Package ‘mixAR’

December 19, 2023

Type Package

Title Mixture Autoregressive Models

Version 0.22.8

Maintainer Georgi N. Boshnakov <georgi.boshnakov@manchester.ac.uk>


License GPL (>= 2)

LazyLoad yes

Depends R (>= 3.5), methods

Imports stats, graphics, utils, stats4, BB, combinat, timeDate, fGarch, Rdpack (>= 0.7), gbutils (>= 0.3-1), MCMCpack, e1071, permute, mvtnorm

Suggests fma, testthat, covr

RdMacros Rdpack


BugReports https://github.com/GeoBosh/mixAR/issues


Encoding UTF-8
RoxygenNote 7.1.1

NeedsCompilation no

Author Georgi N. Boshnakov [aut, cre],
Davide Ravagli [aut]

Repository CRAN

Date/Publication 2023-12-19 01:40:02 UTC

R topics documented:
mixAR-package .................................................. 3
bayes_mixAR .................................................... 7
Choose_pk ....................................................... 9
cond_loglik .................................................... 10
dist_norm ....................................................... 11
e_m_est_dist ................................................... 12
e_m_est_sigma ................................................ 13
e_m_rinit ....................................................... 14
est_templ ....................................................... 15
exampleModels ................................................. 16
fit_mixAR-methods ............................................. 17
fit_mixARreg-methods ......................................... 19
fit_mixVAR-methods .......................................... 22
fnoise ......................................................... 23
get_edist-methods ........................................... 25
inner .......................................................... 26
isStable ....................................................... 27
label_switch .................................................. 28
lik_params ..................................................... 29
make_fcond_lik-methods ..................................... 30
marg_loglik ................................................... 31
MixAR-class ................................................... 33
mixAR-methods ............................................... 34
mixARemFixedPoint .......................................... 35
MixARGaussian-class ........................................ 37
MixARGen-class ............................................... 39
mixARnoise_sim ............................................... 41
mixAR_BIC ..................................................... 42
mixARCond_probs ............................................ 44
mixAR_diag ................................................... 44
mixAR_diag ................................................... 47
mixAR_switch ................................................ 48
MixComp-class ............................................... 49
mixFilter ...................................................... 52
mixgenMstep .................................................. 53
mixMstep ...................................................... 54
mixSARfit ...................................................... 55
MixVAR-class .......................................................... 56
mixVARfit .......................................................... 57
MixVARGaussian-class ............................................. 58
mixVAR_sim .......................................................... 59
mix_ek ................................................................. 60
mix_hatk ............................................................... 62
mix_moment .......................................................... 62
mix_ncomp-methods .................................................. 65
mix_pdf-methods ...................................................... 65
mix_qf-methods ...................................................... 66
mix_se-methods ...................................................... 67
multiStep_dist-methods ............................................. 69
noise_dist ........................................................... 72
noise_dist-methods ................................................... 73
noise_params-methods .............................................. 73
noise_rand-methods ................................................ 74
parameters .......................................................... 74
percent_of ............................................................ 75
permn_cols ............................................................ 77
PortfolioData1 ......................................................... 78
predict_coef ........................................................ 79
ragged ................................................................. 80
raggedCoef-class ...................................................... 81
raggedCoefS-class .................................................... 84
raggedCoefV-class .................................................... 86
raghat1 ................................................................. 87
randomArCoefficients .............................................. 88
row_lengths-methods ............................................... 90
sampZpi ............................................................... 90
show_diff ............................................................. 92
simuExperiment ...................................................... 93
stdnormmoment ...................................................... 95
tomarparambyComp ................................................ 97

Index 99

mixAR-package  Mixture Autoregressive Models

Description

Details

Package mixAR provides functions for modelling with mixture autoregressive (MixAR) models. The S4 class "MixARGaussian" can be used when the error distributions of the components are standard Gaussian. The class "MixARgen" admits arbitrary (well, within reason) distributions for the error components. Both classes inherit from the virtual class "MixAR".

Estimation can be done with \texttt{fit_mixAR}. Currently, the EM algorithm is used for estimation.

For "MixARGaussian" the M-step of the EM algorithm reduces to a system of linear equations. For "MixARgen" the problem is substantially non-linear. The implementation is fairly general but currently not optimised for efficiency. The specification of the error distributions went through several stages and may still be reviewed. However, backward compatibility will be kept.

Author(s)

Georgi N. Boshnakov [aut, cre], Davide Ravagli [aut]

Maintainer: Georgi N. Boshnakov <georgi.boshnakov@manchester.ac.uk>

References


mixAR-package

See Also

fit several types of mixAR models: fit_mixAR, bayes_mixAR, fit_mixARreg, mixSARfit;
Predictive distributions and summaries: mix_pdf, mix_cdf, mix_qf, mix_location, mix_variance, mix_central_moment, mix_moment, mix_kurtosis, mix_ekurtosis

multi-step prediction: multiStep_dist

Examples

```r
## object 'exampleModels' contains a number of models for examples and testing
names(exampleModels)
exampleModels$WL.ibm

## some of the models below are available in object 'exampleModels';
## the examples here show how to create them from scratch
mo_WLprob <- c(0.5439, 0.4176, 0.0385)  # model coefficients from Wong&Li
mo_WLsigma <- c(4.8227, 6.0082, 18.1716)
mo_WLar <- list(c(0.6792, 0.3208), c(1.6711, -0.6711), 1)

mo_WL <- new("MixARGaussian", prob = mo_WLprob, scale = mo_WLsigma, arcoef = mo_WLar)

mo_WL_A <- new("MixARGaussian" # WongLi, model A
  , prob = c(0.5, 0.5)
  , scale = c(5, 1)
  , shift = c(0, 0)
  , arcoef = list(c(0.5), c(1.1))
)

mo_WL_B <- new("MixARGaussian" # WongLi, model B
  , prob = c(0.75, 0.25)
  , scale = c(5, 1)
  , shift = c(0, 0)
  , arcoef = list(c(0.5), c(1.4))
)

mo_WL_I <- new("MixARGaussian" # WongLi, model I
  , prob = c(0.4, 0.3, 0.3)
  , scale = c(1, 1, 5)
  , shift = c(0, 0, -5)
  , arcoef = list(c(0.9, -0.6), c(-0.5), c(1.50, -0.74, 0.12))
)

mo_WL_II <- new("MixARGaussian" # WongLi, model II
  , prob = c(0.4, 0.3, 0.3)
  , scale = c(1, 1, 5)
  , shift = c(5, 0, -5)
  , arcoef = list(c(0.9, -0.6), c(-0.7, 0), c(0, 0.80))
)

## MixAR models with arbitrary dist. of the components
## (user interface not finalized)
```
## Gaussian

```r
mo_WLgen <- new("MixARgen", prob = mo_WLprob, scale = mo_WLsigma, arcoef = mo_WLar,
                 dist = list(dist_norm))
```

## \( t_3 \)

```r
mo_WLt3v <- new("MixARgen", prob = mo_WLprob, scale = mo_WLsigma, arcoef = mo_WLar,
                 dist = list(fdist_std(3, fixed = FALSE)))
```

## \( t_20, t_30, t_40 \) (can be used to start estimation)

```r
mo_WLtf <- new("MixARgen", prob = mo_WLprob, scale = mo_WLsigma, arcoef = mo_WLar,
                 dist = list(generator =
                     function(par)
                       fn_std(par, fixed = FALSE), param = c(20, 30, 40)))
```

## data(ibmclose, package = "fma") # for \`ibmclose\`

## The examples below are quick but some of them are marked as 'not run'
## to avoid cumulative time of more than 5s on CRAN.

## fit a MAR(2,2,1) model

```r
a0a <- fit_mixAR(as.numeric(fma::ibmclose), c(2, 2, 1), crit = 1e-4)
```

## same with 2 sets of automatically generated initial values.

```r
a0b <- fit_mixAR(as.numeric(fma::ibmclose), c(2, 2, 1), 2, crit = 1e-4)
```

## fix the shift parameters:

```r
a1a <- fit_mixAR(as.numeric(fma::ibmclose), c(2, 2, 1), fix = "shift", crit = 1e-4)
```

## ... with 3 sets of automatically generated initial values.

```r
a1b <- fit_mixAR(as.numeric(fma::ibmclose), c(2, 2, 1), 3, fix = "shift", crit = 1e-4)
```

## specify the model using a MixAR model object

```r
a1c <- fit_mixAR(as.numeric(fma::ibmclose), a1a$model, init = a0a$model, fix = "shift",
                 crit = 1e-4)
```

## fit a model like \( mo_{-}WL \) using as initial values 2 automatically generated sets.

```r
a2 <- fit_mixAR(as.numeric(fma::ibmclose), mo_WL, 2, fix = "shift", permute = TRUE,
                 crit = 1e-4)
```

```r
moT_B3 <- new("MixARgen"
             , prob = c(0.3, 0.3, 0.4)
             , scale = c(2, 1, 0.5)
             , shift = c(5, -5, 0)
             , arcoef = list(c(0.5, 0.24), c(-0.9), c(1.5, -0.74, 0.12))
             , t4, t4, t10
             , dist = distlist("stdt", c(4,10), fixed = c(FALSE, TRUE), tr = c(1, 1, 2))
             )
```

```r
moT_C1 <- new("MixARgen"
```
bayes_mixAR

Bayesian sampling of mixture autoregressive models

Description

Samples parameters of a mixture autoregressive model from respective posterior distributions.

Usage

bayes_mixAR(y, model, fix_shift = FALSE, a = .2, c = 2, tau, nsim, burnin)

Arguments

y a time series (currently a numeric vector).
model an object of class MixAR. Currently only handles MixARGaussian objects.
fix_shift should shift be kept fixed? If FALSE (default) shift is sampled.
a, c numeric hyperparameters, default values are from Richardson and Green (1997).
tau numeric vector of length g, the number of components in the mixture. Tuning parameter for M-H move in updating AR parameters. If length(tau) is 1, same tuning parameter is taken for all components.
nsim numeric, the number of iterations.
burnin numeric, the number of iterations taken as burn-in period.

Details

For details see Ravagli and Boshnakov (2020).
Value

a list with following elements:

- **mix_weights**: a g columns matrix with samples from the posterior distributions of the mixing weights.
- **scale**: a g columns matrix with samples from posterior distributions of scale parameters.
- **precision**: a g columns matrix with samples from posterior distributions of precision parameters, defined as $1 / (\text{scale}^2)$.
- **shift**: a g columns matrix with samples from posterior distributions of shift parameters, namely $\phi_k0$.
- **mu**: a g columns matrix with samples from posterior distributions of component means, calculated as $\phi_k0 / (1 - \phi_k1 - \phi_k2 - \ldots)$.
- **ARcoeff**: a list which elements are matrices, one for each AR component in the mixture.
- **acc_rate**: numeric vector, the acceptance rate for M-H moves.
- **n_samp**: the sample size, calculated as $\text{nsim} - \text{burnin}$.
- **LatentZ**: the latest Z variables drawn (for utility only).
- **n_comp**: the number of components in the mixture.
- **fix_shift**: same as input, whether the shift parameter was kept fixed or not.

Author(s)

Davide Ravagli

References


Examples

```r
prob <- c(0.5, 0.5)
sigma <- c(1, 2)
ar <- list(-0.5, 1)

model <- new("MixARGaussian", prob = prob, scale = sigma, arcoef = ar)

## MAR(1,1) model
y <- mixAR_sim(model, 300, rep(0, max(model@order)))

bayes_mixAR(y, model, fix_shift = FALSE, tau = c(.15,.25), nsim = 20, burnin = 10)
```
Choose_pk

Choose the autoregressive order of MixAR components

Description

Reversible Jump MCMC algorithm to choose the optimal autoregressive order of each component of a mixture autoregressive model.

Usage

Choose_pk(y, model, fix_shift = FALSE, tau, pmax, method, par = NULL, nsim)

Arguments

- **y**: a time series. Currently a numeric vector.
- **model**: an object inheriting from class "MixAR".
- **fix_shift**: whether the shift/mean parameter should be kept fixed to its starting value or not. Default is FALSE.
- **tau**: tuning parameters for Metropolis-Hastings algorithm in sampling AR coefficients.
- **pmax**: the largest autoregressive order allowed for each component.
- **method**: character vector of length 1. Method for calculating probability of new AR order to be increased/decreased by 1 unit. Currently available "Ratio", "Poisson" and "NULL". Default is "NULL".
- **par**: numeric, parameter for tuning probabilities according to method. Ignored if method is "NULL".
- **nsim**: numeric, the number of iterations.

Value

- **out**: a dataframe with g+1 columns. The first g columns contain the autoregressive orders of the components, the last column how often a model is preferred, divided by nsim.
- **fix_shift**: the choice made for the shift/mean parameters.
- **method**: the method used to increase/decrease AR orders.

Note

Choose_pk currently supports class "MixARGaussian" only.

Author(s)

Davide Ravagli
References


See Also

bx_dx for more details on the method

Examples

```r
model <- new("MixARGaussian",
    prob = exampleModels$WL_At@prob, # c(0.5, 0.5)
    scale = exampleModels$WL_At@scale, # c(1, 2)
    arcoef = list(-0.5, 1) )
# note: arcoef != list(-0.5, 1) == exampleModels$WL_At@arcoef@a

set.seed(1234)
n <- 50 # 200
y <- mixAR_sim(model, n, rep(0, max(model@order)), nskip = 100)

nsim <- 25 # 100
pk_star <- Choose_pk(y, model, tau = c(.15, .25), pmax = 5, method = "NULL", nsim = nsim)
```

cond_loglik

Log-likelihood of MixAR models

Description

Compute the log-likelihood of a MixAR model for a univariate time series.

Usage

```r
cond_loglik(model, x, index)
cond_loglikS(model, x, index)
```

Arguments

- `model`: a MixAR model.
- `x`: a time series or numeric vector.
- `index`: a vector of integers giving the indices in `x` over which to compute the sum for the log-likelihood, default is `(p+1):length(x)`, where `p` is the maximum AR order of the components of the model.
Details

cond_loglik computes the conditional log-likelihood of a MixAR model. Conditional here means conditional on the first \( p \) values being fixed, where \( p \) is the maximum AR order of the components of the model.

Argument \texttt{index} can be used to compute the sum over a subset of time points.

\texttt{cond_loglikS} is a variant of \texttt{cond_loglik} for the case when the input model contains seasonal AR coefficients.

Value

the log-likelihood, a numeric value

Author(s)

Georgi N. Boshnakov and Davide Ravagli

Examples

```r
## data(ibmclose, package = "fma") # doesn't work with fma v2.4, using `::`
cond_loglik(exampleModels$WL_ibm, as.numeric(fma::ibmclose))
cond_loglik(exampleModels$WL_ibm_gen, as.numeric(fma::ibmclose))

data(lynx) # for `lynx`
sar <- new("raggedCoefS", a = list(c(1.1022, -0.2835), c(1.5279, -0.8871)),
as = list(c(0, 0), 0), s = 10)

## SMAR(2; 2, 2)(2, 1)_{10}
model_s10 <- new("MixARGaussian", prob = c(.3, .7), scale = c(.08, .202),
arcoeff = sar, shift = c(.7, 1))

cond_loglikS(model_s10, log(lynx))
cond_loglikS(model_s10, log(lynx), index = 45:114) # on reduced dataset
```

---

\textit{dist_norm} Functions for the standard normal distribution

Description

The noise distributions are specified by a list of functions for the density, quantiles, etc. This object demonstrates this for the standard normal distribution.

Usage

dist_norm

Format

This is a list of functions or names of functions for calculations related to the standard normal distribution. Currently it has elements with the following names: "pdf", "cdf", "rand", "logpdf", "Fscore", "xFscore", "Parscore", "get_param", "set_param", "any_param", "show".
Details

dist_norm may be used to specify the noise distribution for MixAR models. It can be used as a template if other distributions are needed, see also fdist_stdnorm.

See Also

fdist_stdnorm

Examples

dist_norm
dist_norm$pdf
dist_norm$cdf

em_est_dist Optimise scale parameters in MixARgen models

Description

Optimise the scale parameters in MixAR models from class MixARgen. Internal function.

Usage

em_est_dist(tau, etk, parscore, sigma, nu, logpdf)

Arguments

tau conditional probabilities, an object of class "MixComp", see 'Details'.
etk component residuals, see 'Details'.
parscore the score function(s), see 'Details'.
sigma current values of the scale parameters, a numeric vector.
nu current values of the parameters. w.r.t. which optimisation is done.
logpdf the log of pdf as a function of the parameters.

Details

One or more of the error distributions of a MixAR model may have parameters that are considered unknown. In that case em_est_dist can be used to optimise with respect to them.

The representation of the error distributions in "MixARgen" models carries all the necessary information about parameters. em_est_dist works by extracting their current values from logpdf, passes them to the optimisation function (or equation solver) and stores the result back into logpdf. em_est_dist is quite general, as long as logpdf is prepared according to the conventions it expects (this is so if they are valid elements of the dist slot of "MixARgen" objects).

Value

the new values of the parameters
em_est_sigma

**Update the scale parameters of MixAR models**

**Description**

Calculates estimates of scale parameters of MixAR models from conditional probabilities and mixture 'residuals'. Used in EM algorithm.

**Usage**

```r
tauetk2sigmahat(tau, etk)
em_est_sigma(tau, etk, Fscore, sigma,
  dontfix = rep(TRUE, length(sigma)), compwise = FALSE)
```

**Arguments**

- **tau**: the conditional probabilities for the groups, a "MixComp" object.
- **etk**: component "residuals", MixComp object(?)
- **Fscore**: the score function(s) of the noise distributions.
- **sigma**: current values of the scale parameters.
- **compwise**: if TRUE solve the equations component-wise, see 'Details.
- **dontfix**: a logical vector containing TRUE in the positions of elements of sigma that are to be estimated.

**Details**

- `tauetk2sigmahat` calculates estimates of the scale parameters for a MixAR time series with Gaussian components. There is an explicit formula in that case.
- `em_est_sigma` calculates estimates of the scale parameters in the general case. The non-linear equations are solved using functions from package BB. The equations for the components can often be solved independently. When that is the case, compwise may speed things up a little.

**Value**

The new values of the scale parameters, a numeric vector
Description

Gaussian EM-step with random initialisation.

Usage

\texttt{em\_rinit}(y, order, partempl)
\texttt{etk2tau}(etk)

Arguments

\begin{itemize}
\item \texttt{y}: time series.
\item \texttt{order}: \text{MixAR} order, vector of length the number of components.
\item \texttt{partempl}: parameter template, a list containing one element for each mixture component, see \texttt{randomArCoefficients}.
\item \texttt{etk}: \text{MixAR} component residuals, a matrix.
\end{itemize}

Details

\texttt{em\_rinit} generates random MAR residuals, performs a non-distributional E-step, and a Gaussian M-step.

\texttt{etk2tau} estimates \texttt{tau} from component residuals only. Note that this is unlike \texttt{em\_tau}, which also needs the noise pdf’s, as well as estimates of the mixture probabilities.

\texttt{em\_rinit} uses \texttt{etk2tau} to start the EM algorithm.

Value

\begin{itemize}
\item for \texttt{em\_rinit}, an object from class \texttt{"MixARGaussian"}
\item for \texttt{etk2tau}, a matrix representing \texttt{tau} (i-th row contains probabilities corresponding to the i-th observation)
\end{itemize}

Author(s)

Georgi N. Boshnakov
Create estimation templates from MixAR model objects.

Usage

```r
est_templ(model, shift = TRUE, ...)
```

Arguments

- `model`: a "MixAR" object.
- `shift`: logical, see Details.
- `...`: currently not used.

