Package 'EasyMx'

January 30, 2023

Date 2023-01-27

Type Package

Title Easy Model-Builder Functions for 'OpenMx'

Imports methods

Description Utilities for building certain kinds of common matrices and models in the extended structural equation modeling package, 'OpenMx'.

Depends R (>= 3.0.0), OpenMx

Suggests rpf (>= 0.45)

License GPL

Version 0.3-2

NeedsCompilation no

URL https://bitbucket.org/mhunter/easymx

BugReports https://bitbucket.org/mhunter/easymx/issues

Author Michael D. Hunter [aut, cre] (<https://orcid.org/0000-0002-3651-6709>), Joshua N. Pritikin [ctb]

Maintainer Michael D. Hunter <mhunter.ou@gmail.com>

Repository CRAN

Date/Publication 2023-01-30 22:00:07 UTC

R topics documented:

EasyMx-package   *EasyMx: Easy modeling in OpenMx*

Description

EasyMx is a package for extended structural equation modeling. It is built as a higher-level frontend on top of OpenMx. It is intended as an Easy introduction to OpenMx: Easy Mx. Try the example below.

Details

All of the functions in the EasyMx package create OpenMx objects. These are most often MxMatrix, MxAlgebra, or MxModel objects. The primary difference between EasyMx and OpenMx is design philosophy. OpenMx has its foundation in WYSIWID: What you say is what it does. This requires the user to be very explicit. The EasyMx package is not as strong or flexible as OpenMx, but it places less burden on the user. Many decisions are made automatically for the user. Some of them are modifiable within EasyMx; for others the user is encouraged to use OpenMx, where nearly everything is modifiable.

The package is broadly divided into two styles of functions: matrix builders and model builders.

The matrix builder functions are utilities for building common structural equation model matrices. These include `emxLoadings` for factor loadings, `emxResiduals` for residual variances, `emxCovariances` for latent or manifest covariances, `emxMeans` for means and intercepts matrices, and `emxThresholds` for thresholds matrices when ordinal data are involved.

The model builder functions are higher-level utilities for building common kinds of structural equation models. The model builders often call several matrix builders. The model builders include `emxFactorModel` for (multiple) factor models, `emxGrowthModel` for latent growth curve models, and `emxRegressionModel` for full-information likelihood estimation of regression for observed variables.

There are also a few model builder functions for non-standard structural equation models. In particular, the emxVARModel function creates vector autoregressive models, and the emxStateSpaceMixtureModel function creates state space mixture models.

A third category of functions encompasses special functions for behavior genetics modeling. Some of these functions are matrix builders, and others are model builders. The lowest-level functions for behavior genetics are emxCholeskyVariance, emxGeneticFactorVariance, emxRelatednessMatrix, and emxKroneckerVariance.

A higher-level set of behavior genetics matrix builders create all the matrices and algebraic statements needed for e.g. the A component of an ACE model. These functions are emxCholeskyComponent and emxGeneticFactorComponent.

The highest-level of behavior genetics functions builds some basic twin models. The primary function for this is emxTwinModel.

Finally, a mixture model helper is provided: emxMixtureModel.

Examples

```
# Make and run a one factor model
## Not run:
require(EasyMx)

data(demoOneFactor)
fmod <- list(G=names(demoOneFactor))
fit1 <- emxFactorModel(fmod, demoOneFactor, run=TRUE)
summary(fit1)

## End(Not run)
```

---

emxCholeskyComponent   *Creates component for a Biometric Cholesky Model*

Description

This function creates all the objects needed for a Cholesky component.

Usage

```
emxCholeskyComponent(x, xname, xvalues, xfree,
                     h=2, hname=paste0('H', xname), hvalues, hlabels)
```

Arguments

x	character vector. The base names of the variables used for the matrix with no repetition for twins (x, y, z not x1, y1, z1, x2, y2, z2).
xname	character. Name of the component matrix.
xvalues	numeric vector. Values of the component matrix.

xfree	logical vector. Whether each element of the component matrix is freely esti-mates.
h	numeric. The number of variables for the relatedness matrix, i.e. the number of critters with relationships
hname	character. Name of the relatedness matrix.
hvalues	numeric vector. Values for the relatedness matrix.
hlabels	character vector. Labels for the relatedness matrix.

Details

This function is a combination of emxCholeskyVariance, emxRelatednessMatrix, and emxKroneck-erVariance.

Value

A list with elements (1) the lower triangular matrix for the Cholesky, (2) the full positive definite variance matrix, (3) the relatedness matrix, and (4) the Kronecker product of the variance matrix and the relatedness matrix.

