

# Package: synlik (via r-universe)

October 23, 2024

**Type** Package

**Title** Synthetic Likelihood Methods for Intractable Likelihoods

**Version** 0.1.6

**Date** 2023-03-02

**Author** Matteo Fasiolo and Simon Wood

**Maintainer** Matteo Fasiolo <matteo.fasiolo@gmail.com>

**Description** Framework to perform synthetic likelihood inference for models where the likelihood function is unavailable or intractable.

**URL** [http://mfasiolo.github.io/synlik\\_release/](http://mfasiolo.github.io/synlik_release/)

**License** GPL (>= 2)

**Depends** R (>= 2.10), Rcpp (>= 0.12.0)

**Imports** methods, graphics, Matrix, compiler, stats, parallel

**Suggests** knitr, markdown, stabledist

**LinkingTo** Rcpp, RcppArmadillo

**VignetteBuilder** knitr

**RoxygenNote** 5.0.1

**NeedsCompilation** yes

**Repository** CRAN

**Date/Publication** 2023-03-02 14:10:02 UTC

## Contents

synlik-package . . . . .	2
ANYOrNULL-class . . . . .	4
bf . . . . .	4
blowSimul . . . . .	6
blow_sl . . . . .	7
checkNorm . . . . .	8
continue . . . . .	9

extractCorr	10
functionOrNULL-class	11
internal_C	11
nlar	12
numericOrNULL-class	13
orderDist	13
plot-smcmc	14
plot-synlik	15
rickerSimul	16
ricker_sl	17
robCov	18
simulate,synlik-method	19
slAcf	20
slice	21
slik	22
smcmc	23
smcmc-class	24
synlik-class	26

<b>Index</b>	<b>28</b>
--------------	-----------

---

synlik-package

*Synthetic Likelihood Methods for Intractable Likelihoods*

---

## Description

Package that provides Synthetic Likelihood methods for intractable likelihoods. The package is meant to be as general purpose as possible: as long as you are able to simulate data from your model you should be able to fit it.

## Details

Package: synlik  
 Type: Package  
 Version: 0.1.2  
 Date: 2018-05-22  
 License: GPL (>=2)

The package allows users to create objects of class `synlik` (S4), which are essentially constituted of a simulator function and a function (`summaries`) that transforms the data into summary statistics. The simulator can output any kind of data (vector, list, etc) and this will be passed directly to the `summaries` function. This allow the package to fit a large variety of models.

Once the model of interest has been set up as a `synlik` object, it is possible several methods on it. The function most useful function is `slik`, which can be used to evaluate the synthetic likelihood. The `slice.synlik` function allows to obtain and plot slices of the synthetic likelihood with respect

to model parameters. It is possible to simulate data or statistics from the model using the generic `simulate`, and to check the normality of the statistics using the `checkNorm` function. Unknown parameters can be estimated by MCMC, through the `smcmc` function. This function will return an object of class `smcmc` (S4), which contains all the inputs and results of the MCMC procedure.

Many functions in the package support parallel simulation on multiple cores.

### Author(s)

Matteo Fasiolo and Simon N. Wood

Maintainer: Matteo Fasiolo <matteo.fasiolo@gmail.com>

### References

Simon N Wood. Statistical inference for noisy nonlinear ecological dynamic systems. *Nature*, 466(7310):1102–1104, 2010.

### See Also

For some examples see the Vignettes (type `vignette("synlik")`).

### Examples

```
## Not run:
#### Here I put a simple example,
#### if you want to see more type: vignette("synlik")

## End(Not run)

#### Create synlik object
ricker_sl <- synlik(simulator = rickerSimul,
                  summaries = rickerStats,
                  param = c(logR = 3.8, logSigma = log(0.3), logPhi = log(10)),
                  extraArgs = list("nObs" = 50, "nBurn" = 50),
                  plotFun = function(input, ...){
                    plot(drop(input), type = 'l', ylab = "Pop", xlab = "Time", ...)
                  }
)

#### Simulate from the object
ricker_sl@data <- simulate(ricker_sl)
ricker_sl@extraArgs$obsData <- ricker_sl@data

#### Simulate statistics (each row is a vector of statistics)
simulate(ricker_sl, seed = 523, nsim = 10, stats = TRUE)

#### Plotting the data
plot(ricker_sl)

#### Checking multivariate normality of the statistics
checkNorm(ricker_sl)
```

```
#### Evaluate the likelihood
set.seed(4234)
slik(ricker_sl,
     param = c(logR = 3.8, logSigma = log(0.3), logPhi = log(10)),
     nsim = 1e3)

#### Plotting a slice of the log-Likelihood possibly using multiple cores
slice(object = ricker_sl,
      ranges = list("logR" = seq(3.5, 3.9, by = 0.02),
                   "logPhi" = seq(2, 2.6, by = 0.02),
                   "logSigma" = seq(-2, -0.5, by = 0.05)),
      param = c(logR = 3.8, logSigma = log(0.3), logPhi = log(10)),
      nsim = 500, multicore = FALSE)

#### MCMC estimation possibly using multiple cores
set.seed(4235)
ricker_sl <- smcmc(ricker_sl,
                  initPar = c(3.2, -1, 2.6),
                  niter = 50,
                  burn = 3,
                  priorFun = function(input, ...) 0,
                  propCov = diag(c(0.1, 0.1, 0.1))^2,
                  nsim = 1e3,
                  multicore = FALSE)

# Continue with additional 50 iterations
ricker_sl <- continue(ricker_sl, niter = 50)

# Plotting results on transformed scale (exponential)
trans <- rep("exp", 3)
names(trans) <- names(ricker_sl@param)

plot(ricker_sl)
```

---

ANYOrNULL-class

*Dummy class*

---

### Description

Class unions for internal use only

---

bf

*Nicholson's 1954 blowfly data*

---

### Description

Data from figures 3 and 4 of Nicholson, 1954.

