Package 'LAM'

January 20, 2025

Type Package

Title Some Latent Variable Models

Version 0.7-22

Date 2024-07-15 11:56:28

Author Alexander Robitzsch [aut,cre]

Maintainer Alexander Robitzsch `<robitzsch@ipn.uni-kiel.de>`

Description Includes some procedures for latent variable modeling with a particular focus on multilevel data.
The 'LAM' package contains mean and covariance structure modelling for multivariate normally distributed data (mlnormal(); Longford, 1987; <doi:10.1093/biomet/74.4.817>), a general Metropolis-Hastings algorithm (amh(); Roberts & Rosenthal, 2001, <doi:10.1214/ss/1015346320>) and penalized maximum likelihood estimation (pmle(); Cole, Chu & Greenland, 2014; <doi:10.1093/aje/kwt245>).

Depends R (>= 3.1)

Imports CDM, graphics, Rcpp, sirt, stats, utils

Suggests coda, expm, MASS, numDeriv, TAM

Enhances lavaan, lme4

LinkingTo Rcpp, RcppArmadillo

URL `https://github.com/alexanderrobitzsch/LAM`,
`https://sites.google.com/site/alexanderrobitzsch2/software`

NeedsCompilation yes

Repository CRAN

Date/Publication 2024-07-15 10:30:02 UTC

Contents

`LAM-package` *Some Latent Variable Models*

Description

Includes some procedures for latent variable modeling with a particular focus on multilevel data. The 'LAM' package contains mean and covariance structure modelling for multivariate normally distributed data (mlnormal(); Longford, 1987; <doi:10.1093/biomet/74.4.817>), a general Metropolis-Hastings algorithm (amh(); Roberts & Rosenthal, 2001, <doi:10.1214/ss/1015346320>) and penalized maximum likelihood estimation (pmle(); Cole, Chu & Greenland, 2014; <doi:10.1093/aje/kwt245>).

Details

The **LAM** package contains the following main functions:

- A general fitting method for mean and covariance structure for multivariate normally distributed data is the `mlnormal` function. Prior distributions or regularization methods (lasso penalties) are also accommodated. Missing values on dependent variables can be treated by applying the full information maximum likelihood method implemented in this function.
- A general (but experimental) Metropolis-Hastings sampler for Bayesian analysis based on MCMC is implemented in the `amh` function. Deterministic optimization of the posterior distribution (maximum posterior estimation or penalized maximum likelihood estimation) can be conduction with the `pmle` function which is based on `stats::optim`.

Author(s)

Alexander Robitzsch [aut,cre]

Maintainer: Alexander Robitzsch <robitzsch@ipn.uni-kiel.de>

References

Cole, S. R., Chu, H., & Greenland, S. (2013). Maximum likelihood, profile likelihood, and penalized likelihood: a primer. *American Journal of Epidemiology, 179*(2), 252-260. doi:10.1093/aje/kwt245

Longford, N. T. (1987). A fast scoring algorithm for maximum likelihood estimation in unbalanced mixed models with nested random effects. *Biometrika, 74*(4), 817-827. doi:10.1093/biomet/74.4.817

Roberts, G. O., & Rosenthal, J. S. (2001). Optimal scaling for various Metropolis-Hastings algorithms. *Statistical Science, 16*(4), 351-367. doi:10.1214/ss/1015346320

Examples

```
##  > library(LAM)
##  ## LAM 0.0-4 (2017-03-03 16:53:46)
##
##   __         ______     __    __
##  /\ \       /\  __ \   /\ "-./  \
##  \ \ \____  \ \  __ \  \ \ \-./\ \
##   \ \_____\  \ \_\ \_\  \ \_\ \ \_\
##    \/_____/   \/_/\/_/   \/_/  \/_/
##
```

`amh`

Bayesian Model Estimation with Adaptive Metropolis Hastings Sampling (`amh`) or Penalized Maximum Likelihood Estimation (`pmle`)

Description

The function `amh` conducts a Bayesian statistical analysis using the adaptive Metropolis-Hastings as the estimation procedure (Hoff, 2009; Roberts & Rosenthal, 2001). Only univariate prior distributions are allowed. Note that this function is intended just for experimental purpose, not to replace general purpose packages like **WinBUGS**, **JAGS**, **Stan** or **MHadaptive**.

The function `pmle` optimizes the penalized likelihood (Cole, Chu & Greenland, 2014) which means that the posterior is maximized and the maximum a posterior estimate is obtained. The optimization functions stats::optim or stats::nlminb can be used.

Usage

```
amh(data, nobs, pars, model,  prior, proposal_sd,  pars_lower=NULL,
      pars_upper=NULL, derivedPars=NULL, n.iter=5000, n.burnin=1000,
      n.sims=3000, acceptance_bounds=c(.45,.55), proposal_refresh=50,
      proposal_equal=4, print_iter=50, boundary_ignore=FALSE )

pmle( data, nobs, pars, model,  prior=NULL, model_grad=NULL, pars_lower=NULL,
    pars_upper=NULL, method="L-BFGS-B", control=list(), verbose=TRUE, hessian=TRUE,
      optim_fct="nlminb", h=1e-4, ... )

## S3 method for class 'amh'
summary(object, digits=3, file=NULL, ...)

## S3 method for class 'amh'
plot(x, conflevel=.95, digits=3, lag.max=.1,
    col.smooth="red", lwd.smooth=2, col.split="blue", lwd.split=2,
    lty.split=1, col.ci="orange", cex.summ=1, ask=FALSE, ... )

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

```
## S3 method for class 'amh'
logLik(object, ...)