Details

Argument `model` is used as a template to specify values of parameters and/or which parameters to estimate or fix. In general, if a value of a parameter in `model` is `NA`, then it is to be estimated. Otherwise the parameter is taken as is.

The current implementation is incomplete. In particular, the AR parameters are always designated for estimation.

Argument `shift` can be used to overwrite some or values component `shift` in `model`. If `shift` has length one, it is replicated to the number of MixAR components. If `shift[k]` is `TRUE`, then the shift coefficient for the k-th component is set to `NA` to request its estimation. Otherwise, the value of the shift for the k-th component in `model` is taken.

Argument `shift` has a default of `TRUE` which causes the shift coefficients to be estimated irrespectively of their values in `model`.

`est_templ` returns a list with as many components as there are MixAR components in the model. The k-th component of the list is itself a list specifying which parameters of the i-th MixAR component to estimate or fix.

Value

a list, as described in Details.

Examples

```r
eexampleModels$WL_A
est_templ(exampleModels$WL_A)
est_templ(exampleModels$WL_A, shift = FALSE)

eexampleModels$WL_I
est_templ(exampleModels$WL_I)
```
MixAR models for examples and testing.

Usage

eexampleModels

Details

Coefficients of models from the examples in Wong and Li (2000). Variations on these with different noise distributions are used throughout the examples in mixAR. The models are from classes inheriting from class "MixAR".

eexampleModels is a list with the following components:

- WL_ibm
- WL_A
- WL_B
- WL_I
- WL_II
- WL_ibm_gen
- WL_ibm_t3v
- WL_ibm_tf
- WL_At
- WL_Bt_1
- WL_Bt_2
- WL_Bt_3
- WL.Ct_1
- WL.Ct_2
- WL.Ct_3

Each component is a MixAR model, i.e. an object inheriting from class "MixAR".

Source


Examples

```r
## use these instead of moWL, moWL_A, moWL_B, etc.
exampleModels$WL_ibm

exampleModels$WL_A
```
fit_mixAR-methods

Estimate a MixAR model for a time series. This is a generic function. The methods defined in package **mixAR** are described here.

**Usage**

```r
fit_mixAR(x, model, init, fix, ...)
```

---

**Description**

Estimate a MixAR model for a time series. This is a generic function. The methods defined in package **mixAR** are described here.
Arguments

- **x**: a time series.
- **model**: model, object inheriting from MixAR class.
- **init**: what initializations to do, see Details.
- **fix**: which parameters to fix, see Details.
- **...**: additional arguments for the methods.

Details

Method dispatch is done on the first three arguments: x, model and init.

- **model**: specifies the model to fit. If model inherits from "MixAR", it is used as a template. If init is missing, the parameters of model are also used as initial values. Model can also be a numeric vector specifying the order of a MixAR model with Gaussian components.
- **init**: can be used to give initial values in variety of ways. If it is a MixAR object it doesn’t need to be of the same class as model, to allow using as initial values common parameters of different MixAR models. A positive integer value of init asks to run the fitting procedure init times, each time generating random initial values.
- **init** can also be a list. In that case, each component of the list should itself be an acceptable value for init and the fitting procedure is run with each component of init.
- **fix**: can be given in a number of ways. Note however that currently there is no method dispatch on it.
- Currently the default method for fit_mixAR just throws error, since there seems no suitable default task to do.
- See individual methods for further details.

Value

- a MixAR model or a list of MixAR models, depending on the arguments.

Methods

- signature(x = "ANY", model = "ANY", init = "ANY") The default method throws error.
- signature(x = "ANY", model = "MixAR", init = "missing") This is equivalent to setting init = model.
- signature(x = "ANY", model = "MixAR", init = "MixAR") model is a template for the result, init specifies initial values for the parameters. In principle, model and init may be from different classes, to allow for example using AR coefficients from a Gaussian fit for other distributions.
- signature(x = "ANY", model = "MixAR", init = "numeric") init must be a single positive integer here. The model is fitted init times, each time starting with a new set of randomly generated initial values. If select is TRUE, the default, the model with the largest likelihood is returned, otherwise a list containing the init fitted models is returned.
- signature(x = "ANY", model = "MixAR", init = "list") Each element of the list init should be an acceptable value for init. The model is fitted with the initial value set to each element of init. A list containing the fitted models is returned.
signature(x = "ANY", model = "MixARGaussian", init = "MixAR")
signature(x = "ANY", model = "numeric", init = "missing") This is equivalent to setting init = 1.
signature(x = "ANY", model = "numeric", init = "numeric") A numeric model should be a vector of non-negative integers specifying the order of the MixAR model. The distribution of the components is assumed Gaussian.

Examples

```r
## model coefficients from Wong&Li (IBM fit)
prob <- exampleModels$WL_ibm@prob # c(0.5439, 0.4176, 0.0385)
sigma <- exampleModels$WL_ibm@scale # c(4.8227, 6.0082, 18.1716)
ar <- exampleModels$WL_ibm$arcoef@a # list(c(0.6792, 0.3208), c(1.6711, -0.6711), 1)

## data(ibmclose, package = "fma") # 'ibmclose'

mot30 <- new("MixARgen", prob = prob, scale = sigma, arcoef = ar, 
       dist = distlist("stdt", c(30, 30, 30)))

mot20_30_40 <- new("MixARgen", prob = prob, scale = sigma, arcoef = ar, 
       dist = distlist("stdt", c(20, 30, 40)))

mo_t20_t30_norm <- new("MixARgen", prob = prob, scale = sigma, arcoef = ar, 
       dist = distlist(c("stdt", "stdt", "stdnorm"), c(20, 30)))

## Gaussian components
fi0 <- fit_mixAR(fma::ibmclose, exampleModels$WL_ibm, fix = "shift", crit = 1e-4)
fi0$model

if(FALSE){ # don't run on CRAN to save a couple of seconds
  ## remove minniter/maxniter below for realistic results.

  ## std-t components
  fi1 <- fit_mixAR(fma::ibmclose, mot30, fix = "shift", 
                  crit = 1e-4, minniter = 1, maxniter = 3)

  fi1$model

  ## 1st and 2nd components std-t, 3rd Gaussian
  fi2 <- fit_mixAR(fma::ibmclose, mo_t20_t30_norm, fix = "shift", 
                  crit = 1e-4, minniter = 1, maxniter = 3)

  fi2$model
}
```

fit_mixARReg-methods

Fit time series regression models with mixture autoregressive residuals

Description

Estimate a linear regression model for time series with residuals from a mixture autoregressive process.
Usage

fit_mixARreg(x, y, mixARmodel, EMinit, ...)

mixARreg(x, y, mixARmodel, tol = 1e-6, niter = 200)

Arguments

x
the response time series (currently a numeric vector).

y
data.frame, matrix or numeric vector. If either of the first two, each column
must contain one covariate (currently numeric). A check for matching lengths
between x and y is done.

mixARmodel
An object inheriting from class "MixAR", giving initial values for EM-estimation
of mixture autoregressive parameters. Currently only "MixARGaussian" is sup-
ported.

EMinit
starting values for EM estimation of MixAR parameters. If present, must be a
named list, containing at least prob and scale as numeric vectors, and a list for
arcoef.

tol
threshold for convergence criterion.

...
passed on to MixARreg.

niter
maximal number of iterations.

Details

fit_mixARreg is a generic function. Currently there is no default method for fit_mixARreg. Ar-
guments y, mixARmodel, EMinit can be given in a number of ways, see individual methods for
details.

Argument mixARmodel gives the details of the the MixAR part of the model and initial values for
the parameters. For fit_mixARreg this can alternatively be done with a list using argument EMinit.
Currently, at least one of the two must be supplied, and if both are present EMinit is ignored.

mixARreg performs a two-step estimation of a linear regression model with mixture autoregressive
residuals. It is the workhorse for fit_mixARreg which calls it to do the computations.

Value

reg
The summary output of the regression part of the model.

mixARmodel
Estimates of the mixture autoregressive part of the model.

niter
The number of iterations until convergence.

Methods

signature(x = "ANY", y = "data.frame", mixARmodel = "MixAR", EMinit = "missing") Covariates
y are supplied as data.frame: each column corresponds to one covariate. Initialization of
MixAR paramters is done using input mixARmodel

signature(x = "ANY", y = "matrix", mixARmodel = "MixAR", EMinit = "missing") Covariates
y are supplied as matrix: each column corresponds to one covariate. Initialization of MixAR
paramters is done using input mixARmodel
signature(x = "ANY", y = "numeric", mixARmodel = "MixAR", EMinit = "missing") Covariates y is supplied as numeric: this method handles the simple regression case with a single covariate. Initialization of mixAR parameters is done using input mixARmodel

signature(x = "ANY", y = "ANY", mixARmodel = "missing", EMinit = "list") EMinit must be a named list (see 'Arguments').

signature(x = "ANY", y = "ANY", mixARmodel = "MixAR", EMinit = "list") When both mixARmodel and EMinit are supplied, the second is ignored.

**Note**
Estimation is done using the function mixARreg within each method.

**Author(s)**
Davide Ravagli and Georgi N. Boshnakov

**See Also**

fit_mixAR

**Examples**

```r
## Simulate covariates
set.seed(1234)
n <- 50 # for CRAN
y <- data.frame(rnorm(n, 7, 1), rt(n, 3), rnorm(n, 3, 2))

## Build mixAR part
model <- new("MixARGaussian",
  prob = exampleModels$WL_At@prob, # c(0.5, 0.5)
  scale = exampleModels$WL_At@scale, # c(1, 2)
  arcoef = exampleModels$WL_At@arcoef @a ) # list(-0.5, 1.1)

## Simulate from MixAR part
u <- mixAR_sim(model, n, 0)
x <- 10 + y[, 1] + 3 * y[, 2] + 2 * y[, 3] + u

## Fit model

## Using MixARGaussian
fit_mixARreg(x = x, y = y, mixARmodel = model, niter = 3)

## Using EMinit
EMinit <- list(prob = exampleModels$WL_At@prob, scale = exampleModels$WL_At@scale,
  arcoef = exampleModels$WL_At@arcoef @a)
fit_mixARreg(x = x, y = y, EMinit = EMinit, niter = 3)
```

Description

Estimate a MixVAR model for a multivariate time series. This is a generic function. The methods defined in package **MixAR** are described here.

Usage

```
fit_mixVAR(x, model, fix, ...)
```

Arguments

- `x`: a multivariate time series (currently a numeric matrix).
- `model`: model, object inheriting from MixVAR class.
- `fix`: if TRUE, fix the shift parameters.
- `...`: additional arguments for the methods (not currently used).

Details

`model` specifies the model to fit. If `model` inherits from "MixVAR", it is used as a template. Estimation is done via EM-Algorithm, using the function `mixVARfit`.

Currently the default method for `fit_mixAR` just throws error, since there seems no suitable default task to do.

Value

a MixVAR model.

Methods

```
signature(x = "ANY", model = "MixVAR")
signature(x = "ANY", model = "ANY")
```

See Also

`mixVARfit`

Examples

```
AR <- list()
AR[[1]] <- array(c(0.5, -0.3, -0.6, 0, 0, 0.5, 0.4, 0.5, -0.3), dim = c(3, 3, 1))
AR[[2]] <- array(c(-0.5, 0.3, 0, 1, 0, -0.5, -0.4, -0.2, 0.5), dim = c(3, 3, 1))

prob <- c(0.75, 0.25)
shift <- cbind(c(0, 0, 0), c(0, 0, 0))
```
Sigma1 <- cbind(c(1, 0.5, -0.4), c(0.5, 2, 0.8), c(-0.4, 0.8, 4))
Sigma2 <- cbind(c(1, 0.2, 0), c(0.2, 2, -0.15), c(0, -0.15, 4))
Sigma <- array(c(Sigma1, Sigma2), dim = c(3, 3, 2))

m <- new("MixVARGaussian", prob = prob, vcov = Sigma, arcoef = AR, shift = shift)
set.seed(1234)
y <- mixVAR_sim(m, n = 100, init = matrix(0, ncol = 3), nskip = 50, flag = FALSE)
fit_mixVAR(y, m, tol = 1e-3)
mixVARfit(y, m, tol = 1e-3)

---

fnoise

Generator functions for noise distributions

Description

These functions and objects are mostly internal and should not be needed for routine use. Generate noise distribution, currently standard normal and standardised t-distributions. These functions can be used as templates for new distributions.

Usage

fdist_stdnorm()
fdist_stdt(df, fixed = TRUE)
fn_stdt(df, fixed = TRUE)
b_show(x)
distlist(type, param, ncomp = NULL, fixed = FALSE, tr = NULL, ...)
ed_nparam
ed_parse(s)
ed_skeleton(df, fixed = FALSE, n = length(df), tr = NULL)
ed_src
ed_stdnorm
ed_stdt
ed_stdt0
ed_stdt1

ft_stdt

Arguments

df degrees of freedom
fixed if TRUE, the parameters are fixed, otherwise they are variable, see Details.
x a fitted object.
type list of distributions.
param parameters.
ncomp number of components.
tr transformation.
... not used.
s named vector.
n number of different degrees of freedom.

Details

If argument fixed is TRUE, estimation functions assume that the parameter(s) are fixed, otherwise they estimate it. The support is incomplete, see below.

fdist_stdnorm is for the standard normal distribution. For example dist_norm is generated by it.

fdist_stdt is for the t-distribution with df degrees of freedom.

fn_stdt is also for the t-distribution but the degrees of freedom, df, may be a vector. The value is a list of distributions. Although the list can be obtained by repeated calls of fdist_stdt

The support is incomplete. In particular, if parameter fixed is TRUE, changes to the parameter(s) should probably not be allowed (this can be achieved by simply dropping the corresponding function from the list). However, a thorough rethinking is necessary, as I introduced it on the fly while developing estimation functions and forbidding changes may necessitate changes in the code. Changes are useful for estimation for convenience but also to avoid recreating the whole distributions again and again.

However, there is a major drawback, which in the final version needs to be addressed satisfactorily. Since parameters are held in local environments, changes to the parameters are reflected in copies of the objects. For example, an estimation function (or the user) may call another function with a model containing an object generated by the above functions and assign the result to a new object. However if the parameters of the noise distribution are changed in the process this will be reflected in the original model.

Note that the above effect is valid only if an object generated by the above functions is reused. Objects created by different calls have different environments, so the problem does not arise for them.
Examples

```r
stdt3 <- fdist_stdt(3)
stdt3v <- fdist_stdt(3, fixed = FALSE)
fn_stdt(c(20, 30, 40), fixed = FALSE)

mo_tf <- new("MixARgen", prob = exampleModels$WL_ibm@prob,
              scale = exampleModels$WL_ibm@scale, arcoef = exampleModels$WL_ibm@arcoef@a,
              dist = list(generator = function(par)
              fn_stdt(par, fixed = FALSE), param = c(20, 30, 40)))

mo_tf
str(mo_tf)
noise_dist(mo_tf, "pdf")
parameters(mo_tf)
parameters(mo_tf, names = TRUE)
get_edist(mo_tf)
oise_params(mo_tf)
```

Description

Methods for function get_edist in package mixAR

Methods

get_edist gives the error (or noise) distribution of MixAR objects.
Currently the distribution is returned as a list of functions. The list contains one element for each
component. If the error distributions of all components are the same, then the list may contain a
single element representing the common error distribution.

Note that the distribution is not necessarily stored in slot dist in this format, see the description
of this slot in class "MixARgen". Such a slot may even not exist if the distribution of the error
components is fixed as is the case for class MixARGaussian.

Each subclass of MixAR needs to define a method for get_edist.

signature(model = "MixAR") Issue an error message and stop.
This method is invoked for subclasses of MixAR that have not defined their own method for
get_edist. This is an error.

signature(model = "MixARGaussian") Return an object representing the fact that the error dis-
tributions of the components of MixARGaussian objects are standard normal.

signature(model = "MixARgen") Return an object representing the error distributions of the com-
ponents of MixARgen objects. The distributions are not necessarily the same for such objects.
Description

Generalised inner product and methods for class MixComp. The methods for MixComp provide for very convenient computing with MixAR models.

Usage

inner(x, y, star = "*", plus = .mplus)

Arguments

x the first argument.
y the second argument.
star function to apply to pairs of elements from x and y, default is multiplication, as for the usual inner product.
plus function to apply to combine the results from the pairs, default is addition, as for the usual inner product.

Details

inner computes a generalised inner product $x . y$, where multiplication and summation can be replaced by other functions.

The default method of inner applies star to the corresponding pairs of elements and combines them with plus. There is no recycling, if x and y have different lengths, an error is raised. The elements of x and y are accessed with "[". plus should be an n-ary operation.

Value

the inner product, the type of the result depends on the arguments

Methods

Methods for inner product between a "MixComp" object and a vector are similar to a product between a matrix and a vector but comply with the conventions of class "MixComp". For this reason they are described in the help page for class "MixComp", along with methods for other functions and operators applied to "MixComp" objects.

signature(x = "ANY", y = "ANY", star = "ANY", plus = "ANY") This is the default method, see section Details.
signature(x = "MixComp", y = "missing", star = "missing", plus = "missing") see "MixComp".
signature(x = "MixComp", y = "numeric", star = "missing", plus = "missing") see "MixComp".
signature(x = "numeric", y = "MixComp", star = "missing", plus = "missing") see "MixComp".
signature(x = "MixComp", y = "numeric", star = "ANY", plus = "ANY") see "MixComp".
signature(x = "MixComp", y = "numeric", star = "ANY", plus = "missing") see "MixComp".
isStable

See Also

"MixComp"

Examples

inner(1:3, 2:4) # [1] 20
class(inner(1:3, 2:4)) # [1] "integer"
# compare to:
1:3 %*% 2:4 # 20, but (1,1)-matrix
class(1:3 %*% 2:4) # matrix

isStable Check if a MixAR model is stable

Description

Checks if a MixAR model is stable. This is also the second order stationarity condition.

Usage

isStable(x)

Arguments

x the model

Details

If each component of a MixAR model corresponds to a stable autoregression model, then the MixAR model is also stable. However, the MixAR model may be stable also when some of its components correspond to integrated or explosive AR models, see the references.

Value

True if the model is stable (second order stationary), FALSE otherwise.

References


Examples

isStable(exampleModels$WL_I)
isStable(exampleModels$WL_II)
Description

Takes the output from a MCMC simulation of parameters of a mixture, and detects whether labels switch has occurred while sampling, using the method by Celeux (2000).

Usage

\[ \text{label_switch}(x, m) \]

Arguments

- \( x \): output from an MCMC sampling of a mixture. A matrix, each column corresponds to one component of the mixture.
- \( m \): the number of observations in the sample that will be used to initialise the algorithm. \( m \sim 100 \) is recommended.

Details

Function can be directly executed when \( x \) is one of \texttt{mix_weights}, \texttt{scale}, \texttt{precision}, \texttt{shift} or \texttt{mu} from \texttt{bayes_mixAR} output. \texttt{ARcoeff} cannot be input as is, but element from the list may be used.

Value

A list of 2:

- \( x \): The input matrix, with adjusted labels
- \( \text{true_perm} \): The "true" permutation at each iteration.

Note

There is no absolute choice on what \( x \) should be to obtain the "true" permutation at any given point. User is subject to make the most suitable choice, given output of their MCMC.

Author(s)

Davide Ravagli

References


See Also

\texttt{bayes_mixAR}
Examples

```r
model <- new("MixARGaussian",
    prob = exampleModels$WL_At@prob, # c(0.5, 0.5)
    scale = exampleModels$WL_At@scale, # c(1, 2)
    arcoef = exampleModels$WL_At@arcoef@a ) # list(-0.5, 1.1)

y <- mixAR_sim(model, n = 300, init = rep(0, which.max(model@order)))

## just examples, use larger numbers in practice
nsim <- 30 # 200
burnin <- 10 # 100
x <- bayes_mixAR(y, model, fix_shift = FALSE, tau = c(.15, .25),
    nsim = nsim, burnin = burnin)

label_switch(x$mix_weights, m = 5)
```

---

### lik_params

**Vector of parameters of a MixAR model**

**Description**

Give a numeric vector containing non-redundant parameters of a MixAR model in a form suitable for use by optimisation routines. The methods defined in package `mixAR` for this generic function are described here.

