Package 'glinvci'

November 30, 2022

Type Package

Title Phylogenetic Comparative Methods with Uncertainty Estimates

Version 1.1.21

Date 2022-11-28

Maintainer Hao Chi Kiang <hello@hckiang.com>

Description A framework for analytically computing the asymptotic confidence intervals and maximum-likelihood estimates of a class of continuous-time Gaussian branching processes defined by Mitov V, Bartoszek K, Asimomitis G, Stadler T (2019) <doi:10.1016/j.tpb.2019.11.005>. The class of model includes the widely used Ornstein-Uhlenbeck and Brownian motion branching processes. The framework is designed to be flexible enough so that the users can easily specify their own sub-models, or re-parameterizations, and obtain the maximum-likelihood estimates and confidence intervals of their own custom models.

License GPL-3

RoxygenNote 7.1.2

VignetteBuilder utils

Encoding UTF-8

URL https://git.sr.ht/~hckiang/glinvci,
https://github.com/hckiang/glinvci

Depends R (>= 3.3.0)

Imports Rcgmin, lbfgsb3c, BB, ape, numDeriv, plyr, rlang, utils, stats

Suggests testthat

NeedsCompilation yes

Author Hao Chi Kiang [cre, aut]

Repository CRAN

Date/Publication 2022-11-30 08:10:07 UTC

R topics documented:

`clone_model` *Clone a GLInv model*

Description

The `clone_model` function is a S3 generic method for either the `glinv` or `glinv_gauss` class.

Usage

```
clone_model(mod, ...)

## S3 method for class 'glinv_gauss'
clone_model(mod, ...)

## S3 method for class 'glinv'
clone_model(mod, ...)
```

Arguments

`mod`	An object of either `glinv` or `glinv_gauss` class.
`...`	Further arguments to be passed to the S3 methods. Not used currently.

Details

Because `glinv` or `glinv_gauss` object is mutable, the assignment `model2 = model1` will not make a copy your model. The correct way to copy a model is to use the `clone_model` function.

Value

A new model that is a clone of mod.

Examples

```
repar = get_restricted_ou(H=NULL, theta=c(0,0), Sig='diag', lossmiss=NULL)
mod1 = glinv(tree    = ape::rtree(10),
             x0      = c(0,0),
             X       = NULL,
             repar   = repar)
mod2 = mod1
mod3 = clone_model(mod1)
traits = matrix(rnorm(20), 2, 10)
set_tips(mod1, traits)
print(has_tipvals(mod1))  # TRUE
print(has_tipvals(mod2))  # TRUE
print(has_tipvals(mod3))  # FALSE
```

---

fit *Fitting a GLInv model via numerical optimisation*

Description

fit.glinv finds the maximum likelihood estimate of a glinv model by solving a numerical optimisation problem.

Usage

```
fit(mod, ...)

## S3 method for class 'glinv'
fit(
  mod,
  parinit = NULL,
  method = "L-BFGS-B",
  lower = -Inf,
  upper = Inf,
  use_optim = FALSE,
  project = NULL,
  projectArgs = NULL,
  control = list(),
  ...
)
```

Arguments

mod	An object of class glinv.
...	Not used.
parinit	A vector, parameter for initialisation of the optimisation routine.
method	One of L-BFGS-B, CG, BB, or any other methods which is accepted by optim.
lower	A vector of lower bounds on the parameters.
upper	A vector of upper bounds on the parameters.
use_optim	If true, use optim's version of L-BFGS-B and CG.
project	Passed to BBoptim.
projectArgs	Passed to BBoptim.
control	Options to be passed into each the underlying optimisation routine's control argument.

Details

If method is L-BFGS-B, then lbfgsb3c is used for optimisation; if it is CG then Rcgmin is used; if it is BB then BBoptim is used, otherwise the method argument is passed to optim.

By default, L-BFGS-B declares convergence when the change of function value is small, CG tests stops when change of gradient squared-Euclidean-norm is small, BB stops when either the change of function values, or the infinity norm of a project gradient, is small. These can be changed through the control argument and the user should refer to the optimisation packages' respective documentation for details.

The user can opt for using optim's version of CG and L-BFGS-B. The implementation in optim of the methods does not incorporate improvements of the methods in the recent decades, but they have stood the test of time.

If parinit were not supplied and the distance between lower and upper is infinite, the initialisation point of the optimisation is drawn from a uniform distribution ranging [-1,1] distribution. If initalisation were not supplied, but the distance between lower and upper is finite, then the initialisation is drawn from a uniform distribution ranging [lower, upper].

Value

fit.glinv returns a list containing at least the following elements:

mlepar	The maximum likelihood estimate.
loglik	The log-likelihood at the maximum likelihood estimate.
score	The gradient of log-likelihood at the maximum likelihood estimate.
convergence	Zero if the optimisation routine has converged successfully.
message	A message from the optimisation routine.

get_restricted_ou *Convenience function for constructing restricted/reparameterised OU parameterisation function.*

Description

get_restricted_ou is a convenience function for constructing restricted/reparameterised OU parameterisation.

Usage

```
get_restricted_ou(H = NULL, theta = NULL, Sig = NULL, lossmiss = "halt")
```

Arguments

H	One of NULL, 'symmetric', 'logspd', 'spd', 'diag', 'logdiag', 'zero', or a numerical vector specifying fixed parameters.
theta	One of NULL, 'zero', or a numerical vector specifying fixed parameters.
Sig	One of NULL, 'diag', or a numerical vector specifying fixed parameters.
lossmiss	One of NULL, 'zap', 'halt'.

Details

get_restricted_ou is intended to provide a more convenient way to construct the restrictions functions, restricted Jacobian and Hessian, than the more flexible methods described in parameter_restriction.

If either one of H, theta is 'zero' but not both, the function stops with error. This is because former is statistically not sensible, and the latter can be done by directly passing a vector of zero to the theta argument.