## S3 method for class 'amh'
vcov(object, ...)

## S3 method for class 'amh'
confint(object, parm, level=.95, ... )

## S3 method for class 'pmle'
summary(object, digits=3, file=NULL, ...)

## S3 method for class 'pmle'
coef(object, ...)

## S3 method for class 'pmle'
logLik(object, ...)

## S3 method for class 'pmle'
vcov(object, ...)

## S3 method for class 'pmle'
confint(object, parm, level=.95, ... )
```

Arguments

`data`	Object which contains data
`nobs`	Number of observations
`pars`	Named vector of initial values for parameters
`model`	Function defining the log-likelihood of the model
`prior`	List with prior distributions for the parameters to be sampled (see Examples). See `sirt::prior_model_parse` for more convenient specifications of the prior distributions. Setting the prior argument to `NULL` corresponds to improper (constant) prior distributions for all parameters.
`proposal_sd`	Vector with initial standard deviations for proposal distribution
`pars_lower`	Vector with lower bounds for parameters
`pars_upper`	Vector with upper bounds for parameters
`derivedPars`	Optional list containing derived parameters from sampled chain
`n.iter`	Number of iterations
`n.burnin`	Number of burn-in iterations
`n.sims`	Number of sampled iterations for parameters
`acceptance_bounds`	Bounds for acceptance probabilities of sampled parameters

`proposal_refresh`	Number of iterations for computation of adaptation of proposal standard deviation
`proposal_equal`	Number of intervals in which the proposal SD should be constant for fixing the SD
`print_iter`	Display progress every `print_iter`th iteration
`boundary_ignore`	Logical indicating whether sampled values outside the specified boundaries should be ignored.
`model_grad`	Optional function which evaluates the gradient of the log-likelihood function (must be a function of `pars`.
`method`	Optimization method in `stats::optim`
`control`	Control parameters `stats::optim`
`verbose`	Logical indicating whether progress should be displayed.
`hessian`	Logical indicating whether the Hessian matrix should be computed
`optim_fct`	Type of optimization: `"optim"` (stats::optim) or the default `"nlminb"` (stats::nlminb)
`h`	Numerical differentiation parameter for prior distributions if `model_grad` is provided.
`object`	Object of class `amh`
`digits`	Number of digits used for rounding
`file`	File name
`...`	Further arguments to be passed
`x`	Object of class `amh`
`conflevel`	Confidence level
`lag.max`	Percentage of iterations used for calculation of autocorrelation function
`col.smooth`	Color moving average
`lwd.smooth`	Line thickness moving average
`col.split`	Color split chain
`lwd.split`	Line thickness splitted chain
`lty.split`	Line type splitted chain
`col.ci`	Color confidence interval
`cex.summ`	Point size summary
`ask`	Logical. If `TRUE` the user is asked for input, before a new figure is drawn.
`parm`	Optional vector of parameters.
`level`	Confidence level.

Value

List of class `amh` including entries

`pars_chain`	Data frame with sampled parameters
`acceptance_parameters`	Acceptance probabilities
`amh_summary`	Summary of parameters
`coef`	Coefficient obtained from marginal MAP estimation
`pmle_pars`	Object of parameters and posterior values corresponding to multivariate maximum of posterior distribution.
`comp_estimators`	Estimates for univariate MAP, multivariate MAP and mean estimator and corresponding posterior estimates.
`ic`	Information criteria
`mcmcobj`	Object of class `mcmc` for **coda** package
`proposal_sd`	Used proposal standard deviations
`proposal_sd_history`	History of proposal standard deviations during burn-in iterations
`acceptance_rates_history`	History of acceptance rates for all parameters during burn-in phase
`...`	More values

References

Cole, S. R., Chu, H., & Greenland, S. (2013). Maximum likelihood, profile likelihood, and penalized likelihood: a primer. *American Journal of Epidemiology, 179*(2), 252-260. doi:10.1093/aje/kwt245

Hoff, P. D. (2009). *A first course in Bayesian statistical methods*. New York: Springer.

Roberts, G. O., & Rosenthal, J. S. (2001). Optimal scaling for various Metropolis-Hastings algorithms. *Statistical Science, 16*(4), 351-367. doi:10.1214/ss/1015346320

See Also

See the Bayesian CRAN Task View for lot of information about alternative R packages.

`sirt::prior_model_parse`

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: Constrained multivariate normal distribution
#############################################################################
```

```
#--- simulate data
Sigma <- matrix( c(
    1, .55, .5,
    .55, 1, .45,
    .5, .45, 1 ), nrow=3, ncol=3, byrow=TRUE )
mu <- c(0,1,1.2)
N <- 400
set.seed(9875)
dat <- MASS::mvrnorm( N, mu, Sigma )
colnames(dat) <- paste0("Y",1:3)
S <- stats::cov(dat)
M <- colMeans(dat)

#-- define maximum likelihood function for normal distribution
fit_ml <- function( S, Sigma, M, mu, n, log=TRUE){
    Sigma1 <- solve(Sigma)
    p <- ncol(Sigma)
    det_Sigma <- det( Sigma )
    eps <- 1E-30
    if ( det_Sigma < eps ){
            det_Sigma <- eps
    }
    l1 <- - p * log( 2*pi ) - t( M - mu ) %*% Sigma1 %*% ( M - mu ) -
                  log( det_Sigma )  - sum( diag( Sigma1 %*% S ) )
    l1 <- n/2 * l1
    if (! log){
        l1 <- exp(l1)
    }
    l1 <- l1[1,1]
    return(l1)
}
# This likelihood function can be directly accessed by the loglike_mvnorm function.

#--- define data input
data <- list( "S"=S, "M"=M, "n"=N )

#--- define list of prior distributions
prior <- list()
prior[["mu1"]] <- list( "dnorm", list( x=NA, mean=0, sd=1 ) )
prior[["mu2"]] <- list( "dnorm", list( x=NA, mean=0, sd=5 ) )
prior[["sig1"]] <- list( "dunif", list( x=NA, 0, 10 ) )
prior[["rho"]] <- list( "dunif", list( x=NA,-1, 1  ) )

#** alternatively, one can specify the prior as a string and uses
#   the 'prior_model_parse' function
prior_model2 <- "
   mu1 ~ dnorm(x=NA, mean=0, sd=1)
   mu2 ~ dnorm(x=NA, mean=0, sd=5)
   sig1 ~ dunif(x=NA, 0,10)
   rho ~ dunif(x=NA,-1,1)
   "
# convert string
prior2 <- sirt::prior_model_parse( prior_model2 )
```

```
prior2  # should be equal to prior

#--- define log likelihood function for model to be fitted
model <- function( pars, data ){
    # mean vector
    mu <- pars[ c("mu1", rep("mu2",2) ) ]
    # covariance matrix
    m1 <- matrix( pars["rho"] * pars["sig1"]^2, 3, 3 )
    diag(m1) <- rep( pars["sig1"]^2, 3 )
    Sigma <- m1
    # evaluate log-likelihood
    ll <- fit_ml( S=data$S, Sigma=Sigma, M=data$M, mu=mu, n=data$n)
    return(ll)
}

#--- initial parameter values
pars <- c(1,2,2,0)
names(pars) <- c("mu1", "mu2", "sig1", "rho")
#--- initial proposal distributions
proposal_sd <- c( .4, .1, .05, .1 )
names(proposal_sd) <- names(pars)
#--- lower and upper bound for parameters
pars_lower <- c( -10, -10, .001, -.999 )
pars_upper <- c( 10, 10, 1E100, .999 )

#--- define list with derived parameters
derivedPars <- list( "var1"=~ I( sig1^2 ), "d1"=~ I( ( mu2 - mu1 ) / sig1 ) )

#*** start Metropolis-Hastings sampling
mod <- LAM::amh( data, nobs=data$n, pars=pars, model=model,
          prior=prior, proposal_sd=proposal_sd,
          n.iter=1000, n.burnin=300, derivedPars=derivedPars,
          pars_lower=pars_lower, pars_upper=pars_upper )

# some S3 methods
summary(mod)
plot(mod, ask=TRUE)
coef(mod)
vcov(mod)
logLik(mod)

#--- compare Bayesian credibility intervals and HPD intervals
ci <- cbind( confint(mod), coda::HPDinterval(mod$mcmcobj)[-1, ] )
ci
# interval lengths
cbind( ci[,2]-ci[,1], ci[,4] - ci[,3] )

#--- plot update history of proposal standard deviations
graphics::matplot( x=rownames(mod$proposal_sd_history),
          y=mod$proposal_sd_history, type="o", pch=1:6)

#**** compare results with lavaan package
library(lavaan)
```

```
lavmodel <- "
    F=~ 1*Y1 + 1*Y2 + 1*Y3
    F ~~ rho*F
    Y1 ~~ v1*Y1
    Y2 ~~ v1*Y2
    Y3 ~~ v1*Y3
    Y1 ~ mu1 * 1
    Y2 ~ mu2 * 1
    Y3 ~ mu2 * 1
    # total standard deviation
    sig1 :=sqrt( rho + v1 )
    "
# estimate model
mod2 <- lavaan::sem( data=as.data.frame(dat), lavmodel )
summary(mod2)
logLik(mod2)

#*** compare results with penalized maximum likelihood estimation
mod3 <- LAM::pmle( data=data, nobs=data$n, pars=pars, model=model,  prior=prior,
            pars_lower=pars_lower, pars_upper=pars_upper, verbose=TRUE  )
# model summaries
summary(mod3)
confint(mod3)
vcov(mod3)

#*** penalized likelihood estimation with provided gradient of log-likelihood

library(CDM)
fct <- function(x){
    model(pars=x, data=data )
}
# use numerical gradient (just for illustration)
grad <- function(pars){
    CDM::numerical_Hessian(par=pars, FUN=fct, gradient=TRUE, hessian=FALSE)
}
#- estimate model
mod3b <- LAM::pmle( data=data, nobs=data$n, pars=pars, model=model,  prior=prior, model_grad=grad,
            pars_lower=pars_lower, pars_upper=pars_upper, verbose=TRUE  )
summary(mod3b)

#--- lavaan with covariance and mean vector input
mod2a <- lavaan::sem( sample.cov=data$S, sample.mean=data$M, sample.nobs=data$n,
                model=lavmodel )
coef(mod2)
coef(mod2a)

#--- fit covariance and mean structure by fitting a transformed
#    covariance structure
#* create an expanded covariance matrix
p <- ncol(S)
S1 <- matrix( NA, nrow=p+1, ncol=p+1 )
S1[1:p,1:p] <- S + outer( M, M )
S1[p+1,1:p] <- S1[1:p, p+1] <- M
```

```
S1[p+1,p+1] <- 1
