Package ‘coalescentMCMC’

July 22, 2025

Version 0.4-4

Date 2022-04-22

Title MCMC Algorithms for the Coalescent

Depends ape, coda, lattice

Imports Matrix, phangorn, splines, stats, utils

ZipData no

Description Flexible framework for coalescent analyses in R. It includes a main function running the MCMC algorithm, auxiliary functions for tree rearrangement, and some functions to compute population genetic parameters. Extended description can be found in Paradis (2020) <doi:10.1201/9780429466700>. For details on the MCMC algorithm, see Kuhner et al. (1995) <doi:10.1093/genetics/140.4.1421> and Drummond et al. (2002) <doi:10.1093/genetics/161.3.1307>.

License GPL (>= 2)

NeedsCompilation yes

Author Emmanuel Paradis [aut, cre, cph]

Maintainer Emmanuel Paradis <Emmanuel.Paradis@ird.fr>

Repository CRAN

Date/Publication 2022-04-22 09:30:05 UTC

Contents

coalescentMCMC *Run MCMC for Coalescent Trees*

Description

These are the main function of the package to run a Markov chain Monte Carlo (MCMC) to generate a set of trees which is returned with their likelihoods, the coalescent likelihoods and the respective parameter(s).

The `logLik` method returns the average log-likelihood of the coalescent model. `AIC`, `BIC`, and `anova` use this average log-likelihood.

Usage

```
coalescentMCMC(x, ntrees = 3000, model = "constant", tree0 = NULL,
               printevery = 100, degree = 1, nknots = 0,
               knot.times = NULL, moves = 1:6)
## S3 method for class 'coalescentMCMC'
logLik(object, ...)
## S3 method for class 'coalescentMCMC'
AIC(object, ..., k = 2)
## S3 method for class 'coalescentMCMC'
BIC(object, ...)
## S3 method for class 'coalescentMCMC'
anova(object, ...)
```

Arguments

`x`	a set of DNA sequences, typically an object of class `"DNAbin"` or `"phyDat"`.
`ntrees`	the number of trees to output.
`tree0`	the initial tree of the chain; by default, a UPGMA tree with a JC69 distance is generated.
`model`	the coalescent model to be used for resampling. By default, a constant-THETA is used.
`printevery`	an integer specifying the frequency at which to print the numbers of trees proposed and accepted; set to 0 to cancel all printings.
`degree`, `nknots`, `knot.times`	parameters used if `model = "splines"`.
`moves`	the tree moves used by the MCMC (see details).
`...`	options passed to other methods.
`object`	an bject of class `"coalescentMCMC"`.
`k`	the coefficient used to calculate the AIC (see `AIC`).

Details

Six tree moves are programmed and one is chosen randomly at each step of the MCMC. The steps are: (1) NeighborhoodRearrangement (Kuhner et al., 1995), (2) ScalingMove, (3) branchSwapping, (4) subtreeExchange, (5) NodeAgeMove, and (6) randomWalkThetaMu (all five from Drummond et al., 2002). In practice, it appears that in many situations `moves = c(1, 3)` is a good selection resulting in around 50% acceptance rate.

Value

`coalescentMCMC` returns an object of class `c("coalescentMCMC", "coda")` with the log-likelihood and the parameters of each tree.

`logLik`, `AIC` and `BIC` return a numeric vector.

`anova` return an object of class `"anova"`.

Author(s)

Emmanuel Paradis

References

Drummond, A. J., Nicholls, G. K., Rodrigo, A. G. and Solomon, W. (2002) Estimating mutation parameters, population history and genealogy simultaneously from temporally spaced sequence data. *Genetics*, **161**, 1307–1320.

Hastings, W. K. (1970) Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, **57**, 97–109.

Kuhner, M. K., Yamato, J. and Felsenstein, J. (1995) Estimating effective population size and mutation rate from sequence data using Metropolis-Hastings sampling. *Genetics*, **140**, 1421–1430.

See Also

`getMCMCtrees`, `dcoal`, `treeOperators`

Examples

```
## Not run:
data(woodmouse)
out <- coalescentMCMC(woodmouse)
plot(out)
getMCMCtrees() # returns 3000 trees

## End(Not run)
```

`dcoal` *Density Functions of Some Time-Dependent Coalescent Models*

Description

These functions compute the (log-)likelihood values for various coalescent models, including the constant-Θ model and various time-dependent models.

Usage

```
dcoal(bt, theta, log = FALSE)
dcoal.step(bt, theta0, theta1, tau, log = FALSE)
dcoal.linear(bt, theta0, thetaT, log = FALSE)
dcoal.time(bt, theta0, rho, log = FALSE)
dcoal.time2(bt, theta0, rho1, rho2, tau, log = FALSE)
```

Arguments

`bt`	a vector of coalescent times (typically from `branching.times`).
`theta`	population parameter THETA.
`log`	a logical value specifying whether the probabilities should be returned log-transformed.
`theta0`, `theta1`, `thetaT`	THETA parameter for the time-dependent models.
`tau`	breakpoint in time when the parameters change.
`rho`, `rho1`, `rho2`	population (exponential) growth rates.

