How to use mvMORPH?

I) Trees and Time series

1. Phylogenetic trees

The trees used by the **mvMORPH** functions must be objects of class "phylo" as provided by the **ape** package. Such trees are directly used with the "EB", "BM1", and "OU1" models (and also with the SHIFT model if the shift age is given in the param list; see ?mvSHIFT). To fit models with multiple groups (or selective regimes) such as the "BMM" and "OUM" models (functions *mvBM* and *mvOU*), the evolutionary history of the selective regimes (i.e., the ancestral state reconstruction - see the next vignette) must be "painted" on the tree. For this purpose **mvMORPH** uses trees in SIMMAP format as provided by the **phytools** package. Several functions allows creating SIMMAP trees:

- *make.simmap* (Ancestral state reconstruction using stochastic mapping)
- *make.era.map* (Creates temporal map on the tree)
- *paintBranches* (Assigns to a particular branch a given discrete state)
- *paintSubTree* (Assigns to a particular sub-tree a given discrete state)

For instance, users may be interested in directly using prior knowledge to specify different states on particular parts of the tree (or use results from other ancestral reconstructions methods). The user must refer to the **phytools** documentation for more details.

```
# Load the package and dependencies (ape, phytools, corpcor, subplex)
library(mvMORPH)
```

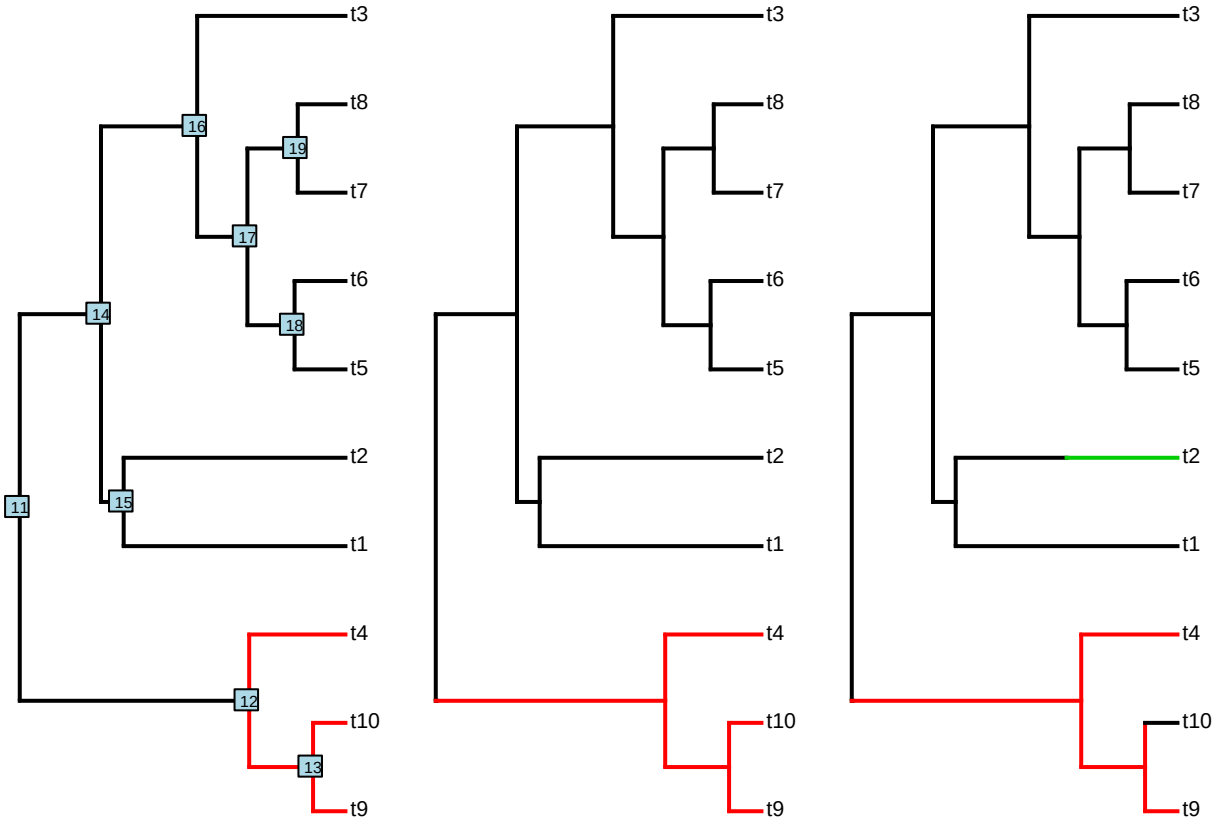
```r
# Use a specified random number seed for reproducibility
set.seed(14)
par(mfrow=c(1,3))
tree <- pbtree(n=10)

# Set a different regime to the monophyletic group on node "12"
tree2 <- paintSubTree(tree, node=12, state="group_2", anc.state="group_1")
plot(tree2);nodelabels(,cex=0.8)

# We can set the character state to the stem branch leading to the subtree
tree3 <- paintSubTree(tree, node=12, state="group_2", anc.state="group_1", stem=TRUE)
plot(tree3)

# Finally we can also set a different regime to the branch leading to the tip "t10"
branch_1 <- match("t10",tree3$tip.label) # alternatively: which(tree$tip.label=="t10")
tree4 <- paintBranches(tree3, branch_1, state="group_1")

# set also a change to a third state along the branch
# leading to "t2" using the "stem" argument
branch_2 <- match("t2",tree3$tip.label)
tree4 <- paintSubTree(tree4, branch_2, state="group_3", stem=0.5)
plot(tree4)
```



Note that the tree length can be scaled to unity using the *"scale.height = TRUE"* option. Why? Scaling the tree will not change the value of the maximum log-likelihood (or AIC), but will change the scale of the parameters estimates. This is particularly useful to relatively compare analysis on different trees or to directly interpret estimates such as the phylogenetic halflife in % of the tree size (see ?*halflife*). Besides, working on

scaled trees can sometime speed up the computations.

2. Time series

The *mvRWTS* and *mvOUTS* functions allow fitting multivariate models of continuous traits evolution with time-series data (such as ancestor-descendent fossil lineages). Such models can be used, for instance, to compare how traits evolve within an ancestor-descendent lineage or between lineages. The times-series must be provided as vectors of ordered ages relative to the starting point (i.e., the oldest, which should be "0").

For instance:

```
# Make a tiny time-serie object from fossil ages between [55; 32.5] Ma
fossil_ages <- c(55,46,43,38,35,32.5)

# The time serie becomes:
timeserie <- max(fossil_ages)-fossil_ages
timeserie
```

```
> [1]  0.0  9.0 12.0 17.0 20.0 22.5
```

The *"scale.height=TRUE"* option will scale the time serie between 0 and 1 to provide the relative ages. Similarly to the analysis on phylogenetic trees, this change the scale of the parameter estimates (such as sigma or alpha) but not the log-likelihood.