See Also

emxGeneticFactorComponent

Examples

```
# Create an ACE model in 22 lines
require(EasyMx)
require(OpenMx)
data(twinData)
twinVar = names(twinData)
selVars <- c('ht1', 'bmi1','ht2','bmi2')
mzdzData <- subset(twinData, zyg %in% c(1, 3), c(selVars, 'zyg'))
mzdzData$RCoef <- c(1, NA, .5)[mzdzData$zyg]
nVar = length(selVars)/2
x <- paste0('x', 1:nVar)

acomp <- emxCholeskyComponent(x, 'A', hvalues=c(1, .5, 1), hlabels=c(NA, 'data.RCoef', NA))
ccomp <- emxCholeskyComponent(x, 'C', hvalues=c(1, 1, 1))
ecomp <- emxCholeskyComponent(x, 'E', hvalues=c(1, 0, 1))
totalVar <- mxAlgebra(AKron + CKron + EKron, 'V', dimnames=list(selVars, selVars))
totalMean <- emxMeans(selVars, type='twin')
expect <- mxExpectationNormal(totalVar$name, totalMean$name)
fitfun <- mxFitFunctionML()

comlist <- c(acomp, ccomp, ecomp, list(totalVar, totalMean, expect, fitfun))

model <- mxModel('model', comlist, mxData(mzdzData, 'raw'))
## Not run:
run2 <- mxRun(model)
```

```
## End(Not run)
```

---

emxCholeskyVariance   *Create a variance matrix in Cholesky form*

Description

This function creates a Cholesky variance matrix and associated MxMatrix and MxAlgebra objects.

Usage

```
emxCholeskyVariance(x, name, values=.8, free=TRUE)
```

Arguments

x	character vector. The names of the variables used for the matrix.
name	character. The name of the variance matrix created.
values	numeric vector. The starting values for the lower triangular matrix.
free	logical vector. Whether the lower triangular elements are free.

Details

This is a helper function for creating a matrix that is symmetric and positive definite. Full co-variance matrices are the most common case of these. In a behavior genetics modeling context, Cholesky components can be created for Additive genetics, Common environments, and unique Environments. These are unrestrictive models of the covariances of multiple phenotypes.

Value

A list with two components. The first component is the lower triangular MxMatrix. The second component is an MxAlgebra, the result of which is the positive definite variance matrix.

See Also

[emxGeneticFactorVariance](#)

Examples

```
# Create a Cholesky variance matrix called 'A'
require(EasyMx)
nVar <- 3
x <- paste0('x', 1:nVar)
amat <- emxCholeskyVariance(x, 'A')
```

---

emxCommonPathwayComponent

Creates component for a Biometric Common Pathway Model

Description

UNDER ACTIVE DEVELOPMENT. DO NOT TRUST. This function creates all the objects needed for a Common Pathway component.

Usage

```
emxCommonPathwayComponent(x, xname, xvalues=.8, xfree=TRUE, xlbound=NA, xubound=NA,
                          h=2, hname=paste0('H', xname), hvalues, hlabels)
```

Arguments

x	character vector. The base names of the variables used for the matrix with no repetition for twins (x, y, z not x1, y1, z1, x2, y2, z2).
xname	character. Name of the component matrix.
xvalues	Matrix of fixed and/or starting values of parameters
xfree	Matrix of TRUE (for free) and FALSE (for fixed) values
xlbound	Matrix of numeric lower bounds for parameters
xubound	Matrix of numeric upper bounds for parameters
h	numeric. The number of variables for the relatedness matrix, i.e. the number of critters with relationships
hname	character. Name of the relatedness matrix.
hvalues	numeric vector. Values for the relatedness matrix.
hlabels	character vector. Labels for the relatedness matrix.

Details

This function is a combination of emxCholeskyVariance, emxRelatednessMatrix, and emxKroneckerVariance.

Value

A list with elements (1) the lower triangular matrix for the Cholesky, (2) the full positive definite variance matrix, (3) the relatedness matrix, and (4) the Kronecker product of the variance matrix and the relatedness matrix.

See Also

[emxGeneticFactorComponent](#)

Examples

```
# Create an ACE model in 22 lines
require(EasyMx)
require(OpenMx)
data(twinData)
twinVar = names(twinData)
selVars <- c('ht1', 'bmi1','ht2','bmi2')
mzdzData <- subset(twinData, zyg %in% c(1, 3), c(selVars, 'zyg'))
mzdzData$RCoef <- c(1, NA, .5)[mzdzData$zyg]
nVar = length(selVars)/2
x <- paste0('x', 1:nVar)

acomp <- emxCholeskyComponent(x, 'A', hvalues=c(1, .5, 1), hlabels=c(NA, 'data.RCoef', NA))
ccomp <- emxCholeskyComponent(x, 'C', hvalues=c(1, 1, 1))
ecomp <- emxCholeskyComponent(x, 'E', hvalues=c(1, 0, 1))
totalVar <- mxAlgebra(AKron + CKron + EKron, 'V', dimnames=list(selVars, selVars))
totalMean <- emxMeans(selVars, type='twin')
expect <- mxExpectationNormal(totalVar$name, totalMean$name)
fitfun <- mxFitFunctionML()

comlist <- c(acomp, ccomp, ecomp, list(totalVar, totalMean, expect, fitfun))

model <- mxModel('model', comlist, mxData(mzdzData, 'raw'))
## Not run:
run2 <- mxRun(model)

## End(Not run)
```

---

emxCovariances	*Create a set of covariances*

Description

This function creates a covariance matrix as an MxMatrix or MxPath object.