If lossmiss is NULL, the returned functions does not have capability to handle missing or lost values.

Value

A list containing the following elements:

par	A reparameterisation function conforming to the format required by the parfns argument of glinv.
jac	A Jacobian function of the above reparameterisation function conforming to the format required by the parjacs argument of glinv.
hess	A Hessian function of the above reparameterisation function conforming to the format required by the parhess argument of glinv.
nparams	A function which accepts one integer argument, the total number of dimensions of the multivariate traits, and returns the number of parameters of the restricted model.

Examples

```
### --- STEP 1: Make an example tree and trait data
ntips = 200
k     = 2                # No. of trait dimensions
tr    = ape::rtree(ntips)
X     = matrix(rnorm(k*ntips), k, ntips)
x0    = rnorm(k)

### --- STEP 2: Make a model which has unrestricted H, fixed theta and diagonal Sigma_x'.
repar = get_restricted_ou(H=NULL, theta=c(3,1), Sig='diag', lossmiss=NULL)
mod   = glinv(tr, x0, X,
              pardims =repar$nparams(k),
              parfns  =repar$par,
              parjacs =repar$jac,
              parhess =repar$hess)
# Actually, to save typing, the following short-cut call is the same as the above:
# mod = glinv(tr, x0, X, repar=repar)

### --- STEP 3: Use the model as usual, say, we compute the likelihood at a specified parameter.
H     = matrix(c(1,0,0,-1), k)
theta = c(3,1)
sig   = matrix(c(0.25,0,0,0.25), k)
sig_x = t(chol(sig))
LIK   = lik(mod)(c(H, c(0.5,0.5)))

### --- STEP 4: Confirm the restricted model does indeed match the unrestricted.
mod_unrestricted = glinv(tr, x0, X,
                         pardims=nparams_ou(k),
                         parfns=oupar,
                         parjacs=oujac,
                         parhess=ouhess)
LIK_unrestricted = lik(mod_unrestricted)(c(H,theta,sig_x[lower.tri(sig_x, diag=TRUE)]))
print(LIK == LIK_unrestricted)
# [1] TRUE

### --- STEP 5: Confirm the this is indeed the same as making everything manually
mod_manual = glinv(tr, x0, X,
                   pardims  = nparams_ou_fixedtheta_diagSig(k),
                   parfns   = ou_fixedtheta_diagSig(oupar,   theta=c(3,1)),
                   parjacs  = dou_fixedtheta_diagSig(oujac,  theta=c(3,1)),
                   parhess  = hou_fixedtheta_diagSig(ouhess, theta=c(3,1)))
LIK_manual = lik(mod_manual)(c(H=H, sig_x=c(0.5,0.5)))
print(LIK == LIK_manual)
# [1] TRUE
```

---

glinv *Construct an GLInv model with respect to user-specified parametrisation*

Description

The `glinv` function construct an object of class `glinv`, which represents a GLInv model with respect to a user-specified parametrisation.

The `lik.glinv` function returns a function which accepts a parameter vector, which is of length `mod$nparams`, and returns the log-likelihood.

The `grad.glinv` function returns a function which accepts a parameter vector, which is of length `mod$nparams`, and returns the gradient of log-likelihood with respect to this parametrisation.

The `hess.glinv` function returns a function which accepts a parameter vector, which is of length `mod$nparams`, and returns the Hessian matrix of log-likelihood with respect to this parametrisation.

Usage

```
glinv(
  tree,
  x0,
  X,
  parfns = NULL,
  pardims = NULL,
  regimes = NULL,
  parjacs = NULL,
  parhess = NULL,
  repar = NULL
)

## S3 method for class 'glinv'
print(x, ...)

## S3 method for class 'glinv'
lik(mod, ...)

## S3 method for class 'glinv'
grad(
  mod,
  numDerivArgs = list(method = "Richardson", method.args = list(d = 0.5, r = 3)),
  ...
)

## S3 method for class 'glinv'
hess(
  mod,
  numDerivArgs = list(method = "Richardson", method.args = list(d = 0.5, r = 3)),
  store_gaussian_hessian = FALSE,
  ...
)

## S3 method for class 'glinv'
plot(x, internal_nodes = TRUE, ...)
```

Arguments

tree	A tree of class phylo.
x0	A vector representing the root's trait vector. Must not contain NA and NaN.
X	Optional. A matrix of trait values, in which X[p,n] stores the p-th dimension of the multivariate trait of the n-th tip of the phylogeny. NA and NaN has special meanings (See Details).
parfns	A list of functions that maps from the user-parametrisation to the underlying Gaussian parameters. Each of them returns a vector of concatenated (Φ, w, V'), where V' is the lower triangular part of V, and accepts four arguments: a vector of parameters whose length is specified by the pardims argument to the glinv_gauss function, the branch length leading to the currently processing node, a vector of factors with three levels indicating which dimensions are missing or lost in the mother of the current node, and a vector of factors with the same three levels indicating missingness of the current node.
pardims	A vector of integers, which has the same amount elements as the length of parfns. pardims[i] indicates the length of the parameter vector that parfns[i] accepts.
regimes	A list of length-two integer vectors. Each of these length-two vectors specifies an evolutionary regime and consists of a named element start, which specifies the node ID at which an evolutionary regime starts, and another named element fn, which is an index of parfns, indicating which parametrisation function this evolutionary regime should use.
parjacs	A list of functions, which has the same amount elements as that of parfns. parjacs[i] accepts the same arguments as parfns[i] and returns the Jacobian of parfns[i].
parhess	A list of functions, which has the same amount elements as that of parfn[i]. parhess[i] accepts the same arguments as parfns[i] and returns a list of three 3D arrays, named Phi, w, V respectively inside the list. ((parhess[[i]])(...))$Phi[m,i,j] contains the cross second-order partial derivative of Φ_m (here we treat the matrix Φ as a column-major-flattened vector) with respect to the i-th and$j$-th user parameters; while ((parhess[[i]])(...))$w[m,i,j] and ((parhess[[i]])(...))$V[m,i,j] analogously contains second-order derivative of w_m and V'_m.
repar	Optional. One or a list of object returned by get_restricted_ou. This is a convenient short-cut alternative to supplying pardims, parfns, parjacs, and parhess one-by-one.
x	An object of class glinv.
...	Not used.
mod	An object of class glinv.
numDerivArgs	Arguments to pass to numDeriv::jacobian. Only used the user did not specify the parjacs arguments when creating mod.
store_gaussian_hessian	If TRUE and method is not mc, the returned list will contain a (usually huge) Hessian matrix gaussian_hessian with respect to the Gaussian parameters Φ, w, V'. This option significantly increases the amount of memory the function uses, in order to store the matrix.
internal_nodes	Boolean, whether to plot the internal nodes's numbers