**Usage**

```
data(bf1)
```

**Arguments**

```
bf1          the dataset name
```

**Details**

bf1 is Nisbet and Gurney's run 1, and Nicholson's (1954) figure 3 (adult food limitation). The data are actually from the global population dynamics database at Silwood. They are daily: Nicholson's figure 3 plots data every other day, but the text says that measurements were taken daily. However elsewhere they are reported every other day. Probably best to assume that they have been interpolated to daily.

bf2 and bf3 are digitized from Nicholson's (1954) figure 4. bf2 is the upper series: larval food limitation, with 50g per day of larval food provided. bf3 is the lower series: same set up, half as much food. These are Nisbet and Gurney's runs 2 and 3, respectively.

**Value**

matrix of replicate data series

**Author(s)**

Simon N. Wood, maintainer Matteo Fasiolo <matteo.fasiolo@@gmail.com>

**References**

Alexander J Nicholson. An outline of the dynamics of animal populations. Australian Journal of Zoology, 2(1):9–65, 1954.

**See Also**

blowfly

**Examples**

```
library(synlik)
data(bf1)
data(bf2)
data(bf3)
par(mfrow=c(3,1),mar=c(4,4,1,1))
with(bf1,plot(day,pop,type="l"))
with(bf1,points(day,pop,pch=20,cex=.8))
abline(mean(bf1$pop),0,col=2); abline(quantile(bf1$pop),0,col=3);
with(bf2,plot(day,pop,type="l"))
with(bf2,points(day,pop,pch=20,cex=.8))
abline(mean(bf2$pop),0,col=2); abline(quantile(bf2$pop),0,col=3);
with(bf3,plot(day,pop,type="l"))
with(bf3,points(day,pop,pch=20,cex=.8))
abline(mean(bf3$pop),0,col=2); abline(quantile(bf3$pop),0,col=3);
```

---

blowSimul	<i>Simulates from the blowfly model</i>
-----------	---

---

**Description**

Simulator for the blowfly model proposed by Wood (2010).

**Usage**

```
blowSimul(param, nsim, extraArgs, ...)
```

**Arguments**

param	vector of log-parameters: delta, P, N0, var.p, tau and var.d. The interpretation of these parameters is described in Wood (2010).
nsim	Number of simulations from the model.
extraArgs	A named list of additional arguments: <ul style="list-style-type: none"> <li>• nObs = Length of each simulated time series.</li> <li>• nBurn = Number of initial steps to be discarded before saving the following nObs steps.</li> <li>• steps = Positive integer. If steps == n the observations correspond to n time steps.</li> </ul>
...	Need for compatibility with synlik, but not used.

**Value**

A matrix nsim by nObs, where each row is a simulated path.

**Author(s)**

Simon Wood and Matteo Fasiolo <matteo.fasiolo@gmail.com>.

**References**

Simon N Wood. Statistical inference for noisy nonlinear ecological dynamic systems. *Nature*, 466(7310):1102–1104, 2010.

Brillinger, D. R., J. Guckenheimer, P. Guttorp, and G. Oster. 1980. Empirical modelling of population time series data: the case of age and density dependent vital rates. *Lectures on Mathematics in the Life Sciences*13:65-90.

Nicholson, A. J. 1957. The self-adjustment of populations to change. *Cold Spring Harbor Symposia on Quantitative Biology*22:153-173.

**See Also**

[blow\\_sl](#)

**Examples**

```
tmp <- blowSimul(param = log( c( "delta" = 0.16, "P" = 6.5, "N0" = 400,
                               "var.p" = 0.1, "tau" = 14, "var.d" = 0.1) ),
               nsim = 2,
               extraArgs = list("nObs" = 200, "nBurn" = 1000, "steps" = 2))
matplot(t(tmp), type = 'l', ylab = "Y", xlab = "Time")
```

blow\_sl

*Blowfly model***Description**

synlik object containing the blowfly model proposed by Wood (2010). The main components are the simulator [blowSimul](#) and the statistics blowStats, described in the same reference.

**Author(s)**

Simon Wood and Matteo Fasiolo <matteo.fasiolo@gmail.com>.

**References**

Simon N Wood. Statistical inference for noisy nonlinear ecological dynamic systems. *Nature*, 466(7310):1102–1104, 2010.