II) Data and Errors

Matrix or data frame with species in rows and continuous traits (variables) in columns, rownames must match tip names. Missing cases are currently allowed as **NA** values with the "rpf", "inverse", and "pseudoinverse" methods. The measurement error or sampling variance is a matrix of similar size (note that the sampling variance is given by the square of the standard-errors of species mean estimates). Measurement error missing values (**NA** or "0") are allowed.

For some time-serie models; *mvOUTS* with fixed root vcv (see the "param" list section below) and *mvRWTS*; the sampling variance for the first obsevations in the time-serie must be provided otherwise an arbitrary value of 0.01 is set automatically to avoid singularity issues.

III) Optimization methods

mvMORPH rely on the *optim* and *subplex* functions to optimize the log-likelihood of the models. The "L-BFGS-B" and "Nelder-Mead" algorithms are used by default for most models. The "L-BFGS-B" algorithm is generally faster and seem to be more efficient, but when the optimizer don't converge or if the fit seem unreliable (i.e., when $convergence or the $hess.value in the results are not 0) it may help to switch between these methods or change the options of these methods in the "control" argument (see ?optim or ?subplex). The *subplex* algorithm seem particularly efficient for the user-defined models. When the optimization is turn to "fixed" only the log-likelihood function is returned without evaluation (see X - log-likelihood function below).

IV) Likelihood computation methods

mvMORPH uses various methods based on generalized least square (GLS) or the prunning algorithm (contrasts) to compute the log-likelihood. While GLS based approaches are very flexible and allow fitting almost all models, to deal with missing cases and measurement error very easily, and are particularly safe for optimization (e.g., the "pseudoinverse" method), their naive implementations are computationally intensive (e.g., "inverse" and "pseudoinverse" methods). Two alternative and efficient algorithms also based on GLS are proposed ("rpf" and "sparse"). The "sparse" method is both efficient in terms of computational time and memory use, but is limited by the sparsity structure of the variance-covariance (vcv) of the phylogenetic tree while the "rpf" is not. While these approaches are up to ten time faster than the "inverse" or the "pseudoinverse" methods, they may suffer of singularity issues during optimization in some cases. The "pic" method is the fastest one (thousand of time faster than the GLS based methods implemented here), but currently available for a couple of models only.

The various methods available for each models currently (stay tuned on gitHub) on mvMORPH are presented in the table below (from the fastest to the slowest)

functions	*pic*	*sparse*	*univarpf*	*rpf*	*inverse*	*pseudo*
mvBM:BM1	X	X	-	X	X	X
mvBM:BMM	U	X	-	X	X	X
mvOU:OU1	U	X	U	X	X	X
mvOU:OUM	-	X	U	X	X	X
mvEB	X	X	-	X	X	X
mvSHIFT	-	X	-	X	X	X
mvOUTS	-	-	-	X	X	X
mvRWTS	-	-	-	X	X	X
NA values	-	-	X	X	X	X
error	-	X	X	X	X	X

Notes: the "sparse" method can be used only with the "fixedRoot" vcv for the "OU1" and "OUM" models. This is because the "randomRoot" vcv is not sparse. This is also the case for the time-serie models. U = method currently implemented for the univariate case.

V) Models comparison

Models can be compared using Akaike criterions (AIC) and Akaike weights or likelihood ratio test (LRT) when they are nested. Several functions are available on **mvMORPH**: *LRT*, *AIC*, *logLik*, and *aicw*.

```
set.seed(1)
tree <- pbtree(n=50)
# 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, theta=theta), model="BM1", nsim=1)

# Fit three nested models
fit_1 <- mvBM(tree, data, model="BM1")
fit_2 <- mvBM(tree, data, model="BM1", param=list(constraint="equal"))
fit_3 <- mvBM(tree, data, model="BM1", param=list(constraint="diagonal"))
```

```
# Compare their AIC values
AIC(fit_1); AIC(fit_2); AIC(fit_3)
```

```
> [1] 15.21573
```

```
> [1] 14.17575
```

```
> [1] 29.89082
```

```
# Likelihood Ratio Test:
LRT(fit_1, fit_2) # test non-significant as expected!
```

```
> -- Log-likelihood Ratio Test --
> Model BM1  versus  BM1  equal variance/rates
> Number of degrees of freedom : 1
> LRT statistic: 0.960028  p-value: 0.3271798
> ---
> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
LRT(fit_1, fit_3)
```

```
> -- Log-likelihood Ratio Test --
> Model BM1  versus  BM1  diagonal
> Number of degrees of freedom : 1
> LRT statistic: 16.67509  p-value: 4.435969e-05 ***
> ---
> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
# Compute the Akaike weights:
results <- list(fit_1,fit_2,fit_3)
weights <- aicw(results)
weights
```

```
> -- Akaike weights --
>                Rank  AIC  diff       wi     AICw
> BM1 equal 2       1 14.2  0.00 1.000000 0.626992
> BM1 default 1     2 15.2  1.04 0.594529 0.372765
> BM1 diagonal 3    3 29.9 15.72 0.000387 0.000243
```

```
# Model averaging of the evolutionary covariance
mdavg <- lapply(1:3, function(x) results[[x]]$sigma*weights$aicw[x])
Evol_cov <- Reduce('+',mdavg)
Evol_cov
```

```
>
>  0.07542106 0.01685019
>  0.01685019 0.09541013
```

```
# Is the model averaging better than the best fitting model in this example?
# which.min(c(mean((sigma-fit_2$sigma)^2),mean((sigma-Evol_cov)^2)))

# Get the evolutionary correlations:
cov2cor(Evol_cov)
```

```
>
>  1.0000000 0.1986375
>  0.1986375 1.0000000
```

```
# with more than 2 traits you can compute the conditional (or partial) correlations
cor2pcor(Evol_cov)
```

```
>           [,1]      [,2]
> [1,] 1.0000000 0.1986375
> [2,] 0.1986375 1.0000000
```

```
# Model averaging for the root state
mdavg <- lapply(1:3, function(x) results[[x]]$theta*weights$aicw[x])
Evol_theta <- Reduce('+',mdavg)
```

VI) Treatment of the root

1. Ancestral state and primary optimum

The "root" argument in the "param" *list* of the *mvOU* function allows specifying if the root state and the optimum must be estimated (root=TRUE), or if we assume that the root state and the primary optimum are distributed according to the stationary distribution of the process (root=FALSE; Fig. 2). Indeed, the root state and the primary optimum are not identifiable separately on extant species. A slight variant of root=FALSE is root="stationary" where the root state is explicitely dropped and we assume the optimum is stationary (e.g., the implementation used in the OUwie package).