vars <- c( colnames(S), "MY" )
rownames(S1) <- colnames(S1) <- vars
#* lavaan model
lavmodel <- "
    # indicators
    F=~ 1*Y1 + 1*Y2 + 1*Y3
    # pseudo-indicator representing mean structure
    FM=~ 1*MY
    MY ~~ 0*MY
    FM ~~ 1*FM
    F ~~ 0*FM
    # mean structure
    FM=~ mu1*Y1 + mu2*Y2 + mu2*Y3
    # variance structure
    F ~~ rho*F
    Y1 ~~ v1*Y1
    Y2 ~~ v1*Y2
    Y3 ~~ v1*Y3
    sig1 :=sqrt( rho + v1 )
    "

# estimate model
mod2b <- lavaan::sem( sample.cov=S1,sample.nobs=data$n,
                model=lavmodel )
summary(mod2b)
summary(mod2)

#############################################################################
# EXAMPLE 2: Estimation of a linear model with Box-Cox transformation of response
#############################################################################

#*** simulate data with Box-Cox transformation
set.seed(875)
N <- 1000
b0 <- 1.5
b1 <- .3
sigma <- .5
lambda <- 0.3
# apply inverse Box-Cox transformation
  # yl=( y^lambda - 1 ) / lambda
  # -> y=( lambda * yl + 1 )^(1/lambda)
x <- stats::rnorm( N,  mean=0, sd=1 )
yl <- stats::rnorm( N, mean=b0, sd=sigma ) + b1*x
# truncate at zero
eps <- .01
yl <- ifelse( yl < eps, eps, yl )
y <- ( lambda * yl + 1 ) ^(1/lambda )

#-- display distributions of transformed and untransformed data
   graphics::par(mfrow=c(1,2))
graphics::hist(yl, breaks=20)
graphics::hist(y, breaks=20)
```

```
    graphics::par(mfrow=c(1,1))

#*** define vector of parameters
pars <- c( 0, 0,  1, -.2 )
names(pars) <- c("b0", "b1", "sigma", "lambda" )
#*** input data
data <- list( "y"=y, "x"=x)
#*** define model with log-likelihood function
model <- function( pars, data ){
    sigma <- pars["sigma"]
    b0 <- pars["b0"]
    b1 <- pars["b1"]
    lambda <- pars["lambda"]
    if ( abs(lambda) < .01){ lambda <- .01 * sign(lambda) }
    y <- data$y
    x <- data$x
    n <- length(y)
    y_lambda <- ( y^lambda - 1 ) / lambda
    ll <- - n/2 * log(2*pi) - n * log( sigma ) -
            1/(2*sigma^2)* sum( (y_lambda - b0 - b1*x)^2 ) +
            ( lambda - 1 ) * sum( log( y ) )
    return(ll)
}
#-- test model function
model( pars, data )

#*** define prior distributions
prior <- list()
prior[["b0"]] <- list( "dnorm", list( x=NA, mean=0, sd=10 ) )
prior[["b1"]] <- list( "dnorm", list( x=NA, mean=0, sd=10 ) )
prior[["sigma"]] <- list( "dunif", list( x=NA, 0, 10  ) )
prior[["lambda"]] <- list( "dunif", list( x=NA, -2, 2 ) )
#*** define proposal SDs
proposal_sd <- c( .1, .1, .1, .1 )
names(proposal_sd) <- names(pars)
#*** define bounds for parameters
pars_lower <- c( -100, -100, .01, -2 )
pars_upper <- c( 100, 100, 100, 2 )

#*** sampling routine
mod <- LAM::amh( data, nobs=N, pars, model,  prior, proposal_sd,
        n.iter=10000, n.burnin=2000, n.sims=5000,
        pars_lower=pars_lower, pars_upper=pars_upper )
#-- S3 methods
summary(mod)
plot(mod, ask=TRUE )

#*** estimating Box-Cox transformation in MASS package
library(MASS)
mod2 <- MASS::boxcox( stats::lm( y ~ x ), lambda=seq(-1,2,length=100) )
mod2$x[ which.max( mod2$y ) ]

#*** estimate Box-Cox parameter lambda with car package
```

```
library(car)
mod3 <- car::powerTransform( y ~ x )
summary(mod3)
# fit linear model with transformed response
mod3a <- stats::lm( car::bcPower( y, mod3$roundlam) ~ x )
summary(mod3a)

#############################################################################
# EXAMPLE 3: STARTS model directly specified in LAM or lavaan
#############################################################################

## Data from Wu (2016)

library(LAM)
library(sirt)
library(STARTS)

## define list with input data
## S ... covariance matrix, M ... mean vector

# read covariance matrix of data in Wu (older cohort, positive affect)
S <- matrix( c( 12.745, 7.046, 6.906, 6.070, 5.047, 6.110,
    7.046, 14.977, 8.334, 6.714, 6.91, 6.624,
    6.906, 8.334, 13.323, 7.979, 8.418, 7.951,
    6.070, 6.714, 7.979, 12.041, 7.874, 8.099,
    5.047, 6.91, 8.418, 7.874, 13.838, 9.117,
    6.110, 6.624, 7.951, 8.099, 9.117, 15.132 ),
    nrow=6, ncol=6, byrow=TRUE )
#* standardize S such that the average SD is 1 (for ease of interpretation)
M_SD <- mean( sqrt( diag(S) ))
S <- S / M_SD^2
colnames(S) <- rownames(S) <- paste0("W",1:6)
W <- 6   # number of measurement waves
data <- list( "S"=S, "M"=rep(0,W), "n"=660, "W"=W  )

#*** likelihood function for the STARTS model
model <- function( pars, data ){
    # mean vector
    mu <- data$M
    # covariance matrix
    W <- data$W
    var_trait <- pars["vt"]
    var_ar <- pars["va"]
    var_state <- pars["vs"]
    a <- pars["b"]
    Sigma <- STARTS::starts_uni_cov( W=W, var_trait=var_trait,
                var_ar=var_ar, var_state=var_state, a=a )
    # evaluate log-likelihood
    ll <- LAM::loglike_mvnorm( S=data$S, Sigma=Sigma, M=data$M, mu=mu,
                n=data$n, lambda=1E-5)
    return(ll)
}
#** Note:
```

```
#   (1) The function starts_uni_cov calculates the model implied covariance matrix
#       for the STARTS model.
#   (2) The function loglike_mvnorm evaluates the loglikelihood for a multivariate
#       normal distribution given sample and population means M and mu, and sample
#       and population covariance matrix S and Sigma.

#*** starting values for parameters
pars <- c( .33, .33, .33, .75)
names(pars) <- c("vt","va","vs","b")
#*** bounds for acceptance rates
acceptance_bounds <- c( .45, .55 )
#*** starting values for proposal standard deviations
proposal_sd <- c( .1, .1, .1, .1 )
names(proposal_sd) <- names(pars)
#*** lower and upper bounds for parameter estimates
pars_lower <- c( .001, .001, .001, .001 )
pars_upper <- c( 10, 10, 10, .999 )
#*** define prior distributions | use prior sample size of 3
prior_model <- "
    vt ~ dinvgamma2(NA, 3, .33 )
    va ~ dinvgamma2(NA, 3, .33 )
    vs ~ dinvgamma2(NA, 3, .33 )
    b ~ dbeta(NA, 4, 4 )
        "
#*** define number of iterations
n.burnin <- 5000
n.iter <- 20000
set.seed(987)    # fix random seed
#*** estimate model with 'LAM::amh' function
mod <- LAM::amh( data=data, nobs=data$n, pars=pars, model=model,
            prior=prior_model, proposal_sd=proposal_sd, n.iter=n.iter,
            n.burnin=n.burnin, pars_lower=pars_lower, pars_upper=pars_upper)
#*** model summary
summary(mod)
  ##  Parameter Summary (Marginal MAP estimation)
  ##    parameter   MAP    SD  Q2.5 Q97.5  Rhat SERatio effSize accrate
  ##  1        vt 0.352 0.088 0.122 0.449 1.014   0.088     128   0.557
  ##  2        va 0.335 0.080 0.238 0.542 1.015   0.090     123   0.546
  ##  3        vs 0.341 0.018 0.297 0.367 1.005   0.042     571   0.529
  ##  4         b 0.834 0.065 0.652 0.895 1.017   0.079     161   0.522
  ##
  ##  Comparison of Different Estimators
  ##
  ##  MAP: Univariate marginal MAP estimation
  ##  mMAP: Multivariate MAP estimation (penalized likelihood estimate)
  ##  Mean: Mean of posterior distributions
  ##
  ##    Parameter Summary:
  ##    parm   MAP  mMAP  Mean
  ##  1   vt 0.352 0.294 0.300
  ##  2   va 0.335 0.371 0.369
  ##  3   vs 0.341 0.339 0.335
  ##  4    b 0.834 0.822 0.800
```

```
#* inspect convergence
plot(mod, ask=TRUE)

#---------------------------
# fitting the STARTS model with penalized maximum likelihood estimation
mod2 <- LAM::pmle( data=data, nobs=data$n, pars=pars, model=model,  prior=prior_model,
            pars_lower=pars_lower, pars_upper=pars_upper, method="L-BFGS-B",
            control=list( trace=TRUE )  )
# model summaries
summary(mod2)
  ##  Parameter Summary
  ##    parameter   est    se      t     p active
  ##  1        vt 0.298 0.110  2.712 0.007      1
  ##  2        va 0.364 0.102  3.560 0.000      1
  ##  3        vs 0.337 0.018 18.746 0.000      1
  ##  4         b 0.818 0.074 11.118 0.000      1

#---------------------------
# fitting the STARTS model in lavaan

library(lavaan)

## define lavaan model
lavmodel <- "
     #*** stable trait
     T=~ 1*W1 + 1*W2 + 1*W3 + 1*W4 + 1*W5 + 1*W6
     T ~~ vt * T
     W1 ~~ 0*W1
     W2 ~~ 0*W2
     W3 ~~ 0*W3
     W4 ~~ 0*W4
     W5 ~~ 0*W5
     W6 ~~ 0*W6
     #*** autoregressive trait
     AR1=~ 1*W1
     AR2=~ 1*W2
     AR3=~ 1*W3
     AR4=~ 1*W4
     AR5=~ 1*W5
     AR6=~ 1*W6
     #*** state component
     S1=~ 1*W1
     S2=~ 1*W2
     S3=~ 1*W3
     S4=~ 1*W4
     S5=~ 1*W5
     S6=~ 1*W6
     S1 ~~ vs * S1
     S2 ~~ vs * S2
     S3 ~~ vs * S3
     S4 ~~ vs * S4
     S5 ~~ vs * S5
```

```
        S6 ~~ vs * S6
        AR2 ~ b * AR1
        AR3 ~ b * AR2
        AR4 ~ b * AR3
        AR5 ~ b * AR4
        AR6 ~ b * AR5
        AR1 ~~ va * AR1
        AR2 ~~ v1 * AR2
        AR3 ~~ v1 * AR3
        AR4 ~~ v1 * AR4
        AR5 ~~ v1 * AR5
        AR6 ~~ v1 * AR6
        #*** nonlinear constraint
        v1==va * ( 1 - b^2 )
        #*** force variances to be positive
        vt > 0.001
        va > 0.001
        vs > 0.001
        #*** variance proportions
        var_total :=vt + vs + va
        propt :=vt / var_total
        propa :=va / var_total
        props :=vs / var_total
       "
# estimate lavaan model
mod <- lavaan::lavaan(model=lavmodel, sample.cov=S, sample.nobs=660)
# summary and fit measures
summary(mod)
lavaan::fitMeasures(mod)
coef(mod)[ ! duplicated( names(coef(mod))) ]
  ##          vt          vs           b          va          v1
  ##  0.001000023 0.349754630 0.916789054 0.651723144 0.103948711

## End(Not run)
```