**See Also**

[blowSimul](#)

**Examples**

```
data(blow_sl)

plot(blow_sl)
simulate(blow_sl, stats = TRUE)

slik(blow_sl,
     param = log( c( "delta" = 0.16, "P" = 6.5, "N0" = 400,
                    "var.p" = 0.1, "tau" = 14, "var.d" = 0.1) ),
     nsim   = 1e3)

# Using Nicholson's data
data(bf1)

plot(blow_sl)

blow_sl@data <- bf1$pop
blow_sl@extraArgs$obsData <- bf1$pop #Important: blow_sl@blowStats uses the observed data

slik(blow_sl,
```

```
param = log( c( "delta" = 0.16, "P" = 6.5, "N0" = 400,
               "var.p" = 0.1, "tau" = 14, "var.d" = 0.1) ),
nsim   = 1e3)
```

---

checkNorm	<i>Checking the multivariate normal approximation.</i>
-----------	--

---

## Description

Given an object of class `synlik` this routine provides a graphical check of whether the distribution of the random summary statistics is multivariate normal.

## Usage

```
checkNorm(object, param = object@param, nsim = 1000, observed = NULL,
          cex.axis = 1, cex.lab = 1, ...)
```

## Arguments

<code>object</code>	An object of class <code>synlik</code> or a matrix where each row is a random vector.
<code>param</code>	A vector of model's parameters at which the summary statistics will be simulated.
<code>nsim</code>	number of summary statistics to be simulated if <code>object</code> is of class <code>synlik</code> , otherwise it is not used.
<code>observed</code>	A vector of observed summary statistics. By default <code>NULL</code> , so <code>object@data</code> will be used as observed statistics. It will be looked at only if <code>object</code> is a matrix.
<code>cex.axis</code>	Axis scale expansion factor.
<code>cex.lab</code>	Axis label expansion factor.
<code>...</code>	additional arguments to be passed to <code>object@simulator</code> and <code>object@summaries</code> . In general I would avoid using it and including in <code>object@extraArgs</code> everything they need.

## Details

The method is from section 7.5 of Krzanowski (1988). The replicate vectors of summary statistic  $S$  are transformed to variables which should be univariate chi squared r.v.s with DoF given by the number of rows of  $S$ . An appropriate QQ-plot is produced, and the proportion of the data differing substantially from the ideal line is reported. Deviations at the right hand end of the plot indicate that the tail behaviour of the Normal approximation is poor: in the context of synthetic likelihood this is of little consequence. Secondly,  $s$  is transformed to a vector which should be i.i.d.  $N(0,1)$  under multivariate normality, and a QQ plot is produced. Unfortunately this approach is not very useful unless the dimension of  $s$  is rather large. In simulations, perfectly MVN data produce highly variable results, so that the approach lacks any real power.

**Value**

Mainly produces plots and prints output. Also an array indicating proportion of simulated statistics smaller than observed.

**Author(s)**

Simon N. Wood, maintained by Matteo Fasiolo <matteo.fasiolo@gmail.com>.

**References**

Krzanowski, W.J. (1988) Principles of Multivariate Analysis. Oxford.

**Examples**

```
#### Create Object
data(ricker_sl)

#### Simulate from the object
ricker_sl@data <- simulate(ricker_sl)
ricker_sl@extraArgs$obsData <- ricker_sl@data

#### Checking multivariate normality
checkNorm(ricker_sl)

# With matrix input
checkNorm(matrix(rnorm(200), 100, 2))
```

---

continue

*Continuing estimation.*

---

**Description**

Generic function, that given the results of an estimation procedure (ex. MCMC or maximum likelihood optimization) continues the procedure for some more iterations.

**Usage**

```
continue(object, ...)

## S4 method for signature 'smcmc'
continue(object, niter = object@niter, nsim = object@nsim,
  propCov = object@propCov, targetRate = object@targetRate,
  recompute = object@recompute, multicore = object@multicore,
  ncores = object@ncores, cluster = NULL, control = object@control, ...)
```

**Arguments**

object	An object representing the results of an estimation procedure which we wish to continue. For example it might represents an MCMC chain.
...	additional arguments to be passed to <code>slik</code> function, see <a href="#">slik</a> .
niter	see <a href="#">smcmc-class</a> .
nsim	see <a href="#">smcmc-class</a> .
propCov	see <a href="#">smcmc-class</a> .
targetRate	see <a href="#">smcmc-class</a> .
recompute	see <a href="#">smcmc-class</a> .
multicore	see <a href="#">smcmc-class</a> .
ncores	see <a href="#">smcmc-class</a> .
cluster	an object of class <code>c("SOCKcluster", "cluster")</code> . This allows the user to pass her own cluster, which will be used if <code>multicore == TRUE</code> . The user has to remember to stop the cluster.
control	see <a href="#">smcmc-class</a> .

**Details**

When `is("smcmc", object) == TRUE` continues MCMC estimation of an object of class `smcmc`. All input parameters are defaulted to the corresponding slots in the input object, with the exception of `cluster`. The chain restarts were it ended, burn-in is set to zero, the same prior (if any) is used.

**Value**

An object of the same class as `object`, where the results of the estimation have been updated.

**See Also**

For examples, see [smcmc-class](#).

---

extractCorr	<i>Extracting correlations from a covariance matrix</i>
-------------	---

---

**Description**

Extracting correlations from a covariance matrix

**Usage**

```
extractCorr(mat)
```

**Arguments**

mat	A covariance matrix.
-----	----------------------

**Value**

The correlation matrix embedded in mat.

**Examples**

```
# 2 dimensional case
d <- 2
tmp <- matrix(rnorm(d^2), d, d)
mcov <- tcrossprod(tmp, tmp)

# Covariance matrix
mcov

# Correlation matrix
extractCorr(mcov)
```

---

functionOrNULL-class *Dummy class*

---

**Description**

Class unions for internal use only

---

internal\_C *Internal C and C++ function*

---

**Description**

This functions are for internal use only.