Usage

```
emxCovariances(x, values, free, path=FALSE, type, name='Variances')
```

Arguments

x	character vector. The names of the variables for which covariances are created.
values	numeric vector. See Details.
free	logical vector. See Details.
path	logical. Whether to return the MxPath object instead of the MxMatrix.
type	character. The kind of covariance structure to create. See Details.
name	The name of the matrix created.

Details

Possible values for the `type` argument are 'independent', 'full', and 'corr'. When `type='independent'`, the remaining arguments are passes to `emxResiduals`. The `values` and `free` arguments are only used when the `type` argument is 'independent'. For all other cases, they are ignored.

When `type='full'`, a full covariance matrix is created. That is, a symmetric matrix is created with all unique elements freely estimated. The starting values for the variances are all 1; for the covariances, all 0.5.

When `type='corr'`, a full correlation matrix is created. That is, a symmetric matrix is created with all unique elements not on the diagonal freely estimated. The starting values for the correlations are all 0.5. The variances are fixed at 1.

Value

Depending on the value of the `path` argument, either an MxMatrix or and MxPath object that can be inspected, modified, and/or included in MxModel objects.

See Also

emxFactorModel, emxGrowthModel

Examples

```
# Create a covariance matrix
require(EasyMx)
manVars <- paste0('x', 1:6)
latVars <- paste0('F', 1:2)
emxCovariances(manVars, type='full')
emxCovariances(latVars, type='corr', path=TRUE)
```

---

emxFactorModel   *Create a factor model*

Description

This function creates a factor model as an MxModel object.

Usage

```
emxFactorModel(model, data, name, run=FALSE, identification, use, ordinal,
    ..., parameterization=c("lisrel", "ifa"), weight = as.character(NA))
emxModelFactor(model, data, name, run=FALSE, identification, use, ordinal,
    ..., parameterization=c("lisrel", "ifa"), weight = as.character(NA))
```

Arguments

`model`	named list. Gives the factor loading pattern. See Details.
`data`	data used for the model
`name`	character. Optional name of the model created.
`run`	logical. Whether to run the model before returning.
`identification`	Not yet implemented. How the model is identified. Currently ignored.
`use`	character vector. The names of the variables to use.
`ordinal`	character vector. The names of the ordinal variables.
`...`	Force later arguments to be named. ... is ignored.
`parameterization`	
	character. Whether to specify the model as a LISREL SEM or as an Item Factor Analysis
`weight`	character. Name of the data column used for sample weights.

Details

The `model` argument must be a named list. The names of the list give the names of the latent variables. Each list element gives the names of the variables that load onto that latent variable. This may sound complicated, but the example below makes this more clear. It is intended to be visually intuitive.

Value

An MxModel.

See Also

emxLoadings

Examples

```
# Example
require(EasyMx)
data(myFADataRaw)
xmap <- list(F1=paste0('x', 1:6), F2=paste0('y', 1:3), F3=paste0('z', 1:3))
## Not run:
mod <- emxFactorModel(xmap, data=myFADataRaw, run=TRUE)

## End(Not run)
```

---

emxGeneticFactorComponent

Creates component for a Genetic Factor Model

Description

This function creates all the objects needed for Genetic Factor component

Usage

```
emxGeneticFactorComponent(x, xname, xvalues=.8, xfree=TRUE, xlbound=NA, xubound=NA,
                          h=2, hname=paste0('H', xname), hvalues, hlabels)
```

Arguments

x	character vector. The base names of the variables used for the matrix with no repetition for twins (x, y, z not x1, y1, z1, x2, y2, z2).
xname	character. Name of the component matrix.
xvalues	numeric vector. Values of the genetic factor loadings.
xfree	logical vector. Whether the genetic factor loadings are free.
xlbound	numeric vector. Lower bounds of the factor loadings.
xubound	numeric vector. Upper bounds of the factor loadings.
h	numeric. The number of variables for the relatedness matrix, i.e. the number of critters with relationships
hname	character. Name of the relatedness matrix.
hvalues	numeric vector. Values for the relatedness matrix.
hlabels	character vector. Labels for the relatedness matrix.

Details

This function is a combination of emxGeneticFactorVariance, emxRelatednessMatrix, and emxKroneckerVariance.

Value

A list with elements (1) the genetic factor loadings matrix, (2) the full positive definite variance matrix, (3) the relatedness matrix, and (4) the Kronecker product of the variance matrix and the relatedness matrix.

See Also

[emxCholeskyComponent](#)

Examples

```
# Create genetic factor A component for DZ twins
require(EasyMx)
xvars <- paste0('x', 1:4)
acomp <- emxGeneticFactorComponent(xvars, 'A', hvalues=c(1, .5, 1))
```

---

emxGeneticFactorVariance

Creates a variance matrix accoring to the Genetic Factor Model

---

Description

This function creates a variance matrix according to the genetic factor model

Usage

```
emxGeneticFactorVariance(x, name, values=.8, free=TRUE, lbound=NA, ubound=NA)
```

Arguments

x	character vector. The names of the variables used for the matrix.
name	character. The name of the variance matrix created.
values	numeric vector. The starting values for the lower triangular matrix.
free	logical vector. Whether the lower triangular elements are free.
lbound	numeric vector. Lower bounds on free parameters.
ubound	numeric vector. Upper bounds on free parameters.

