Package ‘mvMORPH’

June 12, 2020

Type Package

Title Multivariate Comparative Tools for Fitting Evolutionary Models to Morphometric Data

Version 1.1.3

Date 2020-06-08

Author Julien Clavel, with contributions from Aaron King, and Emmanuel Paradis

Maintainer Julien Clavel <julien.clavel@hotmail.fr>


Depends R (>= 3.5.0), phytools, ape, corpcor, subplex

Imports stats, spam, graphics, glassoFast, parallel, pbmcapply

Suggests knitr, car

License GPL (>= 2.0)

URL https://github.com/JClavel/mvMORPH

VignetteBuilder knitr

NeedsCompilation yes

Repository CRAN

Date/Publication 2020-06-12 18:50:03 UTC

R topics documented:

mvMORPH-package .................................................. 2
aicw ................................................................. 3
coef.mvglst ......................................................... 5
EIC ................................................................. 6
**Description**

Fits of multivariate evolutionary models on trees (with one or multiple selective regimes) and time-series dedicated to morphometrics or biometric continuous data with covariation. Testing for a phylogenetic signal in a multivariate dataset (including fossil and/or extant taxa), fitting linear models to high-dimensional multivariate comparative data, changes in rate or mode of evolution of continuous traits, simulating multivariate traits evolution, computing the likelihood of multivariate models, accounts for measurement errors and missing data, and other things...

**Details**

Package: mvMORPH
Type: Package
Version: 1.1.3
Date: 2013-07-22
License: GPL (>=2.0)
**Author(s)**

Julien Clavel  
Maintainer: Julien Clavel <julien.clavel@hotmail.fr>

**References**


**See Also**

mvgls mvgls.pca manova.gls mvOU mvBM mvEB mvSHIFT mvOUTS mvRWTS mvSIM mvLL LRT halflife stationary estim aicw GIC EIC

---

### aicw

**Akaike weights**

**Description**

This function return the Akaike weights for a set of fitted models.

**Usage**

```r
aicw(x,...)
```

**Arguments**

- `x`: A list with the fitted objects or a list/vector of AIC
- `...`: Options to be passed through; e.g. `aicc=TRUE` when a list of fitted objects is provided.

**Details**

This function compute the Akaike weights for a set of model AIC or AICc. Akaike weights can be used for model comparison and model averaging.
Value

models  List of models
AIC  Akaike Information Criterion
diff  AIC difference with the best fit model
wi  Absolute weight
aicweights  Akaike weights (relative weights)

Author(s)

Julien Clavel

References


See Also

AIC mvMORPH

Examples

set.seed(1)
# Generating a random tree
tree<-pbtree(n=50)

#simulate the traits
sigma <- matrix(c(0.01,0.005,0.003,0.005,0.01,0.003,0.003,0.003,0.01),3)
theta<-c(0,0,0)
data<-mvSIM(tree, model="BM1", nsim=1, param=list(sigma=sigma, theta=theta))

## Fitting the models
# BM1 - General structure
fit1 <- mvBM(tree, data, model="BM1", method="pic")

# BM1 - No covariations
fit2 <- mvBM(tree, data, model="BM1", method="pic", param=list(constraint="diagonal"))

# BM1 - Equal variances/rates
fit3 <- mvBM(tree, data, model="BM1", method="pic", param=list(constraint="equal"))

results <- list(fit1,fit2,fit3)

# or
# results <- c(AIC(fit1), AIC(fit2), AIC(fit3))

# Akaike weights
aicw(results)

# AICc weights


```r
aicw(results, aicc=TRUE)
# we can compare the MSE...
# mean((fit1$sigma-sigma)^2)
# mean((fit3$sigma-sigma)^2)
```

---

**coef.mvgls**

*Extract multivariate gls model coefficients*

---

**Description**

Returns the coefficients of a linear model fit of class 'mvgls'.

**Usage**

```r
## S3 method for class 'mvgls'
coef(object, ...)
```

**Arguments**

- `object`:
  - an object of class 'mvgls' obtained from a `mvgls` fit.
- `...`:
  - other arguments (not used).

**Value**

The coefficients extracted from the model.

**Note**

For an intercept only model with phylogenetic structure this correspond to the ancestral states.

**Author(s)**

J. Clavel

**See Also**

`vcov.mvgls` `residuals.mvgls` `fitted.mvgls` `mvgls`
EIC

Example

EIC

Description

The EIC (Ishiguro et al. 1997, Kitagawa & Konishi 2010), uses bootstrap to estimate the bias term of the Extended Information Criterion. This criterion allows comparing models fit by Maximum Likelihood (ML) or Penalized Likelihood (PL).

Usage

```
EIC(object, nboot=100L, nbcores=1L, ...)  
```

Arguments

- **object**: An object of class 'mvgls'. See ?mvgls
- **nboot**: The number of bootstrap replicates used for estimating the EIC.
- **nbcores**: The number of cores used to speed-up the computations (uses the 'parallel' package)
- **...**: Options to be passed through.

Details

The Extended Information Criterion (EIC) allows comparing the fit of various models estimated by Penalized Likelihood or Maximum Likelihood (see ?mvgls). Similar to the GIC or the more common AIC, the EIC has the form:

\[ EIC = -2 \cdot \text{Likelihood} + 2 \cdot \text{bias} \]

Where Likelihood corresponds to either the full or the restricted likelihood (see the note below), and the bias term is estimated by (semi-parametric) bootstrap simulations rather than by using analytical or approximate solutions (see for instance ?GIC). The smaller the EIC, the better is the model. With small sample sizes, the variability around the bootstrap estimates is expected to be high, and one must increase the number of bootstrap replicates. Parallel computation (argument nbcores) allows to speed-up the computations.

Note: for models estimated by REML, it is generally not possible to compare the restricted likelihood when they have different fixed effects. However, it is possible to compare models with different fixed effects by using the full likelihood evaluated with the REML estimates (see e.g. Yafune et al. 2006, Verbyla 2019). Both options - evaluating the restricted likelihood or the full likelihood with parameters estimated by REML - are available through the REML argument in the EIC function. The default has been set to REML=FALSE to allow the comparison of models with different fixed effects using the full likelihood evaluated with the REML estimates (see Verbyla 2019).
**Value**

a list with the following components

- **LogLikelihood**
  the log-likelihood estimated for the model with estimated parameters

- **EIC**
  the EIC criterion

- **se**
  the standard error of the bias term estimated by bootstrap

- **bias**
  the values of the bias term estimated from the bootstrapped replicates to compute the EIC

**Author(s)**

J. Clavel

**References**


**See Also**

GIC mvgls mvgls.pca

**Examples**

```r
set.seed(1)
n <- 32 # number of species
p <- 50 # number of traits
tree <- pbtree(n=n) # phylogenetic tree
R <- crossprod(matrix(runif(p*p), ncol=p)) # a random symmetric matrix (covariance)
# simulate a dataset
Y <- mvSIM(tree, model="BM1", nsim=1, param=list(sigma=R))
```

estim <- mvgls(Y~1, tree=tree, model="BM", method="H&L")
fit2 <- mvgls(Y~1, tree=tree, model="OU", method="H&L")

EIC(fit1); EIC(fit2)

estim

Ancestral states reconstructions and missing value imputation with
phylogenetic/time-series models

Description

This function imputes the missing cases (NA values) according to a given phylogenetic model
(object of class "mvmorph"); it can also do ancestral state reconstruction.

Usage

estim(tree, data, object, error=NULL, asr=FALSE)

Arguments

tree Phylogenetic tree (an object of class "phylo" or "simmap") or a time-series.
data Matrix or data frame with species in rows and continuous traits with missing
cases (NA values) in columns (preferentially with names and in the same order
than in the tree).
object A fitted object from an mvMORPH model (class "mvmorph").
error Matrix or data frame with species in rows and continuous traits sampling vari-
ance (squared standard errors) in columns.
asr If asr=TRUE, the ancestral states are estimated instead of the missing cases.

Details

Missing observations for species in a phylogenetic tree are estimated according to a given evolution-
ary model (and parameters). Multivariate models are useful to recover the variance and covariance
structure of the dataset to be imputed.

When asr=TRUE, the estimates, their variances and standard errors are those of the ancestral states
at each node of the tree (this option is not available for the time-series). Note that if there are
missing cases, they are first imputed before estimating the ancestral states.

Value

estimates The imputed dataset
var Variance of the estimates
se Standard error of the estimates
NA_index Position of the missing cases in the dataset
estim

Author(s)
Julien Clavel

References

See Also
mvMORPH mvOU mvEB mvBM mvSHIFT

Examples

```r
## Simulated dataset
set.seed(14)
# Generating a random tree
tree<-pbtree(n=50)

# Setting the regime states of tip species
sta<-as.vector(c(rep("Forest",20),rep("Savannah",30))); names(sta)<-tree$tip.label

# Making the simmap tree with mapped states
tree<-make.simmap(tree,sta , model="ER", nsim=1)
col<-c("blue","orange"); names(col)<-c("Forest","Savannah")

# Plot of the phylogeny for illustration
plotSimmap(tree,col,fsize=0.6,node.numbers=FALSE,lwd=3, pts=FALSE)

# Simulate two correlated traits evolving along the phylogeny
traits<-mvSIM(tree,nsim=1, model="BMM", param=list(sigma=list(matrix(c(2,1,1,1.5),2,2),
matrix(c(4,1,1,4),2,2)), names_traits=c("head.size","mouth.size")))

# Introduce some missing cases (NA values)
data<-traits
data[8,2]<-NA
data[25,1]<-NA

# Fit of model 1
fit<-mvBM(tree,data,model="BMM")

# Estimate the missing cases
imp<-estim(tree, data, fit)

# Check the imputed data
imp$estim[1:10,]

## We want the ancestral states values at each nodes:
```
nodelabels() # To see where the nodes are situated
imp2<estim(tree, data, fit, asr=TRUE)
# Check the 10 firsts ancestral states
imp2$estim[1:10,]

---

**fitted.mvgls**

*Extract multivariate gls model fitted values*

**Description**

Returns the fitted values of a linear model of class 'mvgls'.

**Usage**

```r
## S3 method for class 'mvgls'
fitted(object, ...)
```

**Arguments**

- `object` an object of class 'mvgls' obtained from a `mvgls` fit.
- `...` other arguments (not used).

**Value**

The fitted values extracted from the model.

**Author(s)**

J. Clavel

**See Also**

`vcov.mvgls` `residuals.mvgls` `coef.mvgls` `mvgls`
**GIC**

*Generalized Information Criterion (GIC) to compare models fit with `mvgls` by Maximum Likelihood (ML) or Penalized Likelihood (PL).*

---

**Description**

The GIC (Konishi & Kitagawa 1996) allows comparing models fit by Maximum Likelihood (ML) or Penalized Likelihood (PL).

**Usage**

```r
GIC(object, ...)```

**Arguments**

- `object` An object of class `mvgls`. See `?mvgls`
- `...` Options to be passed through.

**Details**

The Generalized Information Criterion (GIC) allows comparing the fit of various models estimated by Penalized Likelihood (see `?mvgls`). See also the `gic.criterion` function in the RPANDA package. Note that the current implementation of the criterion has not been tested for multiple predictors comparison. Prefer simulation based comparisons instead.

**Value**

A list with the following components:

- `LogLikelihood` the log-likelihood estimated for the model with estimated parameters
- `GIC` the GIC criterion
- `bias` the value of the bias term estimated to compute the GIC

**Author(s)**

J. Clavel

**References**


See Also

mvgls mvgls.pca

Examples

```r
set.seed(1)
n <- 32 # number of species
p <- 50 # number of traits
tree <- pmtree(n=n) # phylogenetic tree
R <- crossprod(matrix(rnorm(p*p), ncol=p)) # a random symmetric matrix (covariance)
# simulate a dataset
Y <- mvSIM(tree, model="BM1", nsim=1, param=list(sigma=R))

fit1 <- mvgls(Y~1, tree=tree, model="BM", method="H&L")
fit2 <- mvgls(Y~1, tree=tree, model="OU", method="H&L")

GIC(fit1); GIC(fit2)
```

halflife

The phylogenetic half-life for an Ornstein-Uhlenbeck process

Description

This function returns the phylogenetic half-life for an Ornstein-Uhlenbeck process (object of class "ou").

Usage