Details

Models for glinv assumes one or more evolutionary regimes exists in the phylogeny. The regimes parameters defines how many regimes there are, where do the regimes start, and what parameterisation function it has. If regimes were NULL then a single regime starting from the root node is assumed. Multiple regimes could share the same parametrisation function (and thus the same parameters) by specifying the same index; therefore the number of regimes may differs from the number of parametrisation functions. One and only one regime must start from the root of the phylogeny.

If X contains NA in the p-th dimension of the i-th tip (whose node ID is also i) then $X_p i$ is tagged MISSING. No other tags of any other nodes are changed. The p-th dimension of any node j, regardless of whether or not it is an internal node or a tips, is tagged LOST if and only if the p-th dimension of all tips inside the clade started at j are NaN. Any entry that is neither LOST nor MISSING are tagged OK. These tags are then passed into the user-defined functions parfns etc. as arguments; therefore the user is free to specify how these tags are handled. x0 cannot contain missing values, and the vectors of missingness tags passed to parfns, for any nodes, are always of the same length as x0.

Before this package calls the functions in parhess, it adds, into the function's environment, a variable named INFO__ which contains some extra information.

Passing a single function to parfns is equivalent to passing a singleton list; and the same is true for parjacs, parhess, and pardims.

Value

The glinv function returns a model object of S3 class glinv. Elements are:

rawmod	An object of class glinv_gauss.
regimes	Identical to the regimes argument.
regtags	An integer vector of the same length as the number of nodes. The i-th element is the regime ID (corresponding to the index in the regimes argument to the glinv_gauss function) of node i. NA at the root.
misstags	A factor matrix with three ordered levels, LOST, OK, and MISSING. Each column corresponds to a node and row to a trait dimension.
nparams	The sum of the pardims argument, an integer.
pardims	Identical to the pardims arguemnt.
parfntags	An integer vector of the same length as the number of nodes. The i-th element is the index of parfns that corresponds to node i. NA at the root.
parfns	Identical to the parfns argument.
parjacs	Identical to the parjacs argument.
parhess	Identical to the parhess argument.
parsegments	A K-by-2 matrix of integer indicies, where K is the length of parfns. If v is a vector that lik.glinv accepts, then v[parsegments[k,1]:parsegments[k,2]] is the parameter vector should parfns[[k]] accept.
gausssegments	A N-by-2 matrix of integer indicies, where N is the number of nodes. If w is a vector that lik.glinv_gauss accepts, then w[gausssegments[i,1]:gausssegments[i,2]] is the concatenated (Φ, w, V') corresponding to node i.

gaussparams_fn A function that accepts a parameter vector of length nparams and returns a parameter vector of length rawmod$nparams. When called, this function traverses the tree, calls the functions in parfns on each node, and assemble the results into a format that lik.glinv_gauss accepts.

gaussparams_jac A function that accepts a parameter vector of length nparams and returns a p-by-q Jacobian matrix, where p is rawmod$nparams and q is nparams in this object. When called, this function traverses the tree, calls the functions in parjacs on each node, and row-concatenates the result in an order consistent with what lik.glinv_gauss accepts.

X The original data (trait) matrix in a "normalized" format.

References

Mitov V, Bartoszek K, Asimomitis G, Stadler T (2019). "Fast likelihood calculation for multivariate Gaussian phylogenetic models with shifts." Theor. Popul. Biol.. https://doi.org/10.1016/j.tpb.2019.11.005.

Examples

```
### --- STEP 1: Making an example tree and trait data
ntips = 200
k     = 2                 # No. of trait dimensions
tr    = ape::rtree(ntips)
X     = matrix(rnorm(k*ntips), k, ntips)
x0    = rnorm(k)

### --- STEP 2: Making a model object. We use OU as an example.
###             Assume H is a positively definite diagonal matrix.
mod = glinv(tr, x0, X,
            parfns = list(ou_logdiagH(ou_haltlost(oupar))),
            pardims = list(nparams_ou_diagH(k)),
            parjacs = list(dou_logdiagH(dou_haltlost(oujac))),
            parhess = list(hou_logdiagH(hou_haltlost(ouhess))))

### --- STEP 3: Try getting the likelihood, gradient etc.
H     = matrix(c(1,0,0,-1), k)
theta = c(0,0)
sig   = matrix(c(0.5,0,0,0.5), k)
sig_x = t(chol(sig))
# glinvci ALWAYS assumes diagonals of sig_x is in log scale.
diag(sig_x) = log(diag(sig_x))
par_init = c(H=diag(H),theta=theta,sig_x=sig_x[lower.tri(sig_x,diag=TRUE)])
print(par_init)
print(lik(mod)(par_init))
print(grad(mod)(par_init))
print(hess(mod)(par_init))

### --- STEP 4: Fitting a model
fitted = fit(mod, par_init)
print(fitted)
```

```
### --- STEP 5: Estimating variance-covariance of the MLE
v_estimate = varest(mod, fitted)

### --- STEP 6: Get marginal confidence intervals
print(marginal_ci(v_estimate, lvl=0.95))
```

`glinvci` *glinvci: Confidence intervals and hypothesis testing for GLInv model*

Description

The glinvci package provides a framework for computing the maximum-likelihood estimates and asymptotic confidence intervals of a class of continuous-time Gaussian branching processes, including the Ornstein-Uhlenbeck branching process, which is commonly used in phylogenetic comparative methods. The framework is designed to be flexible enough that the user can easily specify their own parameterisation and obtain the maximum-likelihood estimates and confidence intervals of their own parameters.