Value

A list with two components. The first component is the factor loadings matrix. The second component is an MxAlgebra, the result of which is the variance matrix implied by the factor loadings.

See Also

emxCholeskyVariance

Examples

```
# Create a genetic factor variance matrix
require(EasyMx)
xvars <- paste0('x', 1:2)
emxGeneticFactorVariance(xvars, 'D')
```

---

emxGrowthModel   *Create a latent growth curve model*

Description

This function creates a latent growth curve model as an MxModel object.

Usage

```
emxGrowthModel(model, data, name, run=FALSE, identification, use, ordinal, times)
emxModelGrowth(model, data, name, run=FALSE, identification, use, ordinal, times)
```

Arguments

model	character or numeric. See Details.
data	data used for the model
name	character. Optional name of the model created.
run	logical. Whether to run the model before returning.
identification	Not yet implemented. How the model is identified. Currently ignored.
use	character vector. The names of the variables to use.
ordinal	character vector. The names of the ordinal variables.
times	optional character or numeric vector. Either the numeric times of measurement or the names of the variables in data that give the times of measurement.

Details

The model argument can be either a character or a number that tells the kind of growth curve to make. If it is a character it currently must be one of "Intercept", "Linear", "Quadratic", "Cubic", "Quartic", or "Quintic", and it produces a polynomial growth curve of the corresponding type. If it is a number, the function produces a polynomial growth curve of the corresponding order. Zero is an intercept only, one is linear, two is quadratic; and so on.

When missing, the times are assumed to start at zero and increment by one until the number of variables is completed.

Value

An MxModel

See Also

emxFactorModel, emxGrowthModel

Examples

```
# Example
require(EasyMx)
data(myLongitudinalData)
## Not run:
mod <- emxGrowthModel('Linear', data=myLongitudinalData, use=names(myLongitudinalData), run=TRUE)

## End(Not run)
```

---

emxIndependentPathwayComponent

Creates component for a Biometric Independent Pathway Model

Description

UNDER ACTIVE DEVELOPMENT. DO NOT TRUST. This function creates all the objects needed for an Independent Pathway component.

Usage

```
emxIndependentPathwayComponent(x, xname, xvalues=.8, xfree=TRUE, xlbound=NA, xubound=NA,
                        h=2, hname=paste0('H', xname), hvalues, hlabels)
```

Arguments

x	character vector. The base names of the variables used for the matrix with no repetition for twins (x, y, z not x1, y1, z1, x2, y2, z2).
xname	character. Name of the component matrix.
xvalues	Matrix of fixed and/or starting values of parameters
xfree	Matrix of TRUE (for free) and FALSE (for fixed) values
xlbound	Matrix of numeric lower bounds for parameters
xubound	Matrix of numeric upper bounds for parameters
h	numeric. The number of variables for the relatedness matrix, i.e. the number of critters with relationships
hname	character. Name of the relatedness matrix.
hvalues	numeric vector. Values for the relatedness matrix.
hlabels	character vector. Labels for the relatedness matrix.

Details

This function is a combination of emxCholeskyVariance, emxRelatednessMatrix, and emxKroneckerVariance.

Value

A list with elements (1) the lower triangular matrix for the Cholesky, (2) the full positive definite variance matrix, (3) the relatedness matrix, and (4) the Kronecker product of the variance matrix and the relatedness matrix.

See Also

emxGeneticFactorComponent

Examples

```
# Create an ACE model in 22 lines
require(EasyMx)
require(OpenMx)
data(twinData)
twinVar = names(twinData)
selVars <- c('ht1', 'bmi1','ht2','bmi2')
mzdzData <- subset(twinData, zyg %in% c(1, 3), c(selVars, 'zyg'))
mzdzData$RCoef <- c(1, NA, .5)[mzdzData$zyg]
nVar = length(selVars)/2
x <- paste0('x', 1:nVar)

acomp <- emxCholeskyComponent(x, 'A', hvalues=c(1, .5, 1), hlabels=c(NA, 'data.RCoef', NA))
ccomp <- emxCholeskyComponent(x, 'C', hvalues=c(1, 1, 1))
ecomp <- emxCholeskyComponent(x, 'E', hvalues=c(1, 0, 1))
totalVar <- mxAlgebra(AKron + CKron + EKron, 'V', dimnames=list(selVars, selVars))
totalMean <- emxMeans(selVars, type='twin')
expect <- mxExpectationNormal(totalVar$name, totalMean$name)
fitfun <- mxFitFunctionML()

comlist <- c(acomp, ccomp, ecomp, list(totalVar, totalMean, expect, fitfun))

model <- mxModel('model', comlist, mxData(mzdzData, 'raw'))
## Not run:
run2 <- mxRun(model)

## End(Not run)
```

---

emxKroneckerVariance   *Creates a large Variance matrix by Kroneckering two smaller matrices*

Description

This function creates the wide format variance matrix when combined with a relatedness matrix

Usage

```
emxKroneckerVariance(h, v, name)
```

Arguments

h	MxMatrix. Left hand side of the Kronecker product. Typically the relatedness matrix.
v	MxMatrix. Right hand side of the Kronecker product. Typically the variance matrix.
name	character. Name of the resulting large matrix.