**Usage**

```r
lik_params(model)
```

**Arguments**

- `model` a MixAR model.

**Details**

`lik_params` gives the parameters of a MixAR model as a numeric vector. This is a generic function. Parameters common to all MixAR models are arranged as described below. There are no other parameters when the error distributions do not contain parameters of their own. Methods for sub-classes with additional parameters should append them after the common parameters.

If \( k \) is the number of components and \( \pi_i \) is the probability associated with the \( i \)th component, then the parameters are put in a vector as follows:

1. component probabilities, \( \pi_1, \ldots, \pi_{k-1} \) (note: \( \pi_k \) is not included)
2. scales, \( \sigma_1, \ldots, \sigma_k \)
3. shifts, \( \mu_1, \ldots, \mu_k \)
4. AR coefficients of the 1st component,
5. AR coefficients of the 2nd component,
6. ...
7. AR coefficients of the $k$th component.

Value
A numeric vector containing all parameters except the probability associated with the last component.

Methods
signature(model = "MixAR")
signature(model = "MixARgen")

Note
The probability associated with the $k$th component is omitted as it is redundant. This makes it possible to try unconstrained optimisation though it is not likely to give useful results since there are other restrictions on the probabilities.

Author(s)
Georgi N. Boshnakov

Description
Create a function for the computation of the conditional likelihood of MixAR models for a given time series. The methods for this generic function defined in package mixAR are described here.

Usage
make_fcond_lik(model, ts)

Arguments
model a MixAR model
ts the time series
Details

The returned value is a function, say \( f(x) \), whose only argument is a numeric vector of parameters with the arrangement of \( \text{lik_params} \), for which it computes the conditional loglikelihood. \( f \) can be given to optimisation routines.

Argument \( \text{model} \) is an object inheriting from \( \text{MixAR} \) and determines the structure of the \( \text{MixAR} \) model for the function, \( f \), that it creates. So, properties of the model, such as number of components, AR order, and distribution of the noise components are fixed when \( f \) is created and only the numeric values of the parameters are changed by calls to it.

Value

a function of one argument, the parameters of a \( \text{MixAR} \) model as a numeric vector with the arrangement of \( \text{lik_params} \), for which it computes the conditional loglikelihood

Todo

The environment of the returned function contains the time series and the model object (initially argument \( \text{model} \), later the model used in the last call to \( f \)). So, these things can be extracted from \( f \). Is it necessary to create convenience functions?

Methods

signature(model = "MixAR", ts = "numeric")

See Also

\( \text{mix_pdf}, \text{mix_cdf} \)

---

**marg_loglik**

Calculate marginal loglikelihood at high density points of a MAR model.

Description

The function implements the method by Chib (1995) and Chib and Jeliazkov (2001) for calculation of the marginal loglikelihood of a mixture autoregressive model. It automatically finds high density values for model parameters, and evaluates the likelihood at such points.

Usage

\[ \text{marg_loglik}(y, \text{model}, \tau, \text{nsim}, \text{prob_mod}) \]
Arguments

- **y**: a time series (currently a numeric vector).
- **model**: object of formal class `MixAR`, containing initial values for the parameters. Currently available for `MixARGaussian` objects only.
- **tau**: tuning parameter for Metropolis-Hasting move to update autoregressive parameters.
- **nsim**: sample size on which to evaluate highest density values.
- **prob_mod**: this is currently the output from `Choose_pk`: the proportion of times the "best model" was chosen.

Details

- `nsim` is the sample size on which to evaluate highest density values for each set of parameters. For example, choosing `nsim=1000` results in `1000*(g+3)` (1000 iterations for each autoregressive component, plus 1000 for mean and scale parameters and mixing weights).

Value

- A list containing the following elements:
  - `marg_loglik`: value of the marginal loglikelihood.
  - `phi_hd`: set of highest density autoregressive parameters.
  - `prec_hd`: set of highest density precision parameters.
  - `mu_hd`: set of highest density mean parameters.
  - `weig_hd`: set of highest density mixing weights.

Author(s)

- Davide Ravagli

References


Examples

```r
prob <- c(0.5, 0.5)
sigma <- c(1, 2)
arco <- list(-0.5, 1)

model <- new("MixARGaussian", prob = prob, scale = sigma, arcoef = arco)

set.seed(1234)
y <- mixAR_sim(model, 250, rep(0, max(model@order)), nskip = 100) # data

nsim <- 10 # 50
marg_loglik(y, model, tau = c(.15, .25), nsim = nsim, 0.5)
```
MixAR-class

Class "MixAR" — mixture autoregressive models

Description

Mixture autoregressive models

Objects from the Class

A virtual Class: no objects can be created from it.

Derived classes add distribution properties, e.g., use class "MixARGaussian" for MixAR models with Gaussian error components.

Slots

prob: the mixing probabilities, "numeric".
order: the AR orders, "numeric".
shift: intercept terms, "numeric".
scale: scaling factor, "numeric".
arcoef: autoregressive coefficients, an object from class "raggedCoef" containing one row for each mixture component.

Methods

fit_mixAR signature(x = "ANY", model = "MixAR", init = "list"): ...
fit_mixAR signature(x = "ANY", model = "MixAR", init = "missing"): ...
fit_mixAR signature(x = "ANY", model = "MixAR", init = "MixAR"): ...
fit_mixAR signature(x = "ANY", model = "MixAR", init = "numeric"): ...
fit_mixAR signature(x = "ANY", model = "MixARGaussian", init = "MixAR"): ...
ge_dist signature(model = "MixAR"): ... 
initialize signature(.Object = "MixAR"): ...
lik_params signature(model = "MixAR"): ...
make_fcond_lik signature(model = "MixAR", ts = "numeric"): ...
mix_ek signature(model = "MixAR", x = "numeric", index = "numeric", xcond = "missing", scale = "missing"): ...
mix_ek signature(model = "MixAR", x = "numeric", index = "numeric", xcond = "missing", scale = "logical"): ...
mix_ek signature(model = "MixAR", x = "numeric", index = "missing", xcond = "numeric", scale = "missing"): ...
mix_ek signature(model = "MixAR", x = "numeric", index = "missing", xcond = "numeric", scale = "logical"): ...
mix_hatk signature(model = "MixAR", x = "numeric", index = "numeric", xcond = "missing"): ...

mix_ncomp signature(x = "MixAR"): ...

mixAR signature(template = "MixAR"): ...

noise_dist signature(model = "MixAR"): ...

noise_params signature(model = "MixAR"): ...

noise_rand signature(model = "MixAR"): ...

parameters signature(model = "MixAR"): ...

row_lengths signature(x = "MixAR"): ...

Author(s)
Georgi N. Boshnakov

See Also
mixAR, classes "MixARGaussian", "MixARgen"

Examples

## some models from subclasses of (virtual) class "MixAR"
names(exampleModels)
exampleModels$WL_A
exampleModels$WL_At

## modify an existing model, here change the mixture weights
mixAR(exampleModels$WL_A, coef = list((prob = c(0.4, 0.6))))

mixAR-methods Create MixAR objects

Description
Generic function with methods for creating MixAR objects.

Usage
mixAR(template, coef, ..., filler = NA_real_)

Arguments

  template
  an object to be used as a template for the new object, typically inheriting from "MixAR". Alternatively, missing or a numeric vector specifying the order of the MixAR model, see Details.

  coef
  parameters for the new object a list with components "arcoef", "order", "prob", "shift", and "scale".

  ...
  further arguments for methods.

  filler
  value for unspecified parameters, default is NA_real_.
mixAR provides an alternative to the function `new` for specifying MixAR models.

If `template` is numeric vector, it is taken to specify the AR order of the model and the number of mixture components. A Gaussian MixAR model is created with parameters filled initially with NA's and then updated with values given by `coef`. `coef` does not need to have values for all parameters and may be missing altogether. If NA's are not suitable for initialisation, a suitable value can be specified with `filler`.

If `template` is a MixAR object, then the new object will have the class of `template`. The new object is set initially to a copy of `template` and then updated with parameters specified by `coef` (and maybe others for some methods).

In principle, the numeric parameters are vectors of length the number of components of the MixAR model. For convenience, single values are replicated to the number of components. For this to work, at least one component must be specified completely, for example the order. It is an error for the parameters to imply conflicting number of components.

**Methods**

```r
signature(template = "ANY")
signature(template = "MixAR")
```

**See Also**

- class "MixARGaussian", class "MixARgen"

**Examples**

```r
mixAR(coef = list(prob = c(.5,.5), scale = c(1,2),
         arcoef = list(.5, 1.1), shift = c(0,0), order = c(1,1)))

mixAR(template = c(1,1))
mixAR(coef = list(order = c(1,1))) # same

m2 <- new("MixARGaussian", order = c(3, 2, 1),
          arcoef = matrix(c(1:3, c(1:2, 0), c(1, 0, 0)), nrow = 3, byrow = TRUE))
m2a <- mixAR(m2, list(prob = c(0.5, 0.25, 0.25)))
show_diff(m2, m2a)
```

**Description**

Fit a mixture autoregressive model to a univariate time series using the EM algorithm.
Usage

mixARemFixedPoint(y, model, est_shift = TRUE, crit = 1e-14,
maxniter = 200, minniter = 10, verbose = FALSE)

mixARgenemFixedPoint(y, model, crit = 1e-14, maxniter = 200,
minniter = 10, verbose = FALSE, ...)

Arguments

y a univariate time series.
model an object of class MixAR, a mixture autoregressive model providing the model
specifications and initial values for the parameters.
est_shift if TRUE optimise also w.r.t. the shift (constant) terms of the AR components, if
FALSE keep the shift terms fixed.
crit stop iterations when the relative change in the log-likelihood becomes smaller
than this value.
maxniter maximum number of iterations.
minniter minimum number of iterations, do at least that many iterations.
... further arguments to be passed on to the M-step optimiser.
verbose print more details during optimisation.

Details

mixARemFixedPoint and mixARgenemFixedPoint estimate MixAR models with the EM algo-
...
MixARGaussian-class

See Also

fit_mixAR which uses these functions for estimation, classes "MixARGaussian", "MixARgen"

Examples

## data(ibmclose, package = "fma") # ibm data from BJ
m0 <- exampleModels$WL_ibm
m1 <- mixARemFixedPoint(fma::ibmclose, m0)
m1a <- mixARemFixedPoint(fma::ibmclose, m1$model)
show_diff(m1$model, m1a$model)
mixARemFixedPoint(fma::ibmclose, m0, est_shift = FALSE)

## simulate a continuation of ibmclose, assuming m0
ts1 <- mixAR_sim(m0, n = 50, init = c(346, 352, 357), nskip = 0)
m2a <- mixARemFixedPoint(ts1, m0, est_shift = FALSE)$model
m2b <- mixARemFixedPoint(diff(ts1), m0, est_shift = FALSE)$model

MixARGaussian-class  mixAR models with Gaussian noise components

Description

Class "MixARGaussian" represents MixAR models with Gaussian noise components.

Objects from the Class

Objects can be created by calls of the form new("MixARGaussian", ...), giving the elements of the model as named arguments, see the examples below. All elements of the model, except arcoef, are simple numeric vectors. From version 0.19-15 of package MixAR it is possible to create objects using MixARGaussian(...). The two forms are completely equivalent.

arcoef contains the AR coefficients, one numeric vector for each mixture component. It can be given as a "raggedCoef" object or as a list of numeric vectors.

To input a model with seasonal AR coefficients, argument passed to arcoef can be passed as a raggedCoefS object, or as a list of three elements. For the latter, seasonality s must be explicitly indicated. AR coefficients can be given as list or matrix within the main list (one for main AR coefficients, named a, and one for seasonal AR coefficients, as). Each row of a input matrix/element of the list denotes one component of the mixture. If not named, initialisation takes the first passed element to be a and the second to be as.

The AR order of the model is inferred from arcoef argument. If argument order is given, it is checked for consistency with arcoef. The shift slot defaults to a vector of zeroes and the scale slot to a vector of ones.

The distribution of the noise components is standard Gaussian, N(0,1).
MixARGaussian-class

Slots

All slots except arcoef are numeric vectors of length equal to the number of components in the model.

prob: probabilities of the mixture components
order: AR orders of the components
shift: the shift (intercept) terms of the AR components
scale: the standard deviations of the noise terms of the AR components
arcoef: The AR components, object of class "raggedCoef"

Extends

Class "MixAR", directly.

Methods

mix_cdf signature(model = "MixARGaussian", x = "numeric", index = "numeric", xcond = "missing"): ...
mix_cdf signature(model = "MixARGaussian", x = "numeric", index = "missing", xcond = "numeric"): ...
fit_mixAR signature(x = "ANY", model = "MixARGaussian", init = "MixAR"): ...
get_edist signature(model = "MixARGaussian"): ...
mix_cdf signature(model = "MixARGaussian", x = "missing", index = "missing", xcond = "numeric"): ...
mix_pdf signature(model = "MixARGaussian", x = "missing", index = "missing", xcond = "numeric"): ...
mix_pdf signature(model = "MixARGaussian", x = "numeric", index = "missing", xcond = "numeric"): ...
mix_pdf signature(model = "MixARGaussian", x = "numeric", index = "numeric", xcond = "missing"): ...
noise_dist signature(model = "MixARGaussian"): ...
noise_rand signature(model = "MixARGaussian"): ...

Author(s)

Georgi N. Boshnakov

See Also

classes "MixARgen", "MixAR"
MixARgen-class

Examples

```r
showClass("MixARGaussian")

## load ibm data from BJ
## data(ibmclose, package = "fma")

## compute a predictive density, assuming exampleModels$WL_ibm model
## for the first date after the end of the data
pdf1 <- mix_pdf(exampleModels$WL_ibm, xcond = as.numeric(fma::ibmclose))

## plot the predictive density
## (cdf is used to determine limits on the x-axis)
cdf1 <- mix_cdf(exampleModels$WL_ibm, xcond = as.numeric(fma::ibmclose))
gbutils::plotpdf(pdf1, cdf = cdf1, lq = 0.001, uq = 0.999)

## compute lower 5% quantile of cdf1
gbutils::cdf2quantile(0.05, cdf = cdf1)
```

MixARgen-class

Class "MixARgen"

Description

A class for MixAR models with arbitrary noise distributions. "MixARgen" inherits from "MixAR".

Objects from the Class

Objects can be created by calls of the form `new("MixARgen", dist, ...)` or `mixARgen(...)`. The two forms are completely equivalent. The latter is available from version 0.19-15 of package MixAR.

Slots

Most slots are inherited from class "MixAR".

- `prob`: the mixing probabilities, "numeric".
- `order`: the AR orders, "numeric".
- `shift`: intercept terms, "numeric".
- `scale`: scaling factor, "numeric".
- `arcoef`: autoregressive coefficients, an object from class "raggedCoef" containing one row for each mixture component.
- `dist`: Object of class "list", representing the noise distributions. The list contains one element for each component of the MixAR model or a single element if the noise distribution is the same for all components.

If the distributions do not contain parameters (e.g. Gaussian or $t_4$) it is sufficient to give the list of functions in the element `dist` of the list.
If the distributions do contain parameters the recommended arrangement is to give a list with components generator and param, such that a call generator(param) should produce the required list of distributions.
This is not finalised but if changed, backward compatibility with existing objects will be maintained.

**Extends**

Class "MixAR", directly.

**Methods**

- `get_edist` signature(model = "MixARgen"): ...
- `initialize` signature(.Object = "MixARgen"): ...
- `lik_params` signature(model = "MixARgen"): ...
- `mix_cdf` signature(model = "MixARgen", x = "missing", index = "missing", xcond = "numeric"): ...
- `mix_cdf` signature(model = "MixARgen", x = "numeric", index = "missing", xcond = "numeric"): ...
- `mix_cdf` signature(model = "MixARgen", x = "numeric", index = "numeric", xcond = "missing"): ...
- `mix_pdf` signature(model = "MixARgen", x = "missing", index = "missing", xcond = "numeric"): ...
- `mix_pdf` signature(model = "MixARgen", x = "missing", index = "missing", xcond = "numeric"): ...
- `mix_pdf` signature(model = "MixARgen", x = "numeric", index = "missing", xcond = "numeric"): ...
- `mix_pdf` signature(model = "MixARgen", x = "numeric", index = "missing", xcond = "numeric"): ...
- `noise_dist` signature(model = "MixARgen"): ...
- `noise_params` signature(model = "MixARgen"): ...
- `noise_rand` signature(model = "MixARgen"): ...

**Examples**

```
showClass("MixARgen")
exampleModels$WL_ibm_gen@dist
noise_dist(exampleModels$WL_ibm_gen, "cdf")
note_dist(exampleModels$WL_ibm_gen, "pdf")
note_dist(exampleModels$WL_ibm_gen, "pdf", expand = TRUE)
note_dist(exampleModels$WL_ibm_gen, "cdf", expand = TRUE)
```

```r
## data(ibmclose, package = "fma") # for \texttt{Var}

pdf1 <- mix_pdf(exampleModels$WL_ibm, xcond = as.numeric(fma::ibmclose))
cdf1 <- mix_cdf(exampleModels$WL_ibm, xcond = as.numeric(fma::ibmclose))
gbutils::plotpdf(pdf1, cdf = cdf1, lq = 0.001, uq = 0.999)
```
mixARnoise_sim

Simulate white noise series from a list of functions and vector of regimes

Description

Simulate white noise series from a list of functions and vector of regimes. This function is used internally for simulation from MixAR models.

Usage

mixARnoise_sim(rdist, z)
Arguments

rdist  a list of functions for random number generation, see `Details`.

z  a vector of positive integers specifying the `regimes`.

Details

If the length of the list rdist is \( \max(z) \), then \( z[i] \) is the random number generator for regime \( i \). Alternatively, if rdist is of length one, then the same generator will be used for all regimes.

mixARnoise_sim returns a vector, say \( y \), of the same length as \( z \), such that \( y[i] \) is generated by \( z[[i]] \).

Value

a numeric vector

See Also

mixAR_sim

Examples

```r
## MixAR with 2 components: N(0,1) and t_5
set.seed = 1234
z <- sample(2, size = 5, replace = TRUE)
mixARnoise_sim(list(rnorm, function(n) rt(n, 5)), z)
```

---

mixAR_BIC  

\( BIC \) based model selection for MixAR models

Description

\( BIC \) calculations for mixture autoregressive models.