```r
halflife(object)
```

Arguments

```r
object Object fitted with the "mvOU" function.
```

Details

The phylogenetic half-life describes the time to move halfway from the ancestral state to the primary optimum (Hansen, 1997). The multivariate counterpart is computed on the eigenvalues of the "selection" matrix (Bartoszek et al. 2012).

Value

The phylogenetic half-life computed from each eigenvalues (or alpha for the univariate case)
halflife

Author(s)

Julien Clavel

References


See Also

mvMORPH mvOU stationary

Examples

# Simulated dataset
set.seed(14)
# Generating a random tree
tree<-pbtree(n=50)

# Setting the regime states of tip species
sta<as.vector(c(rep("Forest",20),rep("Savannah",30))); names(sta)<-tree$tip.label

# Making the simmap tree with mapped states
tree<-make.simmap(tree,sta, model="ER", nsim=1)
col<-c("blue","orange"); names(col)<-c("Forest","Savannah")

# Plot of the phylogeny for illustration
plotSimmap(tree,col,fsize=0.6,node.numbers=FALSE,lwd=3, pts=FALSE)

# Simulate the traits
alpha<-matrix(c(2,0.5,0.5,1),2)
sigma<-matrix(c(0.1,0.05,0.05,0.1),2)
theta<-c(2,3,1,1.3)
data<-mvSIM(tree, param=list(sigma=sigma, alpha=alpha, ntraits=2, theta=theta, names_traits=c("head.size","mouth.size")), model="OUM", nsim=1)

## Fitting the models
# OUM - Analysis with multiple optima
result<-mvOU(tree, data)

halflife(result)
Likelihood Ratio Test

Description
This function compares the fit of two nested models of trait evolution with a loglikelihood-ratio statistic.

Usage
LRT(model1, model2, echo = TRUE, ...)

Arguments

- **model1**: The most parameterized model. A fitted object from an mvMORPH model.
- **model2**: The second model under comparison (fitted object).
- **echo**: Whether to return the result or not.
- **...**: Options to be passed through. (Not yet available)

Details
The LRT function extracts the log-likelihood of two nested models to compute the loglikelihood-ratio statistic which is compared to a Chi-square distribution. Note that if the models are not nested, the LRT is not an appropriate test and you should rely instead on Information criteria, evidence ratios, or simulated distributions (e.g., Lewis et al. 2011). This can be achieved using the simulate function (see examples below).

Value

- **pval**: The p-value of the LRT test (comparison with Chi-square distribution).
- **ratio**: The LRT (Loglikelihood-ratio test) statistic.
- **ddf**: The number of degrees of freedom between the two models.
- **model1**: Name of the first model.
- **model2**: Name of the second model.

Note
When comparing BM models to OU models, the LRT test might not be at it’s nominal level. You should prefer a simulations based test.

Author(s)
Julien Clavel
References


See Also

mvMORPH mvOU mvEB mvBM mvSHIFT

Examples

```r
## Simulated dataset
set.seed(14)
# Generating a random tree
tree<-pbtree(n=50)

# Setting the regime states of tip species
sta<-as.vector(c(rep("Forest",20),rep("Savannah",30))); names(sta)<-tree$tip.label

# Making the simmap tree with mapped states
tree<-make.simmap(tree,sta , model="ER", nsim=1)
col<-c("blue","orange"); names(col)<-c("Forest","Savannah")

# Plot of the phylogeny for illustration
plotSimmap(tree,col,fsize=0.6,node.numbers=FALSE,lwd=3, pts=FALSE)

# Simulate two correlated traits evolving along the phylogeny
traits<-mvSIM(tree,nsim=1, model="BMM", param=list(sigma=list(matrix(c(2,1,1,1.5),2,2),
                   matrix(c(4,1,1,4),2,2)), ntraits=2, names_traits=c("head.size","mouth.size"))

# Fit of model 1
mod1<-mvBM(tree,traits,model="BMM")

# Fit of model 2
mod2<-mvBM(tree,traits,model="BM1")

# comparing the fit using LRT...
LRT(mod1,mod2)

# Simulation based test
nsim = 500
boot <- simulate(mod2, tree=tree, nsim=nsim)
simulations <- sapply(1:nsim, function(i){
  mod1boot<-mvBM(tree, boot[[i]], model="BMM", diagnostic=FALSE, echo=FALSE)
  mod2boot<-mvBM(tree, boot[[i]], model="BM1", diagnostic=FALSE, echo=FALSE, method="pic")
  2*(mod1boot$LogLik-mod2boot$LogLik)
})
```
# Compute the p-value
LRT_stat<-(2*((mod1$LogLik-mod2$LogLik)))
mean(simulations>=LRT_stat)

plot(density(simulations), main="Non-parametric LRT");
abline(v=LRT_stat, col="red")

---

**manova.gls**

**Multivariate Analysis of Variance**

**Description**

Performs a Multivariate Analysis of Variance (MANOVA) on an object fitted by the `mvgls` function. With the regularized methods by penalized likelihood implemented in `mvgls` (ridgeArch penalty), this function can be used to compare model fit on high-dimensional datasets (where the number of variable is larger than the number of observations).

**Usage**

```r
manova.gls(object, test=c("Pillai", "Wilks", "Hotelling-Lawley", "Roy"),
          type=c("I", "II", "III"), nperm=999L, L=NULL, ...)
```

**Arguments**

- `object`: A model fit obtained by the `mvgls` function.
- `test`: The multivariate test statistic to compute - "Wilks", "Pillai", "Hotelling-Lawley", or "Roy"
- `type`: The type of test (sums of squares and cross-products) - "I", "II", or "III"
- `nperm`: The number of permutations used for building the null distribution of the chosen statistic. Permutation is the only available approach for high-dimensional PL models, but either permutations or parametric tests can be used with maximum likelihood (method "LL" in `mvgls`)
- `L`: A (contrasts) matrix or a vector giving linear combinations of the coefficients rows.
- `...`: Further arguments to be passed through. (e.g., `nbcores=2L` to provide the number of cores used for parallel calculus; `parametric=FALSE` to obtain permutation instead of parametric tests for maximum likelihood fit; `verbose=TRUE` to display a progress bar during permutations; `rhs=0` the "right-hand-side" vector for general linear hypothesis testing. See details)
manova.gls

Details

manova.gls allows performing multivariate tests (e.g. Pillai’s, Wilks, Hotelling-Lawley and Roy largest root) on generalized least squares (GLS) linear model (objects of class "mvgls") fit by either maximum likelihood (method="LL") or penalized likelihood (method="PL-LOO") using the mvgls function.

General Linear Hypothesis of the form:

\[ LB = O \]

an \( L \) matrix specifying linear combinations of the model coefficients (\( B \)) can be provided through the argument \( L \). The right-hand-side matrix \( O \) is a constant matrix (of zeros by default) that can be provided through the argument \( rhs \).

Permutations on high-dimensional datasets is time consuming. You can use the option \( nbcores \) to parallelize the calculus over several cores using forking in UNIX platforms (default is \( nbcores=1 \)). Estimated time to completion is displayed when \( verbose=TRUE \).

Value

An object of class ‘manova.mvgls’ which is usually printed. It contains a list including the following components:

- \( test \) the multivariate test statistic used
- \( type \) the type of tests used to compute the SSCP matrices
- \( stat \) the statistic calculated for each terms in the model
- \( pvalue \) the pvalues calculated for each terms in the model

Note

For PL methods, only the "RidgeArch" penalty is allowed for now.

Author(s)

J. Clavel

References


Clavel, J., Morlon, H. 2020. Reliable phylogenetic regressions for multivariate comparative data: illustration with the MANOVA and application to the effect of diet on mandible morphology in phyllostomid bats. Systematic Biology (DOI:10.1093/sysbio/syaa010)

See Also

mvgls, GICEIC
mv.Precalc

Model parameterization for the various mvMORPH functions

Description
This function allows computing the fixed parameters or objects needed by the mvMORPH functions. This could be useful for bootstrap-like computations (see exemple)

Usage
mv.Precalc(tree, nb.traits = 1, scale.height = FALSE, param = list(pivot = "MMD", method = c("sparse"), smean = TRUE, model = "OUM"))

Arguments
- tree: A "phylo" (or SIMMAP like) object representing the tree for which we want to precalculate parameters.
- nb.traits: The number of traits involved in the subsequent analysis.
- scale.height: Whether the tree should be scaled to unit length or not.
- param: A list of parameters used in the computations (see details)
mv.Precalc

Details

The mv.Precalc function allows the pre-computation of the fixed parameters required by the different mvMORPH models (e.g., the design matrix, the vcv matrix, the sparsity structure...). In the "param" list you should provide the details about the model fit:

- model name (e.g., "OUM", "OU1")
- method (which kind of algorithm is used for computing the log-likelihood).
- smean (whether there is one ancestral state per trait or per selective regimes - for mvBM only).

Additional parameters can be fixed:

- root (estimation of the ancestral state for the Ornstein-Uhlenbeck model; see ?mvOU).
- pivot (pivot method used by the "sparse" matrix method for computing the log-likelihood; see ?spam).

Value

An object of class "mvmorph.precalc" which can be used in the "precalc" argument of the various mvMORPH functions.

Note

This function is mainly used internally; it is still in development. A misuse of this functions can result in a crash of the R session.

Author(s)

Julien Clavel

See Also

mvMORPH mvOU mvEB mvBM mvSHIFT mvLL

Examples

set.seed(14)
# Generating a random tree
tree<-pbtree(n=50)

# Simulate two correlated traits evolving along the phylogeny according to a
# Ornstein-Uhlenbeck process
alpha<-matrix(c(2,1,1,1.3),2,2)
sigma<-matrix(c(1,0.5,0.5,0.8),2,2)
theta<-c(3,1)
nsim<-50
simul<-mvSIM(tree,param=list(sigma=sigma, alpha=alpha, ntraits=2, theta=theta,
                        names_traits=c("head.size","mouth.size")), model="OU1", nsim=nsim)

# Do the pre-calculations
precal<-mv.Precalc(tree,nb.traits=2, param=list(method="sparse",model="OU1", root=FALSE))
mvOU(tree, simul[[x]], method="sparse", model="OU1", precalc=precal,  
param=list(decomp="cholesky"), echo=FALSE, diagnostic=FALSE)

### Bootstrap

# Fit the model to the "nsim" simulated datasets
results<-lapply(1:nsim,function(x){
  mvOU(tree, simul[[x]], method="sparse", model="OU1", precalc=precal,  
    param=list(decomp="cholesky"), echo=FALSE, diagnostic=FALSE)
})

### Use parallel package
library(parallel)
if(.Platform$OS.type == "unix"){
  number_of_cores<-2L # Only working on Unix systems
}else{
  number_of_cores<-1L
}

results<-mclapply(simul, function(x){
  mvOU(tree, x, method="sparse", model="OU1", precalc=precal,  
    param=list(decomp="cholesky"), echo=FALSE, diagnostic=FALSE)
}, mc.cores = getOption("mc.cores", number_of_cores))

# Summarize (we use the generic S3 method "logLik" to extract the log-likelihood)
loglik<-sapply(results,logLik)
hist(loglik)

---

**mvBM**

*Multivariate Brownian Motion models of continuous traits evolution*

**Description**

This function allows the fitting of multivariate multiple rates of evolution under a Brownian Motion model. This function can also fit constrained models.

**Usage**

```
mvBM(tree, data, error = NULL, model = c("BMM", "BM1"),  
param = list(constraint = FALSE, smean = TRUE, trend=FALSE),  
method = c("rpf", "pic", "sparse", "inverse", "pseudoinverse"),  
scale.height = FALSE, optimization = c("L-BFGS-B", "Nelder-Mead", "subplex"),  
control = list(maxit = 20000), precalc = NULL, diagnostic = TRUE, echo = TRUE)
```
Arguments

- **tree**: Phylogenetic tree in SIMMAP format by default. A "phylo" object can also be used with the "BM1" model.
- **data**: Matrix or data frame with species in rows and continuous traits in columns (preferentially with names and in the same order than in the tree). NA values are allowed with the "rpf", "inverse", and "pseudoinverse" methods.
- **error**: Matrix or data frame with species in rows and continuous trait sampling variance (squared standard errors) in columns.
- **model**: "BMM" for multi-rate and multi-selective regimes, and "BM1" for a unique rate of evolution per trait.
- **param**: List of arguments to be passed to the function. See details.
- **method**: Choose between "rpf", "sparse", "inverse", "pseudoinverse", or "pic" for log-likelihood computation during the fitting process. See details.
- **scale.height**: Whether the tree should be scaled to unit length or not.
- **optimization**: Methods used by the optimization routines (see ?optim and ?subplex for details). The "fixed" method returns the log-likelihood function only.
- **control**: Max. bound for the number of iteration of the optimizer; other options can be fixed in the list (see ?optim or ?subplex).
- **diagnostic**: Whether the diagnostics of convergence should be returned or not.
- **echo**: Whether the results must be returned or not.

Details

The mvBM function fits a multivariate Brownian Motion (BM) process, with unique or multiple BM rates (see O’Meara et al., 2006; Revell and Collar, 2009). Note that the function uses the non-censored approach of O’Meara et al. (2006) by default (i.e., a common ancestral state is assumed for the different regimes), but it is possible to specify multiple ancestral states (i.e., one for each regimes) through the "smean" parameter (smean=FALSE) in the "param" list.

The "method" argument allows the user to try different algorithms for computing the log-likelihood. The "rpf" and "sparse" methods use fast GLS algorithms based on factorization for avoiding the computation of the inverse of the variance-covariance matrix and its determinant involved in the log-likelihood estimation. The "inverse" approach uses the "stable" standard explicit computation of the inverse and determinant of the matrix and is therefore slower. The "pseudoinverse" method uses a generalized inverse that is safer for matrix near singularity but highly time consuming. The "pic" method uses a very fast algorithm based on independent contrasts. It should be used with strictly dichotomic trees (i.e., no polytomies) and is currently not available for the multivariate "BMM" model. See ?mvLL for more details on these computational methods.

The "param" list arguments:

- **constraint**: The "constraint" argument in the "param" list allows the user to compute the joint likelihood for each trait by assuming they evolved independently (constraint="diagonal", or constraint="equaldiagonal"). If constraint="equal", the sigma values are constrained to be the same for each studied trait using the constrained Cholesky decomposition proposed by Adams...
or a separation strategy based on spherical parameterization (when \(p>2\)) because of an unstable behavior observed for the constrained Cholesky (Clavel et al. 2015).

This approach is extended here to the multi-rate case by specifying that the rates must be the same in different parts of the tree (common selective regime). It's also possible to constraint the rate matrices in the "BMM" model to share the same eigen-vectors (constraint="shared"); the same variance but different covariances (constraint="variance"); the same correlation but different variances (constraint="correlation"); or to fit a model with different but proportional rates matrices (constraint="proportional").

Finally, user-defined constrained models can be specified through a numeric matrix (square and symmetric) with integer values taken as indices of the parameters. For instance, for three traits:

```r
constraint=matrix(c(1,3,3,2,3,3,3,2,3),3).
```

Covariances constrained to be zero are introduced by NA values, e.g.,

```r
constraint=matrix(c(1,4,4,2,NA,4,NA,3),3).
```

Difference between two nested fitted models can be assessed using the "LRT" function. See example below and ?LRT.

"decomp" - For the general case (unconstrained models), the sigma matrix is parameterized by various methods to ensure its positive definiteness (Pinheiro and Bates, 1996). These methods are the "cholesky", "eigen+", and "spherical" parameterizations.

"smean" - Default set to TRUE. If FALSE, the ancestral state for each selective regime is estimated (e.g., Thomas et al., 2006).

"trend" - Default set to FALSE. If TRUE, the ancestral state is allowed to drift linearly with time. This model is identifiable only with non-ultrametric trees. Note that it is possible to provide a vector of integer indices to constrain the estimated trends (see the vignettes).

"sigma" - Starting values for the likelihood estimation. By default the theoretical expected values are used as starting values for the likelihood optimization (for measurement errors, multiple rates,...). The user can specify starting values with a list() object for the "BMM" model (e.g., two objects in the list for a two-regime analysis), or a simple vector of values for the "BM1" model. The parameterization is done using various factorizations for symmetric matrices (e.g., for the "decomp" argument; Pinheiro & Bates, 1996). Thus, you should provide \(p*(p+1)/2\) values, with \(p\) the number of traits (e.g., random numbers or the values from the cholesky factor of a symmetric positive definite sigma matrix; see example below). If a constrained model is used, the number of starting values is \((p*(p-1)/2)+1\).

If no selective regime is specified the function works only with the model "BM1".

N.B.: Mapping of ancestral states can be done using the "make.simmap", "make.era.map" or "paintSubTree" functions from the "phytools" package.

Value

| LogLik | The log-likelihood of the optimal model. |
| AIC | Akaike Information Criterion for the optimal model. |
| AICc | Sample size-corrected AIC. |
| theta | Estimated ancestral states. |
| sigma | Evolutionary rate matrix for each selective regime. |
**convergence**  Convergence status of the optimizing function; "0" indicates convergence (See `?optim` for details).

**hess.values**  Reliability of the likelihood estimates calculated through the eigen-decomposition of the hessian matrix. "0" means that a reliable estimate has been reached. (See `?mvOU`).

**param**  List of model fit parameters (optimization, method, model, number of parameters...).

**llik**  The log-likelihood function evaluated in the model fit "$llik(par, root.mle=TRUE)".

**Note**

The "pic" method is not yet implemented for the multivariate "BMM" model.

**Author(s)**

Julien Clavel

**References**


**See Also**

mvMORPH mvglss mvOU mvEB mvOUTS mvSIM LRT optim brownie.lite evol.vcv make.simmap make.era.map paintSubTree

**Examples**

```r
# Simulated dataset
set.seed(14)
# Generating a random tree
tree<-pbtree(n=50)

# Setting the regime states of tip species
sta<-as.vector(c(rep("Forest",20),rep("Savannah",30))); names(sta)<-tree$tip.label

# Making the simmap tree with mapped states
```
tree<-make.simmap(tree, sta, model="ER", nsim=1)
col<-c("blue","orange"); names(col)<-c("Forest","Savannah")

# Plot of the phylogeny for illustration
plotSimmap(tree, col, fsize=0.6, node.numbers=FALSE, lwd=3, pts=FALSE)

# Simulate the traits
sigma<-matrix(c(0.1,0.05,0.05,0.1),2)
theta<-c(0,0)
data<-mvSIM(tree, param=list(sigma=sigma, ntraits=2, theta=theta,
        names_traits=c("head.size","mouth.size")), model="BM1", nsim=1)

## Fitting the models

# BMM - Analysis with multiple rates
mvBM(tree, data)

# BM1 - Analysis with a unique rate matrix
fit1<-mvBM(tree, data, model="BM1", method="pic")

# BM1 constrained
fit2<-mvBM(tree, data, model="BM1", method="pic", param=list(constraint="equal"))

# Comparison with LRT test
LRT(fit1,fit2)

# Random starting values
mvBM(tree, data, model="BMM", method="sparse", param=list(sigma=list(runif(3), runif(3))))

# Specified starting values (from the Cholesky factor)
chol_factor<-chol(sigma)
starting_values<-chol_factor[upper.tri(chol_factor,TRUE)]
mvBM(tree, data, model="BMM", method="sparse",
         param=list( sigma=list(starting_values, starting_values)))

# Multiple mean
mvBM(tree, data, model="BMM", method="sparse", param=list(smean=FALSE))

# Introduce some missing cases (NA values)
data2<-data
data2[8,2]<-NA
data2[25,1]<-NA
mvBM(tree, data2, model="BM1")

## FAST FOR THE UNIVARIATE CASE!!

set.seed(14)
tree2<-pbtree(n=5416) # Number of Mammal species
# Setting the regime states of tip species
sta<-as.vector(c(rep("group_1",2000),rep("group_2",3416))); names(sta)<-tree2$tip.label
# Making the simmap tree with mapped states

tree2<-make.simmap(tree2, sta , model="ER", nsim=1)
col<-c("blue", "orange"); names(col)<-c("Group_1", "Group_2")
plotSimmap(tree2,col,fsize=0.6,node.numbers=FALSE,lwd=3, pts=FALSE)

# Simulate a trait evolving by brownian motion on the tree
trait<-rTraitCont(tree2)

# Fitting the models
mvBM(tree2, trait, model="BMM", method="pic")
mvBM(tree2, trait, model="BM1", method="pic")

---

mvEB  

**Multivariate Early Burst model of continuous traits evolution**

### Description

This function fits to a multivariate dataset of continuous traits a multivariate Early Burst (EB) or ACDC models of evolution.

### Usage

```r
mvEB(tree, data, error = NULL, param = list(up = 0), method =
c("rpf", "sparse", "inverse", "pseudoinverse", "pic"), scale.height =
FALSE, optimization = c("Nelder-Mead", "L-BFGS-B", "subplex"),
control = list(maxit = 20000), precalc = NULL, diagnostic = TRUE,
echo = TRUE)
```

### Arguments

- **tree**  
  Phylogenetic tree (phylo object).
- **data**  
  Matrix or data frame with species in rows and continuous traits in columns (preferentially with names and in the same order than in the tree). NA values are allowed with the "rpf", "inverse", and "pseudoinverse" methods.
- **error**  
  Matrix or data frame with species in rows and continuous trait sampling variance (squared standard errors) in columns.
- **param**  
  List of arguments to be passed to the function. See details.
- **method**  
  Choose between "rpf", "sparse", "inverse", "pseudoinverse", or "pic" for computing the log-likelihood during the fitting process. See details.
- **scale.height**  
  Whether the tree should be scaled to unit length or not.
- **optimization**  
  Methods used by the optimization routines (see ?optim and ?subplex for details). The "fixed" method returns the log-likelihood function only.
control  Max. bound for the number of iteration of the optimizer; other options can be fixed in the list (see ?optim or ?subplex for details).
diagnostic  Whether the diagnostics of convergence should be returned or not.
echo  Whether the results must be returned or not.

Details

The Early Burst model (Harmon et al. 2010) is a special case of the ACDC model of Blomberg et al. (2003). Using an upper bound larger than zero transform the EB model to the accelerating rates of character evolution of Blomberg et al. (2003).

The "method" argument allows the user to try different algorithms for computing the log-likelihood. The "rpf" and "sparse" methods use fast GLS algorithms based on factorization for avoiding the computation of the inverse of the variance-covariance matrix and its determinant for the log-likelihood estimation. The "inverse" approach uses the "stable" standard explicit computation of the inverse and determinant of the matrix and is therefore slower. The "pseudoinverse" method uses a generalized inverse that is safer for matrix near singularity but highly time consuming. The "pic" method uses a very fast algorithm based on independent contrasts. See ?mvLL for more details on these computational methods.

The "param" list can be used to set the lower (low) and upper (up, default value is 0 - i.e., Early Burst model) bounds for the estimation of the exponential rate (beta). The default lower bound for decelerating rates (as assumed in Early Burst) is fixed as log(min.rate) / T, where T is the depth of the tree and min.rate is the minimum rate that could be assumed for the model (following Slater and Pennell, 2014; log(10^-5)/T). Bounds may need to be adjusted by the user for specific cases.

Starting values for "sigma" and "beta" could also be provided through the "param" list.

Value

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LogLik</td>
<td>The log-likelihood of the optimal model.</td>
</tr>
<tr>
<td>AIC</td>
<td>Akaike Information Criterion for the optimal model.</td>
</tr>
<tr>
<td>AICc</td>
<td>Sample size-corrected AIC.</td>
</tr>
<tr>
<td>theta</td>
<td>Estimated ancestral states.</td>
</tr>
<tr>
<td>beta</td>
<td>Exponent rate (of decay or increase).</td>
</tr>
<tr>
<td>sigma</td>
<td>Evolutionary rate matrix for each selective regimes.</td>
</tr>
<tr>
<td>convergence</td>
<td>Convergence status of the optimizing function; &quot;0&quot; indicates convergence (see ?optim for details).</td>
</tr>
<tr>
<td>hess.values</td>
<td>Reliability of the likelihood estimates calculated through the eigen-decomposition of the hessian matrix. &quot;0&quot; means that a reliable estimate has been reached. (see ?mvOU for details).</td>
</tr>
<tr>
<td>param</td>
<td>List of model fit parameters (optimization, method, model, number of parameters...).</td>
</tr>
<tr>
<td>llk</td>
<td>The log-likelihood function evaluated in the model fit &quot;$llk(par, root.mle=TRUE)&quot;.</td>
</tr>
</tbody>
</table>
mvEB

Note

The derivative-free "Nelder-Mead" optimization method is used as default setting instead of "L-BFGS-B".

Author(s)

Julien Clavel

References


See Also

mvMORPH mvgls mvOU mvBM mvSHIFT mvOUTS mvRWTS mvSIM optim

Examples

# Simulated dataset
set.seed(14)
# Generating a random tree
tree<- pbtree(n=50, scale=10)

# Simulate the traits
sigma<- matrix(c(0.1,0.05,0.05,0.1),2)
theta<- c(0,0)
beta<- -0.34 # 5 phylogenetic half-life ( log(2) / (10/5) )
data<- mvSIM(tree, param=list(sigma=sigma, beta=beta, ntraits=2, theta=theta,
             names_traits=c("head.size","mouth.size")), model="EB", nsim=1)

## Fitting the models
mvEB(tree, data)
mvEB(tree, data, method="pic")
mvEB(tree, data, method="pic", param=list(low=log(10^-5)/10)) # avoid internal estimation

# ACDC
# Note that the AC model is not differentiable from an OU model on ultrametric trees.
beta<- 0.34
data<- mvSIM(tree, param=list(sigma=sigma, beta=beta, ntraits=2, theta=theta,
             names_traits=c("head.size","mouth.size")), model="EB", nsim=1)
fit<-mvEB(tree, data, method="pic", param=list(up=2, low=-2))

logLik(fit)
AIC(fit)
summary(fit)

### mvgl5s

Fit linear model using Generalized Least Squares to multivariate (high-dimensional) data sets.

**Description**

This function uses maximum likelihood (or restricted likelihood) and penalized likelihood approaches to fit linear models where the errors are allowed to be correlated (i.e. a GLS model for serially correlated phylogenetic and time-series data). mvgl5s uses a penalized-likelihood (PL) approach (see descriptions in Clavel et al. 2019) to fit linear models to high-dimensional data sets (where the number of variables \( p \) is approaching or is larger than the number of observations \( n \)). The PL approach generally provides improved estimates compared to ML.

**Usage**