Details

In many behavior genetic models, a relationship matrix is combined with a base variance matrix. The combination is done with a Kronecker product so that the variance exists (possibly weighted by zero or another number) for each member of the relationship.

Value

MxAlgebra

See Also

emxRelatednessMatrix

Examples

```
# Create a loadings matrix
require(EasyMx)
x <- letters[23:26]
amat <- emxCholeskyVariance(x, 'A')
ahmat <- emxRelatednessMatrix(2, c(1, .5, 1), name='AH')
ab <- emxKroneckerVariance(ahmat, amat[[2]], 'AB')
```

---

emxLoadings	*Create a factor loadings matrix*

Description

This function creates a factor loadings matrix as an MxMatrix or MxPath object.

Usage

```
emxLoadings(x, values=.8, free=TRUE, path=FALSE)
```

Arguments

x	named list. Gives the factor loading pattern. See Details.
values	numeric vector. The starting values for the nonzero loadings.
free	logical vector. Whether the nonzero loadings are free.
path	logical. Whether to return the MxPath object instead of the MxMatrix.

Details

The x argument must be a named list. The names of the list give the names of the latent variables. Each list element gives the names of the variables that load onto that latent variable. This may sound complicated, but the example below makes this more clear. It is intended to be visually intuitive.

Value

Depending on the value of the `path` argument, either an MxMatrix or and MxPath object that can be inspected, modified, and/or included in MxModel objects.

See Also

[emxFactorModel](#)

Examples

```
# Create a loadings matrix
require(EasyMx)
xmap <- list(F1=paste0('x', 1:6), F2=paste0('y', 1:3), F3=paste0('z', 1:3))
emxLoadings(xmap)
emxLoadings(xmap, path=TRUE)
```

---

emxMeans   *Create a set of means*

Description

This function creates a means matrix as an MxMatrix or MxPath object.

Usage

```
emxMeans(x, values=0, free=TRUE, path=FALSE, type='saturated', name, column=TRUE, labels)
```

Arguments

x	character vector. The names of the variables for which means are created.
values	numeric vector. See Details.
free	logical vector. See Details.
path	logical. Whether to return the MxPath object instead of the MxMatrix.
type	character. The kind of covariance structure to create. See Details.
name	The name of the matrix created.
column	logical. Whether to create the means vector as a column or row.
labels	character vector. Optional labels for the means.

Details

Possible values for the type argument are 'saturated', 'equal', 'twin', 'special'.

Value

Depending on the value of the path argument, either an MxMatrix or and MxPath object that can be inspected, modified, and/or included in MxModel objects.

See Also

[emxFactorModel](), [emxGrowthModel]()

Examples

```
# Create a covariance matrix
require(EasyMx)
manVars <- paste0('x', 1:6)
emxMeans(manVars, type='saturated')
```

---

emxMixtureModel   *Create a mixture model*

Description

This function creates a mxiture model as an MxModel object.

Usage

```
emxMixtureModel(model, data, run=FALSE, p=NA, ...)
emxModelMixture(model, data, run=FALSE, p=NA, ...)
```

Arguments

model	list. The MxModel objects that compose the mixture.
data	data used for the model
run	logical. Whether to run the model before returning.
p	character. Optional name of the mixing proportions matrix.
...	Further Mx Objects passed into the mixture model.

Details

The model argument is list of MxModel objects. These are the classes over which the mixture model operates.

The p argument is optional. If not specified, the function will create and properly scale the mixing proportions for you. If specified, the Mx Object that gives the mixing proportions should be a column vector (one-column matrix).

Value

An MxModel.

See Also

[emxLoadings](#)

Examples

```
# Factor Mixture Example
require(EasyMx)
data(myFADataRaw)
xmap1 <- list(F1=paste0('x', 1:6), F2=paste0('y', 1:3), F3=paste0('z', 1:3))
mod1 <- emxFactorModel(xmap1, data=myFADataRaw, name='m1')

xmap2 <- list(F1=c(paste0('x', 1:6), paste0('y', 1:3), paste0('z', 1:3)))
mod2 <- emxFactorModel(xmap2, data=myFADataRaw, name='m2')

mod <- emxMixtureModel(list(mod1, mod2), data=myFADataRaw)
# To estimate parameters either
#  1. mod <- mxRun(mod)    or
#  2. include run=TRUE in the arguments above
summary(mod)
coef(mod)

# Latent Profile Example
require(EasyMx)

m1 <- omxSaturatedModel(demoOneFactor)[[1]]
m1 <- mxRename(m1, 'profile1')

m2 <- omxSaturatedModel(demoOneFactor)[[1]]
m2 <- mxRename(m2, 'profile2')

mod <- emxMixtureModel(list(m1, m2), data=demoOneFactor)
# To estimate parameters either
#  1. mod <- mxRun(mod)    or
#  2. include run=TRUE in the arguments above
summary(mod)
coef(mod)

mxGetExpected(mod$profile1, 'covariance')
mxGetExpected(mod$profile1, 'means')
mxGetExpected(mod$profile2, 'covariance')
mxGetExpected(mod$profile2, 'means')
```

emxRegressionModel *Create a regression model*

Description

This function creates a regression model as an MxModel object.

