.data, sum.stat, transformation, weighting, jaatha)

Arguments
sim.data	Results from simulations
sum.stat	Name of the summary statistics
transformation	Transformation function to apply to the data
weighting	Potentially weights for the simulations.
jaatha	A Jaatha Object.

Value
A list of fitted GLMs, one for each function
fitPoiSmoothed  Fits a GLM for a summary statistics of type "poisson.smoothed"

Description
Fits a GLM for a summary statistics of type "poisson.smoothed"

Usage
fitPoiSmoothed(sim.data, sum.stat, weighting, jaatha)

Arguments
- sim.data: Results from simulations
- sum.stat: Name of the summary statistics
- weighting: Potentially weights for the simulations.
- jaatha: A Jaatha Object.

Value
A list with one fitted GLM

Jaatha-class  The "Jaatha" S4 class saves the basic parameters for a Jaatha estimation procedure

Description
Slots:
- simFunc: Function used for simulating
- par.ranges: A nx2 matrix stating the ranges for the n model parameters we want to estimate. The first row gives the lower range for the parameters, the second row the upper ranges. Each row stands for one parameter and the row-names will be used as names for the parameters.
- sum.stats: The observed summary statistics.
- seeds: A set of random seeds. First one to generate the other two. The next one is for the initial search, the last is for the refined search
- cores: The number of CPU cores to use for simulations
- use.shm: Use the shared memory /dev/shm for temporary files (Linux only)
- opts: Placeholder for additional arguments, for instance ones that needs to be passed to simFunc.
- calls: The function calls for the initial & refined search. These are used to rerun the searches with the exactly same settings for generating bootstrap confidence intervals and the likelihood-ratio statistic.
Jaatha.confidenceIntervals

starting.positions A list of the starting positions, returned by the initial search

likelihood.table A matrix with the best composite log likelihood values and corresponding parameters

conf.ints Confidence Intervals for parameter estimates produced by Jaatha.confidenceIntervals

route Tracks the best estimates of each step.

Jaatha.confidenceIntervals

Function for calculating bootstrap confidence intervals for jaatha estimates.

Description

This functions calculates bias-corrected and accelerated (BCa) bootstrap confidence intervals as described in Section 14.3 of "An introduction to the bootstrap" by Efron and Tibshirani. Basically, we simulate many datasets under the model using the estimated parameters and do a complete Jaatha estimation procedure on each dataset. We can then use the thereby estimated values to calculate approximate confidence intervals.

Usage

Jaatha.confidenceIntervals(jaatha, conf.level = 0.95, replicas = 100,
cores = 1, log.folder = tempfile("jaatha-logs"), subset = 1:replicas)

Arguments

jaatha A jaatha object that was returned by Jaatha.refinedSearch.
conf.level The intended confidence level for the interval.
replicas The number of Jaatha runs that we perform for calculating the confidence interval. Should be reasonable large.
cores The number of CPU cores that will be used.
log.folder A folder were log-files for the individual runs are placed. The default is to use a temporary folder, even though it is highly recommended to specify a folder and save the logs.
subset This setting allows you to distribute the CI calculation on multiple machines. You can specify which replicas should be run on this machine. Each run is identified by an integer between one and replicas. All runs which integers are passed as a vector in this arguments are actually executed. After all runs have finished, you need to manually copy all logs into a single folder and call Jaatha.getCIsFromLogs on this folder.

Details

Warning: This requires a large number of Jaatha runs and should be best executed on a small cluster rather than on a desktop computer.
Jaatha.getCIsFromLogs

Value
The Jaatha Object with confidence intervals included if 'subset' was not used. Nothing otherwise.

jaatha.getCIsFromLogs Function for calculating bootstrap confidence intervals from logs of a previous run of Jaatha.confidenceIntervals.

Description
Function for calculating bootstrap confidence intervals from logs of a previous run of Jaatha.confidenceIntervals.

Usage
Jaatha.getCIsFromLogs(jaatha, conf_level = 0.95, log_folder)

Arguments
jaatha The Jaatha object that was used when calling Jaatha.confidenceIntervals.
conf_level The intended confidence level of the interval. The actual level can vary slightly.
log_folder The folder with logs from the previous run. Just use the folder that was given as 'log.folder' argument there.

Value
The Jaatha Object with confidence intervals included.

jaatha.getLikelihoods Gives the best estimates after a Jaatha search

Description
This method extracts the best estimates with log composite likelihood values from an Jaatha object.