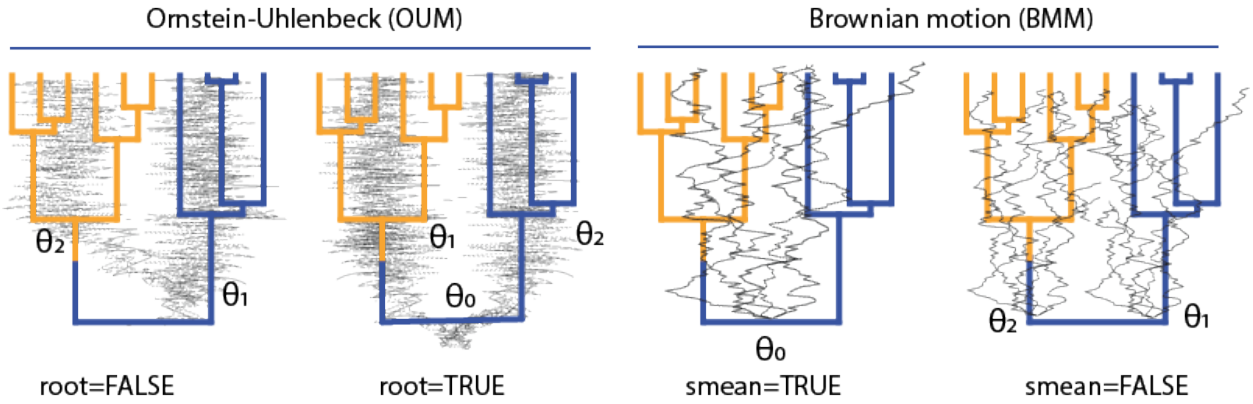


Figure 2. *Treatment of the root state options in the* mvOU *and* mvBM *functions. For the OU process, when root=FALSE the ancestral state at the root is directly derived from the oldest regime state (the ancestral state and the optimum are not estimated separately). With root=TRUE the ancestral state and the optimum are explicitely estimated. It should be noted that the root=TRUE option provides unreliable estimates without fossil data. For the BM model, the smean=TRUE option estimate a single ancestral state (the phylogenetic mean) while smean=FALSE estimate separates phylogenetic mean*

```r
set.seed(100)
tree<-rtree(100)

# Simulate the traits
alpha<-matrix(c(0.2,0.05,0.05,0.1),2)
sigma<-matrix(c(0.1,0.05,0.05,0.1),2)
theta<-c(0,2,0,1.3)
data<-mvSIM(tree, param=list(sigma=sigma, alpha=alpha,
                             theta=theta, root=TRUE), model="OU1", nsim=1)

mvOU(tree, data, model="OU1", param=list(root=TRUE))
```

For the *mvOUTS* function, the path from the initial state to the optimum is explicitely estimated (*root=TRUE*, the default value), otherwise the ancestral (initial) state and the optimum are assumed to be the same *root=FALSE*.

```r
# timeserie
ts <- 0:49
sigma<-matrix(c(0.1,0.05,0.05,0.1),2)
alpha <- matrix(c(1,0.5,0.5,0.8),2,2)
theta<-c(0,2,0,1)
data<-mvSIM(ts, param=list(sigma=sigma, alpha=alpha,
                             theta=theta, root=TRUE), model="OUTS", nsim=1)

mvOUTS(ts, data, param=list(root=TRUE))
mvOUTS(ts, data, param=list(root=FALSE))
```

2. Multiple phylogenetic mean

The "smean" argument in the "param" *list* of the *mvBM* function allows estimating either one ancestral state (smean=TRUE) for the whole tree (even if there are multiple regime states) or multiple ancestral states for each regimes (smean=FALSE). Note: contrary to the OU model, the reconstructed history of the discrete state don't affects the model fit.

When differences between groups/regimes states are only related to the phylogenetic mean, a multiple rate model can be misleadingly preferred over a single rate model:

```r
# BM model with two selective regimes
set.seed(1)
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)

# Simulate the traits
sigma<-matrix(c(0.1,0.05,0.05,0.1),2)
theta<-c(0,2,0,2)
data<-mvSIM(tree, param=list(sigma=sigma, theta=theta, smean=FALSE), model="BM1", nsim=1)
```

```
# fit the models with and without multiple phylogenetic mean, and different rates matrix
fit_1 <- mvBM(tree, data, model="BM1")
fit_2 <- mvBM(tree, data, model="BMM")
fit_3 <- mvBM(tree, data, model="BM1", param=list(smean=FALSE))
fit_4 <- mvBM(tree, data, model="BMM", param=list(smean=FALSE))
```

Unless we consider the multiple mean models, the multipe rate matrix is misleadingly preferred (AIC>4) over the single rate matrix (the generating) model.

```
# Compare the fitted models.
results <- list(fit_1,fit_2,fit_3,fit_4)
aicw(results, aicc=TRUE)
```

```
> -- Akaike weights --
>               Rank   AIC  diff     wi  AICw
> BM1 default 3    1  51.9  0.00 1.0000 0.954
> BMM default 4    2  58.0  6.06 0.0482 0.046
> BMM default 2    3 103.8 51.88 0.0000 0.000
> BM1 default 1    4 107.5 55.60 0.0000 0.000
```

3. directional trends

The "trend" argument in the "param" *list* allows estimating a BM model with a trend (in either *mvBM* or *mvRWTS*). It should be noted that with *mvBM*, a trend can be estimated reliably only on non-ultrametric trees.

```
# Simulated dataset
set.seed(1)
# Generating a random non-ultrametric tree
tree<-rtree(100)

# Simulate the traits
sigma<-matrix(c(0.1,0.05,0.05,0.1),2)
theta<-c(0,0)
trend<-c(0.2,0.2)
data<-mvSIM(tree, param=list(sigma=sigma, trend=trend, theta=theta,
                  names_traits=c("head.size","mouth.size")), model="BM1", nsim=1)

# model without trend
fit_1 <- mvBM(tree, data, param=list(trend=FALSE), model="BM1")
# model with independent trends
fit_2 <- mvBM(tree, data, param=list(trend=TRUE), model="BM1")
# model with similar trend for both traits
fit_3 <- mvBM(tree, data, param=list(trend=c(1,1)), model="BM1")
```

```
results <- list(fit_1,fit_2,fit_3)
aicw(results)
```

```
> -- Akaike weights --
>               Rank   AIC  diff      wi     AICw
> BM1 default 3    1  87.5  0.00 1.00000 0.721314
> BM1 default 2    2  89.4  1.91 0.38533 0.277944
> BM1 default 1    3 101.2 13.76 0.00103 0.000742
```