Usage

```
emxRegressionModel(model, data, type='Steven', run, ...)
emxModelRegression(model, data, type='Steven', run, ...)
```

Arguments

model	formula. See Details.
data	data used for the model
run	logical. Whether to run the model before returning.
type	character. Either 'Steven' or 'Joshua'. See Details.
...	Further named arguments to be passed to lm for the formula

Details

The model argument is a formula identical to what is used in lm.

The type argument switches the kind of regression model that is specified. When there are no missing data, the two versions will estimate the same regression parameters but type='Steven' will estimate addition parameters that are not estimated by type='Joshua'. The type='Steven' model is due to Steven Boker and many others. It estimates more parameters than a typical regression analysis and has a different set of assumptions. More exactly, type='Steven' models the outcome and all of the predictors as a multivariate Normal distribution. By contrast, type='Joshua' is due to Joshua Pritikin and exactly replicates the typical regression model with its usual assumptions. In particular, type='Joshua' models the regression residual as a univariate Normal distribution. Predictors are assumed to have no measurement error (see Westfall & Yarkoni, 2016).

The benefit of type='Steven' is that it handles missing data with full-information maximum likelihood (FIML; Enders & Bandalos, 2001), at the cost of using a different model with different assumptions from ordinary least squares regression. The benefit of type='Joshua' is that it exactly replicates regression as a maximum likelihood model, at the cost of having the same weakness in terms of missing data as OLS regression.

Value

An MxModel.

References

Enders, C. K. & Bandalos, D. L. (2001). The relative performance of full information maximum likelihood estimation for missing data in structural equation models. <i>Structural Equation Modeling, 8</i>(3), 430-457.

Westfall, J. & Yarkoni, T. (2016). Statistically controlling for confounding constructs is harder than you think. <i>PLoS ONE, 11</i>(3). doi:10.1371/journal.pone.0152719

See Also

lm

Examples

```
# Example
require(EasyMx)
data(myRegDataRaw)
myrdr <- myRegDataRaw
myrdr[1, 4] <- NA

## Not run:
run <- emxRegressionModel(y~1+x*z, data=myrdr, run=TRUE)
summary(run)

## End(Not run)

summary(lm(y~1+x*z, data=myrdr))
```

---

emxRelatednessMatrix   *Create a relatedness matrix*

Description

This function creates a relatedness matrix as an MxMatrix, often used in behavior genetics modeling.

Usage

```
emxRelatednessMatrix(nvar, values, labels, name='h')
```

Arguments

nvar	numeric. The number of variables for the matrix, i.e. the number of rows or columns.
values	numeric vector. Values used in the matrix.

`labels`	character vector. Labels of the elements in the matrix. See Details.
`name`	character. The name of the matrix created.

Details

The `labels` argument can be used to create a "definition variable" which populates the value from one of the data columns for each row in the data. In this context, if the genetic relatedness coefficient between a pair of individuals is given by a column in the data then that information can be used to create in the relatedness matrix. Alternatively, multiple groups can be created

Value

An MxMatrix object.

See Also

[emxGeneticFactorVariance](#)

Examples

```
# Create a Cholesky variance matrix called 'A'
require(EasyMx)
ahmat <- emxRelatednessMatrix(2, c(1, .5, 1), labels=c(NA, 'data.RCoef', NA), name='AH')
# data.RCoef creates a definition variable and ignores the .5 value.
```

---

emxResiduals	*Create a residual variances matrix*

Description

This function creates a factor loadings matrix as an MxMatrix or MxPath object.

Usage

```
emxResiduals(x, values=.2, free=TRUE, lbound=NA, ubound=NA, path=FALSE, type='unique')
```

Arguments

`x`	character vector. The names of the variables for which residual variances are created.
`values`	numeric vector. The starting values for the variances.
`free`	logical vector. Whether the variances are free.
`lbound`	numeric vector. Lower bounds for the variances.
`ubound`	numeric vector. Upper bounds for the variances.
`path`	logical. Whether to return the MxPath object instead of the MxMatrix.
`type`	character. The kind of residual variance structure to create. See Details.

Details

Possible values for the `type` argument are 'unique' and 'identical'. When `type='unique'`, each residual variances is a unique free parameter. When `type='identical'`, all of the residual variances are given by a single free parameter. In this case, all the residual variances are constrained to be equal. However, no linear or non-liniear contraint function is used. Rather, a single parameter occurs in multiple locations by using the same label.