`clpm_to_ctm` *Transformation of Path Coefficients of Cross-Lagged Panel Model*

Description

Transforms path coefficients $\boldsymbol{\Phi}(\Delta t_1)$ of a cross-lagged panel model (CLPM) based on time interval Δt_1 into a time interval Δt_2. The transformation is based on the assumption of a continuous time model (CTM; Voelkle, Oud, Davidov, & Schmidt, 2012) including a drift matrix $\boldsymbol{A}$. The transformation relies on the matrix exponential function (see Kuiper & Ryan, 2018), i.e. $\boldsymbol{\Phi}(\Delta t_1) = \exp(\boldsymbol{A}\Delta t_1)$.

Usage

```
clpm_to_ctm(Phi1, delta1=1, delta2=2, Phi1_vcov=NULL)
```

Arguments

`Phi1`	Matrix of path coefficients $\mathbf{\Phi}(\Delta t_1)$
`delta1`	Numeric Δt_1
`delta2`	Numeric Δt_2
`Phi1_vcov`	Optional covariance matrix for parameter estimates of $\mathbf{\Phi}(\Delta t_1)$

Value

A list with following entries

`A`	Drift matrix
`A_se`	Standard errors of drift matrix
`A_vcov`	Covariance matrix of drift matrix
`Phi2`	Path coefficients $\mathbf{\Phi}(\Delta t_2)$
`Phi2_se`	Standard errors for $\mathbf{\Phi}(\Delta t_2)$
`Phi2_vcov`	Covariance matrix for $\mathbf{\Phi}(\Delta t_2)$

References

Kuiper, R. M., & Ryan, O. (2018). Drawing conclusions from cross-lagged relationships: Reconsidering the role of the time-interval. *Structural Equation Modeling, 25*(5), 809-823. doi:10.1080/10705511.2018.1431046

Voelkle, M. C., Oud, J. H., Davidov, E., & Schmidt, P. (2012). An SEM approach to continuous time modeling of panel data: Relating authoritarianism and anomia. *Psychological Methods, 17*(2), 176-192. doi:10.1037/a0027543

Examples

```
#############################################################################
# EXAMPLE 1: Example of Voelkle et al. (2012)
#############################################################################

library(expm)

# path coefficient matrix of Voelkle et al. (2012), but see
# also Kuiper and Ryan (2018)
Phi1 <- matrix( c( .64, .18,
                  .03, .89 ), nrow=2, ncol=2, byrow=TRUE )
# transformation to time interval 2
mod <- LAM::clpm_to_ctm(Phi1, delta1=1, delta2=2)
print(mod)

## Not run:
#############################################################################
# EXAMPLE 2: Example with two dimensions
#############################################################################

library(STARTS)
```

```
library(lavaan)

data(data.starts02, package="STARTS")
dat <- data.starts02$younger_cohort
cormat <- cov2cor(as.matrix(dat$covmat))

#-- estimate CLPM
lavmodel <- "
       a2 ~ a1 + b1
       b2 ~ a1 + b1
       "
mod <- lavaan::sem(lavmodel, sample.cov=cormat, sample.nobs=500)
summary(mod)

#- select parameters
pars <- c("a2~a1", "a2~b1", "b2~a1", "b2~b1")
Phi1 <- matrix( coef(mod)[pars], 2, 2, byrow=TRUE)
Phi1_vcov <- vcov(mod)[ pars, pars ]

# conversion to time interval 1.75
LAM::clpm_to_ctm(Phi1=Phi1, delta1=1, delta2=1.75, Phi1_vcov=Phi1_vcov)

#############################################################################
# EXAMPLE 3: Example with three dimensions
#############################################################################

library(STARTS)
library(lavaan)

data(data.starts02, package="STARTS")
dat <- data.starts02$younger_cohort
cormat <- cov2cor(as.matrix(dat$covmat))

#-- estimate CLPM
lavmodel <- "
       a4 ~ a1 + b1 + c1
       b4 ~ a1 + b1 + c1
       c4 ~ a1 + b1 + c1
       "
mod <- lavaan::sem(lavmodel, sample.cov=cormat, sample.nobs=500)
summary(mod)

#- select parameters
pars <- 1:9
Phi1 <- matrix( coef(mod)[pars], 3, 3, byrow=TRUE)
Phi1_vcov <- vcov(mod)[ pars, pars ]

# conversion frpm time interval 3 to time interval 1
LAM::clpm_to_ctm(Phi1=Phi1, delta1=3, delta2=1, Phi1_vcov=Phi1_vcov)

## End(Not run)
```

`data.HT` *Datasets from Heck and Thomas (2015)*

Description

Selected datasets from Heck and Thomas (2015).