Usage

```r
mixAR_BIC(y, model, fix = NULL, comp_loglik = TRUE, index)
BIC_comp(x, y)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>a time series.</td>
</tr>
<tr>
<td>model</td>
<td>the model for which to calculate ( BIC ), an object inheriting from class MixAR. Alternatively, an output list from fit_mixAR.</td>
</tr>
<tr>
<td>fix</td>
<td>If <code>fix = &quot;shift&quot;</code> shift parameters are not included in calculation of ( BIC ). Default is <code>NULL</code>, i.e. shift parameters are included.</td>
</tr>
</tbody>
</table>
**Details**

`mixAR_BIC` calculates the BIC criterion of a given MixAR object with respect to a specified time series.

If `index` is specified, it has to be at least equal to the largest autoregressive order. The function calculates BIC on the last `(index + 1):n` data points.

`BIC_comp` calculates the value of BIC for the models listed in `x` with respect to the specified time series `y`.

If the distributions of the components contain estimated parameters, then their number is included in the number of parameters for the calculation of BIC.

**Value**

If `comp_loglik = TRUE`, the function calculates BIC based on the given model, data and `index`.

If `comp_loglik = FALSE` and model is output from `fit_mixAR`, it returns object `vallogf` from that list.

**Author(s)**

Davide Ravagli

**Examples**

```r
model1 <- new("MixARGaussian", prob = c(0.5, 0.5), scale = c(1, 2),
               arcoef = list(-0.5, 1.1))
model2 <- new("MixARGaussian", prob = c(0.5, 0.3, 0.2), scale = c(1, 3, 8),
               arcoef = list(c(-0.5, 0.5), 1, 0.4))
set.seed(123)
y <- mixAR_sim(model1, 400, c(0, 0, 0), nskip = 100)
mixAR_BIC(y, model1)
model_fit1 <- fit_mixAR(y, model1)
model_fit2 <- fit_mixAR(y, model2, crit = 1e-4)
mixAR_BIC(y, model_fit1)
mixAR_BIC(y, model_fit2)
BIC_comp(list(model1, model2, model_fit1, model_fit2), y)
mixAR_BIC(y, model_fit1, index = 20)
mixAR_BIC(y, model_fit2, index = 20)
```
mixAR_cond_probs  
*The E-step of the EM algorithm for MixAR models*

**Description**

Compute conditional probabilities for the E-step of the EM algorithm for MixAR models. Internal function.

**Usage**

`mixAR_cond_probs(model, y, indx = NULL)`

**Arguments**

- `model`: an object from a sub-class of "MixAR".
- `y`: the time series, a numeric vector.
- `indx`: indices of elements for which to compute residuals.

**Details**

This is essentially the E-step for the MixAR models.

**Value**

the conditional probabilities, an object from class "MixComp".

---

mixAR_diag  
*Diagnostic checks for mixture autoregressive models*

**Description**

Carry out diagnostic checks and tests on fitted mixAR models.

**Usage**

```r
## S3 method for class 'MixAR'
mixAR_diag(object, gof.lag = NULL, y, ask = interactive(), ..., plot = interactive(), std.resid = FALSE)
```

```r
mixAR_diag(model, y, ...)
```
Arguments

model, object  the model on which to perform the checks, an object from class MixAR. model can also be the output list from `fit_mixAR`.

gof.lag  Goodness of fit lag(s) for the Ljung-Box tests. Vector containing one or more positive integers. `max(gof.lag)` is the maximal lag in the acf and pacf plots. how many lags to compute for acf and pacf? The default is as that of `lag.max` for acf.

y  a time series, currently a numeric vector.

ask  if TRUE, ask (using a menu) which plot to present. Otherwise just plot the selected plots. `ask` is ignored if only one plot is selected with argument `plot`.

plot  if TRUE, the default, produce diagnostic plots. If FALSE don’t produce plots. Otherwise, a numeric vector of integers defining a subset of plots to consider, see Details.

std.resid  if TRUE standardise the ordinary residuals using the conditional standard deviations. NOTE: the default is currently FALSE but it may soon be changed to TRUE.

...  for `mixAR_diag`, passed on to `tsdiag`.

Details

It is recommended to use `tsdiag`. `mixAR_diag` is essentially deprecated and is still here for compatibility with old code. Moreover, the `tsdiag` method is more flexible. The only advantage of `mixAR_diag` is that it accepts also a list for argument `model` but this is equivalent to calling `tsdiag` with `object = model$model`.

The function calculates several types of residuals, provides diagnostic plots for each of them, and returns numerical results. The following choices are currently available:

1. ACF/PACF of residuals,
2. ACF/PACF of U_residuals,
3. ACF/PACF of tau_residuals,
4. ACF/Histogram of tau_residuals.

In interactive sessions the user is presented with a menu to select plot(s) from the available ones. The choice can be restricted to a subset of them by giving argument `plot` a vector of integers. This is most useful to select a particular plot, with something like `plot = 2` in the call to `tsdiag`. `plot` is used as an index vector, so `plot = -1` would remove the first item listed above from the offered alternatives.

Transformations on the data are performed, as described in Smith (1985).

Four types of residuals are computed:

**ordinary residuals** difference (possibly scaled) between observed values and point predictions.

**U_residuals/PIT residuals** probability integral transform of the data using the CDF of the conditional distributions implied by the fitted model. For a good model these should resemble an IID sequence uniformly distributed on (0,1).
**Value**

returns invisibly a list with class "tsdiagMixAR", currently containing the following components:

- residuals ordinary residuals,
- U_residuals see Details,
- V_residuals see Details,
- tau_residuals see Details,
- BIC the value of the BIC criterion, a number.

Each component, except BIC, is a list containing the residuals in component value, Ljung-Box test in "Ljung-Box" and possibly other tests suitable for the corresponding type of residuals.

**Note**

This function should be used for diagnostic checking of MixARGaussian objects only.

**Author(s)**

Davide Ravagli and Georgi N. Boshnakov

**References**


**See Also**

mixAR_BIC, tsdiag
Examples

```r
model1 <- new("MixARGaussian", prob = c(0.5, 0.5), scale = c(1, 2),
              arcoef = list(-0.5, 1.1))
set.seed(123)
y <- mixAR_sim(model1, 400, c(0,0,0), nskip = 100)

fit1 <- fit_mixAR(y, model1)
d <- tsdiag(fit1$model, c(10, 20, 50), y)
```

```r
## This will put each plot in a separate file (mydiag01.pdf, ..., mydiag04.pdf)
## pdf("mydiag%02d.pdf", onefile = FALSE)
## d <- tsdiag(fit1$model, c(10, 20, 50), y, ask = FALSE)
## dev.off()
```

---

mixAR_sim

Simulate from MixAR models

Description

Simulate from MixAR models

Usage

```r
mixAR_sim(model, n, init, nskip = 100, flag = FALSE)
```

```r
mixAny_sim(model, n, init, nskip=100, flag = FALSE,
 theta, galpha0, galpha, gbeta)
```

Arguments

- **model**: model from which to simulate, an object inheriting from class MixAR.
- **init**: initial values, numeric vector.
- **n**: size of the simulated series.
- **nskip**: number of burn-in values, see Details.
- **flag**: if TRUE return also the regimes.
- **theta**: ma coef, a list.
- **galpha0**: alpha0[k], k=1,...,g.
- **galpha**: garch alpha.
- **gbeta**: garch beta.
mixAR_sim simulates a series of length nskip+n and returns the last n values.

mixAny_sim simulates from a MixAR model with GARCH innovations. mixAny_sim was a quick fix for Shahadat and needs consolidation.

The vector init provides the initial values for \( t = \ldots, -1, 0 \). Its length must be at least equal to the maximal AR order. If it is longer, only the last \( \max(\text{model@order}) \) elements are used.

Value

a numeric vector of length n. If flag = TRUE it has attribute regimes containing z.

Examples

eexampleModels$WL_ibm
## simulate a continuation of BJ ibm data
ts1 <- mixAR_sim(eexampleModels$WL_ibm, n = 30, init = c(346, 352, 357), nskip = 0)

# a simulation based estimate of the 1-step predictive distribution
# for the first date after the data.
s1 <- replicate(1000, mixAR_sim(eexampleModels$WL_ibm, n = 1, init = c(346, 352, 357), nskip = 0))
plot(density(s1))

# load ibm data from BJ
## data(ibmclose, package = "fma")

# overlay the 'true' predictive density.
pdf1 <- mix_pdf(eexampleModels$WL_ibm, xcond = as.numeric(fma::ibmclose))
curve(pdf1, add = TRUE, col = 'blue')

# estimate of 5% quantile of predictive distribution
quantile(s1, 0.05)

# Monte Carlo estimate of "expected shortfall"
# (but the data has not been converted into returns...)
mean(s1[ s1 <= quantile(s1, 0.05) ])

mixAR_switch

Relabel the components of a MixAR model

Description

Relabel the components of a MixAR model.

Usage

mixAR_switch(model, perm)
mixAR_permute(model, perm)
Arguments

- `model`: a MixAR model
- `perm`: a permutation for relabeling

Details

If the permutation is the identity permutation the model is returned as is. Otherwise the order of the components is changed according to `perm`. Basically, `perm` is used as index, e.g. `prob[perm]`, etc.

Note

Currently the function only reorders the "usual" components. Subclasses of "MixAR" may contain other parameters (e.g. different error distributions). So this function may not be appropriate for them.

MixComp-class

Class "MixComp" — manipulation of MixAR time series

Description

Class "MixComp" represents components of mixture autoregressive time series and their transformations obtained by arithmetic and related operations. Methods are provided to allow convenient computation with such time series.

Objects from the Class

Objects can be created by calls of the form `new("MixComp", ...)`. It is more usual however to obtain such objects initially from functions such as `mix_ek`. Methods are defined to allow for convenient and intuitive further manipulation of such objects.

Internally, an object of class `MixComp` is a matrix with one column for each component. However, methods for arithmetic operations involving `MixComp` objects are defined to perform natural operations for mixture objects. For example, multiplication by vectors is commutative and “does the right thing”.

Slots

- `m`: Object of class "matrix" with one column corresponding to each component of the mixture AR model.

Methods

Arithmetic operations involving `MixComp` objects are defined to allow for convenient execution of computations for mixture autoregressive models, see class "MixComp".

- `signature(e1 = "MixComp", e2 = "missing")`: unary minus for "MixComp" objects.
- signature(e1 = "numeric", e2 = "MixComp"):
  If e2 is thought of as a matrix, m, then the number of elements of e1 must be the same as 
  the number of rows of m and each column of m is subtracted from e1, see also "mix_ek", 
  "mix_hatk".
  As a special case, if m has only one row, then it is subtracted from each element of e1, i.e. 
  that row is replicated to obtain a matrix with as many rows as the length of e1 and then subtracted 
  from e1 as above.
  The result is a MixComp object.
- signature(e1 = "MixComp", e2 = "numeric"): This is analogous to the above method. **(FIXME: 
  the code of this function does not deal with the special case as in the above method. Is this an 
  omission or I have done it on purpose?)**
%of% signature(e1 = "function", e2 = "MixComp"): This applies the function e1 to each el-
 element of e2. Together with the arithmetic operations this allows for easy computation with 
MixComp objects (e.g. pdfs, likelihoods).
%of% signature(e1 = "character", e2 = "MixComp"):
%of% signature(e1 = "list", e2 = "MixComp"): If e1 is of length one it specifies a function to 
be applied to each element of e2, otherwise it is a list of functions, such that the i-th function 
is applied to the i-th column of e2@m.
* signature(e1 = "MixComp", e2 = "MixComp"): ...
* signature(e1 = "MixComp", e2 = "numeric"): see the following.
* signature(e1 = "numeric", e2 = "MixComp"): Column i of the MixComp object is multiplied 
by the i-th element of the numeric vector, i.e. each "row" of the MixComp object is multiplied 
by the vector (or, the vector is replicated to a matrix to be multiplied by the MixComp object).
* signature(e1 = "function", e2 = "MixComp"): Multiplying a function by a MixComp object 
actually applies the function to each element of the object. This is a misuse of methods, prefer 
operator %of% which does the same.
* signature(e1 = "character", e2 = "MixComp"): The first argument is a name of a function 
which is applied to each element of the MixComp object. This is a misuse of methods, see 
operator %of% which does the same.
/ signature(e1 = "MixComp", e2 = "numeric"): 
/ signature(e1 = "numeric", e2 = "MixComp"): Division works analogously to "/".
^ signature(e1 = "MixComp", e2 = "numeric"): If k is a scalar, raise each element of e1@m to 
power k.
  (For consistency this operation should have the semantics of "+" and "/" but this operator 
probably makes sense only for scalar 'e2', where the semantics doesn’t matter. So, don’t 
bother for now.)
+ signature(e1 = "numeric", e2 = "MixComp"): 
+ signature(e1 = "MixComp", e2 = "numeric"): Addition involving MixComp objects works analog-
ously to subtraction.

inner signature(x = "MixComp", y = "missing", star = "missing", plus = "missing"): With 
one argument inner computes the sum of the columns of the argument. This is conceptually 
equivalent to y being a vector of ones.
inner signature(x = "MixComp", y = "numeric", star = "missing", plus = "missing"): 

MixComp-class
inner signature(x = "numeric", y = "MixComp", star = "missing", plus = "missing"): The number of elements of the numeric argument should be equal to the number of rows of the MixComp object. Effectively, computes the inner product of the two arguments. The order of the arguments does not matter.

Returns a numeric vector.

inner signature(x = "MixComp", y = "numeric", star = "ANY", plus = "ANY"): Computes a generalised inner product of x with y using the specified functions in place of the usual "+" and "+" operations. The defaults for star and + are equivalent to multiplication and addition, respectively.

Note that "+" is a binary operation (not n-ary) in R. So technically the correct way to specify the default operation here is "sum" or sum. Since it is easy to make this mistake, if plus == "+", it is replaced by "sum". (In fact, plus is given a single argument, the vector of values to work on. Since "+" works as a unary operator on one argument, it would give surprising results if left as is.)

inner signature(x = "MixComp", y = "numeric", star = "ANY", plus = "missing"): This is a more efficient implementation for the case when plus = sum.

mix_ncomp signature(x = "MixComp"): Number of components.

signature(x = "MixComp") A "MixComp" object is essentially a matrix. This method gives the dimension of the underlying matrix. This method indirectly ensures that nrow() and ncol() work naturally for "MixComp" objects.

Author(s)

Georgi N. Boshnakov

Examples

```r
## dim, nrow, ncol
a <- new("MixComp", m = matrix(c(1:7, 11:17, 21:27), ncol = 3))
a
dim(a)
nrow(a)
ncol(a)
mix_ncomp(a)

-a
a - 1:7
1:7 + a
2*a

b <- new("MixComp", m = matrix(rnorm(18), ncol = 3))

## apply a function to the columns of a MixComp object
pnorm %of% b

## apply a separate function to to each column
flist <- list(function(x) pnorm(x),
```
mixFilter

Filter time series with MixAR filters

Description
Filter time series with MixAR filters, a generic function with no default method (currently).

Usage
mixFilter(x, coef, index, shift = 0, residual = FALSE, scale = 1)

Arguments
x time series
coef the filter coefficients
index indices for which to calculate the filtered values.
shift optional shifts (intercept) terms.
residual If FALSE (default) calculate “predictions”, if TRUE calculate “residuals”.
scale optional scale factor(s), makes sense only when residual=TRUE, corresponds to scale in the specification of a MixAR model.

Value
a MixComp object

Methods
signature(x = "ANY", coef = "ANY", index = "ANY") This method simply prints an error message and stops.
signature(x = "numeric", coef = "raggedCoef", index = "numeric")

Author(s)
Georgi N. Boshnakov

See Also
raghat1 mix_ek mix_hatk
mixgenMstep

M-step for models from class MixARgen

Description

M-step for models from class MixARgen. This function is for use by other functions.

Usage

mixgenMstep(y, tau, model, index, fix = NULL, comp_sigma = FALSE, method = "BBsolve", maxit = 100, trace = FALSE, lessverbose = TRUE, ...)

Arguments

y  
time series, a numeric vector.

tau  
conditional probabilities, an object of class "MixComp".

model  
the current model, an object from a subclass of class "MixAR".

index  
indices of observations for which to compute residuals, a vector of positive integers, see 'Details'.

method  
optimisation or equation solving method for package BB

...  
arguments to pass on to optimisation functions, not thought over yet. Do not use until this notice is removed.

comp_sigma  
If TRUE optimise the scale parameters using univariate optimisation. (note: does not work with argument 'fix' yet.)

fix  
specify parameters to be held fixed during optimisation, see 'Details'.

maxit  
maximal number of iterations for BB optimisers and solvers. Meant mainly for testing.

trace  
if TRUE, BB optimisers and solvers will print information about their proceedings. Meant mainly for testing.

lessverbose  
if TRUE, print a dot instead of more verbose information.

Details

mixgenMstep is an implementation of the M-step of the EM algorithm for mixture autoregressive models specified by objects of class "MixARgen". The function was build and modified incrementally with the main goal of providing flexibility. Speed will be addressed later.

By default optimisation is done with respect to all parameters. Argument fix may be a list with elements "prob", "shift", "scale" and "arcoef". These elements should be logical vectors containing TRUE in the positions of the fixed parameters. Elements with no fixed parameters may be omitted. (Currently the "prob" element is ignored, i.e. it is not possible to fix any of the component probabilities.)

If fix = "shift" the shift parameters are kept fixed. This is equivalent to fix = list(shift = rep(TRUE,g)).
The parameters (if any) of the distributions of the error components are estimated by default. Currently the above method cannot be used to fix some of them. This can be achieved however by modifying the distribution part of the model since that incorporates information about the parameters and whether they are fixed or not.

See Also

`fit_mixAR` and `mixARgenemFixedPoint` which are meant to be called by users.

---

### Description

Internal functions for EM estimation of MixAR models with Gaussian components: sums of products and crossproducts; M-step for MixAR estimation; estimation of autoregressive part of the model.

### Usage

- `tauCorrelate(y, tau, order)`
- `tau2arcoef(y, tau, order, est_shift = TRUE)`
- `mixMstep(y, tau, order, index, est_shift = TRUE)`

### Arguments

- `y` time series.
- `tau` conditional probabilities for the observations to belong to each of the components, a `MixComp` object.
- `order` order of the MixAR model, numeric vector of length the number of mixture components.
- `index` indices of the observations to include in the likelihood calculations, typically `(p+1):n`, where `p = max(order)` and `n = length(y)`.
- `est_shift` if TRUE include shifts (intercepts) in the AR components, otherwise set them to zero.

### Details

- `mixMstep` performs an M-step for estimation of MixAR models with Gaussian components.
- `tauCorrelate` computes crossproducts needed for EM estimation of MixAR models with Gaussian components.
- `tau2arcoef` computes the AR coefficients by solving Yule-Walker-type equations for each component.
mixSARfit

Value

For `mixMstep`, a MixAR model, an object of class `MixARGaussian`.
For `tauCorrelate`, a named list with the following components:

- `Stau`
- `Stauy`
- `Stauyy`

For `tau2arcoef`, a list with two components:

- `shift` the shift (intercept) terms, a numeric vector
- `arcoef` the AR coefficients as a list, whose i-th component contains the coefficients for component i (as a numeric vector)

MixSARfit  Fit mixture autoregressive models with seasonal AR parameters

Description

Provides estimation via EM-Algorithm for mixture autoregressive models including seasonal AR parameters.

Usage

```
mixSARfit(y, model, est_shift = FALSE, tol = 10^-14)
```

Arguments

- `y` a time series (currently a numeric vector).
- `model` an object of class "MixAR" including seasonal components.
- `est_shift` if missing or `FALSE`, fix the intercepts to zero, otherwise estimate them.
- `tol` threshold for stopping criterion.

Details

This function only works for "MixAR" objects in which slot `arcoef` is of class "raggedCoefS".

Value

A list of 2:

- `model` an object of class "MixAR". The estimated model.
- `vallogf` the value of the loglikelihood function for the returned model.

Author(s)

Davide Ravagli and Georgi N. Boshnakov
MixVAR-class

Class "MixVAR" — mixture vector autoregressive models

Description

Mixture vector autoregressive models

Objects from the Class

A virtual Class: No objects may be created from it.

Derived classes add distribution properties, e.g. use class "MixVARGaussian" for MixVAR models with Gaussian error components.

Slots

prob: the mixing probabilities, an object of class "numeric"
order: Object of class "numeric" ~~
shift: Object of class "matrix" ~~
vcov: Object of class "array" ~~
arcoef: Object of class "raggedCoefV" ~~

Methods

  fit_mixVAR signature(x = "ANY", model = "MixAR"): ...

Author(s)

Davide Ravagli

Examples