**Author(s)**

Simon Wood and Matteo Fasiolo <matteo.fasiolo@gmail.com>.

---

`nlar`*Estimate non-linear autoregressive coefficients*

---

**Description**

Function that, given time series data, transforms them into summary statistics using polynomial autoregression.

**Usage**

```
nlar(x, lag, power)
```

**Arguments**

<code>x</code>	a matrix. Each column contains a replicate series.
<code>lag</code>	vector of lags, for rhs terms.
<code>power</code>	vector of powers, for rhs terms.

**Value**

a matrix where each column contains the coefficients for a different replicate.

**Author(s)**

Simon N. Wood, maintainer Matteo Fasiolo <matteo.fasiolo@gmail.com>.

**Examples**

```
library(synlik)
set.seed(10)
x <- matrix(runif(200),100,2)
beta <- nlar(x,lag=c(1,1),power=c(1,2))
y <- x[,1]
y <- y - mean(y)
z <- y[1:99];y <- y[2:100]
lm(y~z+I(z^2)-1)
beta

## NA testing
x[5,1] <- x[45,2] <- NA
beta <- nlar(x,lag=c(1,1),power=c(1,2))
y <- x[,1]
y <- y - mean(y,na.rm=TRUE)
z <- y[1:99];y <- y[2:100]
lm(y~z+I(z^2)-1)
beta

## higher order...
set.seed(10)
```

```
x <- matrix(runif(100),100,2)
beta <- nlar(x,lag=c(6,6,6,1,1),power=c(1,2,3,1,2))
k <- 2
y <- x[,k]
y <- y - mean(y)
ind <- (1+6):100
y6 <- y[ind-6];y1 <- y[ind-1];y <- y[ind]
beta0 <- coef(lm(y~y6+I(y6^2)+I(y6^3)+y1+I(y1^2)-1))
as.numeric(beta[,k]);beta0;beta0-as.numeric(beta[,k])
```

---

numericOrNULL-class     *Dummy class*

---

### Description

Class unions for internal use only

---

orderDist                     *Summarize marginal distribution of (differenced) series.*

---

### Description

Summarizes (difference) distribution of replicate series, by regressing ordered differenced series on a reference series (which might correspond to observed data).

### Usage

```
orderDist(x, z, np = 3, diff = 1)
```

### Arguments

x	a matrix. Each column contains a replicate series.
z	vector of lags, for rhs terms.
np	maximum power on rhs of regression.
diff	order of differencing (zero for none).

### Value

a matrix where each column contains the coefficients for a different replicate.

### Author(s)

Simon N. Wood, maintainer Matteo Fasiolo <matteo.fasiolo@gmail.com>.

**Examples**

```

library(synlik)
set.seed(10)
n <- 100;nr <- 3
x <- matrix(runif(n*nr),n,nr)
z <- runif(n)
beta <- orderDist(x,z,np=3,diff=1)

zd <- z;xd <- x[,3]
zd <- diff(zd,1);xd <- diff(xd,1)
zd <- sort(zd);zd <- zd - mean(zd)
xd <- sort(xd);xd <- xd - mean(xd)
lm(xd~zd+I(zd^2)+I(zd^3)-1)

```

---

plot-smcmc

*Plotting objects of class smcmc.*


---

**Description**

Method for plotting an object of class smcmc.

**Usage**

```

## S4 method for signature 'smcmc,missing'
plot(x, trans = NULL, addPlot1 = NULL,
     addPlot2 = NULL, ...)

```

**Arguments**

x	An object of class smcmc.
trans	Name list or vector containing names of transforms for some parameters (ex: <code>list("par1" = "exp", "par2" = "log")</code> ). The transformations will be applied before plotting.
addPlot1	Name of additional plotting function that will be call after the MCMC chain have been plotted. It has to have prototype <code>fun(nam, ...)</code> where <code>nam</code> will be the parameter name. See "examples".
addPlot2	Name of additional plotting function that will be call after the histograms have been plotted. It has to have prototype <code>fun(nam, ...)</code> where <code>nam</code> will be the parameter name. See "examples".
...	additional arguments to be passed to the plotting functions.

**See Also**

[smcmc-class](#), [plot](#).

**Examples**

```
data(ricker_smcmc)

# Functions for additional annotations (true parameters)
addline1 <- function(parNam, ...){
  abline(h = exp(ricker_smcmc@param[parNam]), lwd = 2, lty = 2, col = 3)
}
addline2 <- function(parNam, ...){
  abline(v = exp(ricker_smcmc@param[parNam]), lwd = 2, lty = 2, col = 3)
}

# Transformations (exponentials)
trans <- rep("exp", 3)
names(trans) <- names(ricker_smcmc@param)

plot(ricker_smcmc,
     trans = trans,
     addPlot1 = "addline1",
     addPlot2 = "addline2")
```

---

plot-synlik

*Method for plotting an object of class synlik.*

---

**Description**

It basically calls the slot `object@plotFun` with input `object@data`, if it has been provided by the user. Otherwise it tries to use the `plot(x = object@data, y, ...)` generic.