Usage
Jaatha.getLikelihoods(jaatha, max.entries = NULL)

Arguments
jaatha The Jaatha options
max.entries If given, no more than this number of entries will be returned.

Value
A matrix with log composite likelihoods and parameters of The best estimates
Jaatha.getStartingPoints

*Print Start points*

**Description**
Method to print the start Points given by an initial Jaatha search sorted by score.

**Usage**
```
Jaatha.getStartingPoints(jaatha)
```

**Arguments**
- **jaatha**: The Jaatha options

**Value**
a matrix with score and parameters of each start point

Jaatha.initialize

*Initialization of a Jaatha estimation for population genetics*

**Description**
This function sets the basic parameters for an analysis with Jaatha and is the first step for each application of it.

**Usage**
```
Jaatha.initialize(demographic.model, jsfs, seed, cores = 1,
    scaling.factor = 1, use.shm = FALSE, folded = FALSE,
    smoothing = FALSE)
```

**Arguments**
- **demographic.model**: The demographic model to use
- **jsfs**: Your observed Joint Site Frequency Spectrum (JSFS). Jaatha uses the JSFS as summary statistics.
- **seed**: An integer used as seed for both Jaatha and the simulation software
- **cores**: The number of cores to use in parallel. If 0, it tries to guess the number of available cores and use them all.
- **scaling.factor**: You can use this option if you have a large dataset. If so, Jaatha only simulates only a fraction 1/scaling.factor of the dataset and interpolates the missing data.
Jaatha.initialSearch

**use.shm** Logical. Many modern linux distributions have a shared memory file system available under /dev/shm. Set this to TRUE to use it for temporary files. Usually gives a huge performance boost. Warning: This option will be removed in a future version of jaatha. The cleaner way to achieve this is to move your complete R-temp directory to the memory drive. This is explained on http://www.paulstaab.de/2013/11/r-shm.

**folded** If 'TRUE', Jaatha will assume that the JSFS is folded.

**smoothing** If set to true, Jaatha uses a different way to summaries the JSFS. Instead of binning certain areas, and fitting a glm per area, only one glm is fitted for the complete JSFS, and the position of the different entries is treated as a model parameter. This feature is still experimental and not recommended for productive use at the moment.

**Value**

A S4-Object of type jaatha containing the settings

**Examples**

```r
dm <- dm.createThetaTauModel(c(20,25), 100)
jsfs <- matrix(rpois(21*26, 5), 21, 26)
jaatha <- Jaatha.initialize(dm, jsfs)
```

---

Jaatha.initialSearch  **Search the parameter space for good starting positions**

**Description**

This functions divides the parameter space in different parts (blocks). In each block, simulations for different parameter combinations are run to roughly predict the combination with the highest score (which is equivalent to the highest composite log likelihood). This points can later be used as starting positions for the second estimation phase of Jaatha (**Jaatha.refinedSearch**)

**Usage**

```r
Jaatha.initialSearch(jaatha, sim = 200, blocks.per.par = 2, rerun = FALSE)
```

**Arguments**

- **jaatha** The Jaatha settings (create with **Jaatha.initialize**)
- **sim** An integer stating the number of simulations that are performed for each block.
- **blocks.per.par** Integer. For each parameter axis in turn, we divide the parameter space into blocks.per.par equally sized blocks by restricting them to only a fraction of this axis.
- **rerun** You can repeat a previously done initial search in Jaatha. Do do so, just call the initial search function with the jaatha object result of the first initial search and set rerun to 'TRUE'.

Value

The jaatha object with starting positions

Jaatha.refinedSearch  Iterative search for the maximum composite likelihood parameters

Description

This function searches for the parameter combination with the highest composite likelihood. Therefore it iteratively searches the area (=block) around the last found value for a new maximum using simulations and a generalized linear model.