```r
ar1 <- list(c(0.5, -0.5), c(1.1, 0, -0.5))
ar12 <- list(0, c(-0.3, 0.1))
s = 12

rag <- new("raggedCoefS", a = ar1, as = ar12, s = s)

model <- new("MixARGaussian", prob = exampleModels$WL_A@prob, # c(0.5, 0.5)
  scale = exampleModels$WL_A@scale, # c(5, 1)
  arcoef = rag)

set.seed(1234)
y <- mixAR_sim(model, n = 100, init = rep(0, 24))

mixSARfit(y, model)
## fix the intercepts to zero
mixSARfit(y, model, est_shift = FALSE, tol = 1e-4)
```
mixVARfit

See Also

class "MixVARGaussian"

---

**Description**

Provides EM-estimation of mixture autoregressive models for multivariate time series

**Usage**

```r
mixVARfit(y, model, fix = FALSE, tol = 10^-6, verbose = FALSE)
```

**Arguments**

- `y` a data matrix.
- `model` an object of class "MixVAR" with initial values of parameter for EM estimation.
- `tol` Threshold for convergence criterion.
- `fix` if TRUE, fix the shift parameters.
- `verbose` if TRUE print information during the optimisation.

**Details**

Estimation is done under the assumption of multivariate Gaussian innovations.

**Value**

An object of class `MixVARGaussian` with EM estimates of model parameters.

**Author(s)**

Davide Ravagli

**References**


**See Also**

`fit_mixVAR-methods` for examples
MixVARGaussian-class

MixVAR models with multivariate Gaussian noise components

Description

Class MixVARGaussian represents MixAR models with multivariate Gaussian noise components.

Objects from the Class

Objects can be created by calls of the form `new("MixVARGaussian", ...)`, giving the elements of the model as named arguments, see the examples below.

- `arcoef` contains the AR coefficients, one numeric array for each mixture component. It can be given as a "raggedCoefV" object or as a list of numeric arrays.

  The AR order of the model is inferred from `arcoef` argument. If argument `order` is given, it is checked for consistency with `arcoef`. The `shift` slot defaults to a matrix of zeroes and the `vcov` slot to an array of identity matrices, one for each component.

  The distribution of the noise components is standard multivariate Gaussian, N(0,1).

Slots

All slots except `arcoef` are numeric vectors of length equal to the number of components in the model.

- `prob`: probabilities of the mixture components,
- `order`: AR orders of the components,
- `shift`: the shift (intercept) terms of the AR components,
- `vcov`: covariance matrices of the noise terms of the AR components,
- `arcoef`: The AR components, object of class "raggedCoefV".

Extends

Class "MixAR", directly.

Methods

- `fit_mixAR` signature(x = "ANY", model = "MixVARGaussian"): ...

Author(s)

Davide Ravagli

See Also

- class "MixAR"
### Examples

```r
classify("MixVARGaussian")
```

```r
## Create array of covariance matrices
Sigma1 <- cbind(c(0.0013, 0.0011), c(0.0011, 0.0012))
Sigma2 <- cbind(c(0.0072, 0.0047), c(0.0047, 0.0039))
Sigma <- array(c(Sigma1, Sigma2), dim=c(2,2,2))
```

```r
## Create list of AR coefficients
AR <- list()
AR[[1]] <- array(c(0.0973, -0.0499, 0.2927, 0.4256, ## VAR(2;4)
                   -0.0429, 0.0229, -0.1515, -0.1795,
                   -0.0837, -0.1060, -0.1530, 0.1947,
                   -0.1690, -0.0903, 0.1959, 0.0955), dim=c(2,2,4))
AR[[2]] <- array(c(0.3243, 0.2648, 0.4956, 0.2870, ## VAR(2;3)
                   -0.1488, 0.0454, -0.0593, -0.3629,
                   0.1314, 0.0274, 0.0637, 0.0485), dim=c(2,2,3))
```

```r
## Create vector of mixing weights
prob <- c(0.6376, 0.3624)
```

```r
## Create matrix of shift parameters
shift <- cbind(c(0.0044, 0.0020), c(-0.0039, -0.0014))
```

```r
## Build "MixVARGaussian" model
new("MixVARGaussian", prob=prob, vcov=Sigma, arcoef=AR, shift=shift)
```

---

### Description

Simulate data from multivariate MixAR models under the assumptions of multivariate Gaussian innovation.

### Usage

```
mixVAR_sim(model, n, init, nskip = 100, flag = FALSE)
```

### Arguments

- **model**: model from which to simulate, an object inheriting from class `MixVAR`.
- **n**: size of simulated multivariate series.
- **init**: initial values, a numeric matrix. If missing, a matrix of 0 values is generated.
- **nskip**: number of burn-in values.
- **flag**: if TRUE returns also the regimes.
Details

mixVAR_sim simulates a series of length nskip + n and returns the last n values. init provides initial values for the algorithm. Each row is considered as a time point. The number of rows must be at least equal to the maximal AR order.

Value

a numeric matrix with n rows.

Author(s)

Davide Ravagli

See Also

mixAR_sim

Examples

AR <- list()
AR[[1]] <- array(c(0.5,-0.3,-0.6,0,0.5,0.4,0.5,-0.3), dim = c(3,3,1))
AR[[2]] <- array(c(-0.5,0.3,0,1,0,-0.5,-0.4,-0.2, 0.5), dim = c(3,3,1))

prob <- c(0.75, 0.25)
shift <- cbind(c(0,0,0), c(0,0,0))

Sigma1 <- cbind(c(1, 0.5, -0.4), c(0.5, 2, 0.8), c(-0.4, 0.8, 4))
Sigma2 <- cbind(c(1,0.2, 0), c(0.2, 2, -0.15), c(0, -0.15, 4))
Sigma <- array(c(Sigma1, Sigma2), dim = c(3,3,2))

m <- new("MixVARGaussian", prob=prob, vcov=Sigma, arcoef=AR, shift=shift)
mixVAR_sim(m, n=500, init=matrix(rep(0,3), ncol=3), nskip=100, flag=FALSE)

mix_ek

Function and methods to compute component residuals for MixAR models

Description

Compute component residuals for MixAR models.

Usage

mix_ek(model, x, index, xcond, scale)
Arguments

model  a model.
x     time series.
index  a vector of positive integer specifying the indices for which to compute the residuals, has a natural default.
xcond the past values needed for the conditional distribution, a numeric vector of length at least the maximal AR order of the components.
scale logical or missing, if TRUE standardise the residuals.

Details

mix_ek computes component residuals from MixAR models.

It is highly desirable to use it along with mix_hatk and the underlying function mixFilter. Doing this ensures transparent code and easy maintenance. Also, more efficient implementation can be introduced without changing other code.

Methods

signature(model = "MixAR", x = "numeric", index = "missing", xcond = "numeric", scale = "logical")
signature(model = "MixAR", x = "numeric", index = "missing", xcond = "numeric", scale = "missing")
signature(model = "MixAR", x = "numeric", index = "numeric", xcond = "missing", scale = "logical")
signature(model = "MixAR", x = "numeric", index = "numeric", xcond = "missing", scale = "missing")

Author(s)

Georgi N. Boshnakov

See Also

mixFilter which is used by mix_ek to do the job, MixComp-class for easy manipulation of the returned object.
class "MixAR"
**mix_hatk**  
*Compute component predictions for MixAR models*

**Description**  
Function and methods to compute component predictions for MixAR models

**Usage**  
```r  
mix_hatk(model, x, index, xcond)  
```

**Arguments**  
- `model`: a model.  
- `x`: time series.  
- `index`: a vector of positive integers specifying the indices for which to compute the residuals, has a natural default.  
- `xcond`: the past values needed for the conditional distribution, a numeric vector of length at least the maximal AR order of the components.

**Methods**  
```r  
signature(model = "MixAR", x = "numeric", index = "numeric", xcond = "missing")  
```

**Author(s)**  
Georgi N. Boshnakov

**See Also**  
`class "MixAR"`

---

**mix_moment**  
*Conditional moments of MixAR models*

**Description**  
Conditional moments of MixAR models.

**Usage**  
```r  
mix_location(model, x, index, xcond)  
mix_variance(model, x, index, xcond)  
mix_central_moment(model, x, index, xcond, k)  
mix_moment(model, x, index, xcond, k)  
mix_kurtosis(...)  
mix_ekurtosis(...)  
```
Arguments

- **model**: a MixAR object.
- **x**: a time series.
- **index**: a vector of indices in x for which to compute the requested property. If missing, the computation is done for all indices greater than max(model@order).
- **xcond**: a time series, the point prediction is computed for the first value after the end of the time series. Only the last max(model@order) values in xcond are used.
- **k**: a positive integer specifying the moment to compute.
- ... passed on to `mix_central_moment`.

Details

These functions compute conditional moments and related quantities. `kurtosis` and `ekurtosis` compute conditional kurtosis and excess kurtosis, respectively. Effectively, they have the same parameters as `mix_central_moment`, since they pass "..." to it along with k = 4. It is an error to supply argument k to the kurtosis functions.

Value

- when called with one argument (model), a function with argument xcond; otherwise if xcond is not missing, a single numeric value; otherwise a vector of length length(index).

Note

I wrote the above description recently from reading six years old code, it may need further verification.

Author(s)

Georgi N. Boshnakov

References


See Also

`mix_pdf`, `mix_cdf`, `mix_qf` for the predictive distributions (pdf, cdf, quantiles);

Examples

```r
## data(ibmclose, package = "fma") # 'ibmclose'
ibmclose <- as.numeric(fma::ibmclose)
length(ibmclose) # 369
max(exampleModels$WL_ibm@order) # 2

## compute point predictions for t = 3,...,369
```
```r
pred <- mix_location(exampleModels$WL_ibm, ibmclose)
plot(pred)
## compute one-step point predictions for t = 360,...369
mix_location(exampleModels$WL_ibm, ibmclose, index = 369 - 9:0)

f <- mix_location(exampleModels$WL_ibm) # a function
## predict the value after the last
f(ibmclose)

## a different way to compute one-step point predictions for t = 360,...369
sapply(369 - 10:1, function(k) f(ibmclose[1:k]))

## the results are the same, but notice that xcond gives past values
## while index above specifies the times for which to compute the predictions.
identical(sapply(369 - 10:1, function(k) f(ibmclose[1:k])),
        mix_location(exampleModels$WL_ibm, ibmclose, index = 369 - 9:0))

## conditional variance
f <- mix_variance(exampleModels$WL_ibm) # a function
## predict the value after the last
f(ibmclose)

## a different way to compute one-step point predictions for t = 360,...369
sapply(369 - 10:1, function(k) f(ibmclose[1:k]))

## the results are the same, but notice that xcond gives past values
## while index above specifies the times for which to compute the predictions.
identical(sapply(369 - 10:1, function(k) f(ibmclose[1:k])),
        mix_variance(exampleModels$WL_ibm, ibmclose, index = 369 - 9:0))

# interesting example
# bimodal distribution, low kurtosis, 4th moment not much larger than 2nd
moWL <- exampleModels$WL_ibm
mix_location(moWL,xcond = c(500,450))
mix_kurtosis(moWL,xcond = c(500,450))

f1pdf <- mix_pdf(moWL,xcond = c(500,450))
f1cdf <- mix_cdf(moWL,xcond = c(500,450))
gbutils::plotpdf(f1pdf,cdf=f1cdf)
gbutils::plotpdf(f1cdf,cdf=f1cdf)
f1cdf(c(400,480))
mix_variance(moWL,xcond = c(500,450))
mix_central_moment(moWL,xcond = c(500,450), k=2)
sqrt(mix_variance(moWL,xcond = c(500,450)))
sqrt(mix_central_moment(moWL,xcond = c(500,450), k=2))
```
**mix_ncomp-methods**

**Number of rows or columns of a MixComp object**

**Description**
Function and methods to get the number of component in a mixture object. For "MixComp" objects this is equivalent to `ncol`.

**Usage**

```r
mix_ncomp(x)
```

**Arguments**

- `x` an object, such as "MixComp" or "MixAR".

**Value**

a number

**Methods**

```r
signature(x = "MixAR")
signature(x = "MixComp")
```

**Author(s)**
Georgi N. Boshnakov

**See Also**

`MixComp-class`, `MixAR-class`

**mix_pdf-methods**

**Conditional pdf’s and cdf’s of MixAR models**

**Description**

Gives conditional probability densities and distribution functions of mixture autoregressive models.
Methods

mix_pdf gives a probability density, mix_cdf a distribution function. If argument x is supplied, the functions are evaluated for the specified values of x, otherwise function objects are returned and can be used for further computations, eg for graphs.

mix_pdf and mix_cdf have methods with the following signatures.

signature(model = "MixARGaussian", x = "missing", index = "missing", xcond = "numeric")
Return (as a function of one argument) the conditional density (respectively cdf), \( f(x|x_{cond}) \), of \( X_{t+1} \) given the past values xcond. The values in xcond are in natural time order, e.g. the last value in xcond is \( x_t \). xcond must contain enough values for the computation of the conditional density (cdf) but if more are given, only the necessary ones are used.

signature(model = "MixARGaussian", x = "numeric", index = "missing", xcond = "numeric")
Compute the conditional density (respectively cdf) at the values given by x.

signature(model = "MixARGaussian", x = "numeric", index = "numeric", xcond = "missing")
Compute conditional densities (respectively cdf) for times specified in index. For each \( t \in index \) the past values needed for the computation of the pdf (cdf) are \( \ldots, x[t-2], x[t-1] \).

signature(model = "MixARgen", x = "missing", index = "missing", xcond = "numeric")
signature(model = "MixARgen", x = "numeric", index = "missing", xcond = "numeric")
signature(model = "MixARgen", x = "numeric", index = "numeric", xcond = "missing")

Author(s)

Georgi N. Boshnakov

See Also

mix_moment for examples and computation of summary statistics of the predictive distributions
mix_qf for computation of quantiles.

---

**mix_qf-methods**

*Conditional quantile functions of MixAR models*

**Description**

Gives conditional quantile functions of mixture autoregressive models.

**Usage**

mix_qf(model, p, x, index, xcond)
mix_se-methods

Arguments

model       mixAR model.
p          vector of probabilities.
x          time series.
index       vector of positive integers.
indexcond   the past values needed for the conditional distribution, a numeric vector of length
            at least the maximal AR order of the components.

Value

depending on the arguments, a function for computing quantiles or a numeric vector representing
quantiles, see sections 'Details' and 'Methods'

Methods

signature(model = "MixARGaussian", p = "missing", x = "missing", index = "missing", xcond = "numeric")
signature(model = "MixARGaussian", p = "numeric", x = "missing", index = "missing", xcond = "numeric")
signature(model = "MixARGaussian", p = "numeric", x = "numeric", index = "numeric", xcond = "missing")

Author(s)

Georgi N. Boshnakov

See Also

mix_pdf, mix_cdf;
mix_moment for examples

Description

Compute standard errors of estimates of MixAR models.

Usage

mix_se(x, model, fix_shift)
Arguments

x  time series.
model  MixAR model, an object inheriting from class “MixAR”.
fix_shift  logical. Should the shift parameters be fixed? Default is FALSE.

Details

For formulas used in the computation, see Wong (1998).

Value

a list with components:

standard_errors  Standard error of parameter estimates,
covariance_matrix  The covariance matrix, obtained as inverse of the information matrix,
Complete_Information  Complete information matrix,
Missing_Information  Missing information matrix.

Methods

signature(x = "ANY", model = "list")
signature(x = "ANY", model = "MixAR")
signature(x = "ANY", model = "MixARGaussian")

Author(s)

Davide Ravagli

References


Examples

## Example with IBM data
## data(ibmclose, package = "fma")

moWLprob <- exampleModels$WL_ibm@prob  # 2019-12-15; was: c(0.5339, 0.4176, 0.0385)
moWLsigma <- exampleModels$WL_ibm@scale  # c(4.8293, 6.0082, 18.1716)
moWLar <- list(-0.3208, 0.6711, 0)  # @Davide - is this from some model?
moWLibm <- new("MixARGaussian", prob = moWLprob, scale = moWLsigma, arcoef = moWLar)
**Description**

Multi-step predictions for MixAR models.

**Usage**

```r
multiStep_dist(model, maxh, N, xcond, ...)
```

**Arguments**

- `model` a MixAR model.
- `maxh` maximal horizon, a positive integer.
- `N` an integer specifying the number of simulation samples to use, see 'Details. This argument is used only by simulation based methods.
- `xcond` the past values needed for the conditional distribution, a numeric vector of length at least the maximal AR order of the components.
- `...` used only in some methods, see the details for the individual methods.

**Details**

The function currently implements two methods: the exact method due to Boshnakov (2009) and a simulation method described by (Wong and Li 2000) for Gaussian MixAR models but valid more generally.

The simulation method is available for any MixAR model, while the exact method is currently implemented only for models with Gaussian components ("MixARGaussian" class).

`multiStep_dist` returns a function which can be used to obtain various properties of the predictive distribution for lags up to `maxh`.

If argument `N` is missing the exact method is tried. Currently an error will result if the exact method is not implemented for `model`.

If argument `N` is given it must be a scalar numeric value, the number of simulations to be performed to construct an approximation for the predictive distributions.

The simulation is done by `multiStep_dist`. The properties obtained later from the function returned by `multiStep_dist` use the samples generated by the call to `multiStep_dist`. To do a simulation with different parameters (e.g., with larger `N`) call `multiStep_dist` again.
Details on the returned function:
If xcond is missing multiStep_dist returns a function with arguments h, what and xcond.
If xcond is supplied, then it is fixed to that value and the arguments of the returned function are
h, what and '...'. The dots argument is currently used in the case of the simulation method, see
below.
Let f be the function returned by multiStep_dist. Argument h is the required prediction hori-
zon and can be a number in the interval [1, maxh]. Argument what is the required property of
the predictive distribution for lag h. If what is a function, it is applied to the simulated sample
for the requested horizon (currently available only for the simulation method). If what is a char-
acter string, the corresponding property of the predictive distribution for horizon h is returned.
Currently possible values for what are:
"pdf" the probability density function.
"cdf" the cumulative distribution function.
"location" the location (conditional mean).
"variance" the conditional variance, a.k.a (squared) volatility.
"sd" the conditional standard deviation, a.k.a volatility.
"skewness" the conditional skewness.
"kurtosis" the conditional kurtosis.
Note that what = "pdf" and what = "cdf" return functions even in the simulation case. For "pdf"
the function is constructed using density and the "..." arguments passed to f will be passed on
to density if finer control is needed.
If what is none of the above, the raw object is returned currently (but this may change).

Value

a function as described in sections ‘Details’ and ‘Methods’

Methods

The Details section gives a rather detailed description of the function, so the descriptions below are
brief.

signature(model = "MixAR", maxh = "numeric", N = "numeric", xcond = "numeric") Non-missing
N requests the simulation method. The predictive distribution is approximated by simulating
N of future paths up to horizon maxh and using a non-parametric estimate. Arguments "..."
are passed to density to allow finer control.

signature(model = "MixARGaussian", maxh = "numeric", N = "missing", xcond = "missing")
Computes the predictive distribution using the exact method. Returns a function with argu-
ments h, what and xcond.

signature(model = "MixARGaussian", maxh = "numeric", N = "missing", xcond = "ANY") Computes
the predictive distribution using the exact method. Returns a function with arguments h and
what. (i.e., xcond is fixed to the supplied argument xcond).

Author(s)

Georgi N. Boshnakov
References


See Also

`predict_coef`

Examples

```r
## exact method, without xcond
dist <- multiStep_dist(exampleModels$WL_ibm, maxh = 3)

tfpdf <- dist(3, "pdf", xcond = c(560, 600)) # xcond is argument to 'dist' here
tfcdf <- dist(3, "cdf", xcond = c(560, 600))
## plot the pdf (gbutils::plotpdf determines suitable range automatically)
gbutils::plotpdf(tfpdf, cdf = tfcdf)

args(dist(3, "pdf", xcond = c(500, 600))) # x

## use a simulation method with N = 1000
tf <- multiStep_dist(exampleModels$WL_ibm, maxh = 3, N = 1000, xcond = c(560, 600))
args(tf) # (h, what, ...)