```r
mvgl5s(formula, data, tree, model, method=c("PL-LOOCV","LL"),
       REML=TRUE, ...)
```

**Arguments**

- **formula**: An object of class "formula" (a two-sided linear formula describing the model to be fitted. See for instance `?lm`).
- **data**: An optional list, data.frame or environment containing the variables in the model. If not found in `data` the variables are taken from the current environment. Prefer `list` for blocks of multivariate responses unless you’re specifying the response variables by their names using `cbind` with data.frame.
- **tree**: Phylogenetic tree (an object of class "phylo") or a time-series object (not yet available).
- **model**: The evolutionary model: "BM" is Brownian Motion, "OU" is Ornstein-Uhlenbeck, "EB" is Early Burst, and "lambda" is Pagel’s lambda transformation.
- **method**: The method used to fit the model. "PL-LOOCV" (or equivalently just "LOOCV") is the nominal leave one out cross-validation of the penalized log-likelihood, "LL" is the log-likelihood (used in the conventional ML and REML estimation). Two approximated LOOCV methods are also available: "H&L" and "Mahalanobis". The method "H&L" is a fast LOOCV approach based on Hoffbeck and Landgrebe (1996) tricks, and "Mahalanobis" is an approximation of the LOOCV score proposed by Theiler (2012). Both "H&L" and "Mahalanobis" work only with the "RidgeArch" penalty and for intercept only models (see details).
- **REML**: Use REML (default) or ML for estimating the parameters.
Options to be passed through. For instance the type of penalization: penalty="RidgeArch" (default), penalty="RidgeAlt", or penalty="LASSO". The target matrices used by "RidgeArch" and "RidgeAlt" penalizations: target="unitVariance", target="Variance" or target="null"... etc. (see details)

Details

mvgl is allows fitting various multivariable linear models to high-dimensional datasets (i.e. where the number of variables \( p \) is larger than \( n \) for which the residuals have a correlated structure (e.g. evolutionary models such as BM and OU). Models estimates are generally more accurate than maximum likelihood methods. Models fit can be compared using the GIC or EIC criterion (see ?GIC and ?EIC) and hypothesis testing can be performed using the manova.gls function.

The tree is assumed to be fully dichotomic and in "postorder", otherwise the functions multi2di and reorder.phylo are used internally.

The various arguments that can be passed through "...":

"penalty" - The "penalty" argument allow specifying the type of penalization used for regularization (described in Clavel et al. 2019). The various penalizations are: penalty="RidgeArch" (the default), penalty="RidgeAlt" and penalty="LASSO". The "RidgeArch" penalization shrink linearly the covariance matrix toward a target structure (see below for target). This penalization is generally fast and the tuning parameter is bounded between 0 and 1 (see van Wieringen & Peeters 2016). The "RidgeAlt" penalization scheme uses a quadratic ridge penalty to shrink the covariance matrix toward a specified target matrix (see target below and also see van Wieringen & Peeters 2016). Finally, the "LASSO" regularize the covariance matrix by estimating a sparse estimate of its inverse (Friedman et al. 2008). The computation of the solution for this penalization is computationally intensive. Moreover, this penalization scheme is not invariant to arbitrary rotations of the data.

"target" - This argument allows specifying the target matrix toward which the covariance matrix is shrunk for "Ridge" penalties. target="unitVariance" (for a diagonal target proportional to the identity) and target="Variance" (for a diagonal unequal variance target) can be used with both "RidgeArch" and "RidgeAlt" penalties. target="null" (a null target matrix) is only available for "RidgeAlt". Penalization with the "Variance" target shrink the eigenvectors of the covariance matrix and is therefore not rotation invariant. See details on the various target properties in Clavel et al. (2019).

"error" - If TRUE the measurement error (or intra-specific variance) is estimated from the data as a nuisance parameter (like in mixed models). It should probably be systematically used with empirical data. See also Housworth et al. 2004 and Clavel et al. 2019 for details on the proposed implementation.

"scale.height" - Whether the tree should be scaled to unit length or not.

"echo" - Whether the results must be returned or not.

"grid_search" - A logical indicating whether or not a preliminary grid search must be performed to find the best starting values for optimizing the log-likelihood (or penalized log-likelihood). User-specified starting values can be provided through the start argument. Default is TRUE.

"upper" - The upper bound for the parameter search with the "L-BFGS-B" method. See optim for details.

"lower" - The lower bound for the parameter search with the "L-BFGS-B" method. See optim for details.
"tol" - Minimum value for the regularization parameter. Singularities can occur with a zero value in high-dimensional cases. (default is NULL)

Value

An object of class `mvgls`. It contains a list including the following components:

- **coefficients**: a named vector of coefficients
- **residuals**: the residuals ("raw") of the model. That is response minus fitted values. Uses the `residuals(x, type="normalized")` function to obtain the normalized residuals.
- **fitted.values**: the fitted mean values
- **variables**: the variables used for model fit
- **sigma**: the estimated covariance (Pi) and precision (P) matrix, as well as the sample estimate (S)
- **model**: the evolutionary model. But more generally, the model used to specify the structure within the residuals
- **logLik**: either the (negative) log-likelihood when method="LL" or the cross-validated penalized likelihood
- **param**: the (evolutionary) model parameter estimates
- **tuning**: the regularization/tuning parameter of the penalized likelihood
- **mserr**: the estimated standard error when error=TRUE
- **start_values**: the starting parameters used for the optimization of the LL or PL
- **corrSt**: a list including the transformed tree, the determinant obtained from its covariance matrix and the normalized variables (by the inverse square root of the covariance matrix of the phylogenetic tree or the time-series)
- **penalty**: the penalty used for the penalized likelihood approach
- **target**: the target used with the "RidgeArch" or "RidgeAlt" penalized likelihood approaches
- **REML**: logical indicating if the REML (TRUE) or ML (FALSE) method has been used
- **opt**: optimizing function output. See optim

Author(s)

Julien Clavel

References


See Also

manova.gls EIC GIC mvgls.pca fitted.mvgls residuals.mvgls coef.mvgls vcov.mvgls

Examples

```r
set.seed(1)
n <- 32 # number of species
p <- 50 # number of traits (p>n)

tree <- pbtree(n=n, scale=1) # phylogenetic tree
R <- crossprod(matrix(runif(p*p), ncol=p)) # a random covariance matrix
# simulate a BM dataset
Y <- mvSIM(tree, model="BM1", nsim=1, param=list(sigma=R, theta=rep(0,p)))
data=list(Y=Y)

fit1 <- mvgls(Y~1, data=data, tree, model="BM", penalty="RidgeArch")
fit2 <- mvgls(Y~1, data=data, tree, model="OU", penalty="RidgeArch")
fit3 <- mvgls(Y~1, data=data, tree, model="EB", penalty="RidgeArch")

GIC(fit1); GIC(fit2); GIC(fit3) # BM have the lowest GIC value

# Testing for phylogenetic signal with model fit
signal <- mvgls(Y~1, data=data, tree, model="lambda", penalty="RidgeArch")
summary(signal)

# A High-dimensional dataset
p <- 200 # number of traits (p>n)

R <- crossprod(matrix(runif(p*p), ncol=p)) # a random symmetric matrix (covariance)
# simulate a BM dataset
Y <- mvSIM(tree, model="BM1", nsim=1, param=list(sigma=R, theta=rep(0,p)))
data=list(Y=Y)

# Fast LOOCV using "H&L" with RidgeArch penalization
summary(mvgls(Y~1, data=data, tree, model="BM", penalty="RidgeArch", method="H&L"))
```
Principal Component Analysis (PCA) based on GLS estimate of the traits variance-covariance matrix (possibly regularized).

Description

Performs a principal component analysis (PCA) on a regularized variance-covariance matrix obtained using the `mvgls` function. With "evolutionary" models, this performs the so-called phylogenetic PCA.

Usage

```r
mvgls.pca(object, plot=TRUE, ...)
```

Arguments

- `object`: A model fit obtained by the `mvgls` function.
- `plot`: Plot of the PC's axes. Default is TRUE (see details).
- `...`: Options to be passed through. (e.g., `axes=c(1,2)`, `col`, `pch`, `cex`, `mode="cov"` or `"corr"`, etc.)

Details

`mvgls.pca` allows computing a principal component analysis based on a GLS estimate of the covariance matrix (see `mvgls`). The phylogenetic PCA (following Revell 2009) is a special case obtained from the (possibly regularized) evolutionary variance-covariance matrix (see also the `phyl.pca_pl` function in RPANDA). In the high-dimensional case the contribution of the firsts PC axes tend to be overestimated with traditional maximum likelihood approaches. Penalized/regularized model fits reduce this bias and allow the estimation of various error structure models (see Clavel et al. 2018). Plotting options, the number of axes to display (`axes=c(1,2)` is the default), and whether the covariance (mode="cov") or correlation (mode="corr") should be used can be specified through the ellipsis "..." argument.

Value

a list with the following components

- `scores`: the PC scores
- `values`: the eigenvalues of the variance-covariance matrix estimated by `mvgls`
- `vectors`: the eigenvectors of the variance-covariance matrix estimated by `mvgls`
- `rank`: the rank of the estimated variance-covariance matrix
Note
Contrary to conventional PCA, the principal axes of the gls PCA are not orthogonal, they represent the main axes of independent (according to a given phylogenetic or time-series model) changes.

Author(s)
J. Clavel

References

See Also
mvgl.s, GIC, EIC

Examples

set.seed(1)
n <- 32 # number of species
p <- 30 # number of traits
tree <- pbtree(n=n) # phylogenetic tree
R <- crossprod(matrix(runif(p*p),p)) # a random symmetric matrix (covariance)

# simulate a dataset
Y <- mvSIM(tree, model="BM1", nsim=1, param=list(sigma=R))

# The conventional phylogenetic PCA
phylo_pca <- mvgl.s(Y~1, tree=tree, model="BM", method="LL")
mvgl.s.pca(phylo_pca, plot=TRUE)

# fit a multivariate Pagel lambda model with Penalized likelihood
fit <- mvgl.s(Y~1, tree=tree, model="lambda", method="LOO", penalty="RidgeAlt")

# Perform a regularized phylogenetic PCA using the model fit (Pagel lambda model)
pca_results <- mvgl.s.pca(fit, plot=TRUE)

# retrieve the scores
head(pca_results$scores)
mvLL

Multivariate (and univariate) algorithms for log-likelihood estimation of arbitrary covariance matrix/trees

Description

This function allows computing the log-likelihood and estimating ancestral states of an arbitrary tree or variance-covariance matrix with different algorithms based on GLS (Generalized Least Squares) or Independent Contrasts. Works for univariate or multivariate models. Can be wrapped for maximizing the log-likelihood of user-defined models.

Usage

mvLL(tree, data, error = NULL, method = c("pic", "rpf", "sparse", "inverse", "pseudoinverse"), param = list(estim = TRUE, mu = 0, sigma = 0, D = NULL, check = TRUE), precalc = NULL)

Arguments

tree   A phylogenetic tree of class "phylo" or a variance-covariance matrix (vcv) of that tree (or time-series).
data    Matrix or data frame with species in rows and continuous traits in columns. NA values are allowed with the "rpf", "inverse" and "pseudoinverse" methods.
error   Matrix or data frame with species in rows and continuous trait sampling variance (squared standard errors) in columns.
method  Method used for computing the log-likelihood. Could be "pic", "sparse", "rpf", "inverse", or "pseudoinverse". See details below.
param   List of additional arguments to be passed through the function. The "estim", "mu" and "sigma" arguments are only used with the "pic" method. The "D" argument is used with the others to specify the design matrix. See details below.

Details

The mvLL function computes the log-likelihood and the ancestral states (mean at the root-theta) for an arbitrary variance-covariance matrix (or trees for the pruning algorithm based on independent contrasts "pic") provided by the user. This function can be wrapped for optimizing various multivariate models of trait evolution (by transforming the branch lengths of a tree for the "pic" method, or feeding it with variance-covariance and design matrices for the other methods).

Five methods are proposed to compute the log-likelihood:

- "pic" is a very fast pruning algorithm based on independent contrasts which should be used with strictly dichotomic trees (i.e., no polytomies). This method can neither be used with measurement errors nor for multiple ancestral states estimation (theta values).

- "rpf" is a GLS algorithm using the rectangular packed format Cholesky factorization for solving the linear system without computing the inverse of the variance-covariance matrix and its determinant.
(normally used in the loglikelihood estimation). This algorithm uses fast BLAS 3 routines with half storage in packed format for computing the Cholesky upper factor. This method is more efficient than the "inverse" method and can be used with dense matrices (no zero entries).

"sparse" is a GLS algorithm using Cholesky factorization for sparse matrices (including zero entries). The matrices are stored in the "old Yale sparse format" internally. Depending on the sparsity structure of the variance-covariance matrix this algorithm can be more efficient than the "rpf" method.

"inverse" is a GLS algorithm that uses explicit inversion of the variance-covariance matrix (through QR decomposition) as well as computation of its determinant in the log-likelihood estimation. This is the "textbook" method, that is computationally more intensive than the previous approaches.

"pseudoinverse" is a GLS method that uses a generalized inverse (through SVD) for computing the log-likelihood. This method is safer when the matrix is near singularity, but it is the most time-consuming.

The user must provide a variance-covariance matrix (e.g., vcv.phylo(tree)) or a multivariate variance-covariance matrix (e.g., kronecker(matrix(c(2,1,1,1.5),2),vcv.phylo(tree)) as well as a design matrix (or multivariate design matrix) with the "rpf", "sparse", "inverse", and "pseudoinverse" methods.

Use the "param" list of arguments to define whether or not the brownian rate should be estimated and returned (estim=TRUE) with the "pic" method. Otherwise, the rate parameter (also called sigma) is fixed to 1. The arguments "mu" and "sigma" can be used to specify (e.g., in a MCMC setting) the mean at the root and the (squared) brownian rate, respectively.

You can choose to provide differently scaled trees for multivariate data with the "pic" method. In such a case, the trees (one per trait) should be embedded within a list() object. See example below.

Value

logl Estimated log-likelihood for the data with the given matrix/tree.
theta Estimated ancestral states at the root. They are defined by the design matrix (D) for all the methods but "pic".
sigma Estimated (squared) rate parameters. Only when param$estim=TRUE with the "pic" method.

Author(s)

Julien Clavel

References


See Also

mvMORPH mvglss mvOU mvEB mvBM mvSHIFT mvSIM

Examples

```r
## Simulated dataset
set.seed(14)
# Generating a random tree with 50 tips
n=50
tree<-pbtree(n=n)

# Simulated trait
data=rTraitCont(tree)

# Design matrix
D=matrix(rep(1,n),ncol=1)

## Compute the log-likelihood
# Inverse
mvLL(vcv.phylo(tree),data,method="inverse",param=list(D=D))

# Pseudoinverse
mvLL(vcv.phylo(tree),data,method="pseudoinverse",param=list(D=D))

# Sparse
mvLL(vcv.phylo(tree),data,method="sparse",param=list(D=D))

# RPF
mvLL(vcv.phylo(tree),data,method="rpf",param=list(D=D))

# Pic
mvLL(tree,data,method="pic",param=list(estim=TRUE))

# Pic with arbitrary values
mvLL(tree,data,method="pic",param=list(estim=FALSE, mu=0, sigma=1))
mvLL(tree,data,method="pic",param=list(estim=FALSE))
mvLL(tree,data,method="pic",param=list(estim=FALSE, sigma=1)) # similar to mu=NULL