Details

The models are detailed in a vignette: `vignette("CoalescentModels")`.

Value

a numeric vector with (log-)likelihood values.

Author(s)

Emmanuel Paradis

References

Griffiths, R. C. and Tavaré, S. (1994) Sampling theory for neutral alleles in a varying environment. *Philosophical Transactions of the Royal Society of London. Series B. Biological Sciences*, **344**, 403–410.

Kuhner, M. K., Yamato, J. and Felsenstein, J. (1998) Maximum likelihood estimation of population growth rates based on the coalescent. *Genetics*, **149**, 429–434.

See Also

`coalescentMCMC`, `branching.times`

`getMCMCtrees` *Managing List of Trees From MCMC*

Description

These functions help to manage trees output from MCMCs.

`getMCMCtrees` extracts the trees from previous MCMC runs.

`saveMCMCtrees` saves the lists of trees from previous runs on the user's hard disk.

`cleanMCMCtrees` deletes the lists of trees from previous runs (the files possibly on the hard disk are not changed).

`getLastTree` extracts the last tree from a list of trees (object of class `"multiPhylo"`).

`getMCMCstats` returns the summary data for the different chains run during a session.

Usage

```
getMCMCtrees(chain = NULL)
saveMCMCtrees(destdir = ".", format = "RDS", ...)
cleanMCMCtrees()
getLastTree(X)
getMCMCstats()
```

Arguments

`chain`	an integer giving which lists of trees to extract
`destdir`	a character string giving the location where to save the files; by default, this is the current working directory.
`format`	the format of the tree files. Three choices are possible (cae-insensitive): `"RDS"`, `"Newick"`, `"NEXUS"`, or any unambiguous abbreviation of these.
`...`	options passed to the function used to write the tree files (see below) or passed to other methods.
`X`	an bject of class `"multiPhylo"`.

Details

The list of trees is returned in a specific environment and can be extracted with `getMCMCtrees`.

`saveMCMCtrees` saves the files with, by default, the RDS format using `saveRDS`. If `format = "Newick"`, `write.tree` is used.; if `format = "NEXUS"`, `write.nexus` is used. Options can be passed to any of these functions with `...`.

`getLastTree(X)` is a short-cut to `X[[length(X)]]`.

Most functions from the package **coda** can also be used to analyse the MCMC outputs.

Value

`getLastTree` returns an object of class `"phylo"`.

`getMCMCstats` returns a data frame.

Author(s)

Emmanuel Paradis

See Also

`coalescentMCMC`, `treeOperators`, `subset.coalescentMCMC`

`plotTHETA` *Plot THETA From coalescentMCMC Output*

Description

This function plots the values of Θ predicted from model fitted with `coalescentMCMC`.

Usage

```
plotTHETA(x, phy, add = FALSE, rightwards = TRUE, col = "blue",
          transparency = 50/length(phy), xlab = "Time",
          ylab = expression(Theta), ylim = NULL, x.scale = 1,
          y.scale = 1, show.present = TRUE, ...)
```

Arguments

`x`	an output from.
`phy`	an object of class `"phylo"` or `"multiPhylo"`.
`add`	a logical value: whether to add the curves on an existing plot?
`rightwards`	a logical value: whether to draw the time (horizontal) axis rightwards (the default).
`col`	the base colour of the curves.
`transparency`	the degree of transparency of the curves.
`xlab, ylab`	character strings giving the labels for the x- and y-axes.
`ylim`	a numeric vector with two values giving the limits of the y-axis (useful if several curves are drawn).
`x.scale`	the scaling factor for the x-axis; typically, the inverse of the mutation rate.
`y.scale`	the scaling factor for the y-axis; typically, the inverse of twice the mutation rate.
`show.present`	a logical value: whether to indicate the present in italics (near the zero coordinate on the x-axis).
`...`	further arguments passed to `plot`.

Value

NULL

Author(s)

Emmanuel Paradis

See Also

coalescentMCMC

`proba.coalescent` *Probability of Coalescence*

Description

This function calculates the probability that two lineages coalesce out of a sample of size n in a population of size N after t generations.

Usage

```
proba.coalescent(t, N = 1e4, n = 2, exact = TRUE)
```

Arguments

`t`	a vector of generations (rounded to integers if needed).
`N`	the size of population (10,000 by default).
`n`	the sample size (2 by default).
`exact`	a logical value specifying whether exact calculation should be done or an approximation (Hudson, 1991, eq. 3).

Author(s)

Emmanuel Paradis

References

Hudson, R. R. (1991). Gene genealogies and the coalescent process. *Oxford Surveys in Evolutionary Biology*, **7**, 1–44.