## the exact method may also be used with fixed xcond:
tfe <- multiStep_dist(exampleModels$WL_ibm, maxh = 3, xcond = c(560, 600))

## get pdf and cdf for horizon 3
tfepdf <- tfe(3, "pdf")
tfecdf <- tfe(3, "cdf")
## plot the pdf
gbutils::plotpdf(tfepdf, cdf = tfecdf)

tf(3, "location")
tf(1, "location")
mix_location(exampleModels$WL_ibm, xcond = c(560, 600))

## larger simulation gives better approximation, in general
tf <- multiStep_dist(exampleModels$WL_ibm, maxh = 3, N = 10000, xcond = c(560, 600))
tf(1, "location")

tf1000pdf <- tf(3, "pdf")
tf1000cdf <- tf(3, "cdf")
gbutils::plotpdf(tf1000pdf, cdf = tf1000cdf)

## plot the exact and simulated pdf's together for comparison
gbutils::plotpdf(tfepdf, cdf = tfecdf)
curve(tf1000pdf, add = TRUE, col = "red")
```
## get the raw data

tfs <- tf(1, "sampled")
apply(tfs, 2, mean) # location for lags from 1 to max(h) (here 3)

tf(1, "location")
tf(1, "variance")
tf(1, "sd")
mix_variance(exampleModels$WL.ibm, xcond = c(560, 600))
sqrt(mix_variance(exampleModels$WL.ibm, xcond = c(560, 600)))

mix_kurtosis(exampleModels$WL.ibm, xcond = c(359, 200))
mix_kurtosis(exampleModels$WL.ibm, xcond = c(359, 400))

---

### noise_dist

**Internal mixAR functions**

#### Description

Functions for the distributions of the components of MixAR models

#### Usage

```r
get_edist(model)
noise_dist(model, what, expand = FALSE)
noise_rand(model, expand = FALSE)
noise_params(model)
set_noise_params(model, nu)
```

#### Arguments

- `model` a model.
- `what` the property, a character string.
- `expand` if TRUE, expand the list to length equal to the number of components, see Details.
- `nu` degrees of freedom.

#### Details

- `get_edist` gives the distributions of the noise components of `model`. `noise_dist` gives property `what` of the noise distribution. `noise_rand` gives a list of functions for simulation from the component distributions.

In each case, the list contains one element for each component but if it is of length one, then the only element is common for all components. To force a complete list even in this case, use `expand = TRUE`.

`noise_params` gives the parameters of the model as a numeric vector.
The distribution is specified as a list. Element "dist" contain the distribution. Element "generator" is a function that generates a distribution like the one specified. If "dist" is absent or NULL, the generator is called to generate a distribution object.

Initially the distribution itself was used for slot dist. For compatibility with old code using that format, this is still supported.

See Also

fdist_stdnorm, fdist_stdt, fn_stdt.
Methods for function `noise_rand` in package `mixAR`

**Description**

Methods for function `noise_rand` in package `mixAR`

**Methods**

- `signature(model = "MixAR")`
- `signature(model = "MixARGaussian")`
- `signature(model = "MixARgen")`

**parameters**

Set or extract the parameters of MixAR objects

**Description**

Set or extract the parameters of MixAR objects.

**Usage**

```r
parameters(model, namesflag = FALSE, drop = character(0))
```

```r
parameters(model) <- value
## S4 replacement method for signature 'MixAR'
parameters(model) <- value
## S4 replacement method for signature 'ANY'
```

**Arguments**

- `model` a model.
- `namesflag` if TRUE, generate names.
- `drop` names of parameters not to include in the returned value, a character vector. The default is to return all parameters, see Details.
- `value` values of the parameters, numeric.

**Details**

This is a generic function. The dispatch is on argument `model`. The default calls `coef`.

`parameters` extracts the parameters of a MixAR object. It returns a numeric vector. If `namesflag` is TRUE the returned vector is named, so that the parameters can be referred to by names. Argument `drop` is a character vector giving names of parameters not to be included in the returned value.

This function can be useful for setting parameters from optimisation routines.

`set_parameters` is deprecated, use `parameters(model) <- value` instead.
Value

a vector of parameters, maybe with names.

Methods

signature(model = "ANY")
signature(model = "MixAR")

Examples

parameters(exampleModels$WL_ibm)
parameters(exampleModels$WL_ibm, namesflag = TRUE)
## drop orders
parameters(exampleModels$WL_ibm, namesflag = TRUE, drop = "order")
## drop orders and mixing weights
parameters(exampleModels$WL_ibm, namesflag = TRUE, drop = c("order", "prob"))

parameters(exampleModels$WL_I, namesflag = TRUE)
parameters(exampleModels$WL_II, namesflag = TRUE)

Description

The infix operator %of% is a generic function which applies functions to objects. This page describes the function and the methods defined in package mixAR.

Usage

"%of%"(e1, e2)
e1 %of% e2

Arguments

e1 usually a function, the name of a function, a character vector, or a list of functions, see Details.
e2 an object, usually matrix-like.

Details

%of% is a generic function with dispatch on both arguments. It is intended to be used mainly in infix form.
%of% transforms each “column” of a matrix-like object by a function. If e1 specifies a single function, that is applied to all columns. Otherwise length(e1) should equal the number of “columns” of e2 and e1[[i]] is applied to the i-th “column” of e2.
The mental model is that the first argument, e1, is (converted to) a list of functions containing one function for each column of e2. The i-th function is applied to each element of the i-th column.

The methods for "MixComp" objects allow for very transparent and convenient computing with "MixAR" objects.

**Value**

- for the default method, a matrix;
- for methods with e2 from class MixComp, a MixComp object with its slot m replaced by the result of applying e1 to its elements, see the descriptions of the individual methods for details;

**Methods**

Below are the descriptions of the methods for %of% defined by package mixAR.

signature(e1 = "ANY", e2 = "ANY") This is the default method. It uses apply() to evaluate e1 for each element of the matrix e2, without checking the arguments. If the arguments are not suitable for apply(), any error messages will come from it. So, for this method e1 is a function (or the name of a function) and e2 is a matrix or array.

signature(e1 = "function", e2 = "MixComp") Create (and return) a MixComp object with its slot m replaced by the result of applying the function e1 to each element of the MixComp object e2, see class "MixComp".

signature(e1 = "character", e2 = "MixComp") Here e1 contains the names of one or more functions. If length(e1) = 1, this is equivalent to the method for e1 of class "function". If length(e1) > 1, then for each i the function specified by e1[i] is applied to the i-th column of e2@m. In this case there is no recycling: e1 must have ncol(e2@m) elements.

signature(e1 = "list", e2 = "MixComp") Here each element of e1 is a function or the name of a function. It works analogously to the method with e1 from class "character". If length(e1) = 1, then e1[[1]] is applied to each element of e1@m. Otherwise, if length(e1) > 1, then e1[[i]] is applied to the i-th column of e2@m.

**Note**

The code is rather inefficient for some of the methods.

Maybe should require that the functions in the first argument are vectorised. (Some methods effectively assume it.)

**Author(s)**

Georgi N. Boshnakov

**See Also**

class "MixComp"
**Examples**

```r
m <- matrix(rnorm(18), ncol = 3)
## default method
pm1 <- pnorm %of% m
f3 <- list(pnorm, function(x, ...) pnorm(x, mean = 0.1),
           function(x, ...) pnorm(x, mean = -0.1))
## no method for f from "list" yet:
## pm2 <- f3 %of% m
mc <- new(“MixComp”, m = m)
pnorm %of% mc
pmc3 <- f3 %of% mc
## result is equivalent to applying f3[i] to m[, i]:
all.equal(pmc3@m, cbind(f3[[1]](m[, 1]), f3[[2]](m[, 2]), f3[[3]](m[, 3])))
```

---

**permn_cols**

All permutations of the columns of a matrix

**Description**

All permutations of the columns of a matrix

**Usage**

`permn_cols(m)`

**Arguments**

- `m`  
  a matrix

**Details**

This function is a wrapper for `permn` from package ‘combinat’.

**Value**

a list with one element for each permutation of the columns. Each element of the list is an unnamed list with two components:

1. the permutation, a vector of positive integers,
2. a matrix obtained by permuting the columns of `m`.

**Author(s)**

Georgi N. Boshnakov
Examples

```r
m <- matrix(c(11:14,21:24,31:34), ncol=3)
pm <- permn_cols(m)
pm[[2]]
```

---

**PortfolioData1**  
*Closing prices of four stocks*

**Description**

Closing prices of four stocks.

**Usage**

```r
data("PortfolioData1")
```

**Format**

A data frame with 867 observations on the following 4 variables.

- **DELL** numeric, Dell Technologies Inc.
- **MSFT** numeric, Microsoft Corporation.
- **INTC** numeric, Intel Corporation.
- **IBM** numeric, International Business Machine Corporation.

**Details**

Time series of daily adjusted close prices of the above stocks from 2 January 2016 to 29 January 2020 (867 observations).

**Source**

`https://finance.yahoo.com/`

**Examples**

```r
data(PortfolioData1)
dim(PortfolioData1)
head(PortfolioData1)
```
predict_coef

Exact predictive parameters for multi-step MixAR prediction

Description

Exact predictive parameters for multi-step MixAR prediction.

Usage

predict_coef(model, maxh)

Arguments

model a MixAR model.
maxh maximal horizon.

Details

predict_coef() implements the method of Boshnakov (2009) for the h-step prediction of MixAR processes. The h-step predictive distribution has a MixAR distribution with \( g^h \) components and this function computes its parameters.

predict_coef() implements the results by Boshnakov (2009) to compute the parameters of the predictive distributions. predict_coef() is mostly a helper function, use multiStep_dist for prediction/forecasting (the exact method for multiStep_dist uses predict_coef() to do the main work).

predict_coef() returns a list of lists containing the quantities needed for each horizon \( h \), see section Value.

Alternatively, the parameters can be obtained as MixAR models by calling the function generated by the exact method of multiStep_dist with argument what = "MixAR".

Value

a list with components:

**arcoefs**

a list, arcoefs[[h]] gives the ar coefficients for the h-step predictive distribution.

**sigmas**

a list, sigmas[[h]] sigmas[[h]] is a matrix, in which the \( k \)th column contains the theta coefficients needed to compute \( \sigma_{\theta_k} \) in the formula for sigma in Equation (16) (see Boshnakov 2009). In the paper the index is a tuple \((k_1,\ldots,k_h)\) for clarity. In the code each tuple \((k_1,\ldots,k_h)\) is mapped to a linear index in \(1,\ldots,g^h\) (there are \(g^h\) tuples for horizon \( h \), since the mixture has \( g^h \) components).

**probs**

a list, probs[[h]] gives the mixture weights for the h-step predictive distribution.

**sStable**

a list, sigmas[[h]] gives the scale parameters for the h-step predictive distribution.
**Author(s)**

Georgi N. Boshnakov

**References**


**See Also**

`multiStep_dist`

---

**ragged**  
*Small utilities for ragged objects*

**Description**

Small utilities for ragged objects. Modify the elements of raggedCoef objects, extract them as a vector.

**Usage**

```r
rag_modify(rag, v)
ragged2vec(x)
```

**Arguments**

- `rag`: the raggedCoef object to be modified.
- `v`: vector of values to replace the old ones.
- `x`: a raggedCoef object.

**Details**

An error will occur if the length of `v` is not equal to `sum(rag@p)`.  
`rag_modify` is, in a sense, the inverse of `ragged2vec`.

**Value**

- for `rag_modify`, a raggedCoef object of the same order as `rag` but with coefficients replaced by the new values.
- for `ragged2vec`, a numeric vector.
Examples

rag1 <- new("raggedCoef", list(1, 2:3, 4:6))
a1 <- (1:6)^2
rag1a <- rag_modify(rag1, a1)

rag2 <- new("raggedCoef", list(1, numeric(0), 4:6)) # a zero-length component
a2 <- (1:4)^2
rag2a <- rag_modify(rag2, a2)

---

Description

Some models have several vectors of parameters, possibly of different lengths, such that in some circumstances they are thought of as lists, in others as matrices after suitable padding with zeroes. Class "raggedCoef" represents such ragged lists. In package "MixAR" it is used to hold the autoregressive coefficients of MixAR models.

Usage

raggedCoef(p, value = NA_real_)

Arguments

- **p**: orders, vector of integers.
- **value**: typically, a list, but see Details.

Details

Class "raggedCoef" is for objects that can be considered as both, lists and matrices. The elements of the list are vectors, possibly of different lengths. When the object is viewed as a matrix, each element of the list (suitably padded with zeroes or NAs) represents a row of a matrix.

The recommended way to create objects from class "raggedCoef" is with the function `raggedCoef`. If `value` is a "raggedCoef" object it is returned. If `value` is a list, it is converted to "raggedCoef" using `new()`. If argument `p` is missing, it is inferred from the lengths of the elements of the list. If argument `p` is not missing, a consistency check is made to ensure that the order of the object is as specified by `p`.

Otherwise, if `value` is of length one, it is replicated to form a ragged list with i-th element a vector of length `p[i]`. Although not checked, the intention here is that `value` is from some atomic class. The default for `value` is `NA_real_` to give a convenient way to create a ragged list.

Finally, if none of the above applies, `value` is effectively assumed to be a vector of length `sum(p)`, although other cases are admissible (but I don’t remember if this was intended). In this case, `value` is reshaped into a ragged list to match `p`. This is convenient when, for example, the elements of a ragged array are obtained from an optimisation routine which expects plain vector.
Objects from the Class

Below we describe the "initialize" method that underlies `new("raggedCoef", ...)`. The recommended way to create "raggedCoef" objects is with the function `raggedCoef`, see section Details.

Objects can also be created by calls of the form `new("raggedCoef", v)`, where `v` is a list whose elements are numeric vectors, or `new("raggedCoef", v1, v2, ...)`, where `v1, v2, ...` are numeric vectors. The two forms are equivalent if `v = list(v1, v2, ...)`. The elements of the list `v` may be named. Similarly, named arguments can be used in the second form, say `new("raggedCoef", name1 = v1, name2 = v2, ...)`. In both cases the names are preserved in the internal representation, but not used.

If the arguments are not as specified above the result should be considered undefined. Currently, if there are other arguments after the list `v`, they are ignored with a warning. If the first argument is not a list then all arguments must be numeric and an error is raised if this is not the case. For completeness, we mention that exactly two arguments named `a`, and `p` are also accepted by `new()`, eg `new("raggedCoef", p = c(1, 2), a = list(3, 4:5))`, but these are assigned to the slots without any checking. so it is most flexible (and recommended) to use `raggedCoef()` instead.

Slots

- `a`: Object of class "list" containing the values.
- `p`: Object of class "numeric" containing the lengths of the components of `a`.

Methods

Indexing with "[" treats a `raggedCoef` object as a matrix, while "[[" treats the object as list (it works on slot `a`).

Note that there is a difference between `x[2,]` (or the equivalent `x[2]`) and `x[[2]]`—the former gives a vector of length `max(p)`, so potentially padded with zeroes, while the latter gives the component with its "natural" length.

The replacement variants of "[" and "[[" do not change the structure of the object and issue errors if the replacement value would result in that. In situations where the checks are deemed redundant, direct assignments to the corresponding slots may be used.

```
[ signature(x = "raggedCoef", i = "missing", j = "missing", drop = "ANY"):
[ signature(x = "raggedCoef", i = "missing", j = "numeric", drop = "ANY"):
[ signature(x = "raggedCoef", i = "numeric", j = "missing", drop = "ANY"):
[ signature(x = "raggedCoef", i = "numeric", j = "numeric", drop = "ANY"):

"[" treats a `raggedCoef` object as a matrix with one row for each component and number of columns equal to `max(p)`.

However, `x[2]` is equivalent to `x[2,]` which is different from the treatment of matrix objects in base R.

```
[[ signature(x = "raggedCoef", i = "ANY", j = "missing"):
[[ signature(x = "raggedCoef", i = "ANY", j = "ANY"):

"[[" extracts the corresponding element of slot `a`.

```
[<- signature(x = "raggedCoef", i = "ANY", j = "ANY", value = "numeric"):
```

Replace the `j`-th element of `i`-th row with `value`. All arguments must be scalars.
signature(x = "raggedCoef", i = "ANY", j = "missing", value = "numeric"):
- Replace the i-th row with value. Argument i must be a scalar while the length of value must be the same as that of x@a[[i]]. The methods for "[" and "[[" with this signature coinside.

signature(x = "raggedCoef", i = "ANY", j = "missing", value = "list"):
The elements of value must have the same lengths as the elements they are replacing.

signature(x = "raggedCoef", i = "ANY", j = "missing", value = "matrix"):
- This is essentially the reverse of the corresponding non-replacement operator. value must have at least as many columns as the longest element of x that is replaced.

signature(x = "raggedCoef", i = "ANY", j = "ANY", value = "numeric"):
- ...

signature(x = "raggedCoef", i = "missing", j = "missing", value = "list"):
- ...

signature(x = "raggedCoef", i = "missing", j = "missing", value = "matrix"):
- ...

initialize signature(.Object = "raggedCoef"):
- Creates objects of class raggedCoef. This method is used internally by new(). Users should use new() for creation of objects from this class, see the examples.

show signature(object = "raggedCoef"):
- ...

mixFilter signature(x = "numeric", coef = "raggedCoef", index = "numeric"):
- Apply a mixture filter to a time series.

row_lengths signature(x = "raggedCoef"):
- Gives x@p, which is the same as lengths(x@a).

length signature(x = "raggedCoef"):
- Gives the total number of coefficients (sum(x@p)).

anyNA signature(x = "raggedCoef"):
- Are there NA's in x@a?

dim signature(x = "raggedCoef"):
- The dimension of the object, when viewed as a matrix. The presence of this method also ensures that nrow() and related functions give the expected result.

Note
- Slot p is redundant but convenient.

Author(s)
- Georgi N. Boshnakov

See Also
- class "MixARGaussian"

Examples

ragged1 <- list(1, 2:3, 4:6)
ragged2 <- list(a = 1, b = 2:3, c = 4:6)

raggedCoef(1:3) # only order given, fill with NA's
raggedCoef(1:3, 0) # fill with a number (zero in this case)
## init with a list
raggedCoef(ragged1)
raggedCoef(value = ragged1)

## error, since the shape of ragged1 is not c(2, 2, 3):
## raggedCoef(c(2, 2, 3), value = ragged1)

## init with a flattened list
raggedCoef(p = 1:3, value = 1:6)

## specify each component separately
ragA <- new("raggedCoef", 1, 2:3, 4:6)
ragB <- new("raggedCoef", list(1, 2:3, 4:6)) # same
identical(ragA, ragB) # TRUE

## extract as a matrix
ragA[]

## extract the 2nd component
ragA[2] # c(2, 3, 0) ("[" pads with 0's)
ragA[[2]] # c(2, 3) ("[[" does not pad)

## get the 2nd and 3rd components as a matrix
ragA[2:3, ] # "[" treats object (almost) as matrix
ragA[2:3] # same (though not as for "matrix")

## names are kept in the list but currently not used
ragC <- new("raggedCoef", list(a = 1, b = 2:3, c = 4:6))
ragC1 <- new("raggedCoef", a = 1, b = 2:3, c = 4:6)
identical(ragC, ragC1) # TRUE
names(ragC@a) # [1] "a" "b" "c"

length(ragA)
dim(ragA)
c(nrow(ragA), ncol(ragA))
c(NROW(ragA), NCOL(ragA))

---

raggedCoefS-class

Class "raggedCoefS" — ragged list

### Description

Ragged list used to hold coefficients of MixAR models with seasonal AR parameters.

### Objects from the Class

Objects are created by calls of the form

new("raggedCoefS", a = list(v1, v2, ...), as = list(vs1, vs2, ...), s).

If orders p and ps are specified, a consistency check is made.
Slots

- **a**: Object of class "list" containing AR values. Each element of the list must be "numeric".
- **p**: Object of class "numeric" containing the lengths of components in a. If missing, it is generated based on lengths of elements of a.
- **as**: Object of class "list" containing seasonal AR values. Each element of the list must be "numeric".
- **ps**: Object of class "numeric" containing the lengths of elements of as. If missing, it is generated based on lengths of elements of as.
- **s**: A single element "numeric" vector determining the seasonality in the model (monthly, quarterly, etc.).

Methods

Indexing with "[" treats a raggedCoef object as a matrix (one row for each component), while "[[" treats the object as list (it works on slot a). Specifically, "[[1]"] picks the systematic AR parameters, "[[2]"] picks seasonal AR parameters.

The replacement variants of "[" and "[[" do not change the structure of the object.

Replacement methods only work for subsets x[[i]], x[[i]][[j]], x[[i]][[j]][[k]] for suitable i, j, and k.

- i must be equal to 1 for x@a and 2 for x@as.