Usage

```
data(data.HT12)
```

Format

- The format of the dataset `data.HT12` from Chapter 1 is:

  ```
  'data.frame': 120 obs. of 11 variables:
  $ schcode: num 100 100 100 100 100 107 107 107 107 107 ...
  $ read : num 682 644 651 710 673 593 660 640 646 634 ...
  $ math : num 714 661 670 786 719 598 660 622 647 696 ...
  $ lang : num 673 670 648 677 698 596 673 613 618 645 ...
  $ ess : num -2.8 -2.8 -2.8 -2.8 -2.8 3.19 3.19 3.19 3.19 3.19 ...
  $ cses : num -2.4 -2.4 -2.4 -2.4 -2.4 1.67 1.67 1.67 1.67 1.67 ...
  $ female : num 0 0 0 0 0 0 1 1 1 0 ...
  $ lowses : num 0 0 0 0 1 0 0 1 1 0 ...
  $ lgsch : num 0 0 0 0 0 0 0 0 0 0 ...
  $ age : num 135 140 135 151 138 138 140 141 144 146 ...
  $ ncses : num 2.4 2.4 2.4 2.4 2.4 -1.67 -1.67 -1.67 -1.67 -1.67 ...
  ```

Source

https://www.routledge.com/An-Introduction-to-Multilevel-Modeling-Techniques-MLM-and-SEM-Approaches/Heck-Thomas/p/book/9781848725522

References

Heck, R. H. & Thomas, S. L. (2015). *An introduction to multilevel modeling techniques.* Routledge, New York.

`loglike_mvnorm` *Log-Likelihood Value of a Multivariate Normal Distribution*

Description

Computes log-likelihood value of a multivariate normal distribution given the empirical mean vector and the empirical covariance matrix as sufficient statistics.

Usage

```
loglike_mvnorm(M, S, mu, Sigma, n, log=TRUE, lambda=0, ginv=FALSE, eps=1e-30,
       use_rcpp=FALSE )

loglike_mvnorm_NA_pattern( suff_stat, mu, Sigma, log=TRUE, lambda=0, ginv=FALSE,
       eps=1e-30, use_rcpp=FALSE )
```

Arguments

`M`	Empirical mean vector
`S`	Empirical covariance matrix
`mu`	Population mean vector
`Sigma`	Population covariance matrix
`n`	Sample size
`log`	Optional logical indicating whether the logarithm of the likelihood should be calculated.
`lambda`	Regularization parameter of the covariance matrix (see Details).
`ginv`	Logical indicating whether generalized inverse matrix of $\boldsymbol{\Sigma}$ should be used
`eps`	Threshold for determinant value of $\boldsymbol{\Sigma}$
`use_rcpp`	Logical indicating whether **Rcpp** function should be used
`suff_stat`	List with sufficient statistics as generated by `suff_stat_NA_pattern`.

Details

The population covariance matrix $\boldsymbol{\Sigma}$ is regularized if λ (`lambda`) is chosen larger than zero. Let $\boldsymbol{\Delta}_{\boldsymbol{\Sigma}}$ denote a diagonal matrix containing the diagonal entries of $\boldsymbol{\Sigma}$. Then, a regularized matrix $\boldsymbol{\Sigma}^*$ is defined as $\boldsymbol{\Sigma}^* = w\boldsymbol{\Sigma} + (1-w)\boldsymbol{\Delta}_{\boldsymbol{\Sigma}}$ with $w = n/(n+\lambda)$.

Value

Log-likelihood value

Examples

```
#############################################################################
# EXAMPLE 1: Multivariate normal distribution
#############################################################################

library(MASS)

#--- simulate data
Sigma <- c( 1, .55, .5, .55, 1, .5,.5, .5, 1 )
Sigma <- matrix( Sigma, nrow=3, ncol=3 )
mu <- c(0,1,1.2)
N <- 400
set.seed(9875)
dat <- MASS::mvrnorm( N, mu, Sigma )
```

```
colnames(dat) <- paste0("Y",1:3)
S <- stats::cov(dat)
M <- colMeans(dat)

#--- evaulate likelihood
res1 <- LAM::loglike_mvnorm( M=M, S=S, mu=mu, Sigma=Sigma, n=N, lambda=0 )
# compare log likelihood with somewhat regularized covariance matrix
res2 <- LAM::loglike_mvnorm( M=M, S=S, mu=mu, Sigma=Sigma, n=N, lambda=1 )
print(res1)
print(res2)

## Not run:

#############################################################################
# EXAMPLE 2: Multivariate normal distribution with missing data patterns
#############################################################################

library(STARTS)
data(data.starts01b, package="STARTS")
dat <- data.starts01b
dat1 <- dat[, paste0("E",1:3)]

#-- compute sufficient statistics
suff_stat <- LAM::suff_stat_NA_pattern(dat1)
#-- define some population mean and covariance
mu <- colMeans(dat1, na.rm=TRUE)
Sigma <- stats::cov(dat1, use="pairwise.complete.obs")

#-- compute log-likelihood
LAM::loglike_mvnorm_NA_pattern( suff_stat=suff_stat, mu=mu, Sigma=Sigma)

## End(Not run)
```

`mlnormal` *(Restricted) Maximum Likelihood Estimation with Prior Distributions and Penalty Functions under Multivariate Normality*

Description

The `mlnormal` estimates statistical model for multivariate normally distributed outcomes with specified mean structure and covariance structure (see Details and Examples). Model classes include multilevel models, factor analysis, structural equation models, multilevel structural equation models, social relations model and perhaps more.

The estimation can be conducted under maximum likelihood, restricted maximum likelihood and maximum posterior estimation with prior distribution. Regularization (i.e. LASSO penalties) is also accomodated.

Usage

```
mlnormal(y, X, id, Z_list, Z_index, beta=NULL, theta, method="ML", prior=NULL,
    lambda_beta=NULL, weights_beta=NULL, lambda_theta=NULL, weights_theta=NULL,
    beta_lower=NULL, beta_upper=NULL,    theta_lower=NULL, theta_upper=NULL,
    maxit=800, globconv=1e-05, conv=1e-06, verbose=TRUE, REML_shortcut=NULL,
    use_ginverse=FALSE, vcov=TRUE, variance_shortcut=TRUE, use_Rcpp=TRUE,
    level=0.95, numdiff.parm=1e-04, control_beta=NULL, control_theta=NULL)

## S3 method for class 'mlnormal'
summary(object, digits=4, file=NULL, ...)

## S3 method for class 'mlnormal'
print(x, digits=4, ...)

## S3 method for class 'mlnormal'
coef(object, ...)

## S3 method for class 'mlnormal'
logLik(object, ...)

## S3 method for class 'mlnormal'
vcov(object, ...)

## S3 method for class 'mlnormal'
confint(object, parm, level=.95, ... )
```

Arguments

`y`	Vector of outcomes
`X`	Matrix of covariates
`id`	Vector of identifiers (subjects or clusters, see Details)
`Z_list`	List of design matrices for covariance matrix (see Details)
`Z_index`	Array containing loadings of design matrices (see Details). The dimensions are units × matrices × parameters.
`beta`	Initial vector for $\boldsymbol{\beta}$
`theta`	Initial vector for $\boldsymbol{\theta}$
`method`	Estimation method. Can be either `"ML"` or `"REML"`.
`prior`	Prior distributions. Can be conveniently specified in a string which is processed by the function `prior_model_parse`. Only univariate prior distributions can be specified.
`lambda_beta`	Parameter $\lambda_{\boldsymbol{\beta}}$ for penalty function $P(\boldsymbol{\beta}) = \lambda_{\boldsymbol{\beta}} \sum_h w_{\boldsymbol{\beta}h} \lvert \beta_h \rvert$
`weights_beta`	Parameter vector $\boldsymbol{w}_{\boldsymbol{\beta}}$ for penalty function $P(\boldsymbol{\beta}) = \lambda_{\boldsymbol{\beta}} \sum_h w_{\boldsymbol{\beta}h} \lvert \beta_h \rvert$
`lambda_theta`	Parameter $\lambda_{\boldsymbol{\theta}}$ for penalty function $P(\boldsymbol{\theta}) = \lambda_{\boldsymbol{\theta}} \sum_h w_{\boldsymbol{\theta}h} \lvert \theta_h \rvert$
`weights_theta`	Parameter vector $\boldsymbol{w}_{\boldsymbol{\theta}}$ for penalty function $P(\boldsymbol{\theta}) = \lambda_{\boldsymbol{\theta}} \sum_h w_{\boldsymbol{\theta}h} \lvert \theta_h \rvert$