**Usage**

```
## S4 method for signature 'synlik,missing'
plot(x, y, ...)
```

**Arguments**

<code>x</code>	An object of class <code>synlik</code> .
<code>y</code>	Useless argument, only here for compatibility reasons.
<code>...</code>	additional arguments to be passed to <code>object@plotFun</code> .

**Author(s)**

Matteo Fasiolo <matteo.fasiolo@gmail.com>

**See Also**

[synlik-class](#), [plot](#).

**Examples**

```

data(ricker_sl)

# Using ricker_sl@plotFun
plot(ricker_sl)

# Using generic plot, doesn't work well because object@data is a matrix.
ricker_sl@plotFun <- NULL
plot(ricker_sl)

```

---

rickerSimul	<i>Simulates from the ricker model</i>
-------------	--

---

**Description**

Simulator for the stochastic Ricker model, as described by Wood (2010). The observations are  $Y_t \sim \text{Pois}(\Phi * N_t)$ , and the dynamics of the hidden state are given by  $N_t = r * N_{t-1} * \exp(-N_{t-1} + e_t)$ , where  $e_t \sim N(0, \text{Sigma}^2)$ .

**Usage**

```
rickerSimul(param, nsim, extraArgs, ...)
```

**Arguments**

param	vector of log-parameters: logR, logSigma, logPhi. Alternatively a matrix nsim by 3 where each row is a different parameter vector.
nsim	Number of simulations from the model.
extraArgs	A named list of additional arguments: <ul style="list-style-type: none"> <li>• nObs = Length of each simulated time series.</li> <li>• nBurn = Number of initial steps to be discarded before saving the following nObs steps.</li> <li>• randInit = if TRUE (default) the initial state N0 is runif(0, 1), otherwise it is equal to extraArgs\$initVal.</li> <li>• initVal = initial value N0, used only if extraArgs\$randInit == TRUE.</li> </ul>
...	Need for compatibility with synlik, but not used.

**Value**

A matrix nsim by nObs, where each row is a simulated path.

**Author(s)**

Simon Wood and Matteo Fasiolo <matteo.fasiolo@gmail.com>.

## References

Simon N Wood. Statistical inference for noisy nonlinear ecological dynamic systems. *Nature*, 466(7310):1102–1104, 2010.

## See Also

[ricker\\_sl](#)

## Examples

```
tmp <- rickerSimul(c(3.8, -1.2, 2.3), nsim = 2, extraArgs = list("nObs" = 50, "nBurn" = 200))
matplot(t(tmp), type = 'l', ylab = "Y", xlab = "Time")

parMat <- rbind(c(3.8, -1.2, 2.3), # Chaotic
               c(2.5, -1.2, 2.3)) # Not Chaotic

par(mfrow = c(2, 1))
tmp <- rickerSimul(parMat, nsim = 2, extraArgs = list("nObs" = 50, "nBurn" = 200))
plot(tmp[1, ], type = 'l', ylab = "Y", xlab = "Time")
plot(tmp[2, ], type = 'l', ylab = "Y", xlab = "Time")
```

---

ricker\_sl

*Ricker model*

---

## Description

`ricker_sl` is `synlik` object containing the stochastic Ricker model, `ricker_smc` is a `smcmc` object which also contains the results of some MCMC iterations. The model is described [rickerSimul](#) and in Wood (2010). The main components of the object are the simulator [rickerSimul](#) and the statistics `rickerStats`, described in the same reference.

## Author(s)

Simon Wood and Matteo Fasiolo <matteo.fasiolo@gmail.com>.

## References

Simon N Wood. Statistical inference for noisy nonlinear ecological dynamic systems. *Nature*, 466(7310):1102–1104, 2010.

## See Also

[rickerSimul](#)

**Examples**

```

data(ricker_sl)

plot(ricker_sl)
simulate(ricker_sl, stats = TRUE)

slik(ricker_sl,
      param = c(logR = 3.8, logSigma = log(0.3), logPhi = log(10) ),
      nsim   = 1e3)

# Using Nicholson's data
data(ricker_smcmc)

plot(ricker_smcmc)

```

robCov

*Robust covariance matrix estimation***Description**

Obtains a robust estimate of the covariance matrix of a sample of multivariate data, using Campbell's (1980) method as described on p231-235 of Krzanowski (1988).

**Usage**

```
robCov(sY, alpha = 2, beta = 1.25)
```

**Arguments**

sY	A matrix, where each column is a replicate observation on a multivariate r.v.
alpha	tuning parameter, see details.
beta	tuning parameter, see details.

**Details**

Campbell (1980) suggests an estimator of the covariance matrix which downweights observations at more than some Mahalanobis distance  $d_0$  from the mean.  $d_0$  is  $\sqrt{\text{nrow}(sY) + \alpha} / \sqrt{2}$ . Weights are one for observations with Mahalanobis distance,  $d$ , less than  $d_0$ . Otherwise weights are  $d_0 \cdot \exp(-.5 \cdot (d - d_0)^2 / \beta) / d$ . The defaults are as recommended by Campbell. This routine also uses pre-conditioning to ensure good scaling and stable numerical calculations.

**Value**

A list where:

- $E$  square root of the inverse covariance matrix. i.e. the inverse cov matrix is  $t(E) \%*\% E$ ;
- $\text{half.ldet.V}$  Half the log of the determinant of the covariance matrix;

- mYThe estimated mean;
- sdThe estimated standard deviations of each variable.