Value

Depending on the value of the `path` argument, either an MxMatrix or and MxPath object that can be inspected, modified, and/or included in MxModel objects.

See Also

[emxFactorModel](), [emxGrowthModel]()

Examples

```
# Create a residual variance matrix
require(EasyMx)
manVars <- paste0('x', 1:6)
emxResiduals(manVars, lbound=1e-6)
emxResiduals(manVars, type='identical')
emxResiduals(manVars, path=TRUE)
```

---

emxStateSpaceMixtureClassify

Classify time series in a state space mixture model

Description

This function classifies time series (usually people) in a state space mixture model.

Usage

```
emxStateSpaceMixtureClassify(model)
```

Arguments

model   MxModel. The output from [emxStateSpaceMixtureModel]()

Details

This is a helper function for state space mixture modeling. The function will almost exclusively be used in conjunction with emxStateSpaceMixtureModel. The present function takes a state space mixture model as input, and returns detailed information about the most likely class for each unique ID.

Value

A named list with elements

estimated_classes

A vector of the most likely class for each person. Dimension is people.

joint_m2ll   A matrix of joint minus two summed log likelihoods of each person *and* each class. Dimension is people by classes.

m2ll   A matrix of minus two summed log likelihoods of each person *given* each class. Dimension is people by classes.

likelihood   An array of the likelihoods (i.e., probability densities) of each combination of time point, person, and class. Dimension is time points by people by classes.

See Also

emxStateSpaceMixtureModel , emxMixtureModel

Examples

```
# Example
require(EasyMx)
data(myFADataRaw)

ds0 <- myFADataRaw[,1:3]

# Make a VAR Model
vm <- emxVARModel(data=ds0, use=names(ds0), name='varmodel')

# Re-label parameters to have different AR parameters
# for class 1 and class 2
vm1 <- OpenMx::omxSetParameters(vm, labels=vm$Dynamics$labels,
    newlabels=paste0(vm$Dynamics$labels, '_k1'), name='klass1')
vm2 <- OpenMx::omxSetParameters(vm, labels=vm$Dynamics$labels,
    newlabels=paste0(vm$Dynamics$labels, '_k2'), name='klass2')

# Pretend you have a data set of 50 people
# each measured 10 times on 3 variables
ds1 <- myFADataRaw[, 1:3]
ds1$id <- rep(1:50, each=nrow(myFADataRaw)/50)

## Not run:
# Make the state space mixture model
ssmm <- emxStateSpaceMixtureModel(model=list(vm1, vm2),
    data=ds1, ID='id')

# Fit model
ssmmr <- mxRun(ssmm)

# Extract estimated classes and diagnostics
eclasses <- emxStateSpaceMixtureClassify(ssmmr)
```

```
## End(Not run)
```

---

emxStateSpaceMixtureModel

Create a state space mixture model

Description

This function creates a state space mixture model as an MxModel object.

Usage

```
emxStateSpaceMixtureModel(model, data, name, run=FALSE, use, ID, time, ...)
emxModelStateSpaceMixture(model, data, name, run=FALSE, use, ID, time, ...)
```

Arguments

model	list of MxModel objects, each of which should be a state space model
data	data used for the model
name	character. Optional name of the model created.
run	logical. Whether to run the model before returning.
use	character vector. The names of the variables to use. Currently ignored.
ID	character. Name of variable that identifies each unique person.
time	character. Name of the variable that gives the time of each obsevation. Currently ignored.
...	Force later arguments to be named. ... is ignored.

Details

The idea of state space mixture modeling is to model multiple, multivariate time series while allowing for qualitative differences between the time series. Suppose you have a multivariate time series for several people. You think some people should have the same time series model, but not everyone. You think there should be a small number of homogeneous sets of people that follow the same time series model, but you do not know which people or the exact parameter values of the candidate time series models. This function presents one solution to this problem.

State space mixture modeling begins with a set of candidate state space models, and uses these state space models as the mixture classes. The goal is to simultaneously estimate the free parameters of the state space models, and estimate which multivariate time series (e.g., perseon) belongs to which mixture class.

The component state space models may share some free parameters or none. Note that free parameters with the same name are constrained to be equal across all models. Conversely, unnamed free parameters are given unique names and are allowed to differ for each person-mixture combination, which creates a very large number of free parameters. We strongly encourage you to name all of your free parameters in the model list to avoid melting your computer's CPU.

The `model` argument currently must be a list. The elements of the list should be MxModel objects. Each list element forms a mixture class in the final model.

This function creates a multigroup mixture model where the mixture classes are the elements of the `model` list argument. Each unique `ID` forms a group.

The `data` argument can be a list of `data.frame` objects with one element for each `ID`, or a single `data.frame` with an `ID` variable that separates groups.

Value

An MxModel.