Author(s)

Hao Chi Kiang, <hello@hckiang.com>

`glinv_gauss` *Construct an object representing a GLInv model with respect to the underlying Gaussian process parameters.*

Description

The `glinv_gauss` function constructs an object of class `glinv_gauss`, which represents a lower-level GLInv model with respect to the underlying Gaussian process space. The likelihood Hessian of, for example, Brownian motion and Ornstein-Uhlenbeck models can be computed by applying the calculus chain rule to the output of Jacobians and Hessians from this class.

The `lik.glinv_gauss` function computes the likelihood of a full `glinv_gauss` model.

The `grad.glinv_gauss` function computes the log-likelihood gradients of a `glinv_gauss` models. If `par` is NULL, it returns a function that, when called, returns the same thing as if `grad.glinv_gauss` were called with `par` argument.

The `hess.glinv_gauss` function computes the log-likelihood Hessian of a `glinv_gauss` models.

Usage

```
glinv_gauss(tree, x0, dimtab = NULL, X = NULL)

## S3 method for class 'glinv_gauss'
lik(mod, par = NULL, ...)

## S3 method for class 'glinv_gauss'
grad(mod, par = NULL, lik = FALSE, ...)

## S3 method for class 'glinv_gauss'
hess(mod, par = NULL, lik = FALSE, grad = FALSE, directions = NULL, ...)

## S3 method for class 'glinv_gauss'
print(x, ...)
```

Arguments

tree	A tree of class ape::phylo.
x0	A vector representing the root's trait vector.
dimtab	An integer, a vector of integers, or NULL, specifying the number of dimensions of each nodes of the tree. If it is a vector, dimtab[n] is the trait vector dimension of node n. If it is only a single integer than all nodes are assumed to have the same amount of dimensions. If it is NULL then all nodes are asummed to have the same amount of dimensions as x0.
X	Trait values, either a matrix in which X[p,n] stores the p-th dimension of the multivariate trait of the n-th tip of the phylogeny, or a list in which X[[n]] is a numeric vector representing the multivariate trait of the n-th tip. The latter form is required if not all the tips has the same number of dimensions.
mod	A model object of class glinv_gauss.
par	A vector, containing the parameters at which the likelihood should be computed.
...	Not used.
lik	If TRUE, grad.glinv_gauss and hess.glinv_gauss returns also the log-likelihood.
grad	If TRUE, hess.glinv_gauss returns also the gradient.
directions	Either NULL or a matrix with mod$nparams many rows and arbitrarily many columns. If NULL, 'hess.glinv_gauss' returns the Hessian matrix itself, which is typically a huge matrix; otherwise, the funciton returns a square matrix M such that M_ij contains $d_i^T H d_j$, where d_i is the i-th column of directions and H is the huge Hessian matrix, without storing H itself in memory.
x	An object of class glinv_gauss.

Details

The glinv_gauss class does not include any information for dealing with evolutionary regimes, lost traits, and missing data, nor does it facilitate reparametrisation. These are all functionality of the glinv class instead. The member variables of the objects of the glinv_gauss class only are for

the users' convenience to *read* the information about the model, and the user *should not* modify its member variables directly.

For each non-root node i in the phylogeny, the multivariate trait vector x_i follows a Gaussian distribution with mean $\Phi_i x_j + w_i$ and variance V_i when conditioned on the mother's trait vector x_j. The 'parameters' of this model is, therefore, the joint of all $(\Phi_i, w_i V_i')$ for all nodes i. The root does not have any associated parameters.

The parameter vector `par` should be the concatenation of all (Φ_i, w_i, V_i') in accending order sorted by i, the node number (which is the same node numbers as in `tree$edge`). The matrix Φ_i is flattened in column-major order and V_i' is the lower-triangular part of V_i, column-major-flattened. Since the root does not have parameters, its entry is simply skipped. For example, if a binary tree has 10 non-root nodes in total and each of them are 3 dimensional, then each (Φ_i, w_i, V_i') should have $9 + 3 + 6 = 18$ elements; thus after concatenation `par` should be a 180 elements.

Value

An object of S3 class `glinv_gauss` with the following components

ctree A pointer to an internal C structure.

apetree Same as the `tree` argument but with some pre-processing in its edge table

origtree The `tree` argument.

x0 The trait vector at the root of the tree.

dimtab Identical to the `dimtab` argument.

gaussdim The number of dimension of the parameter space of this model.

References

Mitov V, Bartoszek K, Asimomitis G, Stadler T (2019). "Fast likelihood calculation for multivariate Gaussian phylogenetic models with shifts." Theor. Popul. Biol.. https://doi.org/10.1016/j.tpb.2019.11.005.