### Author(s)

Simon N. Wood, maintained by Matteo Fasiolo <matteo.fasiolo@gmail.com>.

### References

Krzanowski, W.J. (1988) Principles of Multivariate Analysis. Oxford. Campbell, N.A. (1980) Robust procedures in multivariate analysis I: robust covariance estimation. JRSSC 29, 231-237.

### Examples

```
p <- 5;n <- 100
Y <- matrix(runif(p*n),p,n)
robCov(Y)
```

---

simulate,synlik-method

*Simulate data or statistics from an object of class synlik.*

---

### Description

Simulate data or statistics from an object of class synlik.

### Usage

```
## S4 method for signature 'synlik'
simulate(object, nsim, seed = NULL, param = object@param,
  stats = FALSE, clean = TRUE, verbose = TRUE, ...)
```

### Arguments

object	An object of class synlik.
nsim	Number of simulations from the model.
seed	Random seed to be used. It is not passed to the simulator, but simply passed to <code>set.seed()</code> from within <code>simulate.synlik</code> .
param	Vector of parameters passed to <code>object@simulator</code> .
stats	If TRUE the function transforms the simulated data into statistics using <code>object@summaries</code> .
clean	If TRUE the function tries to clean the statistics from NaNs or non-finite values. Given that <code>object@summaries</code> has to return a numeric vector or a matrix where each row is a simulation, rows containing non-finite values will be discarded.
verbose	If TRUE the function will complain if, for instance, the simulations contain lots of non-finite values.
...	additional arguments to be passed to <code>object@simulator</code> and <code>object@summaries</code> . In general I would avoid using it and including <code>object@extraArgs</code> everything they need.

**Value**

If `stats == FALSE` the output will that of `object@simulator`, which depends on the simulator used by the user. If `stats == TRUE` the output will be a matrix where each row is vector of simulated summary statistics.

**Author(s)**

Matteo Fasiolo <matteo.fasiolo@gmail.com>

**See Also**

[synlik-class](#), [simulate](#).

**Examples**

```
data(ricker_sl)

# Simulate data
simulate(ricker_sl, nsim = 2)

# Simulate statistics
simulate(ricker_sl, nsim = 2, stats = TRUE)
```

---

slAcf

*Estimate auto-covariances for multiple datasets.*

---

**Description**

Function that, give time series data, transforms them into auto-covariances with different lags.

**Usage**

```
slAcf(x, max.lag = 10)
```

**Arguments**

`x` a matrix. Each column contains a replicate series.  
`max.lag` How many lags to use.

**Value**

a matrix where each column contains the coefficients for a different replicate. The first coefficient corresponds to `lag == 0`, hence it is the variance, the second is the covariance one step ahead and so on.

**Author(s)**

Simon N. Wood, maintainer Matteo Fasiolo <matteo.fasiolo@gmail.com>.

**Examples**

```
library(synlik)
set.seed(10)
x <- matrix(runif(1000),100,10)
acf <- slAcf(x)
```

---

 slice
 

---



---

*Plot slices of the synthetic log-likelihood.*


---

**Description**

Plot slices of the synthetic log-likelihood.

**Usage**

```
slice(object, ranges, nsim, param = object@param, pairs = FALSE,
      draw = TRUE, trans = NULL, multicore = FALSE, ncores = detectCores() -
      1, cluster = NULL, ...)
```

**Arguments**

object	synlik object.
ranges	ranges of values along which we want the slices. If <code>length(parName) == 1</code> than range has a vector, while if <code>length(parName) == 2</code> it have to be a named list of 2 vectors (ex: <code>list("alpha" = 1:10, "beta" = 10:1)</code> ).
nsim	Number of simulations used to evaluate the synthetic likelihood at each location.
param	Named vector containing the value of the ALL parameters (including the sliced one). Parameters that are not in <code>parName</code> will be fixed to the values in <code>param</code> .
pairs	if TRUE the function will produce a 2D slice for every pair of parameters in ranges. FALSE by default.
draw	If TRUE the slice will be plotted.
trans	Named vector or list of transformations to be applied to the parameters in <code>parName</code> before plotting ex: <code>trans = c(s = "exp", d = "exp")/</code>
multicore	If TRUE the <code>object@simulator</code> and <code>object@summaries</code> functions will be executed in parallel. That is the <code>nsim</code> simulations will be divided in multiple cores.
ncores	Number of cores to use if <code>multicore == TRUE</code> .
cluster	An object of class <code>c("SOCKcluster", "cluster")</code> . This allows the user to pass her own cluster, which will be used if <code>multicore == TRUE</code> . The user has to remember to stop the cluster.
...	additional arguments to be passed to <code>slik()</code> , see <a href="#">slik</a> .

**Value**

Either a vector or matrix of log-synthetic likelihood estimates, depending on whether `length(parNames) == 1` or `2`. These are returned invisibly.