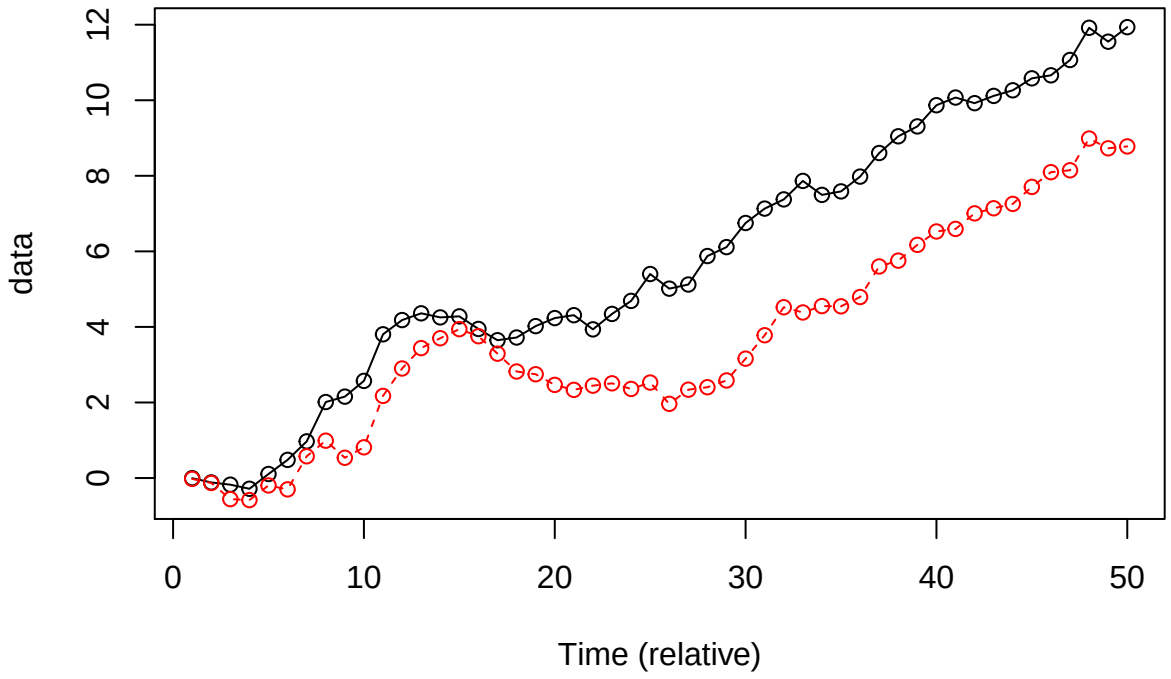
```r
# are the differences between trends significant?
LRT(fit_2,fit_3) # No... as expected
```

```
> -- Log-likelihood Ratio Test --
> Model BM1  versus  BM1
> Number of degrees of freedom : 1
> LRT statistic: 0.09268642  p-value: 0.7607893
> ---
> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

In the same way we can fit a directional random walk on time-series data.

```r
# Simulate the time serie and data
timeserie <- 0:49
error <- matrix(0, nrow=50, ncol=2); error[1,] <- 0.001
data<-mvSIM(timeserie, error=error, model="RWTS",
            param=list(sigma=sigma, theta=theta, trend=trend), nsim=1)

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




```r
# model fit
fit1 <- mvRWTS(timeserie, data, error)
fit2 <- mvRWTS(timeserie, data, error, param=list(trend=c(1,1)))
```

```r
LRT(fit2,fit1)
```

```
> -- Log-likelihood Ratio Test --
> Model RWTS  versus  RWTS
> Number of degrees of freedom : 1
```

```
> LRT statistic: 17.22943  p-value: 3.312656e-05 ***
> ---
> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Note that the theta parameters cannot be estimated analytically for the "trend" model (i.e., option root.mle=TRUE, in the returned log-likelihood function)

VII) Constraints

1. Overview of the various constraints parameterizations in mvMORPH

Comparison of constrained models allows testing various evolutionary hypothesis such as independent evolution, comparing evolutionary rates between traits, common structure, phenotypic integration... etc. The constraints methods currently available on mvMORPH are the following (see also Fig. 1):

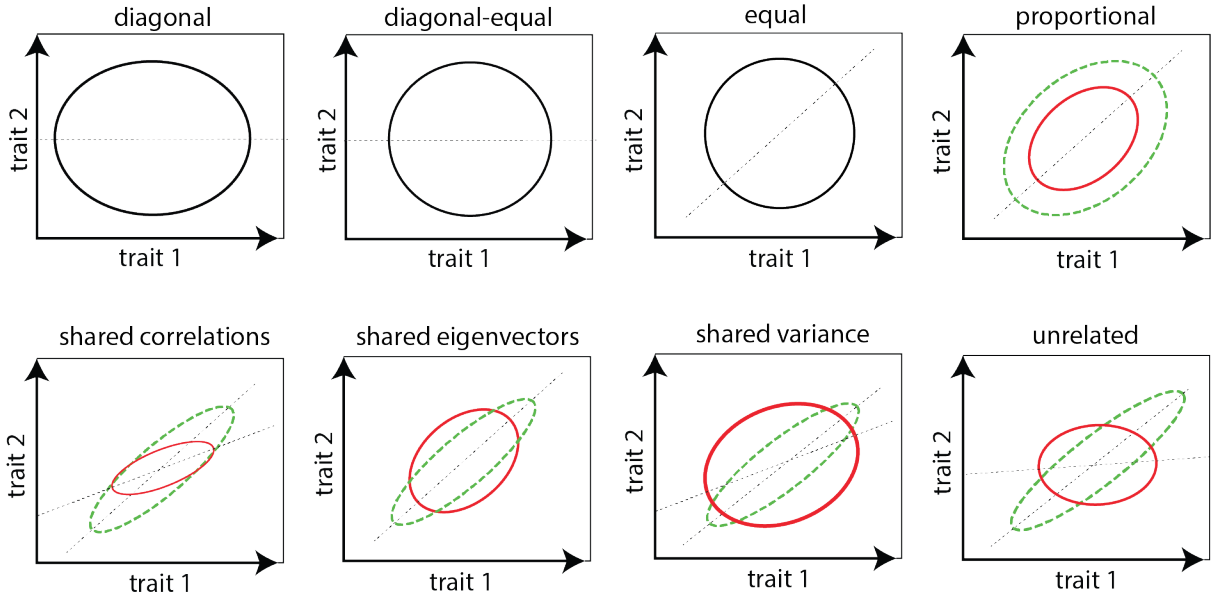


Figure 1. *Bivariate representation of the covariance constraints used in mvMORPH.*

- **"default"** : no constraints imposed on the structure of the rate (evolutionary variance and covariance) matrix
- **"diagonal"** : there is no evolutionary covariances between the traits, the rate matrix is diagonal.
- **"equaldiagonal"** there is no evolutionary covariances and the traits share the same variance.
- **"equal"** the traits covaries and share the same variance (or evolutionary rates).
- **"proportional"** (only with "BMM") the rates matrices are proportional for different regimes/states mapped on the tree.
- **"correlation"** (only with "BMM") the rates matrices share the same correlations between traits for different regimes/states mapped on the tree.
- **"variance"** (only with "BMM") the rates matrices share the same variances (rates) between regimes/states mapped on the tree.
- **"shared"** (only with "BMM") the rates matrices share the same eigen-vectors (orientation) between regimes/states mapped on the tree.
- **"user"** : the design of the rate matrix is directly defined by the user. Note that for 2 traits, all the combinations are readily available from the default methods. See example below.

2. "User-defined" constraints

User-defined constraints are done by providing a symmetric square matrix with integer values taken as indices of the parameters. To constraint covariance values to zero the user must used **NA** values as indices.Because a covariance matrix must be symmetric positive definite, negative eigenvalues are automatically shrunk to a specified tolerance value (=1). This value can be changed through the "param" *list* with the "tol" argument. This value must be changed with various estimates (e.g. 1e-5, 10...) if convergence of the optimizer has not been achieved. The alternative optimization methods may help. **The "subplex" optimization method is advised**.

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

# Simulate 3 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))

# Fit user defined contrained model
user_const <- matrix(c(1,4,4,4,2,5,4,5,3),3)
fit1 <- mvBM(tree, data, model="BM1", method="pic"
             , param=list(constraint=user_const), optimization="subplex")

# only rates/variances are changing
user_const <- matrix(c(1,3,3,3,2,3,3,3,2),3)
fit2 <- mvBM(tree, data, model="BM1", param=list(constraint=user_const)
             , method="pic", optimization="subplex")
```

```
# Some covariances constrained to zero
user_const <- matrix(c(1,4,4,4,2,NA,4,NA,3),3)

print(user_const)
```

```
>      [,1] [,2] [,3]
> [1,]    1    4    4
> [2,]    4    2   NA
> [3,]    4   NA    3
```

```
fit3 <- mvBM(tree, data, model="BM1", method="pic"
             , param=list(constraint=user_const), optimization="subplex")
```

```
> Efficient parameterization is not yet implemented for user-defined constraints
> Please check the results carefully!!

> successful convergence of the optimizer
> a reliable solution has been reached
>
> -- Summary results for user-defined BM1 constrained model --
> LogLikelihood:     141.75
> AIC:   -269.5
> AICc:      -266.8334
```

```
> 7 parameters
>
> Estimated rates matrix
> ----------------------
>
>  0.008590119 0.002877440 0.00287744
>  0.002877440 0.006576635 0.00000000
>  0.002877440 0.000000000 0.01121442
>
> Estimated root state
> --------------------
>
> theta: -0.1187766 -0.151949 -0.05041854
```