Examples

```
tr = ape::rtree(3)
model = glinv_gauss(tr, x0=c(0,0), X=matrix(rnorm(6),2,3))
par = unlist(
 list(
   list('Phi' = c(1,0,0,1), # Parameters for node #1, a tip
        'w'   = c(-1,1),
        'V'   = c(1,0,1)),  # Lower triangular part of a 2D identity matrix
   list('Phi' = c(2,0,0,2), # For node #2, a tip
        'w'   = c(-2,2),
        'V'   = c(2,0,2)),
   list('Phi' = c(3,0,0,3), # For node #3, a tip
        'w'   = c(-3,3),
        'V'   = c(3,0,3)),
   list('Phi' = c(4,0,0,4), # For node #5. Node #4 skipped as it is the root
        'w'   = c(-4,4),
        'V'   = c(4,0,4))
   ))
print(par)
```

```
lik(model, par)
grad(model, par)
hess(model, par)
```

`grad` *Compute the log-likelihood gradients of GLInv models*

Description

For the `glinv` class, which is a high-level user interface, please see `grad.glinv`; and for `glinv_gauss`, which is a lower-level facility, please see `grad.glinv_gauss`.

Usage

```
grad(mod, ...)
```

Arguments

`mod`	An object of either `glinv` or `glinv_gauss` class.
`...`	Further arguments to be passed to the S3 methods.

Value

A numerical vector containing the gradient of `mod`.

`has_tipvals` *Check if a* `glinv_gauss` *model contains trait values at their tips.*

Description

Returns true if and only if the `glinv_gauss` model were initialised with `X=NULL` and the user had never called `set_tips` on it.

Usage

```
has_tipvals(mod)

## S3 method for class 'glinv_gauss'
has_tipvals(mod)

## S3 method for class 'glinv'
has_tipvals(mod)
```

Arguments

`mod`	A `glinv_gauss` object.

Value

A Boolean. True if mod contains tip trait values and false otherwise.

---

hess *Compute the log-likelihood Hessian of GLInv models*

Description

For the glinv class, which is a high-level user interface, please see hess.glinv; and for glinv_gauss, which is a lower-level facility, please see hess.glinv_gauss.

Usage

```
hess(mod, ...)
```

Arguments

mod	An object of either glinv or glinv_gauss class.
...	Further arguments to be passed to the S3 methods.

Value

A numerical square matrix containing the Hessian of mod.

---

lik *Compute the likelihood of a GLInv model*

Description

This is a S3 generic method. For the glinv class, which is a high-level user interface, please see lik.glinv; and for glinv_gauss, which is a lower-level facility, please see lik.glinv_gauss.

Usage

```
lik(mod, ...)
```

Arguments

mod	An object of either glinv or glinv_gauss class.
...	Further arguments to be passed to the S3 methods.

Value

A numerical scalar containing the likelihood of mod.

marginal_ci *Getting marginal confidence interval for GLInv model*

Description

marginal_ci computes the marginal confidence interval for each parameters using the variance-covariance matrix output by 'varest.glinv'.

Usage

```
marginal_ci(varest_result, lvl = 0.95)
```

Arguments

varest_result	The output from 'varest.glinv'.
lvl	Confidence level. Default to 95 percent.

Value

A matrix p-by-2 matrix where p is the number of parameters. The first column is the lower limits and second column is the upper limits.

nparams_ou *Get the number of parameters of the unrestricted OU model*

Description

nparams_ou returns the number of parameters of the unrestricted OU model. For the restricted models, including Brownian motion, see parameter_restriction for details.

Usage

```
nparams_ou(k)
```

Arguments

k	An Integer. The total number of dimensions of the multivariate traits.

Value

A numerical scalar, which is the number of parameters of the the unrestricted OU model.

oupar *Parameterisation functions of Ornstein-Uhlenbeck model*

Description

oupar is a function that maps from the Ornstein-Uhlenbeck model parameters to the Gaussian parametersation.

oujac accepts the same arguments as oupar and returns the Jacobian matrix of oupar.

ouhess accepts the same arguments as oupar and returns all the second derivatives oupar. The returned values are consistent with the format required by glinv.

Usage

```
oupar(par, t, ...)

oujac(par, t, ...)

ouhess(par, t, ...)
```

Arguments

par	A numeric vector containing the joint vector of the Ornstein-Uhlenbeck drift matrix, long-term mean, and volitality matrix, which is a lower-triangular Cholesky factor.
t	Branch length of the currently processing node.
...	Unused in these functions. Their existence is needed because lik.glinv etc. always pass us four arguments. See lik.glinv for details.

Details

By multivariate Ornstein-Uhlenbeck process, we mean

$$dx(t) = -H(x(t) - \theta)dt + \Sigma_x dW(t)$$

where H is a k-by-k matrix with real entries, θ is any real k-vector, Σ_x is a lower-triangular matrix, $W(t)$ is the Brownian motion process. The parameters of this model is (H, θ, Σ_x), therefore $k^2 + k + k(k+1)/2$ dimensional.

This package uses parameterisation (H, θ, Σ'_x), where H and θ is the same as above defined, and Σ'_x is the lower-triangular part of Σ_x, except that, only on diagonal entries, $\Sigma'_x = log(\Sigma_x)$. The use of logarithm is for eliminating multiple local maxima in the log-likelihood.

The par arguemnt is the concatenation of column-major-flattened H, θ, and the column-major-flattened lower-triangular part of Σ'_x.

Value

`oupar` returns the a vector of concatenated (Φ, w, V'), where V' is the lower triangular part of V. `oujac` returns the Jacobian matrix of `oupar`. `ouhess` returns a list of three 3D arrays, named `Phi`, `w`, `V` respectively inside the list, in which `ouhess(...)$Phi[m,i,j]` contains the cross second-order partial derivative of Φ_m (here we treat the matrix Φ as a column-major-flattened vector) with respect to the i-th and j-th user parameters; and `ouhess(...)$w[m,i,j]` and `((parhess[[i]])(...))$V[m,i,j]` analogously contains second-order derivative of w_m and V'_m.