```
[ signature(x = "raggedCoefS", i = "missing", j = "missing") : returns the complete matrix of coefficients, one row corresponding to one component, with '0's to match different orders
[ signature(x = "raggedCoefS", i = "missing", j = "missing") :
[ signature(x = "raggedCoefS", i = "numeric", j = "missing") :
[ signature(x = "raggedCoefS", i = "numeric", j = "numeric") : Indexing with "[" treats a raggedCoef object as a matrix with one row for each component and number of columns equal to max(p) + max(ps) in increasing lag. However, x[2] is equivalent to x[2,,] which is different from the treatment of matrix objects in base R.
[ signature(x = "raggedCoefS"), i = "numeric" : if i=1 selects the list of systematic AR parameters; if i=2 selects the list of seasonal AR parameters.
[ signature(x = "raggedCoefS"), i = "numeric", j = "numeric" :
[ signature(x = "raggedCoefS"), i = "numeric", j = "numeric", k = "numeric" : j and k are used to select specific elements from the list of interest.
```

Author(s)

Davide Ravagli

See Also

class "raggedCoef"
Examples

showClass("raggedCoefS")

ragA <- new("raggedCoefS", a = list( c(0.5, -0.5), 1),
    as = list(0, c(0.3, -0.1)), s = 12)
ragB <- new("raggedCoefS", a = list( c(0.5, -0.5), 1), p = c(2, 1),
    as = list(0, c(0.3, -0.1)), ps = c(1, 2), s = 12)  # same

## Elements selection examples

ragA[]  ## matrix of coefficients
ragA[1]; ragA[1,]  ## vector of coefficients from first component
ragA[[2]][[1]]  ## vector of seasonal AR parameters from first component

## Replacement of values in 'raggedCoefS' objects

ragB[[2]] <- list(1, c(-0.5, 0.5))
ragB[[2]][[2]] <- c(20, 22)
ragB[[1]][[1]][1] <- 0

raggedCoefV-class  Class "raggedCoefV" — ragged list

Description

Ragged list used to hold coefficients of MixVAR models.

Objects from the Class

Objects are created by calls of the form new("raggedCoefV", a = list(v1, v2, ...).

Slots

a: Object of class "list" containing AR values. Each element of the list must be "array".

p: Object of class "numeric" containing the length of arrays in a (AR orders). If missing, it is generated based on lengths of elements of a.

Methods

Indexing with "[]" and "[[" works on slot a.

"[]" and "[[" can be use alternatively. Specifically, "[]" and "[[" produce the same result, the complete list of AR coefficients. Similarly, [i,], [i] and [[i]] all return the i'th element of the list, the array for i'th component. [i,j] returns an array with j'th lag autoregressive parameters for each component.
raghat1

Filter a time series with options to shift and scale

Description

Filter a time series with options to shift and scale. This function is used by mixFilter.

Usage

raghat1(filter, x, index, shift = 0, residual = FALSE, scale = 1)
Arguments

- **filter**: The coefficients of the filter, numeric, see Details.
- **x**: time series, numeric.
- **index**: indices for which to compute the filtered values, numeric.
- **shift**: a constant to be added to each filtered element, a number.
- **residual**: if TRUE calculate a ‘residual’, otherwise calculate a ‘hat’ value.
- **scale**: if scale \(!= 1\) calculate scaled residuals by dividing by this value. Probably meaningful only if residual=TRUE.

Details

This function is used by `mixFilter`. Applies an autoregressive filter to a time series for indices specified by index.

Note that ‘filter’ here is equivalent to calculating one-step predictions (or residuals if residual=TRUE) from autoregressions.

`index` should not specify indices smaller than `length(filter)+1` or larger than `length(x)+1`. The value `length(x)+1` can legitimately be used to calculate a prediction (but not a residual of course) for the first value after the end of the series.

Value

A numeric vector of length equal to `length(index)`.

Note

This should probably use `filter` but for the purposes of this package `filter` is usually short and the calculation is vectorised w.r.t. `index`, so should not be terribly slow.

randomArCoefficients  Random initial values for MixAR estimation

Description

Translations of functions from my Mathematica sources. Not used currently?

Usage

randomArCoefficients(ts, wv, pk, pmax, partempl, sub_size = 10, condthr = 10, nattempt = 10, startfrom = pmax + 1)

randomMarParametersKernel(ts, ww, pk, pmax, partempl, ...)

randomMarResiduals(ts, p, partempl)

tsDesignMatrixExtended(ts, p, ind, partempl)
randomArCoefficients

Arguments

- \texttt{ts} time series.
- \texttt{wv,ww} a vector of weights (?).
- \texttt{pk} the AR order of the requested component.
- \texttt{pmax} the maximal AR order in the model. Needed since it cannot be determined by functions working on a single component.
- \texttt{partempl} parameter template, a list containing one element for each mixture component, see Details.
- \texttt{sub\_size} the size of the subsample to use, default is 10.
- \texttt{condthr} threshold for the condition number.
- \texttt{nattempt} if \texttt{condthr} is not reached after \texttt{nattempt} attempts, the function returns the results from the last subset tried.
- \texttt{startfrom} the starting index (in \texttt{ts}) to use for subsampling, default is \texttt{pmax + 1}.
- \texttt{...} arguments to pass on to \texttt{randomArCoefficients()}.  
- \texttt{p} a vector of non-negative integers, the MixAR order.
- \texttt{ind} a vector of positive integers specifying the indices of the observations to use for the “response” variable.

Details

\texttt{randomArCoefficients} tries small subsamples (not necessarily contiguous) from the observations in search of a cluster hopefully belonging to one mixture component and estimates the corresponding shift and AR parameters.

\texttt{randomMarResiduals} selects random parameters for each mixture component and returns the corresponding residuals. \texttt{randomMarParametersKernel} is a helper function which does the computation for one component.

\texttt{tsDesignMatrixExtended} forms the extended design matrix corresponding to a subsample. This is used for least square estimation of the parameters.

Author(s)

Georgi N. Boshnakov

See Also

tomarparambyComp
Description

Determine the lengths of the ‘rows’ of a ragged object.

Methods

Some objects in this package contain (effectively) lists of vectors. These vectors are considered ‘rows’ and this function returns their lengths (as a vector).

signature(x = "ANY") The default method. Applies length to the elements of the argument (2020-03-28: now using lengths(x)).

signature(x = "raggedCoef") Returns the lengths of the rows of the components, a numeric vector.

signature(x = "MixAR") Returns the AR orders of the model components, a numeric vector.

sampZpi

Sampling functions for Bayesian analysis of mixture autoregressive models

Description

Sampling functions for Bayesian analysis of mixture autoregressive models. Draws observations from posterior distributions of the latent variables $Z_t$s and the parameters of mixture autoregressive models.

Usage

sampZpi(y, pk, prob, mu, AR, sigma, nsim, d)
sampMuShift(y, pk, prec, nk, shift, z, AR, nsim)
sampSigmaTau(y, pk, prec, nk, AR, mu, z, a, c, nsim)

Arguments

y 
a time series (currently a numeric vector).

pk 
numeric vector of length g. The autoregressive order of each component.

prob
numeric vector of length g. Current mixing weights.

mu
numeric vector of length g. Current mean parameters.

shift
numeric vector of length g. Current shift parameters.

AR
list of g elements. Autoregressive coefficients for each component.

sigma
numeric vector of length g. Current scale parameters.
**sampZpi**

nsim  
desired sample size.

d  
numeric vector of length g. Hyperparameters for Dirichlet prior. If missing, a vector of 1s is generated. If length is 1, creates a vector of length g with given number.

prec  
numeric vector of length g. Current precision parameters.

nk  
output from sampZpi. Component sum of latent variables Z_t.

z  
output latentZ from sampZpi. A matrix containing the simulated latent variables.

a,c  
hyerparameters.

**Details**

sampZpi draws observations from posterior distributions of the latent variables Z_ts and mixing weights of a Mixture autoregressive model.

sampSigmaTau draws observations from posterior distributions of the precisions tau_k of a Mixture autoregressive model, and obtains scales sigma_k by transformation.

sampMuShift Draws observations from posterior distributions of the means mu_k of a Mixture autoregressive model, and obtains shifts phi_k0 by transformation.

**Value**

for sampZpi, a list containing the following elements:

mix_weights  
matrix with nrow = nsim and ncol = g: sampled mixing weights.

latentZ  
matrix with nrow = n - p and ncol = g, n equal to length(y) and p equal to max(pk: the simulated latent variables Z_t at the last of nsim iterations (functional). Specifically, each row contains 1 for exactly one component, and is filled with 0.

nk  
Vector of length g. Column sums of latentZ.

for sampMuShift, a list containing the following elements:

shift  
matrix with nrow = nsim and ncol = g: simulated shift parameters, obtained by transformation of the means.

mu  
matrix with nrow = nsim and ncol = g: simulated mean parameters.

for sampSigmaTau, a list containing the following elements:

scale  
matrix with nrow = nsim and ncol = g: scale parameters, obtained by transformation of precisions.

precision  
matrix with nrow = nsim and ncol = g: precision parameters.

lambda  
numeric vector of length nsim simulated values of hyperparameter lambda, due to hierarchical setup.

**Author(s)**

Davide Ravagli
Examples

```r
model <- new("MixARGaussian",
  prob = exampleModels$WL_At@prob, # c(0.5, 0.5)
  scale = exampleModels$WL_At@scale, # c(1, 2)
  arcoef = exampleModels$WL_At@arcoef@a ) # list(-0.5, 1.1)

prob <- model@prob
sigma <- model@scale
prec <- 1 / sigma ^ 2
g <- length(model@prob)
d <- rep(1, g)
pk <- model@arcoef@p
p <- max(pk)
shift <- mu <- model@shift

AR <- model@arcoef@a

model

set.seed(1234)
n <- 50 # 500
nsim <- 50

y <- mixAR_sim(model, n = n, init = 0)
x <- sampZpi(y, pk, prob, shift, AR, sigma, nsim = nsim, d)
x1 <- sampMuShift(y, pk, prec, nk = x$nk, shift, z = x$latentZ, AR, nsim = nsim)
x2 <- sampSigmaTau(y, pk, prec, nk = x$nk, AR, mu = x1$mu, z = x$latentZ,
  a = 0.2, c = 2, nsim = nsim)
```

Description

Show differences between two MixAR models in a way that enables quick comparison between them. This is a generic function, package `mixAR` defines methods for MixAR models.

Usage

```r
show_diff(model1, model2)
```

Arguments

- `model1`, `model2`: the MixAR models to be compared.

Details

`show_diff()` is a generic function with dispatch on both arguments. `show_diff()` prints the differences between two models in convenient form for comparison. The methods for MixAR models allow to see differences between similar models at a glance.
simuExperiment

Value

The function is called for the side effect of printing the differences between the two models and has no useful return value.

Methods

signature(model1 = "MixAR", model2 = "MixAR")
signature(model1 = "MixARGaussian", model2 = "MixARgen")
signature(model1 = "MixARgen", model2 = "MixARGaussian")
signature(model1 = "MixARgen", model2 = "MixARgen")

Author(s)

Georgi N. Boshnakov

Examples

## the examples reveal that the models below
differ only in the noise distributions
show_diff(exampleModels$WL_Ct_3, exampleModels$WL_Bt_1)
show_diff(exampleModels$WL_Bt_1, exampleModels$WL_Ct_3)
show_diff(exampleModels$WL_Ct_2, exampleModels$WL_Bt_3)

simuExperiment  Perform simulation experiments

Description

Perform simulation experiments

Usage

simuExperiment(model, simu, est, N = 100, use_true = FALSE,
raw = FALSE, init_name = "init", keep = identity,
summary_fun = .fsummary, ...)

Arguments

model  the model, see 'Details'.
simu  arguments for the simulation function, a list, see 'Details'.
est  arguments for the estimation function, a list, see 'Details'.
N  number of simulations.
use_true  if TRUE, use also the "true" coefficients as initial values, see 'Details'.
raw  if TRUE, include the list of estimated models in the returned value.
**Details**

Argument model specifies the underlying model and is not always needed, see the examples. Argument simu specifies how to simulate the data. Argument est specifies the estimation procedure. Argument N specifies the number of simulation runs. The remaining arguments control details of the simulations, mostly what is returned.

Basically, simuExperiment does N simulation-estimation runs. The keep function is applied to the value obtained from each run. The results from keep are assembled in a list (these are the 'raw' results). Finally, the summary function (argument summary_fun) is applied to the raw list.

simu and est are lists with two elements: fun and args. fun is a function or the name of a function. args is a list of arguments to that function. The first argument of the estimation function, est$fun, is the simulated data. This argument is inserted by simuExperiment and should not be put in est$args.

The value returned by the summary function is the main part of the result. If raw = TRUE, then the raw list is returned, as well. Further fields may be made possible through additional arguments but 'Summary' and 'Raw' are guaranteed to be as described here.

simuExperiment uses init_name only if use_true is TRUE to arrange a call of the estimation function with initial value model. Obviously, simuExperiment does not know how (or if) the estimation function does with its arguments.

The function specified by argument keep is called with one argument when use_true is FALSE and two arguments otherwise.

**Value**

A list with one or more elements, depending on the arguments.

Summary a summary of the experiment, by default sample means and standard deviations of the estimates.

Raw A list of the estimated models.

**Author(s)**

Georgi N. Boshnakov

**Examples**

```r
## explore dist. of the mean of a random sample of length 5.
## (only illustration, such simple cases hardly need simuExperiment)
sim1 <- list(fun="rnorm", args = list(n=5, mean=3, sd = 2))
```
est1 <- list(fun=mean, args = list())

# a basic report function
fsum1 <- function(x) { wrk <- do.call("c",x)
  c(n = length(wrk), mean = mean(wrk), sd = sd(wrk))}

a1 <- simuExperiment(TRUE, simu = sim1, est = est1, N = 1000, summary_fun = fsum1)

# explore also the dist. of the sample s.d.
est2 <- est1
est2$fun <- function(x) {cbar = mean(x), s = sd(x)}
a2 <- simuExperiment(TRUE, simu = sim1, est = est2, N = 1000)

# keep the raw sample means and s.d.'s for further use
a2a <- simuExperiment(TRUE, simu = sim1, est = est2, N = 1000, raw = TRUE)
a2a$Summary

# replicate a2a$Summary
s5 <- sapply(a2a$Raw, identity)
apply(s5, 1, mean)
apply(s5, 1, sd)

hist(s5[1,], prob=TRUE)
lines(density(s5[1,]))
curve(dnorm(x, mean(s5[1,]), sd(s5[1,])), add = TRUE, col = "red")
mixAR:::.fsummary(a2a$Raw)
mixAR:::.fsummary(a2a$Raw, merge = TRUE)

---

stdnormmoment

Compute moments and absolute moments of standardised-t and normal distributions

Description
Compute moments and absolute moments of standardised-t, t and normal distributions.

Usage
stdnormmoment(k)
stdnormabsmoment(k)
stdtmoment(nu, k)
stdtabsmoment(nu, k)
tabsmoment(nu, k)

Arguments

k numeric vector, moments to compute.

nu a number, degrees of freedom.
Details

These functions compute moments of standardised-t and standard normal distributions. These distributions have mean zero and variance 1. Standardised-t is often preferred over Student-t for innovation distributions, since its variance doesn’t depend on its parameter (degrees of freedom). The absolute moments of the usual t-distributions are provided, as well.

The names of the functions start with an abbreviated name of the distribution concerned: \texttt{stdnorm(N(0,1))}, \texttt{stdt(standardised-t)}, \texttt{t(Student-t)}.

The functions with names ending in \texttt{absmoment()} (\texttt{stdnormabsmoment()}, \texttt{stdtabsmoment()} and \texttt{tabsmoment()}) compute absolute moments. The rest (\texttt{stdnormmoment()} and \texttt{stdtmoment()}) compute ordinary moments.

The absolute moments are valid for (at least) \( k \geq 0 \), not necessarily integer. The ordinary moments are currently intended only for integer moments and return \texttt{NaN}'s for fractional ones, with warnings.

Note that the Student-t and standardised-t with \( \nu \) degrees of freedom have finite (absolute) moments only for \( k < \nu \). As a consequence, standardised-t is defined only for \( \nu > 2 \) (otherwise the variance is infinite).

\texttt{stdtabsmoment} returns \texttt{Inf} for any \( k \geq \nu \). \texttt{stdtmoment} returns \texttt{Inf} for even integer \( k \)'s, such that \( k \geq \nu \). However, for odd integers it returns zero and for non-integer moments it returns \texttt{NaN}. Here is an example, where the first two \( k \)'s are smaller than \( \nu \), while the others are not:

\begin{verbatim}
stdtabsmoment(nu = 5, k = c(4, 4.5, 5, 5.5))
##: [1] 9.00000 29.31405 Inf Inf
stdtmoment(nu = 5, k = c(4, 4.5, 5, 5.5))
##: [1] 9 NaN 0 NaN
\end{verbatim}

These functions are designed to work with scalar \( \nu \) but this is not enforced.

Value

numeric vector of the same length as \( k \).

Author(s)

Georgi N. Boshnakov

References


Examples

\begin{verbatim}
## some familiar positive integer moments
stdnormmoment(1:6)
## fractional moments of N(0,1) currently give NaN
stdnormmoment(seq(1, 6, by = 0.5))
## abs moments don't need to be integer
curve(stdnormabsmoment, from = 0, to = 6, type = "l", col = "blue")
\end{verbatim}
## standardised-t

```r
stdtmoment(5, 1:6)
stdtabsmoment(5, 1:6)
stdtabsmoment(5, 1:6)
```

## Student-t

```r
tabsmoment(5, 1:6)
```

---

### tomarparambyComp

Translations of some of my MixAR Mathematica functions. Not sure if these are still used.

#### Usage