See Also

[emxStateSpaceMixtureClassify](#) , [emxMixtureModel](#)

Examples

```
# Example
require(EasyMx)
data(myFADataRaw)

ds0 <- myFADataRaw[,1:3]

# Make a VAR Model
vm <- emxVARModel(data=ds0, use=names(ds0), name='varmodel')

# Re-label parameters to have different AR parameters
# for class 1 and class 2
vm1 <- OpenMx::omxSetParameters(vm, labels=vm$Dynamics$labels,
    newlabels=paste0(vm$Dynamics$labels, '_k1'), name='klass1')
vm2 <- OpenMx::omxSetParameters(vm, labels=vm$Dynamics$labels,
    newlabels=paste0(vm$Dynamics$labels, '_k2'), name='klass2')

# Pretend you have a data set of 50 people
# each measured 10 times on 3 variables
ds1 <- myFADataRaw[, 1:3]
ds1$id <- rep(1:50, each=nrow(myFADataRaw)/50)

## Not run:
# Make and fit the state space mixture model
ssmm <- emxStateSpaceMixtureModel(model=list(vm1, vm2),
    data=ds1, ID='id', run=TRUE)

## End(Not run)
```

---

emxThresholds   *Create a set of thresholds for ordinal data*

Description

This function creates a threshold matrix as an MxMatrix object.

Usage

```
emxThresholds(data, ordinalCols, deviation=TRUE)
```

Arguments

data   The data frame or matrix for which thresholds should be created.

ordinalCols   optional character vector. The names of the ordinal variables in the data.

deviation   logical. Return the list of OpenMx objects needed for the deviation form of the
              threholds (default) or just the raw thresholds matrix

Value

An MxMatrix giving the thresholds.

See Also

emxFactorModel, emxGrowthModel

Examples

```
# Example
require(EasyMx)
data(jointdata)
jointdata[, c(2, 4, 5)] <- mxFactor(jointdata[,c(2, 4, 5)],
levels=sapply(jointdata[,c(2, 4, 5)], function(x){sort(unique(x))}))
emxThresholds(jointdata, c(FALSE, TRUE, FALSE, TRUE, TRUE))
```

---

emxTwinModel   *Creates behavior genetics Twin Model*

Description

This function creates an MxModel and associated objects for a basic Twin model.

Usage

```
emxTwinModel(model, relatedness, data, run=FALSE, use, name='model', components='ACE')
emxModelTwin(model, relatedness, data, run=FALSE, use, name='model', components='ACE')
```

Arguments

model	Description of the model. Currently ignored.
relatedness	Description of the relatedness patterns. Currently the name of the variable that gives the coefficient of relatedness.
data	data.frame or matrix. The data set used in the model.
run	logical. Whether to run the model before returning.
use	character vector. Names of the variables used in the model.
name	character. Name of the model.
components	character. Name of the variance components to include. Current valid options are 'ACE' and 'ADE'

Details

Because the model argument is ignored and the relatedness argument has limited use, this function only constructs a very basic and rigid Twin model. It creates a Cholesky model with A, C, and E components or a Cholesky model with A, D, and E components. The means are constrained equal across twins.

Value

MxModel.

See Also

[emxFactorModel](#)

Examples

```
# Create an ACE model in 10 lines
# 8 of those are data handling.
# 2 are the actual model.
require(EasyMx)
```

```
require(OpenMx)
data(twinData)
twinVar = names(twinData)
selVars <- c('ht1', 'bmi1','ht2','bmi2')
mzdzData <- subset(twinData, zyg %in% c(1, 3), c(selVars, 'zyg'))
mzdzData$RCoef <- c(1, NA, .5)[mzdzData$zyg]

## Not run:
run3 <- emxTwinModel(model='Cholesky', relatedness='RCoef',
data=mzdzData, use=selVars, run=TRUE, name='TwCh')

## End(Not run)
```

---

emxVarianceComponents   *Creates Variance Components Model*

Description

This function creates a variance components model as an MxModel object.

Usage

```
emxVarianceComponents(model, data, run)
```

Arguments

model	MxModel. See Details.
data	matrix or data.frame. The used in the model
run	logical. Whether to run the model before returning.

Details

This function does not really do anything currently. Do not use it.

Value

MxModel.

See Also

[emxFactorModel](#)

Examples

```
# Create a loadings matrix
require(EasyMx)
```

---

emxVARModel   *Create a vector autoregressive (VAR) model*

Description

This function creates a vector autoregressive (VAR) model as an MxModel object.

Usage

```
emxVARModel(model, data, name, run=FALSE, use, ID)
emxModelVAR(model, data, name, run=FALSE, use, ID)
```

Arguments

model	Currently ignored, but later will specify particular kinds of VAR models
data	data used for the model
name	character. Optional name of the model created.
run	logical. Whether to run the model before returning.
use	character vector. The names of the variables to use. Currently ignored.
ID	character. Name of variable that identifies each unique person.

Details

The purpose of this function is to quickly specify a vector autoregressive model. It is currently in the early stages of development and might change considerable with regard to the model argument and the ID argument.

Value

An MxModel.

See Also

emxStateSpaceMixtureModel , emxFactorModel

Examples

```
# Example
require(EasyMx)
data(myFADataRaw)
ds0 <- myFADataRaw[,1:3]

# Make a VAR Model
vm <- emxVARModel(data=ds0, use=names(ds0), name='varmodel')
```

Index