**Author(s)**

Matteo Fasiolo <matteo.fasiolo@gmail.com>

**Examples**

```
data(ricker_sl)

# Plotting slices of the logLikelihood
slice(object = ricker_sl,
      ranges = list("logR" = seq(3.5, 3.9, by = 0.01),
                   "logPhi" = seq(2, 2.6, by = 0.01),
                   "logSigma" = seq(-2, -0.5, by = 0.01)),
      param = c(logR = 3.8, logSigma = log(0.3), logPhi = log(10)),
      nsim = 500)

## Not run:
# Plotting a contour of the logLikelihood
slice(object = ricker_sl,
      ranges = list("logR" = seq(3.5, 3.9, by = 0.01),
                   "logPhi" = seq(2, 2.6, by = 0.01),
                   "logSigma" = seq(-2, -0.5, by = 0.04)),
      pairs = TRUE,
      param = c(logR = 3.8, logSigma = log(0.3), logPhi = log(10)),
      nsim = 500, multicore = TRUE)

## End(Not run)
```

---

slik

*Evaluates the synthetic log-likelihood.*

---

**Description**

Evaluates the synthetic log-likelihood.

**Usage**

```
slik(object, param, nsim, multicore = FALSE, ncores = detectCores() - 1,
     cluster = NULL, ...)
```

**Arguments**

object	An object of class <code>synlik</code> .
param	Vector of parameters at which the synthetic likelihood will be evaluated.
nsim	Number of simulation from the model.
multicore	(logical) if TRUE the <code>object@simulator</code> and <code>object@summaries</code> functions will be executed in parallel. That is the <code>nsim</code> simulations will be divided in multiple cores.

ncores	(integer) number of cores to use if multicore == TRUE.
cluster	an object of class c("SOCKcluster", "cluster"). This allows the user to pass her own cluster, which will be used if multicore == TRUE. The user has to remember to stop the cluster.
...	additional arguments to be passed to object@simulator and object@summaries. In general I would avoid using it and including object@extraArgs everything they need.

**Value**

The estimated value of the synthetic log-likelihood at param.

**Author(s)**

Matteo Fasiolo <matteo.fasiolo@gmail.com>

**References**

Simon N Wood. Statistical inference for noisy nonlinear ecological dynamic systems. *Nature*, 466(7310):1102–1104, 2010.

**Examples**

```
data(ricker_sl)
set.seed(643)
slik(ricker_sl, param = c(3.8, -1.2, 2.3), nsim = 500)
```

---

smcmc

---

*MCMC parameter estimation for objects of class synlik.*


---

**Description**

MCMC parameter estimation for objects of class synlik.

**Usage**

```
smcmc(object, initPar, niter, nsim, propCov, burn = 0,
       priorFun = function(param, ...) 0, targetRate = NULL, recompute = FALSE,
       multicore = !is.null(cluster), cluster = NULL, ncores = detectCores() -
       1, control = list(), ...)
```

**Arguments**

object	An object of class <code>synlik</code> .
initPar	see <a href="#">smcmc-class</a> .
niter	see <a href="#">smcmc-class</a> .
nsim	see <a href="#">smcmc-class</a> .
propCov	see <a href="#">smcmc-class</a> .
burn	see <a href="#">smcmc-class</a> .
priorFun	see <a href="#">smcmc-class</a> .
targetRate	see <a href="#">smcmc-class</a> .
recompute	see <a href="#">smcmc-class</a> .
multicore	see <a href="#">smcmc-class</a> .
cluster	an object of class <code>c("SOCKcluster", "cluster")</code> . This allows the user to pass her own cluster, which will be used if <code>multicore == TRUE</code> . The user has to remember to stop the cluster.
ncores	see <a href="#">smcmc-class</a> .
control	see <a href="#">smcmc-class</a> .
...	additional arguments to be passed to <code>slik</code> function, see <a href="#">slik</a> .

**Value**

An object of class `smcmc`.

**Author(s)**

Matteo Fasiolo <matteo.fasiolo@gmail.com>, code for adaptive step from the `adaptMCMC` package.

**References**

Vihola, M. (2011) Robust adaptive Metropolis algorithm with coerced acceptance rate. *Statistics and Computing*.

---

smcmc-class

smcmc-class

---

**Description**

Object representing the results of MCMC estimation on an object of class `synlik`, from which it inherits.

**Slots**

- initPar** Vector of initial parameters where the MCMC chain will start (*numeric*).
- niter** Number of MCMC iterations (*integer*).
- nsim** Number of simulations from the simulator at each step of the MCMC algorithm (*integer*).
- burn** Number of initial MCMC iterations that are discarded (*integer*).
- priorFun** Function that takes a vector of parameters as input and the log-density of the prior as output. If the output is not finite the proposed point will be discarded. (*function*). The function needs to have signature `fun(x, ...)`, where `x` represents the input parameters (*function*).
- propCov** Matrix representing the covariance matrix to be used to perturb the parameters at each step of the MCMC chain (*matrix*).
- targetRate** Target rate for the adaptive MCMC sampler. Should be in  $(0, 1)$ , default is `NULL` (no adaptation). The adaptation uses the approach of Vihola (2011). (*numeric*)
- recompute** If `TRUE` the synthetic likelihood will be evaluated at the current and proposed positions in the parameter space (thus doubling the computational effort). If `FALSE` the likelihood of the current position won't be re-estimated (*logical*).
- multicore** If `TRUE` the `object@simulator` and `object@summaries` functions will be executed in parallel. That is the `nsim` simulations will be divided in multiple cores (*logical*).
- ncores** Number of cores to use if `multicore == TRUE` (*integer*).
- accRate** Acceptance rate of the MCMC chain, between 0 and 1 (*numeric*).
- chains** Matrix of size `niter` by `length(initPar)` where the `i`-th row contains the position of the MCMC algorithm in the parameter space at the `i`-th (*matrix*).
- llkChain** Vector of `niter` elements where the `i`-th element is contains the estimate of the synthetic likelihood at the `i`-th iteration (*numeric*).
- control** Control parameters used by the MCMC sampler:
- `theta` = controls the speed of adaption. Should be between 0.5 and 1. A lower gamma leads to faster adaption.
  - `adaptStart` = iteration where the adaption starts. Default 0.
  - `adaptStop` = iteration where the adaption stops. Default `burn + niter`
  - `saveFile` = path to the file where the intermediate results will be stored (ex: `"~/Res.RData"`).
  - `saveFreq` = frequency with which the intermediate results will be saved on `saveFile`. Default 100.
  - `verbose` = if `TRUE` intermediate posterior means will be printer.
  - `verbFreq` = frequency with which the intermediate posterior means will be printer. Default 500.