Usage

Jaatha.refinedSearch(jaatha, best.start.pos = 2,
    sim = length(getParNames(jaatha)) * 25, sim.final = min(sim, 100),
    epsilon = NULL, half.block.size = 0.025, weight = NULL,
    max.steps = 200, rerun = FALSE)

Arguments

jaatha  The Jaatha settings (create with Jaatha.initialize)
best.start.pos  This is the number of best starting positions found in the initial search that we will use. Jaatha runs a separate search starting from each of this points.
sim  The number of simulations that are performed in each step
sim.final  The number of simulations that are performed after the search to estimate the composite log likelihood. If not specified, the value of sim will be used
epsilon  Obsolete. Has no effect anymore and will be remove on next major release.
half.block.size  The size of the new block that is created around a new maximum.
weight  Obsolete. Has no effect anymore and will be remove on next major release.
max.steps  The search will stop at this number of steps if not stopped before (see epsilon).
rerun  You can repeat a previously done refined search in Jaatha. Do do so, just call the refined search function with the jaatha object result of the first refined search and set rerun to 'TRUE'.

Value

An Jaatha object. The found values are written to the slot likelihood.table.
**Jaatha.setSeqgenExecutable**

*Set the path to the executable for seqgen*

**Description**

Set the path to the executable for seqgen

**Usage**

```python
Jaatha.setSeqgenExecutable(sengen.exe)
```

**Arguments**

- `sengen.exe` Path to seqgen’s executable.

**normalize**

*Convert parameters from the natural scale into Jaatha’s internal scale*

**Description**

Convert parameters from the natural scale into Jaatha’s internal scale

**Usage**

```python
normalize(value, jaatha)
```

**Arguments**

- `value` A parameter combination in this natural scale
- `jaatha` The current jaatha object

**Value**

The parameter from value, converted into Jaatha’s interval 0-1 scale
runSimulations  
The function that actually executes the simulations from within Jaatha.

Description
The function that actually executes the simulations from within Jaatha.

Usage
runSimulations(pars, cores, jaatha)

Arguments
pars  
A matrix with parameter values for the simulations. Each row is a set of model parameters. Should be in jaatha internal 0-1 scaling.
cores  
The number of cores to use. Using more than one core requires the `multicore` package.
jaatha  
The current jaatha object

Value
A list, where each entry is a list of summary statistics for a simulation.

setCores  
Function to set the number of cores that Jaatha uses for simulations

Description
Jaatha can distribute the simulation on multiple cores if the operating system supports p-threads (parallel::mclapply is used internally). This function sets the number of cores that should be used.

Usage
setCores(jaatha, cores = 1)

Arguments
jaatha  
The jaatha object for which we set the number of cores
cores  
The number of cores to use. On Windows, only one core can be used.

Value
The jaatha object with set number of cores
**simulateWithinBlock**

This function executes simulations for a number of random parameter values inside a block.

**Description**

This function executes simulations for a number of random parameter values inside a block.

**Usage**

```
simulateWithinBlock(sim, block, jaatha)
```

**Arguments**

- `sim` the number of simulations to execute.
- `block` the block in which the simulations are executed.
- `jaatha` The current jaatha object

**Value**

A list of simulation results, where is simulation result is a list of summary statistics.
Index

*Topic package
  jaatha-package, 3
  Jaatha-class, 27

addFeature, 3

calcJsfs, 4
calculateJsfs, 5
checkType, 5
convertSimResultsToDataFrame, 6

denormalize, 7
dm.addGrowth, 7, 16
dm.addMigration, 8, 18
dm.addMutation, 10
dm.addMutationRateHeterogeneity, 11
dm.addOutgroup, 12
dm.addParameter, 13
dm.addRecombination, 14
dm.addSampleSize, 15
dm.addSizeChange, 8, 15
dm.addSpeciationEvent, 8, 12, 16, 16
dm.addSummaryStatistic, 18
dm.addSymmetricMigration, 18
dm.createDemographicModel, 19
dm.createThetaTauModel, 20
dm.getGroups, 21
dm.getLociLength, 21
dm.getLociNumber, 22
dm.setLociLength, 22
dm.setLociNumber, 23
dm.setMutationModel, 23
dm.simSumStats, 24

estimateLogLikelihood, 25

findBestParInBlock, 25
fitGlm, 26
fitGlmPoiTransformed, 26
fitPoiSmoothed, 27

Jaatha.confidenceIntervals, 28
Jaatha.getClsFromLogs, 28, 29
Jaatha.getLikelihoods, 29
Jaatha.getStartingPoints, 30
Jaatha.initialize, 30, 31, 32
Jaatha.initialSearch, 31
Jaatha.refinedSearch, 31, 32
Jaatha.setSeqgenExecutable, 33

normalize, 33
runSimulations, 34
setCores, 34
simulateWithinBlock, 35