`ou_haltlost` *Handling missing data and lost traits in Ornstein-Uhlenbeck processes*

Description

`ou_haltlost` and `ou_zaplost` handles lost traits and missing data. Each of them wraps the function `oupar` and returns a new function that accepts the same arguments and output the same form of result, but takes into account lost traits and missing data. `dou_haltlost` and `dou_zaplost` wraps the Jacobian function `oujac`, and `hou_haltlost` and `hou_zaplost` wraps the Hessian function `ouhess`.

Usage

```
ou_haltlost(parfn)

dou_haltlost(jacfn)

hou_haltlost(hessfn)

ou_zaplost(parfn)

dou_zaplost(jacfn)

hou_zaplost(hessfn)
```

Arguments

`parfn`	A function that maps from the user-parametrisation to the underlying Gaussian parameters. Each of them returns a vector of concatenated (Φ, w, V'), where V' is the lower triangular part of V, and accepts four arguments: a vector of parameters whose length is specified by the `pardims` argument to the `glinv_gauss` function, the branch length leading to the currently processing node, a vector of factors with three levels indicating which dimensions are missing or lost in the mother of the current node, and a vector of factors with the same three levels indicating missingness of the current node.
`jacfn`	A function that accepts the same arguments as `parfn` and returns the Jacobian of `parfn`.

`hessfn` A function that accepts the same arguments as `parfns` and returns a list of three 3D arrays, named `Phi`, `w`, `V` respectively inside the list. `((hessfn)(...))$Phi[m,i,j]` contains the cross second-order partial derivative of Φ_m (here we treat the matrix Φ as a column-major-flattened vector) with respect to the i-th and$j$-th parameters in the joint (H, θ, Σ_x) vector, and `((hessfn)(...))$w[m,i,j]` and `((hessfn)(...))$V[m,i,j]` analogously contains second-order derivative of w_m and V'_m.

Details

What is missing traits and lost traits: A 'missing' trait refers to a trait value whose data is missing due to data collection problems. Fundamentally, they evolves in the same manner as other traits. An `NA` entry in the data is deemed 'missing'. On the other hand, a lost trait is a trait dimension which had ceased to exists during the evolutionary process. An `NaN` entry in the data indicates a 'lost' trait.

Each nodes has their own missing-ness tags: Each trait dimension of each nodes, either internal or tip, are tagged with one of the three labels: `MISSING`, `LOST`, and `OK`. If the data contains an `NA` in the p-th dimension of the i-th tip then $X_p i$ is tagged `MISSING`. No other tags of any other nodes and dimensions are changed in the case of missing-ness. On the other hands, the p-th dimension of any node j, regardless of whether or not it is an internal node or a tips, is tagged `LOST` if and only if the p-th dimension of all tips inside the clade started at j are `NaN`. Any entry that is neither tagged `LOST` nor `MISSING` are tagged `OK`.

This corresponds to the biological intuition that, if a value is missing only due to data collection problems, the missingness should not influence the random walk process way up the phylogenetic tree; and this is obviously not true if the trait had ceased to exists instead.

Handling of missing data and lost traits: `ou_haltlost` and `ou_zaplost` handles missing data in the same way: they simply marginalises the unobserved dimensions in the joint Gaussian distributions of tip data.

For lost traits, `ou_haltlost` assumes the followings:

1. In the entire branch leading to the earliest node j whose p-th dimension is tagged `LOST`, the lost trait dimension does not evolve at all.
2. In the entire same branch, the magnitude of the p-th dimension at j's mother node has no influence on other dimensions, in any instantaneous moments during the evolution in the branch, neither through the linear combination with the drift matrix nor the Wiener process covariance; in other words, the SDE governing the non-lost dimensions' random walk is invariant of j's mother nodes' p-th dimension.

Therefore, `ou_haltlost` first set the p-th row and column of both of H_j and the p-th row of $Sigma_x$ to zero and marginalise out the degenerate Gaussian dimension.

On the other hands, `ou_zaplost` does not assume the lost trait to stop evolving immediately at moment when the branch leading to j starts, but, instead, simply marginalise out the lost, non-degenerate Gaussian dimensions. This method is the same as the one that is used in the `PCMBase` package.

Usage in combination with parameter restrictions: Without paramter restriction, the following is an example usage in a call to the `glinv` function. It constructs a `glinv` model object which is capable of handling missing data and lost traits.

```
mod.full = glinv(tree, x0, my_data,
                 parfns  = haltlost(oupar),
                 pardims = nparams_ou(k),
                 parjacs = dhaltlost(oujac),
                 parhess = hhaltlost(ouhess))
```

Note that we have the same naming convention that functions wrappers whose nams have prefix `d` wraps the Jacobians, while prefix `d` wraps the Hessians.

If parameter restriction is needed, then `*ou_*lost` should called *before any reparameterisation/restriction functions* because it expects the passed-in function `parfn` to accept the full H matrix, rather than only the diagonal or lower-triangular part of it. Example:

```
f = haltlost(oupar)
g = dhaltlost(oujac)
h = hhaltlost(oujac)
mod.full = glinv(tree, x0, my_data,
                 parfns  = ou_spdH(f),
                 pardims = nparams_ou_spdH(k),
                 parjacs = dou_spdH(g),
                 parhess = ou_spdH(h,g))
```

Value

`ou_haltlost` and `ou_zaplost` returns a wrapped versions of 'parfn', which accepts the same arguments and outputs in the same format. `dou_haltlost` and `dou_zaplost`, analogously, wraps `jacfn`. `hou_zaplost` and `hou_zaplost` wraps `hessfn`.

`parameter_restriction` *Restrict the parameters space of OU and Brownian motion models.*

Description

`ou_diagH`, `ou_diagH_fixedtheta_diagSig`, etc., restricts the OU model's parameters. For example, `ou_diagH` restricts the drift H to diagonal matrix, and `ou_diagH_fixedtheta_diagSig` further restricts theta to be a constant and Σ'_x to be diagonal. A Brownian motion model can be made by these restriction.