**Author(s)**

Matteo Fasiolo <matteo.fasiolo@gmail.com>

**References**

Vihola, M. (2011) Robust adaptive Metropolis algorithm with coerced acceptance rate. *Statistics and Computing*.

**Examples**

```

# Load "synlik" object
data(ricker_sl)

plot(ricker_sl)

# MCMC estimation
set.seed(4235)
ricker_sl <- smcmc(ricker_sl,
                  initPar = c(3.2, -1, 2.6),
                  niter = 50,
                  burn = 3,
                  priorFun = function(input, ...) 1,
                  propCov = diag( c(0.1, 0.1, 0.1) )^2,
                  nsim = 200,
                  multicore = FALSE)

# Continue with additional 50 iterations
ricker_sl <- continue(ricker_sl, niter = 50)

plot(ricker_sl)

```

---

synlik-class

synlik-class

---

**Description**

Basic class for simulation-based approximate inference using Synthetic Likelihood methods.

**Usage**

```
synlik(...)
```

**Arguments**

... See section "Slots".

**Slots**

**param** Named vector of parameters used by `object@simulator` (numeric).

**simulator** Function that simulates from the model (function). It has to have prototype `fun(param, nsim, extraArgs, ...)`. If `summaries()` is not specified the `simulator()` has output a matrix with `nsim` rows, where each row is a vector of simulated statistics. Otherwise it can output any kind of object, and this output will be passed to `summaries()`.

**summaries** Function that transforms simulated data into summary statistics (function). It has to have prototype `fun(x, extraArgs, ...)` and it has to output a matrix with `nsim` rows, where each row is a vector of simulated statistics. Parameter `x` contains the data.

**data** Object containing the observed data or statistics (ANY).

**extraArgs** List containing all the extra arguments to be passed to `object@simulator` and `object@summaries` (list).

**plotFun** Function that will be used to plot `object@data`. Prototype should be `fun(x, ...)` (function).

### Author(s)

Matteo Fasiolo <matteo.fasiolo@gmail.com>

### References

Simon N Wood. Statistical inference for noisy nonlinear ecological dynamic systems. *Nature*, 466(7310):1102–1104, 2010.

### Examples

```
#### Create Object
ricker_sl <- synlik(simulator = rickerSimul,
                  summaries = rickerStats,
                  param = c( logR = 3.8, logSigma = log(0.3), logPhi = log(10) ),
                  extraArgs = list("nObs" = 50, "nBurn" = 50),
                  plotFun = function(input, ...)
                    plot(drop(input), type = 'l', ylab = "Pop", xlab = "Time", ...))

# Simulate from the object
ricker_sl@data <- simulate(ricker_sl)
ricker_sl@extraArgs$obsData <- ricker_sl@data # Needed by WOOD2010 statistics

plot(ricker_sl)
```

# Index

ANYOrNULL-class, 4

bf, 4  
bf1 (bf), 4  
bf2 (bf), 4  
bf3 (bf), 4  
blow\_sl, 6, 7  
blow\_smcnc (blow\_sl), 7  
blowC (internal\_C), 11  
blowSimul, 6, 7  
blowStats (blow\_sl), 7

checkBoundsCpp (internal\_C), 11  
checkNorm, 8  
cleanStats (internal\_C), 11  
continue, 9  
continue, smcnc-method (continue), 9

extractCorr, 10

functionOrNULL-class, 11

internal\_C, 11

nlar, 12  
numericOrNULL-class, 13

order\_reg (internal\_C), 11  
orderDist, 13

plot, 14, 15  
plot, smcnc, missing-method (plot-smcnc), 14  
plot, synlik, missing-method (plot-synlik), 15  
plot-smcnc, 14  
plot-synlik, 15

ricker\_sl, 17, 17  
ricker\_smcnc (ricker\_sl), 17  
rickerSimul, 16, 17

rickerStats (ricker\_sl), 17  
robCov, 18

simpleModelsWrap (internal\_C), 11  
simulate, 20  
simulate, synlik-method, 19  
slAcf, 20  
slacf (internal\_C), 11  
slice, 21  
slik, 10, 21, 22, 24  
slnlar (internal\_C), 11  
smcnc, 23  
smcnc-class, 24  
synlik (synlik-class), 26  
synlik-class, 26  
synlik-package, 2