`beta_lower`	Vector containing lower bounds for $\boldsymbol{\beta}$ parameter
`beta_upper`	Vector containing upper bounds for $\boldsymbol{\beta}$ parameter
`theta_lower`	Vector containing lower bounds for $\boldsymbol{\theta}$ parameter
`theta_upper`	Vector containing upper bounds for $\boldsymbol{\theta}$ parameter
`maxit`	Maximum number of iterations
`globconv`	Convergence criterion deviance
`conv`	Maximum parameter change
`verbose`	Print progress?
`REML_shortcut`	Logical indicating whether computational shortcuts should be used for REML estimation
`use_ginverse`	Logical indicating whether a generalized inverse should be used
`vcov`	Logical indicating whether a covariance matrix of $\boldsymbol{\theta}$ parameter estimates should be computed in case of REML (which is computationally demanding)
`variance_shortcut`	Logical indicating whether computational shortcuts for calculating covariance matrices should be used
`use_Rcpp`	Logical indicating whether the **Rcpp** package should be used
`level`	Confidence level
`numdiff.parm`	Numerical differentiation parameter
`control_beta`	List with control arguments for $\boldsymbol{\beta}$ estimation. The default is `list( maxiter=10, conv=1E-4, ridge=1E-6)`.
`control_theta`	List with control arguments for $\boldsymbol{\theta}$ estimation. The default is `list( maxiter=10, conv=1E-4, ridge=1E-6)`.
`object`	Object of class `mlnormal`
`digits`	Number of digits used for rounding
`file`	File name
`parm`	Parameter to be selected for `confint` method
`...`	Further arguments to be passed
`x`	Object of class `mlnormal`

Details

The data consists of outcomes $\boldsymbol{y}_i$ and covariates $\boldsymbol{X}_i$ for unit i. The unit can be subjects, clusters (like schools) or the full outcome vector. It is assumed that $\boldsymbol{y}_i$ is normally distributed as $N(\boldsymbol{\mu}_i, \boldsymbol{V}_i)$ where the mean structure is modelled as

$$\boldsymbol{\mu}_i = \boldsymbol{X}_i\boldsymbol{\beta}$$

and the covariance structure $\boldsymbol{V}_i$ depends on a parameter vector $\boldsymbol{\theta}$. More specifically, the covariance matrix $\boldsymbol{V}_i$ is modelled as a sum of functions of the parameter $\boldsymbol{\theta}$ and known design matrices $\boldsymbol{Z}_{im}$ for unit i ($m = 1, \ldots, M$). The model is

$$\boldsymbol{V}_i = \sum_{m=1}^{M} \boldsymbol{Z}_{im}\gamma_{im} \qquad \text{with} \qquad \gamma_{im} = \prod_{h=1}^{H} \theta_h^{q_{imh}}$$

where q_{imh} are non-negative known integers specified in `Z_index` and $\boldsymbol{Z}_{im}$ are design matrices specified in `Z_list`.

The estimation follows Fisher scoring (Jiang, 2007; for applications see also Longford, 1987; Lee, 1990; Gill & Swartz, 2001) and the regularization approach is as described in Lin, Pang and Jiang (2013) (see also Krishnapuram, Carin, Figueiredo, & Hartemink, 2005).

Value

List with entries

`theta`	Estimated $\boldsymbol{\theta}$ parameter
`beta`	Estimated $\boldsymbol{\beta}$ parameter
`theta_summary`	Summary of $\boldsymbol{\theta}$ parameters
`beta_summary`	Summary of $\boldsymbol{\beta}$ parameters
`coef`	Estimated parameters
`vcov`	Covariance matrix of estimated parameters
`ic`	Information criteria
`V_list`	List with fitted covariance matrices $\boldsymbol{V}_i$
`V1_list`	List with inverses of fitted covariance matrices $\boldsymbol{V}_i$
`prior_args`	Some arguments in case of prior distributions
`...`	More values

References

Gill, P. S., & Swartz, T. B. (2001). Statistical analyses for round robin interaction data. *Canadian Journal of Statistics, 29*, 321-331. doi:10.2307/3316080

Jiang, J. (2007). *Linear and generalized linear mixed models and their applications.* New York: Springer.

Krishnapuram, B., Carin, L., Figueiredo, M. A., & Hartemink, A. J. (2005). Sparse multinomial logistic regression: Fast algorithms and generalization bounds. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 27*, 957-968. doi:10.1109/TPAMI.2005.127

Lee, S. Y. (1990). Multilevel analysis of structural equation models. *Biometrika, 77*, 763-772. doi:10.1093/biomet/77.4.763

Lin, B., Pang, Z., & Jiang, J. (2013). Fixed and random effects selection by REML and pathwise coordinate optimization. *Journal of Computational and Graphical Statistics, 22*, 341-355. doi:10.1080/10618600.2012.681219

Longford, N. T. (1987). A fast scoring algorithm for maximum likelihood estimation in unbalanced mixed models with nested random effects. *Biometrika, 74*, 817-827. doi:10.1093/biomet/74.4.817

See Also

See **lavaan**, **sem**, **lava**, **OpenMx** or **nlsem** packages for estimation of (single level) structural equation models.

See the **regsem** and **lsl** packages for regularized structural equation models.

See **lme4** or **nlme** package for estimation of multilevel models.

See the **lmmlasso** and **glmmLasso** packages for regularized mixed effects models.

See **OpenMx** and **xxM** packages (*http://xxm.times.uh.edu/*) for estimation of multilevel structural equation models.

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: Two-level random intercept model
#############################################################################

#--------------------------------------------------------------
# Simulate data
#--------------------------------------------------------------

set.seed(976)
G <- 150 ; rg <- c(10,20)   # 150 groups with group sizes ranging from 10 to 20
#* simulate group sizes
ng <- round( stats::runif( G, min=rg[1], max=rg[2] ) )
idcluster <- rep(1:G, ng )
#* simulate covariate
iccx <- .3
x <- rep( stats::rnorm( G, sd=sqrt( iccx) ), ng ) +
            stats::rnorm( sum(ng), sd=sqrt( 1 - iccx) )
#* simulate outcome
b0 <- 1.5 ; b1 <- .4 ; iccy <- .2
y <- b0 + b1*x + rep( stats::rnorm( G, sd=sqrt( iccy) ), ng ) +
         stats::rnorm( sum(ng), sd=sqrt( 1 - iccy) )

#-----------------------
#--- arrange input for mlnormal function

id <- idcluster          # cluster is identifier
X <- cbind( 1, x )      # matrix of covariates
N <- length(id)          # number of units (clusters), which is G

MD <- max(ng)   # maximum number of persons in a group
NP <- 2         # number of covariance parameters theta

#* list of design matrix for covariance matrix
#  In the case of the random intercept model, the covariance structure is
#  tau^2 * J + sigma^2 * I, where J is a matrix of ones and I is the
#  identity matrix
Z <- as.list(1:G)
for (gg in 1:G){
```

```
    Ngg <- ng[gg]
    Z[[gg]] <- as.list( 1:2 )
    Z[[gg]][[1]] <- matrix( 1, nrow=Ngg, ncol=Ngg )  # level 2 variance
    Z[[gg]][[2]] <- diag(1,Ngg)            # level 1 variance
}
Z_list <- Z
#* parameter list containing the powers of parameters
Z_index <- array( 0, dim=c(G,2,2) )
Z_index[ 1:G, 1, 1] <- Z_index[ 1:G, 2, 2] <- 1

#** starting values and parameter names
beta <- c( 1, 0 )
names(beta) <- c("int", "x")
theta <- c( .05, 1 )
names(theta) <- c("tau2", "sig2" )

#** create dataset for lme4 for comparison
dat <- data.frame(y=y, x=x, id=id )

#----------------------------------------------------------------
# Model 1: Maximum likelihood estimation
#----------------------------------------------------------------

#** mlnormal function
mod1a <- LAM::mlnormal( y=y, X=X, id=id, Z_list=Z_list, Z_index=Z_index,
            beta=beta, theta=theta, method="ML" )
summary(mod1a)

# lme4::lmer function
library(lme4)
mod1b <- lme4::lmer( y ~ x + (1 | id ), data=dat, REML=FALSE )
summary(mod1b)

#----------------------------------------------------------------
# Model 2: Restricted maximum likelihood estimation
#----------------------------------------------------------------

#** mlnormal function
mod2a <- LAM::mlnormal( y=y, X=X, id=id, Z_list=Z_list, Z_index=Z_index,
            beta=beta, theta=theta, method="REML" )
summary(mod2a)