Note: the user-defined option can also be used with the *decomp* or *decompSigma* argument for the OU processes (see below). Diagonal elements values are automatically squared to obtain positive definite matrices or positive eigenvalues.

VIII) The "param" list

The "param" list argument allows specifying multiple options in **mvMORPH** functions (starting values estimate for the optimization, constraints, matrix parameterization...)

1. The "decomp" and "decompSigma" options

These options allow specifying the matrix parameterization used for the alpha (A) and sigma (Σ) matrices of parameters for the Ornstein-Uhlenbeck process:

$$dX(t) = A(\theta(t) - X(t))dt + \Sigma dW(t)$$

and Brownian motion:

$$dX(t) = \Sigma dW(t)$$

For symmetric matrices (e.g. the variance-covariance matrix of brownian rates Σ) **mvMORPH** uses various parameterizations to ensure the positive definiteness such as *decomp="cholesky"*, *decomp="spherical"* or *decomp="eigen+"*. The *decomp="eigen"* option parameterize a symmetric matrix. For the *mvOU* function, the sigma matrix parameterization is controlled by the *decompSigma* argument; e.g. *decompSigma="cholesky"*.

Symmetric matrices are parameterized by *p(p+1)/2* parameters. The firsts *p(p-1)/2* parameters are used to compute the cholesky factors of spherical angles, or the orthogonal matrices; the next p parameters are the eigenvalues (in "eigen") or the standard deviations (in "spherical"). The bivariate case is illustrated here:

Cholesky decomposition: *decomp="cholesky"*

$$\Sigma/A = \begin{bmatrix} l_{11} & 0 \\ l_{21} & l_{22} \end{bmatrix} \begin{bmatrix} l_{11} & 0 \\ l_{21} & l_{22} \end{bmatrix}^T$$

Separation strategy: *decomp="spherical"*

$$\Sigma/A = \begin{bmatrix} \sqrt{\nu_1} & 0 \\ 0 & \sqrt{\nu_2} \end{bmatrix} \left(\begin{bmatrix} 1 & \cos(\theta_1) \\ 0 & \sin(\theta_1) \end{bmatrix}^T \begin{bmatrix} 1 & \cos(\theta_1) \\ 0 & \sin(\theta_1) \end{bmatrix} \right) \begin{bmatrix} \sqrt{\nu_1} & 0 \\ 0 & \sqrt{\nu_2} \end{bmatrix}$$

Eigen decomposition: *decomp="eigen"*

$$\Sigma/A = \begin{bmatrix}\cos(\theta_1) & -\sin(\theta_1)\\ \sin(\theta_1) & \cos(\theta_1)\end{bmatrix}\begin{bmatrix}\lambda_1 & 0\\ 0 & \lambda_2\end{bmatrix}\begin{bmatrix}\cos(\theta_1) & -\sin(\theta_1)\\ \sin(\theta_1) & \cos(\theta_1)\end{bmatrix}^T$$

or to force the eigenvalues to be positive (mandatory for the sigma matrix):

$$\Sigma/A = \begin{bmatrix}\cos(\theta_1) & -\sin(\theta_1)\\ \sin(\theta_1) & \cos(\theta_1)\end{bmatrix}\begin{bmatrix}\log(\lambda_1) & 0\\ 0 & \log(\lambda_2)\end{bmatrix}\begin{bmatrix}\cos(\theta_1) & -\sin(\theta_1)\\ \sin(\theta_1) & \cos(\theta_1)\end{bmatrix}^T$$

For the OU process, the alpha matrix can be more general. Matrix parameterization such as singular value decomposition (*decomp="svd"*), QR (*decomp="qr"*), or Schur decomposition (*decomp="schur"*) are nevertheless used to make decomposable and real matrices. The *decomp="svd+"*, *decomp="qr+"*, and *decomp="schur+"* forces the eigenvalues to be positive by taking their logarithm, and ensure uniqueness for the QR factorization. These matrices are parameterized by p^2 parameters. The firsts *p(p-1)/2* parameters are used to compute the first orthogonal matrix, the next p parameters for the eigenvalues ("svd") or diagonal values of the upper triangular matrix ("qr" and "schur"). The last *p(p-1)/2* parameters are used to compute the second othogonal matrix ("svd"), or to fill the upper triangular matrix in the "schur" and "qr" parameterizations.