# Arbitrary value for mu with other methods (similar to mu=0 and sigma=1 with "pic")
mvLL(vcv.phylo(tree),data,method="rpf",param=list(D=D, estim=FALSE, mu=0))

## Multivariate cases
# Simulate traits
data2<-mvSIM(tree,nsim=1,model="BM1",param=list(sigma=diag(2),theta=c(0,0),ntraits=2))
# Design matrix
D<-cbind(rep(c(1,0),each=50),rep(c(0,1),each=50))
# RPF
```
mvOU

Multivariate Ornstein-Uhlenbeck model of continuous traits evolution

Description

This function allows the fitting of a multivariate Ornstein-Uhlenbeck (OU) model by allowing a given tree branch to be subdivided into multiple selective regimes using SIMMAP-like mapping of ancestral states. Species measurement errors or dispersions can also be included in the model.

Usage

mvOU(tree, data, error = NULL, model = c("OUM", "OU1"), param = list(sigma = NULL, alpha = NULL, vcv = "fixedRoot", decomp = c("cholesky", "spherical", "eigen", "qr", "diagonal", "upper", "lower")), method = c("rpf", "sparse", "inverse", "pseudoinverse", "univarpf"), scale.height = FALSE, optimization = c("L-BFGS-B", "Nelder-Mead", "subplex"), control = list(maxit = 20000), precalc = NULL, diagnostic = TRUE, echo = TRUE)
**Arguments**

- **tree**: Phylogenetic tree with mapped ancestral states in SIMMAP format. (See `make.simmap` function from phytools package). A "phylo" object can be used with model "OU1".
- **data**: Matrix or data frame with species in rows and continuous traits in columns. NA values are allowed with the "rpf", "inverse", and "pseudoinverse" methods.
- **error**: Matrix or data frame with species in rows and continuous trait sampling variance (squared standard errors) in columns.
- **model**: Choose between "OUM" for a multiple selective regime model, or "OU1" for a unique selective regime for the whole tree.
- **param**: List of arguments to be passed to the function. See details below.
- **method**: Choose between "rpf", "sparse", "inverse", "pseudoinverse", or "univar" for computing the log-likelihood during the fitting process. See details below.
- **scale.height**: Whether the tree should be scaled to unit length or not.
- **optimization**: Methods used by the optimization routines (see ?optim and ?subplex for details). The "fixed" method returns the log-likelihood function only.
- **control**: Max. bound for the number of iteration of the optimizer; other options can be fixed in the list. (See ?optim or ?subplex for details).
- **diagnostic**: Whether the convergence diagnostics should be returned or not.
- **echo**: Whether the results must be returned or not.

**Details**

The mvOU function fits a multivariate model of evolution according to an Ornstein-Uhlenbeck process. The user can incorporate measurement errors and uses SIMMAP-like mapping of ancestral states (phytools objects of class "simmap"). SIMMAP mapping allows one to assign parts of branchs to different selective regimes, and allows testing for change in trait variance that is not synchronous with the species divergence events. See the package vignette: browseVignettes("mvMORPH").

Mapping of ancestral states can be done using the "make.simmap", "make.era.map" or "paintSubTree" functions from the "phytools" package.

The "method" argument allows the user to try different algorithms for computing the log-likelihood. The "rpf", "univar" (for univariate analysis) and "sparse" methods use fast GLS algorithms based on factorization for avoiding the computation of the inverse of the variance-covariance matrix and its determinant for the log-likelihood estimation. The "inverse" approach uses the "stable" standard explicit computation of the inverse and determinant of the matrix and is therefore slower. The "pseudoinverse" method uses a generalized inverse that is safer for matrix near singularity but highly time consuming. See ?mvLL for details.

Arguments in the "param" list are:

- "sigma" or "alpha" - Starting values for the likelihood search can be specified through the "alpha" and "sigma" arguments in the param list. It is also possible to test for the significance of the off-diagonal sigma (scatter) and alpha (drift) matrix in the full model by making comparison with a constrained model (using sigma="constraint", or alpha="constraint") in the "param" argument list.
You can also provide starting values for the constrained model. For instance, for two traits use
\texttt{sigma=list("constraint", c(0.5,0.5))} (or \texttt{alpha=list("constraint", c(0.5,0.5))}).

"\texttt{decomp}" - You can further constrain the alpha matrix by specifying the decomposition of the
matrix through the "\texttt{decomp}" argument in the "\texttt{param}" list. Indeed, the multivariate Ornstein-
Uhlenbeck model is described by the spectral decomposition of the alpha matrix. Thus it is possible
to parameterize the alpha matrix to be decomposable using various parameterizations (e.g., on its
eigenvalues with different biological interpretations; Sy et al. 1997, Bartoszek et al. 2012). For a
symmetric matrix parameterization the user can choose the "\texttt{cholesky}", "\texttt{eigen}", or "\texttt{spherical}" option.

For general square (non-symmetric) matrices the "\texttt{svd}", "\texttt{qr}", and "\texttt{schur}" parameterizations can
be used. The "\texttt{schur}" parameterization constrains the eigenvalues of the alpha matrix to be real
numbers. The "\texttt{svd+}", "\texttt{qr+}" or "\texttt{eigen+}" options forces the eigenvalues to be positives by taking
their logarithm. It is also possible to specify "\texttt{diagonal}" which is similar to the use of the "\texttt{con-
straint}" argument for "\texttt{alpha}" argument, or to use "\texttt{equal}" and "\texttt{equaldiagonal}". Finally, one can
specify that the alpha matrix is "\texttt{upper}" or "\texttt{lower}" triangular (i.e., one process affect the other
unilaterally). Details can be found in the package vignette: browseVignettes("mvMORPH").

"\texttt{decompSigma}" - The sigma matrix is parameterized by various methods to ensure its positive
definiteness (Pinheiro and Bates, 1996). These methods can be accessed through the "\texttt{decomp-
Sigma}" argument and are the "\texttt{cholesky}", "\texttt{eigen+}", and "\texttt{spherical}" parameterization. The
sigma matrix can also be forced to be diagonal using "\texttt{diagonal}" or "\texttt{equaldiagonal}" and forced
to have the same variances using "\texttt{equal}". Details can be found in the package vignette: browse-
Vignettes("mvMORPH").

"\texttt{vcv}" - It is possible to specify in the "\texttt{param}" list what kind of variance-covariance matrix to
use with the "\texttt{vcv}" argument, depending on how the root is treated. The \texttt{vcv="randomRoot"} option
assumes that the value at the root is a random variable with the stationary distribution of the process.
It cannot be used with the "\texttt{sparse}" method to speed up the computations. The \texttt{vcv="fixedRoot"}
option assumes that the root is a fixed parameter. On ultrametric trees both approaches should
converge on the same results when the OU process is stationary.

"\texttt{root}" - This argument allows the user to specify if the ancestral state at the root (theta 0) should be
estimated (\texttt{root=TRUE}), or assumed to be at the oldest regime state stationary distribution (\texttt{root=FALSE}).
An alternative is to follow Beaulieu et al. (2012) and explicitly drop the root state influence
(\texttt{root="stationary"}). The first option should be used with non-ultrametric trees (i.e., with fossil
species; e.g., Hansen 1997) where information on the ancestral state is directly available from the
data. Indeed, estimating shifts in the ancestral state from extant species could be problematic and it
seems preferable to assume each regime optimum to be at the stationary distribution.

For the "\texttt{decomp}" and "\texttt{decompSigma}" arguments, an user-defined matrix with integer values taken
as indices of the parameters to be estimated can be provided. See \texttt{?mvBM} and \texttt{?mvRWTS}.

Note on the returned Hessian matrix in the result list (\texttt{param$opt$hessian}): The hessian is the
matrix of second order partial derivatives of the likelihood function with respect
to the maximum likelihood parameter values. This matrix provides a measure of the steepness of the
likelihood surface in the vicinity of the optimum. The eigen-decomposition of the hessian matrix
returned by the optimizing function allows assessing the reliability of the fit of the model (even if the
optimizer has converged). When the optimization function does not converge on a stable result, the
user may consider increasing the "\texttt{maxit}" argument in the "\texttt{control}" option, or try a simpler model
with fewer parameters to estimate. Changing the starting values ("\texttt{alpha}" and "\texttt{sigma}" options in
the \texttt{param} list) as well as the optimizing method ("\texttt{optimization}" options) may help sometimes (e.g.,
alpha=runif(3) for a two-trait analysis with random starting values - i.e., the lower triangular alpha matrix). Note that the number of starting values to provide depends on the matrix decomposition chosen for the alpha parameter \((p^2(p+1)/2\) values for symmetric alpha matrix, but \(p^2\) values for non-symmetric ones - with \(p\) the number of traits).

**Value**

- **LogLik** The log-likelihood of the optimal model.
- **AIC** Akaike Information Criterion for the optimal model.
- **AICc** Sample size-corrected AIC.
- **theta** Estimated ancestral states.
- **alpha** Matrix of estimated alpha values (strength of selection).
- **sigma** Evolutionary rate matrix (drift).
- **convergence** Convergence status of the optimizing function; "0" indicates convergence. (see ?optim for details).
- **hess.values** Reliability of the likelihood estimates calculated through the eigen-decomposition of the hessian matrix. "0" means that a reliable estimate has been reached. See details above.
- **param** List of model fit parameters (optimization, method, model, number of parameters...).
- **llik** The log-likelihood function evaluated in the model fit "$llik(par, root.mle=TRUE)".

**Note**

This function partly uses a modified version of the C code from the "OUCH" package built by Aaron King, as well as a C code which is part of the "ape" package by Emmanuel Paradis. I kindly thank those authors for sharing their sources. Note that Bartoszek et al. (2012) proposed the mvSLOUCH package also dedicated to multivariate Ornstein-Uhlenbeck processes, which allows fitting regression models with randomly evolving predictor variables.

The "symmetric", "nsymmetric", "symmetricPositive", and "nsymPositive" options for the "decomp" argument are deprecated.

**Author(s)**

Julien Clavel

**References**


See Also
mvMORPH mvgls halflife stationary mvBM mvEB mvSHIFT mvOUTS mvRWTS mvSIM LRT optim
make.simmap make.era.map paintSubTree

Examples

# Simulated dataset
set.seed(14)
# Generating a random tree
tree<-pbtree(n=50)

# Setting the regime states of tip species
sta<-as.vector(c(rep("Forest",20),rep("Savannah",30))); names(sta)<-tree$tip.label

# Making the simmap tree with mapped states
tree<-make.simmap(tree,sta , model="ER", nsim=1)
col<-c("blue","orange"); names(col)<-c("Forest","Savannah")

# Plot of the phylogeny for illustration
plotSimmap(tree,col,fsize=0.6,node.numbers=FALSE,lwd=3, pts=FALSE)

# Simulate the traits
alpha<-matrix(c(2,0.5,0.5,1),2)
sigma<-matrix(c(0.1,0.05,0.05,0.1),2)
theta<-c(2,3,1,1.3)
data<-mvSIM(tree, param=list(sigma=sigma, alpha=alpha, ntraits=2, theta=theta, names_traits=c("head.size","mouth.size")), model="OUM", nsim=1)

## Fitting the models

# OUM - Analysis with multiple optima
mvOU(tree, data)

# OU1 - Analysis with a unique optimum
mvOU(tree, data, model="OU1", method="sparse")

# various options
mvOU(tree, data, model="OUM", method="sparse", scale.height=FALSE, param=list(decomp="svd", root="stationary"))# non-symmetric alpha
mvOU(tree, data, model="OUM", method="sparse", scale.height=FALSE, param=list(decomp="qr", root=TRUE)) # non-symmetric alpha
mvOU(tree, data, model="OUM", method="sparse", scale.height=FALSE, param=list(decomp="cholesky", root=TRUE)) # symmetric-positive