# lme4::lmer function
mod2b <- lme4::lmer( y ~ x + (1 | id ), data=dat, REML=TRUE )
summary(mod2b)

#----------------------------------------------------------------
# Model 3: Estimation of standard deviation instead of variances
#----------------------------------------------------------------

# The model is now parametrized in standard deviations
# Variances are then modeled as tau^2 and sigma^2, respectively.
Z_index2 <- 2*Z_index       # change loading matrix
```

```
# estimate model
mod3 <- LAM::mlnormal( y=y, X=X, id=id, Z_list=Z_list, Z_index=Z_index2,
            beta=beta, theta=theta )
summary(mod3)

#----------------------------------------------------------------
# Model 4: Maximum posterior estimation
#----------------------------------------------------------------

# specify prior distributions for parameters
prior <- "
    tau2 ~ dgamma(NA, 2, .5 )
    sig2 ~ dinvgamma(NA, .1, .1 )
    x ~ dnorm( NA, .2, 1000 )
    "

# estimate model in mlnormal
mod4 <- LAM::mlnormal( y=y, X=X, id=id, Z_list=Z_list, Z_index=Z_index,
            beta=beta, theta=theta, method="REML", prior=prior, vcov=FALSE )
summary(mod4)

#----------------------------------------------------------------
# Model 5: Estimation with regularization on beta and theta parameters
#----------------------------------------------------------------

#*** penalty on theta parameter
lambda_theta <- 10
weights_theta <- 1 + 0 * theta
#*** penalty on beta parameter
lambda_beta <- 3
weights_beta <- c( 0, 1.8 )

# estimate model
mod5 <- LAM::mlnormal( y=y, X=X, id=id, Z_list=Z_list, Z_index=Z_index,
            beta=beta, theta=theta, method="ML", maxit=maxit,
            lambda_theta=lambda_theta, weights_theta=weights_theta,
            lambda_beta=lambda_beta, weights_beta=weights_beta  )
summary(mod5)

#############################################################################
# EXAMPLE 2: Latent covariate model, two-level regression
#############################################################################

# Yb=beta_0 + beta_b*Xb + eb (between level) and
# Yw=beta_w*Xw + ew (within level)

#----------------------------------------------------------------
# Simulate data from latent covariate model
#----------------------------------------------------------------

set.seed(865)
# regression parameters
```

```
beta_0 <- 1 ; beta_b <- .7 ; beta_w <- .3
G <- 200      # number of groups
n <- 15      # group size
iccx <- .2   # intra class correlation x
iccy <- .35  # (conditional) intra class correlation y
# simulate latent variables
xb <- stats::rnorm(G, sd=sqrt( iccx ) )
yb <- beta_0 + beta_b * xb + stats::rnorm(G, sd=sqrt( iccy ) )
xw <- stats::rnorm(G*n, sd=sqrt( 1-iccx ) )
yw <- beta_w * xw + stats::rnorm(G*n, sd=sqrt( 1-iccy ) )
group <- rep( 1:G, each=n )
x <- xw + xb[ group ]
y <- yw + yb[ group ]
# test results on true data
lm( yb ~ xb )
lm( yw ~ xw )

# create vector of outcomes in the form
# ( y_11, x_11, y_21, x_21, ... )
dat <- cbind( y, x )
dat
Y <- as.vector( t(dat) )    # outcome vector
ny <- length(Y)
X <- matrix( 0, nrow=ny, ncol=2 )
X[ seq(1,ny,2), 1 ] <- 1   # design vector for mean y
X[ seq(2,ny,2), 2 ] <- 1   # design vector for mean x
id <- rep( group, each=2 )

#--------------------------------------------------------------
# Model 1: Linear regression ignoring multilevel structure
#--------------------------------------------------------------

# y=beta_0 + beta_t *x + e
# Var(y)=beta_t^2 * var_x + var_e
# Cov(y,x)=beta_t * var_x
# Var(x)=var_x

#** initial parameter values
theta <- c( 0, 1, .5 )
names(theta) <- c( "beta_t", "var_x", "var_e")
beta <- c(0,0)
names(beta) <- c("mu_y","mu_x")

# The unit i is a cluster in this example.

#--- define design matrices | list Z_list
Hlist <- list(  matrix( c(1,0,0,0), 2, 2 ), # var(y)
                matrix( c(1,0,0,0), 2, 2 ), # var(y) (two terms)
                matrix( c(0,1,1,0), 2, 2 ), # cov(x,y)
                matrix( c(0,0,0,1), 2, 2 ) ) # var(x)

U0 <- matrix( 0, nrow=2*n,ncol=2*n )
Ulist <- list( U0, U0, U0, U0 )
```

```
M <- length(Hlist)
for (mm in 1:M){    # mm <- 1
    for (nn in 1:n){     # nn <- 0
        Ulist[[ mm ]][ 2*(nn-1) + 1:2, 2*(nn-1) + 1:2 ] <- Hlist[[ mm ]]
    }
}
Z_list <- as.list(1:G)
for (gg in 1:G){
    Z_list[[gg]] <- Ulist
}

#--- define index vectors
Z_index <- array( 0, dim=c(G, 4, 3 ) )
K0 <- matrix( 0, nrow=4, ncol=3 )
colnames(K0) <- names(theta)
# Var(y)=beta_t^2 * var_x + var_e  (matrices withn indices 1 and 2)
K0[ 1, c("beta_t","var_x") ] <- c(2,1)  # beta_t^2 * var_x
K0[ 2, c("var_e") ] <- c(1)  # var_e
# Cov(y,x)=beta_t * var_x
K0[ 3, c("beta_t","var_x")] <- c(1,1)
# Var(x)=var_x
K0[ 4, c("var_x") ] <- c(1)
for (gg in 1:G){
    Z_index[gg,,] <- K0
}

#*** estimate model with mlnormal
mod1a <- LAM::mlnormal( y=Y, X=X, id=id, Z_list=Z_list, Z_index=Z_index,
            beta=beta, theta=theta, method="REML", vcov=FALSE )
summary(mod1a)

#*** estimate linear regression with stats::lm
mod1b <- stats::lm( y ~ x )
summary(mod1b)

#----------------------------------------------------------------
# Model 2: Latent covariate model
#----------------------------------------------------------------

#** initial parameters
theta <- c( 0.12, .6, .5, 0, .2, .2 )
names(theta) <- c( "beta_w", "var_xw", "var_ew",
                "beta_b", "var_xb", "var_eb")

#--- define design matrices | list Z_list
Hlist <- list(  matrix( c(1,0,0,0), 2, 2 ), # var(y)
                matrix( c(1,0,0,0), 2, 2 ), # var(y) (two terms)
                matrix( c(0,1,1,0), 2, 2 ), # cov(x,y)
                matrix( c(0,0,0,1), 2, 2 ) ) # var(x)
U0 <- matrix( 0, nrow=2*n,ncol=2*n )
Ulist <- list( U0, U0, U0, U0,  # within structure
               U0, U0, U0, U0  )  # between structure
M <- length(Hlist)
```

```
#*** within structure
design_within <- diag(n)  # design matrix within structure
for (mm in 1:M){    # mm <- 1
    Ulist[[ mm ]] <- base::kronecker( design_within, Hlist[[mm]] )
}
#*** between structure
design_between <- matrix(1, nrow=n, ncol=n)
      # matrix of ones corresponding to group size
for (mm in 1:M){    # mm <- 1
    Ulist[[ mm + M ]] <-  base::kronecker( design_between, Hlist[[ mm ]] )
}
Z_list <- as.list(1:G)
for (gg in 1:G){
    Z_list[[gg]] <- Ulist
}

#--- define index vectors Z_index
Z_index <- array( 0, dim=c(G, 8, 6 ) )
K0 <- matrix( 0, nrow=8, ncol=6 )
colnames(K0) <- names(theta)
# Var(y)=beta^2 * var_x + var_e  (matrices withn indices 1 and 2)
K0[ 1, c("beta_w","var_xw") ] <- c(2,1)  # beta_t^2 * var_x
K0[ 2, c("var_ew") ] <- c(1)  # var_e
K0[ 5, c("beta_b","var_xb") ] <- c(2,1)  # beta_t^2 * var_x
K0[ 6, c("var_eb") ] <- c(1)  # var_e
# Cov(y,x)=beta * var_x
K0[ 3, c("beta_w","var_xw")] <- c(1,1)
K0[ 7, c("beta_b","var_xb")] <- c(1,1)
# Var(x)=var_x
K0[ 4, c("var_xw") ] <- c(1)
K0[ 8, c("var_xb") ] <- c(1)
for (gg in 1:G){
    Z_index[gg,,] <- K0
}