Singular Value Decomposition (SVD): *decomp="svd"*

$$A = \begin{bmatrix}\cos(\theta_1) & -\sin(\theta_1)\\ \sin(\theta_1) & \cos(\theta_1)\end{bmatrix}\begin{bmatrix}\lambda_1 & 0\\ 0 & \lambda_2\end{bmatrix}\begin{bmatrix}\cos(\theta_2) & -\sin(\theta_2)\\ \sin(\theta_2) & \cos(\theta_2)\end{bmatrix}^T$$

QR decomposition: *decomp="qr"*

$$A = \begin{bmatrix}\cos(\theta_1) & -\sin(\theta_1)\\ \sin(\theta_1) & \cos(\theta_1)\end{bmatrix}\begin{bmatrix}\lambda_1 & \theta_2\\ 0 & \lambda_2\end{bmatrix}$$

Schur decomposition: *decomp="Schur"*

$$A = \begin{bmatrix}\cos(\theta_1) & -\sin(\theta_1)\\ \sin(\theta_1) & \cos(\theta_1)\end{bmatrix}\begin{bmatrix}\lambda_1 & \theta_2\\ 0 & \lambda_2\end{bmatrix}\begin{bmatrix}\cos(\theta_1) & -\sin(\theta_1)\\ \sin(\theta_1) & \cos(\theta_1)\end{bmatrix}^T$$

Finally, the alpha matrix can be diagonal (independent effects) or triangular. Triangular parameterizations are for instance used on time-series forecasting to determine if a given variable is helpful in predicting a second one; the so-called Granger causality (e.g., the lower process drives the upper one for a upper triangular matrix). Similarly to symmetric matrices, upper and lower triangular matrices are parameterized by *p(p+1)/2* parameters. The firsts *p(p-1)/2* parameters are used for the upper or lower off-diagonal of the matrices, while the next p parameters are in the diagonal.

Diagonal: *decomp="diagonal"*

$$\Sigma/A = \begin{bmatrix}\sqrt{d_{11}} & 0\\ 0 & \sqrt{d_{22}}\end{bmatrix}\begin{bmatrix}\sqrt{d_{11}} & 0\\ 0 & \sqrt{d_{22}}\end{bmatrix}^T$$

Upper triangular: *decomp="upper"*

$$A = \begin{bmatrix}\log(d_{11}) & u_{12}\\ 0 & \log(d_{22})\end{bmatrix}$$

Lower triangular: *decomp="lower"*

$$A = \begin{bmatrix}\log(d_{11}) & 0\\ l_{21} & \log(d_{22})\end{bmatrix}$$

Note: the package can't handle matrices with similar eigenvalues currently (e.g., the diagonal values must be slightly differents in a trianbgular matrix).

2. the "vcv" argument

For the Ornstein-Uhlenbeck process (OU), the variance-covariance matrix (vcv) is conditioned depending on how the root is treated. The *vcv="randomRoot"* option assume that the root is a random variable with the distribution of the stationary process; the *vcv="fixedRoot"* option assume that the root state is a fixed parameter. While the computation of the "randomRoot" variance-covariance is faster, we cannot use it with the *method="sparse"* option to compute the log-likelihood. However when the process is stationary (i.e., when the alpha value/eigenvalues are large enough) both approaches should converge to the same results:

```
set.seed(1)
# simulate a tree scaled to unit length
tree <- pbtree(n=100, scale=1)

theta <- c(0,2) # the ancestral state and the optimum
sigma <- 1
alpha <- 0.1 # large half-life
trait <- mvSIM(tree, nsim=1, model="OU1",
               param=list(theta=theta, sigma=sigma, alpha=alpha, root=TRUE))

# fit the models
fit_1 <- mvOU(tree, trait, model="OU1", param=list(vcv="randomRoot"))
fit_2 <- mvOU(tree, trait, model="OU1", param=list(vcv="fixedRoot"))

# Simulate with a stronger effect
alpha <- 6 # small half-life ~10% of the tree height
trait <- mvSIM(tree, nsim=1, model="OU1",
               param=list(theta=theta, sigma=sigma, alpha=alpha, root=TRUE))

# fit the models
fit_3 <- mvOU(tree, trait, model="OU1", param=list(vcv="randomRoot"))
fit_4 <- mvOU(tree, trait, model="OU1", param=list(vcv="fixedRoot"))
```

```
# compare the log-likelihood
abs(logLik(fit_1))-abs(logLik(fit_2)) # large half-life
```

```
> [1] 2.034886
```

```
abs(logLik(fit_3))-abs(logLik(fit_4)) # small half-life
```

```
> [1] 0.0002828673
```

IX) Functions

1. The loglikelihood function

The log-likelihood function evaluated during the model fit can be accessed on objects of class "mvmorph" with the *$llik()* expression.

```
set.seed(1)
tree <- pbtree(n=50)
# 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, theta=theta), model="BM1", nsim=1)

# Fit the model
fit <- mvBM(tree, data, model="BM1")
```

```
# the loglik function (the order of parameter is printed:)
fit$llik
```

```
# Use the maximum likelihood parameters from the optimizer
par <- fit$param$opt$par
fit$llik(par)
```

```
> [1] -2.607863
```

```
all.equal(fit$llik(par),logLik(fit)) # similar results
```

```
> [1] TRUE
```

```
# Return also the analytic MLE for theta
fit$llik(par, theta=TRUE)
```

```
> $logl
> [1] -2.607863
>
> $theta
> [1]  0.02265092 -0.10563330
```

On all the **mvMORPH** models, the ancestral states (optimum or theta), are the maximum likelihood estimate (mle) analytic solution. But the user can specify manually the values for the root state (e.g., for mcmc evaluation):

```
# Use the maximum likelihood parameters from the optimizer
fit$llik(c(par,fit$theta), root.mle=FALSE)
```

```
> [1] -2.607863
```

2. The matrix parameterization functions

The model object also return functions in the "param" *list*. The *sigmafun* and *alphafun* functions returns the parameterizations functions used to fit the models (see the "parameterization" section).

```
# Reconstruct sigma with the untransformed values of the optimizer
sigma <- fit$param$sigmafun(par)

all.equal(as.vector(sigma), as.vector(fit$sigma))
```

```
> [1] TRUE
```

Note that all these functions are available without maximum likelihood evaluation by using the "fixed" optimization option.

```
# Return the log-likelihood function only
fit <- mvBM(tree, data, model="BM1", optimization="fixed")
```

```
> No optimizations performed, only the Log-likelihood function is returned with default parameters.
```

X) Fossil data

All the functions in **mvMORPH** can handle datasets with fossil species, either from time-series or phylogenetic trees. Fossil data added to extant ones in non-ultrametric trees frequently greatly improve the reliability and accuracy of parameter estimates and are mandatory for detecting changes in evolutionary modes such as with the *mvSHIFT* model or with *mvBM* and *mvRWTS* with "trend". See also "treatment of the root state" above.