mvOUTS

Multivariate continuous trait evolution for a stationary time series
(Ornstein-Uhlenbeck model)

Description

This function allows the fitting of a multivariate Ornstein-Uhlenbeck (OU) model to a time series. Species measurement errors or dispersions can also be included in the model.

Usage

mvOUTS(times, data, error = NULL, param = list(sigma = NULL, alpha = NULL,
mvOUTS

vcv = "randomRoot", decomp = c("cholesky","spherical","eigen","qr",
"diagonal","upper","lower"), method = c("rpf","inverse","pseudoinverse",
"univarpf"), scale.height = FALSE, optimization = c("L-BFGS-B","Nelder-Mead",
"subplex"), control = list(maxit = 20000), precalc = NULL, diagnostic = TRUE,
echo = TRUE)

Arguments

times  Time series - vector of sample ages.
data   Matrix or data frame with species in rows and continuous traits in columns. NA values are allowed.
error  Matrix or data frame with species in rows and continuous trait sampling variance (squared standard errors) in columns.
param  List of arguments to be passed to the function. See details below.
method Choose between "rpf", "inverse", "pseudoinverse", or "univarpf" for computing the log-likelihood during the fitting process. See details below.
scale.height Whether the time series should be scaled to unit length or not.
optimization Methods used by the optimization routines (see ?optim and ?subplex for details). The "fixed" method returns the log-likelihood function only.
control Max. bound for the number of iteration of the optimizer; other options can be fixed in the list (see ?optim or ?subplex).
diagnostic Whether the convergence diagnostics should be returned or not.
echo   Whether the results must be returned or not.

Details

The mvOUTS function fits a multivariate model of trait evolution on a time series according to an Ornstein-Uhlenbeck process. The user can include measurement errors to the analyzed dataset.

The "method" argument allows the user to try different algorithms for computing the log-likelihood. The "rpf", "univarpf" (for univariate analysis) methods use fast GLS algorithms based on factorization for avoiding the computation of the inverse of the variance-covariance matrix and its determinant for the log-likelihood estimation. The "inverse" approach uses the "stable" standard explicit computation of the inverse and determinant of the matrix and is therefore slower. The "pseudoinverse" method uses a generalized inverse that is safer for matrix near singularity but highly time consuming. See ?mvLL for details.

Arguments in the "param" list are:

"sigma" or "alpha" - Starting values for the likelihood search can be specified through the "alpha" and "sigma" arguments in the param list. It is also possible to test for the significance of the off-diagonal sigma (scatter) and alpha (drift) matrix in the full model by making comparison with a constrained model (using sigma="constraint", or alpha="constraint") in the "param" argument list. You can also provide starting values for the constrained model. For instance, for two traits use sigma=list("constraint", c(0.5,0.5)) (or alpha=list("constraint", c(0.5,0.5))).
"decomp" - You can further constrain the alpha matrix by specifying the decomposition of the matrix through the "decomp" argument in the "param" list. Indeed, the multivariate Ornstein-Uhlenbeck model is described by the spectral decomposition of the alpha matrix. Thus it is possible to parameterize the alpha matrix to be decomposable using various parameterizations (e.g., on its eigenvalues with different biological interpretations; Sy et al. 1997, Bartoszek et al. 2012). For a symmetric matrix parameterization the user can choose the "cholesky", "eigen", or "spherical" option. For general square (non-symmetric) matrices the "svd", "qr" and "schur" parameterizations can be used. The "schur" parameterization constrains the eigenvalues of the alpha matrix to be real numbers. The "svd+", "qr+" or "eigen+" options forces the eigenvalues to be positives by taking their logarithm. It is also possible to specify "diagonal" which is similar to the use of the "constraint" argument for the "alpha" argument, or to use "equal" and "equaldiagonal". Finally, one can specify that the alpha matrix is "upper" or "lower" triangular (i.e., one process affect the other unilateraly). Details can be found in the package vignette: browseVignettes("mvMORPH").

"decompSigma" - The sigma matrix is parameterized by various methods to ensure its positive definiteness (Pinheiro and Bates, 1996). These methods can be accessed through the "decompSigma" argument and are the "cholesky", "eigen+", and "spherical" parameterization. The sigma matrix can also be forced to be diagonal using "diagonal" or "equaldiagonal" and forced to have the same variances using "equal". Details can be found in the package vignette: browseVignettes("mvMORPH").

"vcv" - It is possible to specify in the "param" list what kind of variance-covariance matrix to use with the "vcv" argument, depending on how the root is treated. The vcv="randomRoot" option assumes that the value at the root is a random variable with the stationary distribution of the process. The vcv="fixedRoot" option assumes that the root is a fixed parameter.

"root" - If root=TRUE, the ancestral state and the optimum (stationary mean) are estimated, otherwise (root=FALSE) the ancestral (initial) state and the optimum (long-term expectation) are assumed to be the same.

Note: for the "decomp" and "decompSigma" arguments, an user-defined matrix with integer values taken as indices of the parameters to be estimated can be provided. See ?mvBM and ?mvRWTS.

**Value**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LogLik</td>
<td>The log-likelihood of the optimal model.</td>
</tr>
<tr>
<td>AIC</td>
<td>Akaike Information Criterion for the optimal model.</td>
</tr>
<tr>
<td>AICc</td>
<td>Sample size-corrected AIC.</td>
</tr>
<tr>
<td>theta</td>
<td>Estimated ancestral states.</td>
</tr>
<tr>
<td>alpha</td>
<td>Matrix of estimated alpha values (strength of selection, drift matrix).</td>
</tr>
<tr>
<td>sigma</td>
<td>Evolutionary rate matrix (scatter).</td>
</tr>
<tr>
<td>convergence</td>
<td>Convergence status of the optimizing function; &quot;0&quot; indicates convergence. (See ?optim for details).</td>
</tr>
<tr>
<td>hess.values</td>
<td>Reliability of the likelihood estimates calculated through the eigen-decomposition of the hessian matrix. &quot;0&quot; means that a reliable estimate has been reached. See details on ?mvOU.</td>
</tr>
<tr>
<td>param</td>
<td>List of model fit parameters (optimization, method, model, number of parameters...).</td>
</tr>
<tr>
<td>llk</td>
<td>The log-likelihood function evaluated in the model fit &quot;$lik(par, root.mle=TRUE)&quot;.</td>
</tr>
</tbody>
</table>
Author(s)
Julien Clavel

References

See Also
mvMORPH halflife stationary mvOU mvRWTS mvBM mvEB mvSHIFT mvSIM LRT optim

Examples

# Simulate the time series
set.seed(14)
timeseries <- 0:49
# Parameters with general alpha matrix on two competitive species (or two traits)
# asymmetric (drift) matrix with intervention from the lowest layer
alpha <- matrix(c(0.15,0,0.1,0.1),2,2)
# scatter matrix
sigma <- matrix(c(0.01,0.005,0.005,0.01),2)
# ancestral states and long term optimum expectation
theta <- matrix(c(0,1,0,.5),2) # columns=traits

# Simulate the data
traits <- mvSIM(timeseries, model="OUTS", param=list(theta=theta, alpha=alpha, sigma=sigma))

# Plot the time series
matplot(traits,type="o",pch=1, xlab="Time (relative)"

fit1 <- mvOUTS(timeseries, traits, param=list(decomp="qr"))
fit2 <- mvOUTS(timeseries, traits, param=list(decomp="eigen"))
fit3 <- mvOUTS(timeseries, traits, param=list(decomp="diagonal"))
results <- list(fit1,fit2,fit3)
aicw(results)

# Simulate under the MLE
traits2 <- simulate(fit1, tree=timeseries)
matplot(traits2, type="o", pch=1, xlab="Time (relative)"

mvOUTS(timeseries, traits2, param=list(decomp="eigen"))
mvOUTS(timeseries, traits2, param=list(decomp="diagonal"))
mvOUTS(timeseries, traits2, param=list(decomp="upper"))
mvOUTS(timeseries, traits2, param=list(decomp="lower"))

# try user defined constraints
set.seed(100)
ts <- 49
timeseries <- 1:ts

sigma <- matrix(c(0.01,0.005,0.003,0.005,0.01,0.003,0.003,0.003,0.01),3)
# upper triangular matrix with effect of trait 2 on trait 1.
alpha <- matrix(c(0.4,0,0,-0.5,0.3,0,0,0,0.2),3,3)
theta <- matrix(c(0,0,0,1,0.5,0.5),byrow=TRUE, ncol=3); root=TRUE
data <- mvSIM(timeseries, model="OUTS", param=list(alpha=alpha,
 sigma=sigma, theta=theta, root=root,
 names_traits=c("sp 1", "sp 2", "sp 3")))

# plot
matplot(data, type="o", pch=1, xlab="Time (relative)"
legend("bottomright", inset=.05, legend=colnames(data), pch=19, col=c(1,2,3), horiz=TRUE)

# define an user constrained drift matrix
indice <- matrix(NA,3,3)
diag(indice) <- c(1,2,3)
indice[1,2] <- 4

# fit the model
fit_1 <- mvOUTS(timeseries, data, param=list(vcv="fixedRoot", decomp=indice))
fit_2 <- mvOUTS(timeseries, data, param=list(vcv="fixedRoot", decomp="diagonal"))
LRT(fit_1, fit_2)

---

**mvRWTS**  
*Multivariate Brownian motion / Random Walk model of continuous traits evolution on time series*

**Description**

This function allows the fitting of multivariate Brownian motion/Random walk model on time-series. This function can also fit constrained models.
Usage

mvRWTS(times, data, error = NULL, param =
list(sigma=NULL, trend=FALSE, decomp="cholesky"), method = c("rpf", "inverse", "pseudoinverse"), scale.height = FALSE, optimization = c("L-BFGS-B", "Nelder-Mead", "subplex"),
control = list(maxit = 20000), precalc = NULL, diagnostic = TRUE, echo = TRUE)

Arguments

times Time series - vector of sample ages.
data Matrix or data frame with species/sampled points in rows and continuous traits in columns
error Matrix or data frame with species/sampled points in rows and continuous traits sampling variance (squared standard error) in columns.
param List of arguments to be passed to the function. See details below.
method Choose between "rpf", "inverse", or "pseudoinverse" for log-likelihood computation during the fitting process. See details below.
scale.height Whether the time series should be scaled to unit length or not.
optimization Methods used by the optimization routines (see ?optim and ?subplex for details). The "fixed" method returns the log-likelihood function only.
control Max. bound for the number of iteration of the optimizer; other options can be fixed in the list (see ?optim or ?subplex).
diagnostic Whether the diagnostics of convergence should be returned or not.
echo Whether the results must be returned or not.

details

The mvRWTS function fits a multivariate Random Walk (RW; i.e., the time series counterpart of the Brownian motion process).

The "method" argument allows the user to try different algorithms for computing the log-likelihood. The "rpf" and "sparse" methods use fast GLS algorithms based on factorization for avoiding the computation of the inverse of the variance-covariance matrix and its determinant involved in the log-likelihood estimation. The "inverse" approach uses the "stable" standard explicit computation of the inverse and determinant of the matrix and is therefore slower. The "pseudoinverse" method uses a generalized inverse that is safer for matrix near singularity but highly time consuming. See ?mvLL for more details on these computational methods.

Arguments in the "param" list are:

"constraint" - The "constraint" argument in the "param" list allows the user to compute the joint likelihood for each trait by assuming they evolved independently (constraint="diagonal", or constraint="equaldiagonal"). If constraint="equal", the sigma values are constrained to be the same for each trait using the constrained Cholesky decomposition proposed by Adams (2013) or a separation strategy based on spherical parameterization when p>2 (Clavel et al. 2015).
User-defined constraints can be specified through a numeric matrix (square and symmetric) with integer values taken as indices of the parameters.

For instance, for three traits:

```
constraint=matrix(c(1,3,3,3,2,3,3,3,2),3).
```

Covariances constrained to be zero are introduced by NA values, e.g.,

```
constraint=matrix(c(1,4,4,4,2,NA,4,NA,3),3).
```

Difference between two nested fitted models can be assessed using the "LRT" function. See example below and ?LRT.

"decomp" - For the general case (unconstrained models), the sigma matrix is parameterized by various methods to ensure its positive definiteness (Pinheiro and Bates, 1996). These methods are the "cholesky", "eigen+", and "spherical" parameterizations.

"trend" - Default set to FALSE. If TRUE, the ancestral state is allowed to drift leading to a directional random walk. Note that it is possible to provide a vector of integer indices to constrain the estimated trends when p>1 (see the vignettes).

"sigma" - Starting values for the likelihood estimation. By default the trait covariances are used as starting values for the likelihood optimization. The user can specify starting values as square symmetric matrices or a simple vector of values for the upper factor of the sigma matrix. The parameterization is done using the factorization determined through the "decomp" argument (Pinheiro and Bates, 1996). Thus, you should provide p*(p+1)/2 values, with p the number of traits (e.g., random numbers or the values from the cholesky factor of a symmetric positive definite sigma matrix; see example below). If a constrained model is used, the number of starting values is (p*(p-1)/2)+1.

Value

- **LogLik**: The log-likelihood of the optimal model.
- **AIC**: Akaike Information Criterion for the optimal model.
- **AICc**: Sample size-corrected AIC.
- **theta**: Estimated ancestral states.
- **sigma**: Evolutionary rate matrix for each selective regime.
- **convergence**: Convergence status of the optimizing function; "0" indicates convergence (see ?optim for details).
- **hess.values**: Reliability of the likelihood estimates calculated through the eigen-decomposition of the hessian matrix. "0" means that a reliable estimate has been reached (see ?mvOU).
- **param**: List of model fit parameters (optimization, method, model, number of parameters...).
- **llik**: The log-likelihood function evaluated in the model fit "$llik(par, root.mle=TRUE)".

Author(s)

Julien Clavel
mvRWTS

References


See Also

mvMORPH mvOU mvEB mvSHIFT mvSIM mvOUTS LRT optim

Examples

set.seed(1)
# Simulate the time series
timeseries <- 0:49

# Simulate the traits
sigma <- matrix(c(0.01,0.005,0.005,0.01),2)
theta <- c(0,1)
error <- matrix(0,ncol=2,nrow=50);error[1,]=0.001
data<-mvSIM(timeseries, error=error,
            param=list(sigma=sigma, theta=theta), model="RWTS", nsim=1)

# plot the time series
matplot(data, type="o", pch=1, xlab="Time (relative)")

# model fit
mvRWTS(timeseries, data, error=error, param=list(decomp="diagonal"))
mvRWTS(timeseries, data, error=error, param=list(decomp="equal"))
mvRWTS(timeseries, data, error=error, param=list(decomp="cholesky"))

# Random walk with trend
set.seed(1)
trend <- c(0.02,0.02)
data<-mvSIM(timeseries, error=error,
          param=list(sigma=sigma, theta=theta, trend=trend), model="RWTS", nsim=1)

# plot the time series
matplot(data, type="o", pch=1, xlab="Time (relative)")

# model fit
mvRWTS(timeseries, data, error=error, param=list(trend=TRUE))

# we can specify a vector of indices
mvRWTS(timeseries, data, error=error, param=list(trend=c(1,1)))
Description

This function fits different models of evolution after a fixed point. This allows fitting models of change in mode of evolution following a given event.

Usage

mvSHIFT(tree, data, error = NULL, param = list(age = NULL, sigma = NULL, alpha = NULL, sig = NULL, beta = NULL), model = c("ER", "RR", "EC", "RC", "SR", "EBOU", "OUEB", "EBBM", "BMEB"), method = c("rpf", "sparse", "inverse", "pseudoinverse"), scale.height = FALSE, optimization = c("L-BFGS-B", "Nelder-Mead", "subplex"), control = list(maxit = 20000), precalc = NULL, diagnostic = TRUE, echo = TRUE)

Arguments

tree Phylogenetic tree with a shift mapped (see "make.era.map" function from "phytools" package). A "phylo" object can be used if the "age" argument is provided in the "param" list.
data Matrix or data frame with species in rows and continuous traits in columns. NA values are allowed with the "rpf", "inverse", and "pseudoinverse" methods.
error Matrix or data frame with species in rows and continuous trait sampling variance (squared standard errors) in columns.
param List of arguments to be passed to the function. See details.
model Choose between the different models "OUBM", "BMOU", "EBOU", "OUEB", "BMEB", "EBBM"... See details below.
method Choose between "rpf", "sparse", "inverse", or "pseudoinverse" for computing the log-likelihood during the fitting process. See details below.
scale.height Whether the tree should be scaled to unit length or not.
optimization Methods used by the optimization routines (see ?optim and ?subplex for details). The "fixed" method returns the log-likelihood function only.
control Max. bound for the number of iteration of the optimizer; other options can be fixed in the list (see ?optim and ?subplex for details).
diagnostic Whether the diagnostics of convergence should be returned or not.
echo Whether the results must be returned or not.
mvSHIFT

Details

The mvSHIFT function fits a shift in mode or rate of evolution at a fixed point in time, as previously proposed by some authors (O’Meara et al. 2006; O’Meara, 2012; Slater, 2013). Shift in mode of evolution can be mapped on a modified "phylo" object using the "make.era.map" function from the "phytools" package. Note that only one shift is allowed by the current version of mvMORPH. The age of the shift can be otherwise directly provided (in unit of times of the tree) in the function by the "age" argument in the "param" list.

The function allows fitting model with shift from an Orstein-Uhlenbeck to a Brownian motion process and vice-versa ("OUBM" and "BMOU"), shifts from a Brownian motion to/from an Early Burst (ACDC) model ("BMEB" and "EBBM"), or shifts from an Orstein-Uhlenbeck to/from an Early Burst (ACDC) model ("OUEB" and "EBOU"). Note that the shift models with OU process are relevant only if you use fossil species.

In all these cases it is possible to allow the drift parameter to vary after the fixed point by specifying "i" (for independent) after the model name. For instance, to fit models of "ecological release" or "ecological release and radiate" following Slater (2013), one can use "OUBM" or "OUBMi", respectively.

Alternatively it is also possible to use the shortcuts "ER" or "RR" to fit models of "ecological release" and "ecological release and radiate" respectively, and "EC" for a model of "constrained ecology" (e.g., after invasion of a competitive species in a given ecosystem) where traits are constrained in an Ornstein-Uhlenbeck process after a fixed point in time ("RC" is the same model but assumes an independent rate during the early radiative phase). The "SR" model allows fitting different (Brownian) rates/drift before and after the shift point (note that this model could also be fitted using the mvBM function).

The "param" list can be used to provide lower and upper bounds for the exponential rate parameter of the Early-Burst/ACDC model. See ?mvEB for details.

The "method" argument allows the user to try different algorithms for computing the log-likelihood. The "rpf" and "sparse" methods use fast GLS algorithms based on factorization for avoiding the computation of the inverse of the variance-covariance matrix and its determinant involved in the log-likelihood estimation. The "inverse" approach uses the "stable" standard explicit computation of the inverse and determinant of the matrix and is therefore slower. The "pseudoinverse" method uses a generalized inverse that is safer for matrix near singularity but highly time consuming. See ?mvLL for details.

Value

<table>
<thead>
<tr>
<th>LogLik</th>
<th>The log-likelihood of the optimal model.</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIC</td>
<td>Akaike Information Criterion for the optimal model.</td>
</tr>
<tr>
<td>AICc</td>
<td>Sample size-corrected AIC.</td>
</tr>
<tr>
<td>theta</td>
<td>Estimated ancestral states.</td>
</tr>
<tr>
<td>alpha</td>
<td>Matrix of estimated alpha values (strength of selection).</td>
</tr>
<tr>
<td>beta</td>
<td>Exponent rate (of decay or increase) for the ACDC/Early-Burst model.</td>
</tr>
<tr>
<td>sigma</td>
<td>Evolutionary rate matrix (drift) for the BM process before the shift.</td>
</tr>
<tr>
<td>sig</td>
<td>Evolutionary rate matrix (drift) for the BM process after the shift (only for &quot;i&quot; models).</td>
</tr>
</tbody>
</table>
convergence  Convergence status of the optimizing function; "0" indicates convergence (see ?optim for details).
hess.values Reliability of the likelihood estimates calculated through the eigen-decomposition of the hessian matrix. "0" means that a reliable estimate has been reached (see ?mvOU for details).
param List of model fit parameters (optimization, method, model, number of parameters...).
llik The log-likelihood function evaluated in the model fit "$llik(par, root.mle=TRUE)".

Note
Changes in rate of evolution and optima can also be fitted using the mvBM and mvOU functions using a 'make.era.map' transformed tree.

Author(s)
Julien Clavel

References

See Also
mvMORPH mvOU mvBM mvEB mvOUTS mvRWTS mvSIM optim subplex paintSubTree make.era.map

Examples
# Simulated dataset
set.seed(14)
# Generating a random tree
tree<-rtree(50)

# Providing a tree with the shift mapped on
tot<-max(nodeHeights(tree))
age=tot-3   # The shift occurred 3 Ma ago
tree<-make.era.map(tree,c(0,age))

# Plot of the phylogeny for illustration
plotSimmap(tree,fsize=0.6,node.numbers=FALSE,lwd=3, pts=FALSE)

# Simulate the traits
alpha<-matrix(c(2,0.5,0.5,1),2)
sigma<-matrix(c(0.1,0.05,0.05,0.1),2)
theta<-c(2,3)
data<-mvSIM(tree, param=list(sigma=sigma, alpha=alpha, ntraits=2, theta=theta,
          names_traits=c("head.size","mouth.size")), model="ER", nsim=1)

## Fitting the models
# "Ecological release model"
mvSHIFT(tree, data, model="OUBM") # similar to mvSHIFT(tree, data, model="ER")

# "Release and radiate model"
mvSHIFT(tree, data, model="RR", method="sparse")
# similar to mvSHIFT(tree, data, model="OUBMi")

# More generally...
# OU to/from BM
mvSHIFT(tree, data, model="OUBM", method="sparse")
mvSHIFT(tree, data, model="BMOU", method="sparse")
mvSHIFT(tree, data, model="OUBMi", method="sparse")
mvSHIFT(tree, data, model="BMOUi", method="sparse")

# BM to/from EB
mvSHIFT(tree, data, model="BMEB", method="sparse")
mvSHIFT(tree, data, model="EBBM", method="sparse")
mvSHIFT(tree, data, model="BMEBi", method="sparse")
mvSHIFT(tree, data, model="EBBMI", method="sparse")

# OU to/from EB
mvSHIFT(tree, data, model="OUEB", method="sparse")
mvSHIFT(tree, data, model="OUEBi", method="sparse")
mvSHIFT(tree, data, model="EBOU", method="sparse")
mvSHIFT(tree, data, model="EBOUi", method="sparse")

## Without providing mapped tree
# The shift occured 3Ma ago (param$age=3)
set.seed(14)
tree<-rtree(50)
data<-mvSIM(tree, param=list(sigma=sigma, alpha=alpha, ntraits=2, theta=theta,
          names_traits=c("head.size","mouth.size"), age=3), model="ER", nsim=1)

## Fitting the models without mapped tree but by specifying the age in the param list.
mvSHIFT(tree, data, model="OUBM", param=list(age=3))
Description

This function allows simulating multivariate (as well as univariate) continuous traits evolving according to a BM (Brownian Motion), OU (Ornstein-Uhlenbeck), ACDC (Accelerating rates and Decelerating rates/Early bursts), or SHIFT models of phenotypic evolution.

Usage

mvSIM(tree, nsim = 1, error = NULL, model = c("BM1", "BMM", "OU1", "OUM", "EB"),
param = list(theta = 0, sigma = 0.1, alpha = 1, beta = 0))

Arguments

tree
Phylogenetic tree with mapped ancestral states in SIMMAP format (see make.simmap function from phytools package) or a standard "phylo" object (ape). Or a time-series

nsim
The number of simulated traits (or datasets for multivariate analysis).

error
Matrix or data frame with species in rows and continuous trait sampling variance (squared standard errors) in columns.

model
The model of trait evolution for the simulations. Could be any of the models used by the mvBM, mvEB, mvOU and mvSHIFT functions.

param
List of parameter arguments used for the simulations. You should provide the sigma (values or matrix), alpha (for OU and SHIFT models), beta (EB and SHIFT), theta (ancestral states), ntraits (the number of traits) or others param arguments used in the models. Alternatively you can provide a fitted object of class "mvmorph". See details below.

Details

This function simulates multivariate (as well as univariate) continuous traits evolving along a given phylogenetic tree or time series according to a BM/RW (Brownian Motion/Random walk), OU (Ornstein-Uhlenbeck), ACDC (Accelerating rates and Decelerating rates/Early Bursts), and SHIFT models of phenotypic evolution. The traits are simulated by random sampling from a Multivariate Normal Distribution (Paradis, 2012).

The mvSIM function allows simulating continuous trait (univariate or multivariate) evolution along a phylogeny (or a time-series) with user specified parameters or parameters estimated from a previous fit.

The "simulate" wrapper can also be used with a fitted object of class "mvmorph": simulate(object, nsim=1, tree=tree). See example below.

If parameter values are not provided, the default values are fixed to 1 (sigma, sig, alpha, beta) or to 0 for the mean at the root (ancestral state).

For the "BMM" model were different parts of the tree have their own rate, a list with one rate (or matrix of rates) per selective regime must be provided.

For the "OU1" and "OUM" models, the user can specify if the ancestral state (theta0) should be computed (param$root=TRUE), assumed to be at the oldest regime state (param$root=FALSE), or if there is no root and each regimes is at the stationary point (param$root="stationary"; see also ?mvOU).
For the "BM1", "BMM", and "RWTS" models, a trend can be simulated by providing values to the "trend" argument in the "param" list. Traits names can be provided with the "names_traits" argument in the "param" list. For all the shift models, if the tree is not mapped the age of the shift should be directly provided (in unit of times of the tree) using the "age" argument in the "param" list.

Value

A matrix with simulated traits (columns) for the univariate case, or a list of matrix for the multivariate case (nsim>1).

Note

Ancestral states for Ornstein-Uhlenbeck processes (param$root=TRUE) should be used with non-ultrametric trees. As this method uses Multivariate Normal distribution (MVN) for simulating the traits, it is advised to avoid its use with very large datasets/trees and rely instead on recursive algorithms (see, e.g., ?rTraitCont from "ape").

Author(s)

Julien Clavel

References


See Also

mvMORPH mvglss mvOU mvEB mvBM mvSHIFT mvRWTS mvOUTS mvLL

Examples

```r
## Simulated dataset
set.seed(14)
# Generating a random tree with 50 species
tree<-pbtree(n=50)

# Setting the regime states of tip species
sta<-as.vector(c(rep("Forest",20),rep("Savannah",30))); names(sta)<-tree$tip.label

tree<-make.simmap(tree,sta, model="ER", nsim=1)