Examples

```
proba.coalescent(1:10)
## the approximate formula doesn't work well when n is
## not small compared to N:
proba.coalescent(1, 100, 50, exact = FALSE)
proba.coalescent(1, 100, 50)
```

sim.coalescent *Coalescent Simulation and Visualisation*

Description

This is a pedagogic function to show what is the coalescent in a simple population model with discrete generations and asexual reproduction.

Usage

```
sim.coalescent(n = 5, TIME = 50, growth.rate = NULL, N.0 = 50, N.final = 20,
               col.lin = "grey", col.coal = "blue", pch = NULL, ...)
```

Arguments

`n`	the sample size.
`TIME`	the number of generations.
`growth.rate`	the growth rate of the population.
`N.0`	the initial size of the population.
`N.final`	the final size of the population (i.e., at present).
`col.lin`	the colour used to show links of ancestry in the population.
`col.coal`	the colour used to show the coalescent of the n individuals.
`pch`	the symbol used to show individuals (none by default).
`...`	further arguments passed to `points` if `pch` is used.

Details

The simulation works along the following steps. The number of individuals at each generation is calculated. For each individual, a (unique) parent is randomly chosen at the previous generation. All individuals are then plotted and the ancestry lines are shown; the individuals are eventually ordered to avoid line-crossings. A sample of n individuals are randomly chosen from the last generation, and their shared ancestry is shown with thicker lines.

The first (oldest) generation is at the bottom, and the final (present) one is at the top of the plot.

The population size at each generation is determined from the four arguments: `TIME`, `growth.rate`, `N.0`, and `N.final`. At least three of them must be given by the user. If `TIME` is not given, its value is calculated with `log(N.final/N.0) / growth.rate`.

This code was used to make the figures in Emerson et al. (2001).

Author(s)

Emmanuel Paradis

References

Emerson, B., Paradis, E. and Thebaud, C. (2001). Revealing the demographic histories of species using DNA sequences. *Trends in Ecology and Evolution*, **16**, 707–716.

Examples

```
sim.coalescent()
sim.coalescent(N.0 = 20) # constant population size
```

`subset.coalescentMCMC` *Subset MCMC Output*

Description

This function helps to manipulate outputs from `coalescentMCMC`, particularly it sets the attributes correctly (unlike if you would use `[`).

Usage

```
## S3 method for class 'coalescentMCMC'
subset(x, burnin = 1000, thinning = 10, end = NULL, ...)
```

Arguments

`x`	an object of class `"coalescentMCMC"`.
`burnin`	an integer: the number of generations to drop.
`thinning`	an integer: the sampling frequency.
`end`	an integer: the generations after `end` are removed.
`...`	(unused)

Details

This function has more or less the same effect than the standard indexing operator (`[`). The main difference is that the attributes attached to the output from `coalescentMCMC` are not lost.

The options `end` may help to focus on one part of the MCMC (see examples).

Value

an object of class `"coalescentMCMC"`.

Note

The default values of `burnin` and `thinning` are only indicative: it is recommended to use functions in the package **coda** to help find appropriate values (see examples).

Author(s)

Emmanuel Paradis

See Also

acfplot, effectiveSize

Examples

```
## Not run:
data(woodmouse)
res <- coalescentMCMC(woodmouse, 1e6, moves = c(1, 3)) # ~ 1 hr
plot(res) # surely hard to read
plot(subset(res, end = 1e3)) # plot only the first 1000 generations

acfplot(res)
acfplot(subset(res, 1e4, 100))

## End(Not run)
```

---

treeOperators *Trees Operators for Running MCMC*

Description

These functions provide tools for tree rearrangement to be used as operators in a MCMC run.

Usage

```
NeighborhoodRearrangement(phy, n, THETA, brtimes)
TipInterchange(phy, n)
```

Arguments

`phy`	a tree of class `"phylo"`.
`n`	the number of tips in `phy`.
`THETA`	The estimate of Θ for `phy` at the node 'target'.
`brtimes`	the branching times of `phy`.

Details

`NeighborhoodRearrangement` performs a rearrangement as described by Kuhner et al. (1995).

`TipInterchange` interchanges two tips under the condition that they are not sisters.

`EdgeLengthJittering` alters the branch lengths by adding a random value from a uniform distribution defined by `range(phy$edge.length)` (the ultrametric nature of the tree is conserved).

Value

an object of class "phylo".

Author(s)

Emmanuel Paradis

References

Kuhner, M. K., Yamato, J. and Felsenstein, J. (1995) Estimating effective population size and mutation rate from sequence data using Metropolis-Hastings sampling. *Genetics*, **140**, 1421–1430.

See Also

coalescentMCMC, dcoal

Examples

```
tr <- rcoal(10)
ts <- NeighborhoodRearrangement(tr, 10, 1, branching.times(tr))
layout(matrix(1:2, 2))
plot(tr); plot(ts)
layout(1)
```

Index