Methods specifically dedicated to works with fossil data:

- *mvSHIFT* models
- *mvBM* param=list(trend=TRUE)
- *mvOU* param=list(root=TRUE)
- *mvOUTS* multivariate OU for time-series (coevolution, causal links...)
- *mvRWTS* multivariate random walk (=BM process) for time series
- *mvRWTS* param=list(trend=TRUE); directional random walk

Fossil remains are often altered by post-mortem taphonomic processes such as weathering or distortion. The *estim* function can be used to estimate missing phenotypic values for incomplete fossil specimens according to an evolutionary model.

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

# Simulate two correlated traits evolving along the phylogeny
sigma<-matrix(c(0.1,0.05,0.05,0.1),2); theta<-c(0,0)
traits<-mvSIM(tree, model="BM1", param=list(sigma=sigma, theta=theta))

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

# Fit of model 1
fit_1<-mvBM(tree,data,model="BM1")
```

```
# Estimate the missing cases
imp<-estim(tree, data, fit_1)

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

```
>            [,1]        [,2]
> t49  0.44980939  0.06739505
> t33  0.22877815  0.17143174
> t46  0.20216754 -0.09769158
> t6   0.43515540  0.32572363
> t26 -0.05570371  0.37424794
> t22 -0.09329556  0.34201599
> t38  0.14585869  0.04596017
> t27 -0.04096002  0.08656360
> t5   0.09545650  0.49616115
> t40 -0.01968701  0.32195673
```

XI) Tweak mvMORPH

We can use the returned log-likelihood of the mvMORPH functions, or the *mvLL* function to fit user-defined models. Here we show two simple examples on how the architecture of the package can be used.

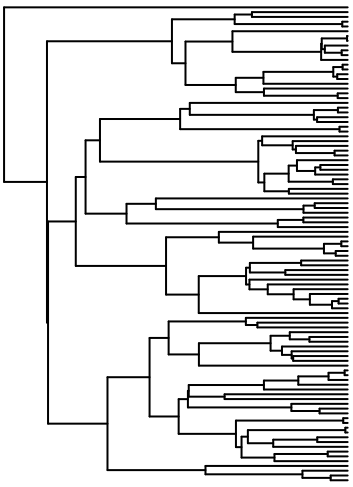
1. Making your own model!

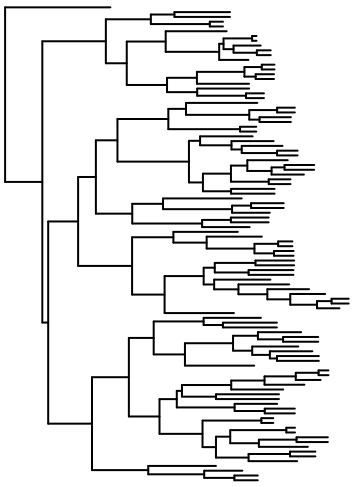
```
set.seed(123)
# Here I show a simple example of model fitting using mvMORPH
tree <- pbtree(n=100)

# 1) Create a model, e.g. the Kappa model of Pagel 1999
phylo_kappa <- function(tree, kappa){
  tree$edge.length <- tree$edge.length ^ kappa
  return(tree)
}

# just to compare the trees
par(mfrow=c(1,2))
plot(tree, show.tip.label=FALSE, main="untransformed tree")
plot(phylo_kappa(tree, 0.5), show.tip.label=FALSE, main="Kappa model")
```

untransformed tree **Kappa model**





```
# 2) Create the log-likelihood function

log_lik <- function(par, tree, data){
  tree_transf <- phylo_kappa(tree, par)
  result <- mvLL(tree_transf, data, method="pic")
  return(list(logl=result$logl, theta=result$theta, sigma=result$sigma))
}

# 3) optimize it!

# create fake data
trait <- rTraitCont(phylo_kappa(tree, 0.7))

# guess for the optimization
guess_value <- 0.5

# optimize the model!
fit <- optim(guess_value, function(par){log_lik(par, tree, trait)$logl},
             method="L-BFGS-B", lower=0, upper=1, control=list(fnscale=-1))

result <- data.frame(logLik=fit$value, kappa=fit$par, sigma2=log_lik(fit$par, tree,
                              trait)$sigma, theta=log_lik(fit$par, tree, trait)$theta )
rownames(result) <- "Model fit"

# print the result
print(result)
```

```
>           logLik     kappa     sigma2      theta
> Model fit 84.743 0.7319548 0.01040997 -0.1733913
```

```
# Do the same for a multivariate model:
# simulate traits independently
trait <-cbind( rTraitCont(phylo_kappa(tree, 0.7)), rTraitCont(phylo_kappa(tree, 0.7)))
```

```
log_lik <- function(par, tree, data){
  # transform the tree for each traits
  tree_transf <- list(phylo_kappa(tree, par[1]), phylo_kappa(tree, par[2]))
  result <- mvLL(tree_transf, data, method="pic")
  return(list(logl=result$logl, theta=result$theta, sigma=result$sigma))
}

# fit
guess_value <- c(0.5,0.5)
fit <- optim(guess_value, function(par){log_lik(par, tree, trait)$logl},
             method="L-BFGS-B", lower=0, upper=1, control=list(fnscale=-1))

result <- list(logLik=fit$value, kappa=fit$par, sigma2=log_lik(fit$par, tree,
                              trait)$sigma, theta=log_lik(fit$par, tree, trait)$theta)
# as expected the covariances are low
print(result)
```

```
> $logLik
> [1] 161.2547
>
> $kappa
> [1] 0.7249301 0.6064902
>
> $sigma2
>             [,1]        [,2]
> [1,] 0.012333692 0.001011377
> [2,] 0.001011377 0.009004207
>
> $theta
> [1] 0.03126802 0.03395208
```