Usage

```
avail_restrictions

brn_diagSig(parfn)

ou_logdiagH(parfn)

dou_logdiagH(jacfn)

hou_logdiagH(hessfn)
```

```
ou_logdiagH_diagSig(parfn)

ou_logspdH_fixedtheta(parfn, theta)

ou_spdH_fixedSig(parfn, Sig)

ou_fixedH_diagSig(parfn, H)

dou_logdiagH_diagSig(jacfn)

dou_logspdH_fixedtheta(jacfn, theta)

dou_spdH_fixedSig(jacfn, Sig)

dou_fixedH_diagSig(jacfn, H)

hou_logdiagH_diagSig(hessfn)

hou_logspdH_fixedtheta(hessfn, jacfn, theta)

hou_spdH_fixedSig(hessfn, jacfn, Sig)

hou_spdH_fixedtheta_fixedSig(hessfn, jacfn, theta, Sig)

hou_fixedH_diagSig(hessfn, H)

nparams_ou_logdiagH(k)

nparams_brn(k)

nparams_ou_spdH_fixedSig(k)
```

Arguments

`parfn`	A function that maps from the user-parametrisation to the underlying Gaussian parameters. Each of them returns a vector of concatenated (Φ, w, V'), where V' is the lower triangular part of V, and accepts four arguments: a vector of parameters whose length is specified by the `pardims` argument to the `glinv_gauss` function, the branch length leading to the currently processing node, a vector of factors with three levels indicating which dimensions are missing or lost in the mother of the current node, and a vector of factors with the same three levels indicating missingness of the current node.
`jacfn`	A function that accepts the same arguments as `parfn` and returns the Jacobian of `parfn`.
`hessfn`	A function that accepts the same arguments as `parfns` and returns a list of three 3D arrays, named `Phi`, `w`, `V` respectively inside the list. `((hessfn)(...))$Phi[m,i,j]` contains the cross second-order partial derivative of Φ_m (here we treat the ma-

trix Φ as a column-major-flattened vector) with respect to the i-th and$j$-th parameters in the joint (H, θ, Σ'_x) vector, and `((hessfn)(...))$w[m,i,j]` and `((hessfn)(...))$V[m,i,j]` analogously contains second-order derivative with respect to w_m and V'_m.

`H`	A numerical vector containing the (flattened) fixed parameter H.
`theta`	A numerical vector containing the (flattened) fixed parameter $theta$.
`Sig`	A numerical vector containing the (flattened) fixed parameter Σ'_x.
`k`	An integer. The total number of dimensions of the multivariate traits.

Format

An object of class `list` of length 4.

Details

How reparametrisation and restriction works:

In the simplest form, without any restriction or reparametrisation, the user typically needs to pass `oupar`, `oujac`, `ouhess`, all of which are simply functions which maps from the OU parameters (H, θ, Σ'_x) to the Gaussian paramters (Φ_i, w_i, V'_i) for each node. For example:

```
mod.full = glinv(tree, x0, my_data,
                 parfns  = oupar,
                 pardims = nparams_ou(k),
                 parjacs = oujac,
                 parhess = ouhess)
```

If one would like to restrict H to only positively definite diagonal matrices, then the call should become

```
mod.pddiag = glinv(tree, x0, my_data,
                   parfns  = ou_logdiagH(oupar),
                   pardims = nparams_ou_logdiagH(k),
                   parjacs = dou_logdiagH(oujac),
                   parhess = hou_logdiagH(ouhess))
```

Note that there is a naming convention that `ou_*` should be applied to 'oupar', `dou_*` to 'oujac', and `hou_*` to 'ouhess'. `d` stands for 'derivative' and `h` stands for 'Hessian'.

In the above call, ou_logdiagH(oupar) accepts the `oupar` function as argument and returns a new function. This new function behaves the same way as oupar itself, except that it expects its first argument (which is the model parameters) to be of lower dimension, only consisting of (h, θ, Σ'_x) where h is the diagonal vector of H. The following example should be illustrative:

```
f = ou_logdiagH(oupar)
par.full = list(H     = matrix(c(3,0,0,2),2,2), # diagonal matrix
                theta = c(4,5),
                sig_x = c(1,0.1,1))
par.restricted = list(H     = log(diag(par.full$H)),
                      theta = par.full$theta,
                      sig_x = par.full$sig_x)
print(all.equal(f(unlist(par.restricted),1,NULL,NULL),
```

```
                                oupar(unlist(par.full),1,NULL,NULL)))
# [1] TRUE
```

Pre-defined restrictions: The following table summarises all the pre-defined ou_* functions. See oupar for precise meaning of the (H, θ, Σ'_x) mentioned below.

R function	Parameter Format after Restriction
brn*	Σ'_x. The Brownian motion. H and θ are zero, thus missing.
*_diagH_*	(h, θ, Σ'_x), with $h = diag(H)$, and H is a diagonal matrix
*_logdiagH_*	$(log(h), \theta, \Sigma'_x)$, with $h = diag(H)$, and H is a diagonal matrix
*_symH_*	(L, θ, Σ'_x), with L being lower-triangular part of the symmetric matrix H
*_spdH_*	(L, θ, Σ'_x), with L being Cholesky factor of the S.P.D. matrix H
*_logspdH_*	(L', θ, Σ'_x) where L' equals L, except that on the diagonals $L'_i = logL_i$
*_fixedH_*	(θ, Σ'_x). H is constant, hence missing
*_fixedtheta_*	(H, Σ'_x). θ is constant, hence missing
*_fixedSig_*	(H, θ). Σ_x is constant, hence missing
*_diagSig_*	(H, θ, s) where $s = diag(\Sigma'_x$, with Σ'_x being a diagonal matrix.