# Plot of the phylogeny for illustration
plotSimmap(tree,col, fsize=0.6, node.numbers=FALSE, lwd=3, pts=FALSE)

## Simulate trait evolution according to a bivariate "BMM" model
# Number of traits
ntraits<-2
```
# Number of simulated (pairs of) traits
nsim<-10
# Rates matrices for the "Forest" and the "Savannah" regimes
sigma<-list(Forest=matrix(c(2,0.5,0.5,1),2), Savannah=matrix(c(5,3,3,4),2))
# ancestral states for each traits
theta<-c(0,0)

# Simulate
simul<-mvSIM(tree,nsim=nsim, model="BMM",param=list(ntraits=ntraits,sigma=sigma,theta=theta))

# Try to fit a "BM1" model to the first simulated dataset
model_fit<-mvBM(tree,simul[[1]],model="BM1")

# Use the estimated parameters to simulate new traits!
simul2<-mvSIM(tree,nsim=nsim,param=model_fit)

# or try with generic "simulate" function
simul3<-simulate(model_fit,nsim=nsim,tree=tree)

## Just-for-fun :Comparing parameters
simul4<-simulate(model_fit,nsim=100,tree=tree)
results<-lapply(simul4,function(x){
  mvBM(tree,x,model="BM1",method="pic", echo=FALSE,diagnostic=FALSE)
})
sigma_simul<-sapply(results,function(x){x$sigma})

# comparison between the simulated (black) and the observed (red) multivariate rates
layout(matrix(1:4, ncol=2))
for(i in 1:4){
  hist(sigma_simul[[i]], main=paste("Estimated sigma on simulated traits"),
       xlab="estimated sigma for 100 replicates");abline(v=mean(sigma_simul[i,]),lwd=2);
  abline(v=model_fit$sigma[i],col="red",lwd=2)
}

---

**phyllostomoid**

*Phylosgeny and trait data for a sample of Phylllostomid bats*

**Description**

Phylogeny, diet, and morphological variables for 49 species of Phylllostomid bats.

**Usage**

data("phyllostomoid")
Details

Illustrative phylogeny (phylllostomid$tree) and morphological data (phylllostomid$mandible - 73 variables composed of the superimposed procrustes 2D-coordinates for the mandible and the condylobasal length) of 49 species of Phyllostomid bats from Monteiro & Nogueira (2011). The firsts 22 coordinates represent anatomical landmarks and the last 50 coordinates are semilandmarks.

The four grouping factor variables (e.g., phylllostomid$grp1, phylllostomid$grp2, ...) are the adaptive regime models for association between mandible morphology and diet considered in Monteiro & Nogueira (2011).

References


Clavel, J., Morlon, H. 2020. Reliable phylogenetic regressions for multivariate comparative data: illustration with the MANOVA and application to the effect of diet on mandible morphology in phylllostomid bats. Systematic Biology (DOI:10.1093/sysbio/syaa010)

Examples

data(phylllostomid)
plot(phylllostomid$tree)
head(phylllostomid$mandible)

# Fit a linear model by PL
fit1 <- mvglss(mandible~grp1, data=phylllostomid, phylllostomid$tree, model="lambda", method="LOO")

# regularized MANOVA test
(manova.gls(fit1, test="Wilks", verbose=TRUE))

pruning

Pruning algorithm to compute the square root of the phylogenetic covariance matrix and it’s determinant.

Description

This function use the pruning algorithm (Felsenstein 1973) to efficiently compute the determinant of the phylogenetic covariance matrix as well as the square root of this matrix (or it’s inverse; Stone 2011, Khabbazian et al. 2016). This algorithm is faster than using "eigen" or "cholesky" function to compute the determinant or the square root (see e.g., Clavel et al. 2015) and can be used to compute independent contrasts and the log-likelihood of a model in linear time.

Usage

pruning(tree, inv=TRUE, scaled=TRUE, trans=TRUE, check=TRUE)
pruning

Arguments

- **tree**: Phylogenetic tree (an object of class "phylo" or "simmap").
- **inv**: Return the matrix square root of either the covariance matrix (inv=FALSE) or its inverse (inv=TRUE, the default). This matrix is a "contrasts" matrix.
- **scaled**: Indicates whether the contrasts should be scaled with their expected variances (default to TRUE).
- **trans**: Return the transpose (trans=TRUE) of the matrix square root/contrasts matrix. (i.e. by default it returns a matrix equivalent to the upper triangular Cholesky factor)
- **check**: Check if the input tree is dichotomous and in "postorder" (see ?is.binary.tree and ?reorder.phylo).

Details

The tree is assumed to be fully dichotomic and in "postorder", otherwise the functions `multi2di` and `reorder.phylo` are used internally when `check=TRUE`.

Value

- **sqrtMat**: The matrix square root (contrasts matrix)
- **varNode**: Variance associated to each node values (similar to "contrasts" variance)
- **varRoot**: Variance associated to the root value (similar to the ancestral state variance)
- **det**: Log-determinant of the phylogenetic covariance of the tree

Author(s)

Julien Clavel

References


See Also

`mvLL`, `mvglm`
Examples

## Simulated dataset
set.seed(14)
# Generating a random tree
tree <- pbtree(n=50)
Y <- mvSIM(tree, model="BM1", param=list(sigma=1, theta=0)) # trait
X <- matrix(1, nrow=Ntip(tree), ncol=1) # design matrix

## Use the GLS trick
# Compute the matrix square root
C <- vcv.phylo(tree)
D <- chol(C)
Cinv <- solve(C)
Di <- chol(Cinv)

# transform the traits
Xi <- Di%*%X
Yi <- Di%*%Y

# Compute the GLS estimate and determinant (see Clavel et al. 2015)
# GLS estimate for the root
print(pseudoinverse(Xi)%*%Yi)

# Determinant of the phylogenetic covariance matrix
print(sum(log(diag(D)^2)))

## Use the pruning algorithm (much faster)
M <- pruning(tree, inv=TRUE)

Xi <- M$sqrtMat%*%X
Yi <- M$sqrtMat%*%Y

# GLS estimate
print(pseudoinverse(Xi)%*%Yi)

# determinant
print(M$det)

## REML determinant (without variance of the root state; see Felsenstein 1973)
# full REML
log(det(C)) + log(det(t(X)%*%Cinv%*%X))

# pruning REML
sum(log(M$varNode))
**Description**

Returns the residuals of a linear model of class 'mvglss'.

**Usage**

```r
## S3 method for class 'mvglss'
residuals(object, type, ...)
```

**Arguments**

- `object`: an object of class 'mvglss' obtained from a `mvglss` fit.
- `type`: an optional character string specifying the type of residuals to be used. To match conventions used in the `nlme` package: if "response", the "raw" residuals (observed-fitted) are used; else, if "normalized", the normalized residuals (the residuals pre-multiplied by the inverse square-root factor of the estimated (between observations) covariance matrix) are used. Note however that there is still between variables correlations with both types.
- `...`: other arguments for this generic function (not used).

**Value**

A matrix with the residuals for the linear model fitted by `mvglss`.

**Author(s)**

J. Clavel

**See Also**

`vcov.mvgls` `residuals.mvgls` `coef.mvgls` `mvgls` `mvglss` `stationary` `The stationary variance of an Ornstein-Uhlenbeck process`

**Description**

This function returns the stationary variance for an Ornstein-Uhlenbeck process (object of class "ou").

**Usage**

```r
stationary(object)
```

**Arguments**

- `object`: Object fitted with the "mvOU" function.
Details

This function computes the dispersion parameter of the Ornstein-Uhlenbeck process (i.e., the expected variance when the process is stationary). The multivariate normal stationary distribution of the Ornstein-Uhlenbeck process is computed following Bartoszek et al. (2012).

Value

The stationary variance-covariance matrix of the OU process

Author(s)

Julien Clavel

References


See Also

mvMORPH mvOU halflife

Examples

# Simulated dataset
set.seed(14)
# Generating a random tree
tree<-pbtree(n=50)

# Setting the regime states of tip species
sta<-as.vector(c(rep("Forest",20),rep("Savannah",30))); names(sta)<-tree$tip.label

# Making the simmap tree with mapped states
trees<-make.simmap(tree,sta, model="ER", nsim=1)
col<-c("blue","orange"); names(col)<-c("Forest","Savannah")

# Plot of the phylogeny for illustration
plotSimmap(trees,col,fsize=0.6,node.numbers=FALSE,lwd=3, pts=FALSE)

# Simulate the traits
alpha<-matrix(c(2,0.5,0.5,1),2)
sigma<-matrix(c(0.1,0.05,0.05,0.1),2)
theta<-c(2,3,1,1.3)
data<-mvSIM(tree, param=list(sigma=sigma, alpha=alpha, ntraits=2, theta=theta, names_traits=c("head.size","mouth.size")), model="OUM", nsim=1)

## Fitting the models
# OUM - Analysis with multiple optima
result<-mvOU(tree, data)
stationary(result)
# Expected values when the process is stationary
expected<-list(alpha=alpha,sigma=sigma)
class(expected)<-c("mvmorph","mvmorph.ou")
stationary(expected)

---

**vcov.mvgls**  
*Calculate variance-covariance matrix for a fitted object of class 'mvgls'*

**Description**

Returns the variance-covariance matrix of the coefficients or the traits.

**Usage**

```r
## S3 method for class 'mvgls'
vcov(object, ...)
```

**Arguments**

- **object**
  - an object of class 'mvgls' obtained from a mvgls fit.
- **...**
  - additional arguments for methods function. See details below.

**Details**

The vcov function returns by default the variance-covariance matrix of the main parameters of a fitted model object. The main parameters are the coefficients (this correspond to the argument type="coef": see also coef.mvgls). With type="covariance", the vcov.mvgls function returns the estimated traits covariance matrix (possibly regularized for PL approaches) while type="precision" return the precision matrix (i.e. the inverse of the covariance).

**Value**

A matrix of the estimated covariances between the parameter estimates (of type "coef", "covari-
ance", or "precision").

**Author(s)**

J. Clavel

**See Also**

coef.mvgls residuals.mvgls fitted.mvgls mvgls
Index

* Topic **AIC**
  aicw, 3
* Topic **Accelerating rates**
  mvEB, 25
* Topic **Akaike weights**
  aicw, 3
* Topic **BM**
  mvMORPH-package, 2
* Topic **Brownian Motion**
  mvBM, 20
  mvRWTS, 46
  mvSHIFT, 50
* Topic **Cholesky constraint**
  mvBM, 20
  mvRWTS, 46
* Topic **Decelerating rates**
  mvEB, 25
* Topic **Determinant**
  pruning, 57
* Topic **EB**
  mvMORPH-package, 2
* Topic **EC**
  mvSHIFT, 50
* Topic **ER**
  mvSHIFT, 50
* Topic **Early burst**
  mvEB, 25
* Topic **Early-Burst**
  mvSHIFT, 50
* Topic **Estim**
  estim, 8
* Topic **Evolutionary rates**
  mvBM, 20
  mvMORPH-package, 2
  mvRWTS, 46
  mvSHIFT, 50
* Topic **GIC**
  manova.gls, 16
  mvglsls, 28
  mvglsls.pca, 32
* Topic **GLS**
  EIC, 6
  GIC, 11
  manova.gls, 16
  mvglsls, 28
  mvglsls.pca, 32
  mvLL, 34
  pruning, 57
* Topic **General Linear Hypothesis**
  manova.gls, 16
* Topic **Hessian**
  mvOU, 37
  mvOUTS, 42
* Topic **High dimensions**
  EIC, 6
  GIC, 11
  manova.gls, 16
  mvglsls, 28
  mvglsls.pca, 32
* Topic **Imputation**
  estim, 8
* Topic **Independent contrasts**
  mvLL, 34
  pruning, 57
* Topic **LRT**
  LRT, 14
* Topic **Loglikelihood ratio test**
  LRT, 14
* Topic **Loglikelihood**
  mvLL, 34
* Topic **MANOVA**
  manova.gls, 16
* Topic **Matrix square root**
  pruning, 57
* Topic **Measurement error**
  mvMORPH-package, 2
* Topic **Methods**
  mvLL, 34
*Topic **Missing values**
estim, 8

*Topic **Model comparison**
EIC, 6
GIC, 11

*Topic **Models comparison**
LRT, 14
manova.gls, 16

*Topic **Multivariate Linear Models**
mvglss, 28

*Topic **Multivariate tests**
manova.gls, 16

*Topic **OU**
thalflife, 12
mvMORPH-package, 2
mvOU, 37
mvOUTS, 42
stationary, 60

*Topic **Ornstein Uhlenbeck**
thalflife, 12
mvOU, 37
mvOUTS, 42
mvSHIFT, 50
stationary, 60

*Topic **PCA**
mvglss.pca, 32

*Topic **Penalized likelihood**
manova.gls, 16
mvglss, 28
mvglss.pca, 32

*Topic **RR**
mvSHIFT, 50

*Topic **Random walk**
mvRWTS, 46

*Topic **Regularization**
manova.gls, 16
mvglss, 28
mvglss.pca, 32

*Topic **SIMMAP**
mvMORPH-package, 2

*Topic **SR**
mvSHIFT, 50

*Topic **Shifts**
mvMORPH-package, 2
mvSHIFT, 50

*Topic **Simulations**
mvMORPH-package, 2

*Topic **Time series**
mvOUTS, 42
mvRWTS, 46

*Topic **User defined constraints**
mvRWTS, 46

*Topic **bats**
phyllostomid, 56

*Topic **datasets**
phyllostomid, 56

*Topic **half-life**
thalflife, 12

*Topic **manova.gls**
phyllostomid, 56

*Topic **mvglss**
mvMORPH-package, 2
phyllostomid, 56

*Topic **mvmmorph object**
mvSIM, 53

*Topic **parameters**
mv.Precalc, 18

*Topic **precalculation**
mv.Precalc, 18

*Topic **simulate traits**
mvSIM, 53

*Topic **stationary**
stationary, 60

AIC, 4
aicw, 3, 3

brownie.lite, 23

coeff.mvglss, 5, 10, 31, 60, 62

EIC, 3, 6, 17, 31, 33
estim, 3, 8
evol.vcv, 23

fitted.mvglss, 5, 10, 31, 62

GIC, 3, 7, 11, 17, 31, 33

halflife, 3, 12, 41, 45, 61

LRT, 3, 14, 23, 41, 45, 49

make.era.map, 23, 41, 52
make.simmap, 23, 41
manova.gls, 3, 16, 31
mv.Precalc, 18
mvBM, 3, 9, 15, 19, 20, 27, 36, 41, 45, 52, 55
mvEB, 3, 9, 15, 19, 23, 25, 36, 41, 45, 49, 52, 55
mvgl, 3, 5, 7, 10, 12, 17, 23, 27, 28, 33, 36, 41, 55, 58, 60, 62
mvgl.pca, 3, 7, 12, 31, 32
mvLL, 3, 19, 34, 55, 58
mvMORPH, 4, 9, 13, 15, 19, 23, 27, 36, 41, 45, 49, 52, 55, 61
mvMORPH (mvMORPH-package), 2
mvMORPH-package, 2
mvOU, 3, 9, 13, 15, 19, 23, 27, 36, 37, 45, 49, 52, 55, 61
mvOUTS, 3, 23, 27, 41, 42, 49, 52, 55
mvRWTS, 3, 23, 27, 41, 45, 46, 52, 55
mvSHIFT, 3, 9, 15, 19, 23, 27, 36, 41, 45, 49, 50, 55
mvSIM, 3, 23, 27, 36, 41, 45, 49, 52, 53
optim, 23, 27, 41, 45, 49, 52
paintSubTree, 23, 41, 52
phyllostomid, 56
pruning, 57
residuals.mvgl, 5, 10, 31, 59, 60, 62
stationary, 3, 13, 41, 45, 60
subplex, 52
vcov.mvgl, 5, 10, 31, 60, 62