#--- estimate model with mlnormal
mod2a <- LAM::mlnormal( y=Y, X=X, id=id, Z_list=Z_list, Z_index=Z_index,
            beta=beta, theta=theta, method="ML" )
summary(mod2a)

#############################################################################
# EXAMPLE 3: Simple linear regression, single level
#############################################################################

#---------------------------------------------------------------
# Simulate data
#---------------------------------------------------------------

set.seed(875)
N <- 300
x <- stats::rnorm( N, sd=1.3 )
y <- .4 + .7 * x + stats::rnorm( N, sd=.5 )
dat <- data.frame( x, y )
```

```
#--------------------------------------------------------------
# Model 1: Linear regression modelled with residual covariance structure
#--------------------------------------------------------------

# matrix of predictros
X <- cbind( 1, x )
# list with design matrices
Z <- as.list(1:N)
for (nn in 1:N){
    Z[[nn]] <- as.list( 1 )
    Z[[nn]][[1]] <- matrix( 1, nrow=1, ncol=1 )  # residual variance
}
#* loading matrix
Z_index <- array( 0, dim=c(N,1,1) )
Z_index[ 1:N, 1, 1] <- 2  # parametrize residual standard deviation
#** starting values and parameter names
beta <- c( 0, 0 )
names(beta) <- c("int", "x")
theta <- c(1)
names(theta) <- c("sig2" )
# id vector
id <- 1:N

#** mlnormal function
mod1a <- LAM::mlnormal( y=y, X=X, id=id, Z_list=Z, Z_index=Z_index,
            beta=beta, theta=theta, method="ML" )
summary(mod1a)

# estimate linear regression with stats::lm
mod1b <- stats::lm( y ~ x )
summary(mod1b)

#--------------------------------------------------------------
# Model 2: Linear regression modelled with bivariate covariance structure
#--------------------------------------------------------------

#** define design matrix referring to mean structure
X <- matrix( 0, nrow=2*N, ncol=2 )
X[ seq(1,2*N,2), 1 ] <- X[ seq(2,2*N,2), 2 ] <- 1

#** create outcome vector
y1 <- dat[ cbind( rep(1:N, each=2), rep(1:2, N ) ) ]
#** list with design matrices
Z <- as.list(1:N)
Z0 <- 0*matrix( 0, nrow=2,ncol=2)
ZXY <- ZY <- ZX <- Z0
# design matrix Var(X)
ZX[1,1] <- 1
# design matrix Var(Y)
ZY[2,2] <- 1
# design matrix covariance
ZXY[1,2] <- ZXY[2,1] <- 1
```

```
# Var(X)=sigx^2
# Cov(X,Y)=beta * sigx^2
# Var(Y)=beta^2 * sigx^2 + sige^2
Z_list0 <- list( ZY, ZY, ZXY, ZX )
for (nn in 1:N){
    Z[[nn]] <- Z_list0
}
#* parameter list containing the powers of parameters
theta <- c(1,0.3,1)
names(theta) <- c("sigx", "beta", "sige" )
Z_index <- array( 0, dim=c(N,4,3) )
for (nn in 1:N){
    # Var(X)
    Z_index[nn, 4, ] <- c(2,0,0)
    # Cov(X,Y)
    Z_index[nn, 3, ] <- c(2,1,0)
    # Var(Y)
    Z_index[nn,1,] <- c(2,2,0)
    Z_index[nn,2,] <- c(0,0,2)
}
#** starting values and parameter names
beta <- c( 0, 0 )
names(beta) <- c("Mx", "My")
# id vector
id <- rep( 1:N, each=2 )

#** mlnormal function
mod2a <- LAM::mlnormal( y=y1, X=X, id=id, Z_list=Z, Z_index=Z_index,
            beta=beta, theta=theta, method="ML" )
summary(mod2a)

#--------------------------------------------------------------
# Model 3: Bivariate normal distribution in (sigma_X, sigma_Y, sigma_XY) parameters
#--------------------------------------------------------------

# list with design matrices
Z <- as.list(1:N)
Z0 <- 0*matrix( 0, nrow=2,ncol=2)
ZXY <- ZY <- ZX <- Z0
# design matrix Var(X)
ZX[1,1] <- 1
# design matrix Var(Y)
ZY[2,2] <- 1
# design matrix covariance
ZXY[1,2] <- ZXY[2,1] <- 1
Z_list0 <- list( ZX, ZY, ZXY  )
for (nn in 1:N){
    Z[[nn]] <- Z_list0
}

#* parameter list
theta <- c(1,1,.3)
names(theta) <- c("sigx", "sigy", "sigxy" )
```

```
Z_index <- array( 0, dim=c(N,3,3) )
for (nn in 1:N){
    # Var(X)
    Z_index[nn, 1, ] <- c(2,0,0)
    # Var(Y)
    Z_index[nn, 2, ] <- c(0,2,0)
    # Cov(X,Y)
    Z_index[nn, 3, ] <- c(0,0,1)
}

#** starting values and parameter names
beta <- c( 0, 0 )
names(beta) <- c("Mx", "My")

#** mlnormal function
mod3a <- LAM::mlnormal( y=y1, X=X, id=id, Z_list=Z, Z_index=Z_index,
            beta=beta, theta=theta, method="ML" )
summary(mod3a)

#---------------------------------------------------------------
# Model 4: Bivariate normal distribution in parameters of Cholesky decomposition
#---------------------------------------------------------------

# list with design matrices
Z <- as.list(1:N)
Z0 <- 0*matrix( 0, nrow=2,ncol=2)
ZXY <- ZY <- ZX <- Z0
# design matrix Var(X)
ZX[1,1] <- 1
# design matrix Var(Y)
ZY[2,2] <- 1
# design matrix covariance
ZXY[1,2] <- ZXY[2,1] <- 1
Z_list0 <- list( ZX, ZXY, ZY, ZY  )
for (nn in 1:N){
    Z[[nn]] <- Z_list0
}

#* parameter list containing the powers of parameters
theta <- c(1,0.3,1)
names(theta) <- c("L11", "L21", "L22" )
Z_index <- array( 0, dim=c(N,4,3) )
for (nn in 1:N){
    Z_index[nn,1,] <- c(2,0,0)
    Z_index[nn,2,] <- c(1,1,0)
    Z_index[nn,3,] <- c(0,2,0)
    Z_index[nn,4,] <- c(0,0,2)
}
#** starting values and parameter names
beta <- c( 0, 0 )
names(beta) <- c("Mx", "My")
# id vector
id <- rep( 1:N, each=2 )
```

```
#** mlnormal function
mod4a <- LAM::mlnormal( y=y1, X=X, id=id, Z_list=Z, Z_index=Z_index,
            beta=beta, theta=theta, method="ML" )
# parameter with lower diagonal entries of Cholesky matrix
mod4a$theta
# fill-in parameters for Cholesky matrix
L <- matrix(0,2,2)
L[ ! upper.tri(L) ] <- mod4a$theta
#** reconstruct covariance matrix
L
stats::cov.wt(dat, method="ML")$cov

## End(Not run)
```

`suff_stat_NA_pattern` *Sufficient Statistics for Dataset with Missing Response Pattern*

Description

Computes sufficient statistics for a dataset with an arbitrary missing response pattern.

Usage

```
suff_stat_NA_pattern(dat)
```

Arguments

`dat`	Numeric data frame

Value

A list with following entries

`nobs`	List with number of observations for each missing response pattern
`M`	List with mean vectors
`S`	List with covariance matrices
`varindex`	List with indices of observed variables
`NP`	Number of missing data patterns
`N`	Total sample size

Examples

```
## Not run:

#############################################################################
# EXAMPLE 1: Toy example for computation of sufficient statistics
#############################################################################
```

```
library(STARTS)

data(data.starts01b, package="STARTS")
dat <- data.starts01b
dat1 <- dat[, paste0("E",1:3)]

#-- compute sufficient statistics
res <- LAM::suff_stat_NA_pattern(dat1)
str(res)

## End(Not run)
```

Index