By Cholesky factor, we mean the only the non-zero part of the lower-triangular Cholesky factor. Restricting Σ'_x to a diagonal matrix means that Σ_x is also diagonal; and the variance of the Brownian motion is $log(diag(\Sigma'_x))$. In other words, the diagonal restriction is placed on Σ'_x, not Σ_x.

Finding a list of these restriction functions: One can use print(avail_restrictions) to see a list of all of these restriction function names.

Calling these restriction functions: All *ou_* or *brn* functions accepts the same arguemnts as ou_logdiagH, dou_logdiagH, hou_logdiagH, nparams_ou_logdiagH as shown in the Usage and Arguments section, except that:

1. If the reparametrisation contains any Cholesky decomposition (in other words, the function name contains spd or logspd) then in the Hessian-level reparameterisation function (named hou_*) an extra argument jacfn is required.
2. If the reparametrisation contains any fixed parameters, extra arguments H, theta, or Sig are required, depending what is fixed.

For example, in the Usage section, ou_logspdH_fixedtheta takes an extra argument theta because of (2), and hou_spdH_fixedSig takes extra argument two extra arguments because of both (1) and (2) are true.

---

rglinv *Simulate random trait values from models.*

Description

Simulate random trait values from the Gaussian branching process specified by mod.

Usage

```
rglinv(mod, par, Nsamp, simplify)

## S3 method for class 'glinv'
rglinv(mod, par, Nsamp = 1, simplify = TRUE)

## S3 method for class 'glinv_gauss'
rglinv(mod, par, Nsamp = 1, simplify = TRUE)
```

Arguments

mod	Either a glinv_gauss or glinv object.
par	Parameters underlying the simulation, in the same format as lik.glinv_gauss or lik.glinv.
Nsamp	Number of sample point to simulate.
simplify	If TRUE, rglinv.glinv returns an Nsamp-element list with each element being a tip-trait matrix; otherwise, rglinv.glinv returns an Nsamp-element list with each element being an n-element list of k-element trait vectors, where n is the number of tips and k is the dimension of each trait vector.

Value

A list containing Nsamp elements, each of which represents a sample point from the model mod. The format of each elements depends on the simplify argument.

---

set_tips *Set trait values at the tip for a* glinv_gauss *model.*

Description

If a glinv_gauss or glinv object were initalised with X=NULL, methods like lik will not work because it lacks actual data. In this case, the user should set the trait values using this method. If trait values were already set before, they will be replaced with the new trait values.

Usage

```
set_tips(mod, X)

## S3 method for class 'glinv_gauss'
set_tips(mod, X)

## S3 method for class 'glinv'
set_tips(mod, X)
```

Arguments

mod	A glinv_gauss or glinv object.
X	A matrix of trait values, in which X[p,n] stores the p-th dimension of the multivariate trait of the n-th tip of the phylogeny.

Details

X can contain any NA nor NaN if set_tips is called on a glinv model but this is will result in error if the method were called on a glinv_gauss model.

This method alters an underlying C structure, therefore has a mutable-object semantic. (See example).

Value

A model whose tip trait values are set.

Examples

```
tr = ape::rtree(10)
model  = glinv_gauss(tr, x0=c(0,0))  # The `X` argument is implicitly NULL
model2 = model                       # This is not copied!
traits = matrix(rnorm(20), 2, 10)
set_tips(model, traits)
```

---

varest *Estimate the variance-covariance matrix of the maximum likelihood estimator.*

Description

varest estimates the uncertainty of an already-computed maximum likelihood estimate.

Usage

```
varest(mod, ...)

## S3 method for class 'glinv'
varest(
  mod,
  fitted,
  method = "analytical",
  numDerivArgs = list(method = "Richardson", method.args = list(d = 0.5, r = 3)),
  store_gaussian_hessian = FALSE,
  control.mc = list(),
  ...
)
```

Arguments

mod	An object of class glinv
...	Not used.
fitted	Either an object returned by fit.glinv or a vector of length mod$nparams that contains the maximum likelihood estimate.
method	Either 'analytical', 'linear' or 'mc'. It specifies how the covariance matrix is computed.
numDerivArgs	Arguments to pass to numDeriv::jacobian. Only used if the user did not supply parjacs when constructing mod.
store_gaussian_hessian	If TRUE and method is not mc, the returned list will contain a (usually huge) Hessian matrix gaussian_hessian with respect to the Gaussian parameters Φ, w, V'. This option significantly increases the amount of memory the function uses, in order to store the matrix.
control.mc	A list of additional arguments to pass to the mc method.

Details

If method is analytical then the covariance matrix is estimated by inverting the negative analytically-computed Hessian at the maximum likelihood estimate; if it is mc then the estimation is done by using Spall's Monte Carlo simultaneous perturbation method; if it is linear then it is done by the "delta method", which approximates the user parameterisation with its first-order Taylor expansion.

The analytical method requires that parhess was specified when 'mod' was created. The linear method does not use the curvature of the reparameterisation and its result is sometimes unreliable; but it does not require the use of parhess. The mc method also does not need parjacs, but the it introduces an additional source complexity and random noise into the estimation; and a large number of sample may be needed.

The control.mc can have the following elements:

Nsamp Integer. Number of Monte Carlo iteration to run. Default is 10000.

c Numeric. Size of perturbation to the parameters. Default is 0.005.

quiet Boolean. Whether to print progress and other information or not. Default is TRUE.

Value

A list containing

vcov	The estimated variance-covariance matrix of the maximum likelihood estimator.
mlepar	The maximum likelihood estimator passed in by the user.
hessian	The Hessian of the log-likelihood at the maximum likelihood estimate. Only exists when method is not mc
gaussian_hessian	Optional, only exists when 'store_gaussian_hessian' is TRUE.

References

Spall JC. Monte Carlo computation of the Fisher information matrix in nonstandard settings. Journal of Computational and Graphical Statistics. 2005 Dec 1;14(4):889-909.

Index