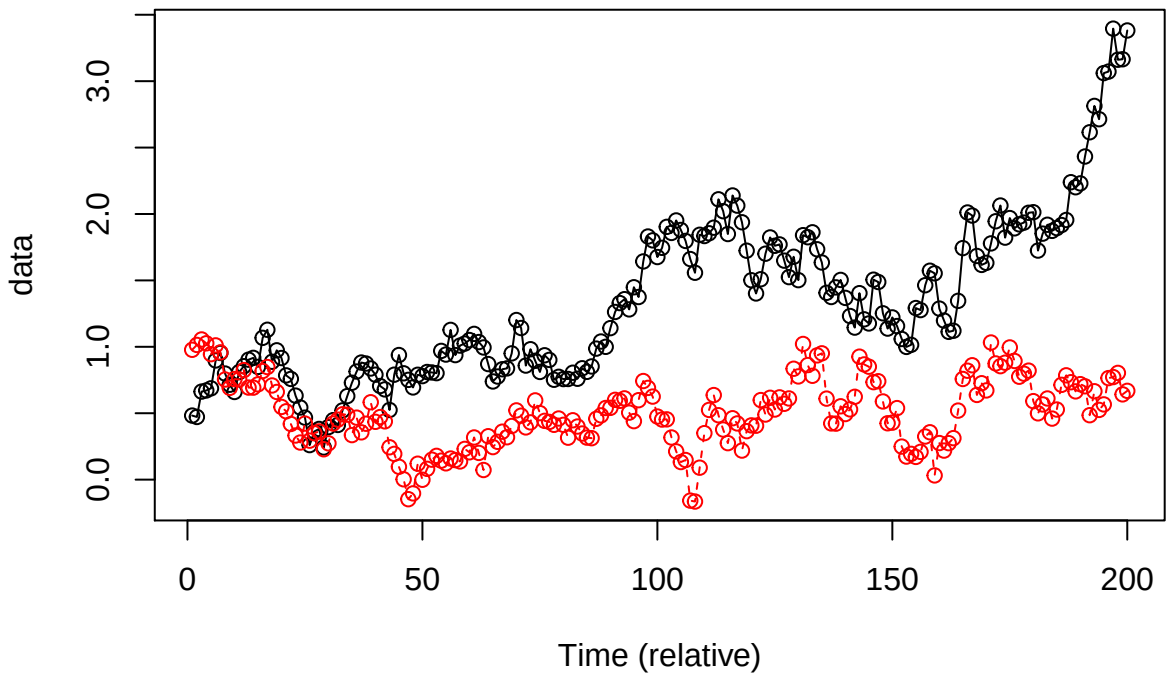
2. Reuse the returned log-likelihood functions

We can also use the likelihood returned by the mvMORPH functions to fit custom models using joint optimizations. Here an example with time-series data to fit a shift in rates and covariances.

```
set.seed(123)
# Simulate the time serie
timeserie <- 0:99

# Simulate the traits
sigma_1 <- matrix(c(0.015,0.005,0.005,0.01),2)
sigma_2 <- matrix(c(0.03,0.01,0.01,0.02),2)
theta <- c(0.5,1)
error <- matrix(0,ncol=2,nrow=100);error[1,]=0.001
data_1<-mvSIM(timeserie, error=error,
              param=list(sigma=sigma_1, theta=theta), model="RWTS", nsim=1)
data_2<-mvSIM(timeserie+100, error=error,
              param=list(sigma=sigma_2, theta=data_1[100,]), model="RWTS", nsim=1)
data <- rbind(data_1,data_2)

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




```
# 1) log-likelihood function
fun_ts_1 <- mvRWTS(timeserie, data_1, error=error, optimization="fixed")
```

```
> No optimizations performed, only the Log-likelihood function is returned.
```

```
fun_ts_2 <- mvRWTS(timeserie+100, data_2, error=error, optimization="fixed")
```

```
> No optimizations performed, only the Log-likelihood function is returned.
```

```
# a model of shift
log_lik_RWshift <- function(par){
  # compute the log-likelihood for the first bin
  part1 <- fun_ts_1$llik(par[1:3], theta=TRUE)
  # compute the log-likelihood for the second bin using the
  # expectation of the first bin and sigma2
  part2 <- fun_ts_2$llik(c(par[4:6],part1$theta), root.mle=FALSE)

  # the log-likelihood
  ll1 <- part1$logl; ll2 <- part2
  return(ll1+ll2)
}

# 2) starting values (only for sigma because theta is computed analytically)
guess_value <- c(chol(sigma_1)[upper.tri(sigma_1, TRUE)],
                 chol(sigma_2)[upper.tri(sigma_2, TRUE)])

# optimize the model!
fit <- optim(guess_value, function(par){log_lik_RWshift(par)},
```

```
            method="Nelder-Mead", control=list(fnscale=-1))

# plot the results
results <- list(sigma_1=fun_ts_1$param$sigmafun(fit$par[1:3]),
                sigma_2=fun_ts_2$param$sigmafun(fit$par[4:6]),
                theta=fun_ts_1$llik(fit$par[1:3], theta=TRUE)$theta,
                llik=fit$value)
print(results)
```

```
> $sigma_1
>             [,1]        [,2]
> [1,] 0.012569443 0.003547747
> [2,] 0.003547747 0.008824665
>
> $sigma_2
>             [,1]        [,2]
> [1,] 0.025662850 0.007850675
> [2,] 0.007850675 0.020154634
>
> $theta
> [1] 0.4803513 0.9819471
>
> $llik
> [1] 277.1485
```

XII) Bayesian mcmc

mvMORPH models can be estimated using Bayesian mcmc. For instance we can use weakly informative priors using the various parameterizations in **mvMORPH**, or directly put priors on the final parameters (e.g., the inverse Wishart for the rate matrix sigma). Here is an example using a separation strategy based on the spherical parameterization (angles in the interval [0,pi] to ensure the uniqueness of the spherical parameterization; and standard deviations expressed as uniform priors on [1e-5,0.4]; see for instance Lu & Ades 2009; and the matrix parameterization section above).

```
set.seed(1)
tree <- pbtree(n=50)
# 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, theta=theta), model="BM1", nsim=1)

# Retrieve the log-likelihood of the model (we can use optimization="fixed")
bm_model <- mvBM(tree, data, model="BM1", method="pic")

# define weakly informative prior for the correlations and standard deviations separately
prior <- function(x){
  a <- dunif(x[1],min=0, max=pi, TRUE) # prior for the angles
  b <- dunif(x[2],min=1e-5, max=0.4, TRUE) # prior for the standard deviations of trait 1
  c <- dunif(x[3],min=1e-5, max=0.4, TRUE) # prior for the standard deviations of trait 2
  return(a+b+c)
}
```

```r
# define the log-likelihood distribution
log_lik <- function(par){
  ll <- bm_model$llik(par)
  pr <- prior(par)
  return(ll+pr)
}

# Use an mcmc sampler
require(spBayes)
require(coda)
start_val <- bm_model$param$opt$par #the ML values
n.batch <- 500
l.batch <- 25

# run the mcmc
fit <- adaptMetropGibbs(log_lik, starting=start_val, batch=n.batch, batch.length=l.batch)

# plot the results
chain <- mcmc(fit$p.theta.samples)
plot(chain)

# check the distribution of the transformed values
rate_matrix <- t(apply(fit$p.theta.samples, 1, bm_model$param$sigmafun))
colnames(rate_matrix) <- c("Sigma [1,1]","Sigma [1,2]","Sigma [2,1]","Sigma [2,2]")
chain2 <- mcmc(rate_matrix)
plot(chain2)
```