```r
tomarparambyComp(params)
tomarparambyType(params)
permuteArpar(params)
```

#### Arguments

- `params` the parameters of the MixAR model, a list, see Details.

#### Details

`tomarparambyComp` is for completeness, my Mathematica programs do not have this currently.

The arrangement of the parameters of MixAR models in package "MixAR" is “by type”: slot `prob` contains the mixture probabilities (weights), `shift` contains intercepts, and so on.

An alternative representation is “by component”: a list whose k-th elements contains all parameters associated with the k-th mixture component. The functions described here use the following order for the parameter of the k-th component: `prob_k`, `shift_k`, `arcoeff_k`, `sigma2_k`.

`tomarparambyType` takes an argument, `params`, arranged “by component” and converts it to “by type”. `tomarparambyComp` does the inverse operation, from “by type” to “by component”.

`permuteArpar` creates all permutations of the components of a MixAR model. It takes a “by component” argument. The autoregressive orders are not permuted, in that if the input model has AR orders `c(2, 1, 3)`, all permuted models are also `c(2, 1, 3)`. The AR coefficients of shorter or longer components are padded with zeroes or truncated, respectively, see the unexported `adjustLengths()`.

#### Value

For `tomarparambyComp`, a list containing the parameters of the model arranged “by component”, see Details.

For `tomarparambyType`, a list containing the parameters of the model arranged “by type”. It contains the following elements.
prob        mixture probabilities, a numeric vector,
shift       shifts, a numeric vector,
arcoef      autoregressive coefficients,
s2          noise variances, a numeric vector.

For permuteArpar, a list with one element (arranged “by type”) for each possible permutation of
the AR parameters.

Author(s)

Georgi N. Boshnakov

See Also

randomArCoefficients

Examples

bycomp <- list(list(0.1, 10, 0.11, 1),
               list(0.2, 20, c(0.11, 0.22), 2),
               list(0.3, 30, c(0.11, 0.22, 0.33), 3))
bytype <- tomarparambyType(bycomp)
identical(bycomp, tomarparambyComp(bytype)) # TRUE

permuteArpar(bycomp)
Index

* MAR model
  mixAR-package, 3
* MixAR
  Choose_pk, 9
  cond_loglik, 10
  exampleModels, 16
  fit_mixAR-methods, 17
  fit_mixARreg-methods, 19
  fnoise, 23
  isStable, 27
  lik_params, 29
  mix_ek, 60
  mix_hatk, 62
  mix_moment, 62
  mix_pdf-methods, 65
  mix_qf-methods, 66
  mix_se-methods, 67
  MixAR-class, 33
  mixAR-methods, 34
  mixAR_BIC, 42
  mixAR_diag, 44
  mixAR_switch, 48
  mixARemFixedPoint, 35
  MixARGAussian-class, 37
  mixFilter, 52
  multiStep_dist-methods, 69
  noise_dist, 72
  parameters, 74
  predict_coef, 79
  randomArCoefficients, 88
  show_diff, 92
  tomarparambyComp, 97
* MixComp
  inner, 26
  mix_ncomp-methods, 65
  MixComp-class, 49
  percent_of, 75
* MixVAR
  MixVAR-class, 56
  mixVAR_sim, 59
  MixVARGaussian-class, 58
* TMAR model
  mixAR-package, 3
* array
  permn_cols, 77
* asymmetry in time series
  mixAR-package, 3
* classes
  MixAR-class, 33
  MixARGAussian-class, 37
  MixARgen-class, 39
  MixComp-class, 49
  MixVAR-class, 56
  MixVARGaussian-class, 58
  raggedCoef-class, 81
  raggedCoefS-class, 84
  raggedCoefV-class, 86
* datagen
  mixAR_sim, 47
* datasets
  PortfolioData1, 78
* distribution
  dist_norm, 11
  mix_pdf-methods, 65
  mix_qf-methods, 66
  stdnormmoment, 95
* em
  em_est_dist, 12
  em_est_sigma, 13
  em_rinit, 14
  mixAR_cond_probs, 44
  mixARemFixedPoint, 35
  mixgenMstep, 53
  mixMstep, 54
* estimation
  fit_mixAR-methods, 17
  marg_loglik, 31
* math
inner, 26
percent_of, 75

* methods
  fit_mixAR-methods, 17
  get_edist-methods, 25
  inner, 26
  lik_params, 29
  make_fcond_lik-methods, 30
  mix_ek, 60
  mix_hatk, 62
  mix_ncomp-methods, 65
  mix_pdf-methods, 65
  mix_qf-methods, 66
  mix_se-methods, 67
  mixAR-methods, 34
  mixFilter, 52
  multiStep_dist-methods, 69
  noise_dist-methods, 73
  noise_params-methods, 73
  noise_rand-methods, 74
  parameters, 74
  percent_of, 75
  row_lengths-methods, 90
  show_diff, 92

* mixture autoregression
  mixAR-package, 3

* mixture autoregressive model
  mixAR-package, 3

* multi-modality in time series
  mixAR-package, 3

* non-gaussian time series
  mixAR-package, 3

* optimize
  mixARemFixedPoint, 35

* package
  mixAR-package, 3

* prediction
  mix_moment, 62
  mix_pdf-methods, 65
  mix_qf-methods, 66
  multiStep_dist-methods, 69
  predict_coef, 79

* ragged
  ragged, 80
  raggedCoef-class, 81
  raggedCoefS-class, 84
  raggedCoefV-class, 86
  raghat1, 87

* regression
  fit_mixARreg-methods, 19

* residuals
  fit_mixARreg-methods, 19

* selection
  marg_loglik, 31

* simulation
  mixARnoise_sim, 41
  mixVAR_sim, 59
  simuExperiment, 93

* ts
  cond_loglik, 10
  mix_pdf-methods, 65
  mix_qf-methods, 66
  mixAR-package, 3
  mixAR_sim, 47
  mixFilter, 52

*,MixComp,MixComp-method
  (MixComp-class), 49

*,MixComp,numeric-method
  (MixComp-class), 49

*,character,MixComp-method
  (MixComp-class), 49

*,function,MixComp-method
  (MixComp-class), 49

*,numeric,MixComp-method
  (MixComp-class), 49

+,MixComp,numeric-method
  (MixComp-class), 49

-,MixComp,missing-method
  (MixComp-class), 49

-,MixComp,numeric-method
  (MixComp-class), 49

-,numeric,MixComp-method
  (MixComp-class), 49

/,MixComp,numeric-method
  (MixComp-class), 49

/,numeric,MixComp-method
  (MixComp-class), 49

[,raggedCoef,missing,missing,ANY-method
  (raggedCoef-class), 81

[,raggedCoef,missing,numeric,ANY-method
  (raggedCoef-class), 81

[,raggedCoef,numerirc,missing,ANY-method
  (raggedCoef-class), 81

[,raggedCoef,numerirc,numeric,ANY-method
(raggedCoef-class), 81
[,raggedCoefS,missing,missing,ANY-method (raggedCoefS-class), 84
[,raggedCoefS,missing,numeric,ANY-method (raggedCoefS-class), 84
[,raggedCoefS,numerics,missing,ANY-method (raggedCoefS-class), 84
[,raggedCoefS,numerics,numeric,ANY-method (raggedCoefS-class), 84
[,raggedCoefV,missing,ANY,ANY-method (raggedCoefV-class), 86
[,raggedCoefV,missing,numeric,ANY-method (raggedCoefV-class), 86
[,raggedCoefV,numeric,ANY,ANY-method (raggedCoefV-class), 86
[,raggedCoefV,numeric,missing,ANY-method (raggedCoefV-class), 86
[,raggedCoefV,numeric,numeric,ANY-method (raggedCoefV-class), 86
[-methods (raggedCoef-class), 81
<-,raggedCoef,ANY,ANY,numeric-method (raggedCoef-class), 81
<-,raggedCoef,ANY,missing,list-method (raggedCoef-class), 81
<-,raggedCoef,ANY,missing,matrix-method (raggedCoef-class), 81
<-,raggedCoef,ANY,missing,numeric-method (raggedCoef-class), 81
<-,raggedCoef,missing,missing,ANY-method (raggedCoef-class), 81
<-,raggedCoef,missing,missing,list-method (raggedCoef-class), 81
<-,raggedCoef,missing,missing,matrix-method (raggedCoef-class), 81
-<-,raggedCoef,missing,missing,numeric-method (raggedCoef-class), 81
[-methods (raggedCoef-class), 81
%of%(percent_of), 75
%of%,ANY,ANY-method (percent_of), 75
%of%,character,MixComp-method (percent_of), 75
%of%,function,MixComp-method (percent_of), 75
%of%,list,MixComp-method (percent_of), 75
anynA,raggedCoef-method (raggedCoef-class), 81
b_show(fnoise), 23
bayes_mixAR,5,7,28
BIC_comp(mixAR_BIC),42
bx_dx,10
Choose_pk,9
cond_loglik,10
cond_loglikS(cond_loglik),10
dim,MixComp-method (MixComp-class), 49
dim,raggedCoef-method (raggedCoef-class), 81
dist_norm,11
distlist(fnoise),23
ed_nparam(fnoise),23
ed_parse(fnoise),23
ed_skeleton (fnoise), 23
ed_src (fnoise), 23
ed_stdnorm (fnoise), 23
ed_stdt (fnoise), 23
ed_stdt0 (fnoise), 23
ed_stdt1 (fnoise), 23
em_est_dist, 12
em_est_sigma, 13
em_rinit, 14
em_tau, 14
est_temp1, 15
etk2tau (em_rinit), 14
exampleModels, 16
fdist_stdnorm, 12, 73
fdist_stdt (fnoise), 23
ft_stdt (fnoise), 23
fit_mixAR, 5, 21, 37, 54
fit_mixAR (fit_mixAR-methods), 17
fit_mixAR, ANY, ANY, ANY-method
  (fit_mixAR-methods), 17
fit_mixAR, ANY, MixAR, list-method
  (fit_mixAR-methods), 17
fit_mixAR, ANY, MixAR, missing-method
  (fit_mixAR-methods), 17
fit_mixAR, ANY, MixAR, MixAR-method
  (fit_mixAR-methods), 17
fit_mixAR, ANY, MixAR, numeric-method
  (fit_mixAR-methods), 17
fit_mixAR, ANY, MixARGaussian, MixAR-method
  (fit_mixAR-methods), 17
fit_mixAR, ANY, numeric, missing-method
  (fit_mixAR-methods), 17
fit_mixAR, ANY, numeric, numeric-method
  (fit_mixAR-methods), 17
fit_mixAR-methods, 17
fit_mixARreg, 5
fit_mixARreg (fit_mixARreg-methods), 19
fit_mixARreg, ANY, ANY, missing, list-method
  (fit_mixARreg-methods), 19
fit_mixARreg, ANY, ANY, MixAR, list-method
  (fit_mixARreg-methods), 19
fit_mixARreg, ANY, data.frame, missing, list-method
  (fit_mixARreg-methods), 19
fit_mixARreg, ANY, data.frame, MixAR, missing-method
  (fit_mixARreg-methods), 19
fit_mixARreg, ANY, matrix, missing, list-method
  (fit_mixARreg-methods), 19
fit_mixARreg, ANY, matrix, MixAR, missing-method
  (fit_mixARreg-methods), 19
fit_mixARreg, ANY, numeric, MixAR, missing-method
  (fit_mixARreg-methods), 19
fit_mixARreg, ANY, numeric, missing, list-method
  (fit_mixARreg-methods), 19
fit_mixARreg, ANY, numeric, MixAR, missing-method
  (fit_mixARreg-methods), 19
fit_mixARreg, ANY, numeric, MixAR, missing-method
  (fit_mixARreg-methods), 19
fit_mixVAR (fit_mixVAR-methods), 22
fit_mixVAR, ANY, ANY-method
  (fit_mixVAR-methods), 22
fit_mixVAR, ANY, MixAR-method
  (fit_mixVAR-methods), 22
fit_mixVAR-methods, 22
fn_stdt, 73
fn_stdt (fnoise), 23
fnoise, 23
ft_stdt (fnoise), 23
get_edist (noise_dist), 72
get_edist, MixAR-method
  (get_edist-methods), 25
get_edist, MixARGaussian-method
  (get_edist-methods), 25
get_edist-methods, 25
inner, 26
inner, ANY, ANY, ANY, ANY-method
  (inner), 26
inner, MixComp, missing, missing, missing-method
  (inner), 26
inner, MixComp, numeric, ANY, ANY-method
  (inner), 26
inner, MixComp, numeric, ANY, missing-method
  (inner), 26
inner, MixComp, numeric, missing, missing-method
  (inner), 26
inner, numeric, MixComp, missing, missing-method
  (inner), 26
inner-methods (inner), 26
isStable, 27
label_switch, 28
length, raggedCoef-method
  (raggedCoef-class), 81
lik_params, 29, 31
lik_params, MixAR-method (lik_params), 29
lik_params, MixARGen-method
  (lik_params), 29
INDEX

lik_params-methods (lik_params), 29
make_fcond_lkl
(make_fcond_lkl-methods), 30
make_fcond_lkl,MixAR,numeric-method
(make_fcond_lkl-methods), 30
make_fcond_lkl-methods, 30
marg_loglik, 31
mix_cdf, 5, 31, 63, 67
mix_cdf (mix_pdf-methods), 65
mix_cdf,MixARGaussian,missing,missing,numeric-method
(mix_pdf-methods), 65
mix_cdf,MixARGaussian,numeric,missing,numeric-method
(mix_pdf-methods), 65
mix_cdf,MixARGaussian,numeric,numeric,missing-method
(mix_pdf-methods), 65
mix_cdf,MixARGaussian,numeric,numeric,numeric-method
(mix_pdf-methods), 65
mix_pdf-methods, 65
mix_qf, 5, 63, 66
mix_qf (mix_qf-methods), 66
mix_qf,MixARGaussian,missing,missing,missing,numeric-method
(mix_qf-methods), 66
mix_qf,MixARGaussian,numeric,missing,missing,numeric-method
(mix_qf-methods), 66
mix_qf,MixARGaussian,numeric,numeric,numeric,numeric-method
(mix_qf-methods), 66
mix_qf-methods, 66
mix_se (mix_se-methods), 67
mix_se,ANY,numeric-method
(mix_se-methods), 67
mix_se,ANY,MixAR-method
(mix_se-methods), 67
mix_variance, 5
mix_variance (mix_variance-methods), 62
mixAny_sim (MixAR_sim), 47
MixAR, 38–40, 58, 61, 62
mixAR, 34
mixAR (mixAR-methods), 34
mixAR,MixAR-method (mixAR-methods), 34
mixAR-class, 33
mixAR-methods, 34
mixAR-package, 3
mixAR_BIC, 42, 46
mixAR_cond_probs, 44
mixAR_diag, 44
mixAR_permute (mixAR_switch), 48
mixAR_sim, 42, 47, 60
mixAR_switch, 48
mixARGaussian, 33–35, 37, 83
mixARGaussian (MixARGaussian-class), 37
mixARGaussian-class, 37
mixARGen, 25, 34, 35, 37, 38
mixARGen (MixARGen-class), 39
mixARGen-class, 39
mixARGenemFixedPoint, 54
mixARnoise_sim, 41
mixARreg (fit_mixARreg-methods), 19
MixComp, 26, 27, 49, 76
MixComp-class, 49
mixFilter, 52, 61
mixFilter,ANY,ANY,ANY-method (mixFilter), 52
mixFilter,numerical,raggedCoef,numerical-method (mixFilter), 52
mixFilter-methods (mixFilter), 52
mixgenMstep, 53
mixMstep, 54
mixSARfit, 5, 55
MixVAR, 87
MixVAR-class, 56
mixVAR_sim, 59
mixVARfit, 22, 57
MixVARGaussian, 56, 57
MixVARGaussian (MixVARGaussian-class), 58
MixVARGaussian-class, 58
mo_T_A (exampleModels), 16
mo_T_B (exampleModels), 16
mo_T_B2 (exampleModels), 16
mo_T_B3 (exampleModels), 16
mo_T_C1 (exampleModels), 16
mo_T_C2 (exampleModels), 16
mo_T_C3 (exampleModels), 16
moWL (exampleModels), 16
moWL_A (exampleModels), 16
moWL_B (exampleModels), 16
moWL_I (exampleModels), 16
moWL_II (exampleModels), 16
moWLar (exampleModels), 16
moWLgen (exampleModels), 16
moWLprob (exampleModels), 16
moWLSigma (exampleModels), 16
moWLT3v (exampleModels), 16
moWLTf (exampleModels), 16
multiStep_dist, 5, 79, 80
multiStep_dist (multiStep_dist-methods), 69
multiStep_dist,MixAR,numeric,numeric,numeric-method (multiStep_dist-methods), 69
multiStep_dist,MixARGaussian,numeric,missing,ANY-method (multiStep_dist-methods), 69
multiStep_dist,MixARGaussian,numeric,missing,missing-method (multiStep_dist-methods), 69
multiStep_dist-methods, 69
noise_dist, 72, 73
noise_dist,MixAR-method (noise_dist-methods), 73
noise_dist,MixARGaussian-method (noise_dist-methods), 73
noise_dist,MixARGen-method (noise_dist-methods), 73
noise_dist-methods, 73
noise_params (noise_dist), 72
noise_params,MixAR-method (noise_params-methods), 73
noise_params,MixARGaussian-method (noise_params-methods), 73
noise_params,MixARGen-method (noise_params-methods), 73
noise_params-methods, 73
noise_rand (noise_dist), 72
noise_rand,MixAR-method (noise_rand-methods), 74
noise_rand,MixARGaussian-method (noise_rand-methods), 74
noise_rand,MixARGen-method (noise_rand-methods), 74
noise_rand-methods, 74
parameters, 74
parameters,ANY-method (parameters), 74
parameters,MixAR-method (parameters), 74
parameters-methods (parameters), 74
parameters<-(parameters), 74
parameters<-,ANY-method (parameters), 74
parameters<-,MixAR-method (parameters), 74
parameters<--methods (parameters), 74
percent_of, 75
perm_cols, 77
permuteArpar (tomarparambyComp), 97
PortfolioData1, 78
predict_coef, 71, 79

rag_modif (ragged), 80
ragged, 80
ragged2vec (ragged), 80
raggedCoef, 37, 85
raggedCoef (raggedCoef-class), 81
raggedCoef-class, 81
raggedCoefS (raggedCoefS-class), 84
raggedCoefS-class, 84
raggedCoefV, 58
raggedCoefV (raggedCoefV-class), 86
raggedCoefV-class, 86
raghat1, 52, 87
randomArCoefficients, 74, 88, 98
randomMarParametersKernel
  (randomArCoefficients), 88
randomMarResiduals
  (randomArCoefficients), 88
row_lengths (row_lengths-methods), 90
row_lengths, ANY-method
  (row_lengths-methods), 90
row_lengths, MixAR-method
  (row_lengths-methods), 90
row_lengths, raggedCoef-method
  (row_lengths-methods), 90
row_lengths-methods, 90

sampMuShift (sampZpi), 90
sampSigmaTau (sampZpi), 90
sampZpi, 90
set_noise_params (noise_dist), 72
set_parameters (parameters), 74
set_parameters, ANY-method (parameters), 74
set_parameters, MixAR-method
  (parameters), 74
set_parameters-methods (parameters), 74
show_diff, 92
show_diff, MixAR, MixAR-method
  (show_diff), 92
show_diff, MixARGaussian, MixARgen-method
  (show_diff), 92
show_diff, MixARgen, MixARGaussian-method
  (show_diff), 92

stdnormabsmoment (stdnormmoment), 95
stdnormmoment, 95
stdtabsmoment (stdnormmoment), 95
stdtmoment (stdnormmoment), 95
tabsmoment (stdnormmoment), 95
tau2arcoef (mixMstep), 54
tauCorrelate (mixMstep), 54
tauetk2sigmahat (em_est_sigma), 13
tomarparambyComp, 89, 97
tomarparambyType (tomarparambyComp), 97
tsDesignMatrixExtended
  (randomArCoefficients), 88
tsdiag, 46
tsdia (mixAR_diag), 44