OpenMx Reference Manual

February 13, 2012

Date 2012-02-03

Title Multipurpose Software for Statistical Modeling

Author Steven M. Boker, Michael C. Neale, Hermine H. Maes, Michael J. Wilde, Michael Spiegel, Timothy R. Brick, Ryne Estabrook, Timothy C. Bates, Paras Mehta, Timo von Oertzen, Ross J. Gore, Michael D. Hunter, Daniel C. Hackett, Julian Karch, Andreas Brandmaier

Maintainer OpenMx Development Team <openmx-developers@list.mail.virginia.edu>

URL http://openmx.psyc.virginia.edu

Description The OpenMx Project intends to rewrite and extend the popular statistical package Mx to address the challenges facing a large range of modern statistical problems such as: the difficulty of measuring behavioral traits; the availability of technologies - such as such as magnetic resonance imaging, continuous physiological monitoring and microarrays - which generate extremely large amounts of data often with complex time-dependent patterning; and increased sophistication in the statistical models used to analyze the data.

License Apache License 2.0

Depends methods

Suggests snowfall

LazyLoad yes

LazyData yes

Collate MxData.R DefinitionVars.R MxReservedNames.R MxNamespace.R MxSearchReplace.R MxFlatSearchReplace.R MxUntitled.R MxCharOrNumber.R MxAlgebraFunctions.R MxExponential.R MxMatrix.R DiagMatrix.R FullMatrix.R IdenMatrix.R LowerMatrix.R SdiagMatrix.R StandMatrix.R SymmMatrix.R UnitMatrix.R ZeroMatrix.R MxAlgebra.R MxCycleDetection.R MxAlgebraConvert.R MxAlgebraTransform.R MxSquareBracket.R MxEval.R MxRename.R MxPath.R MxObjectiveMetaData.R MxRAMMetaData.R MxObjectiveFunction.R MxBounds.R MxConstraint.R MxInterval.R MxTypes.R MxModel.R MxRAMModel.R MxModelDisplay.R MxFlatModel.R MxMultiModel.R MxModelFunctions.R MxModelParameters.R MxUnitTesting.R MxAlgebraObjective.R MxRowObjective.R MxFIMLObjective.R MxMLObjective.R MxRAMObjective.R MxLISRELObjec-

tive.R MxRObjective.R MxApply.R MxRun.R MxRunHelperFunctions.R MxSummary.R MxCompare.R MxSwift.R MxOptions.R MxThreshold.R OriginalMx.R MxGraph.R MxGraphviz.R MxDeparse.R MxCommunication.R MxRestore.R MxVersion.R MxErrorPool.R MxPPML.R MxRAMtoML.R MxDiff.R MxErrorHandling.R MxDetectCores.R zzz.R

Version 1.2.0-1919

R topics documented:

`cvectorize` *Vectorize By Column*

Description

This function returns the vectorization of an input matrix in a column by column traversal of the matrix. The output is returned as a column vector.

Usage

```
cvectorize(x)
```

Arguments

x	an input matrix.

See Also

`rvectorize`, `vech`, `vechs`

Examples

```
cvectorize(matrix(1:9, 3, 3))
cvectorize(matrix(1:12, 3, 4))
```

diag2vec *Extract Diagonal of a Matrix*

Description

Given an input matrix, diag2vec returns a column vector of the elements along the diagonal.

Usage

```
diag2vec(x)
```

Arguments

x	an input matrix.

Details

Similar to the function diag, except that the input argument is always treated as a matrix (i.e., it doesn't have diag()'s functions of returning an Identity matrix from an nrow specification, nor to return a matrix wrapped around a diagonal if provided with a vector). To get vector2matrix functionality, call vec2diag.

See Also

vec2diag

Examples

```
diag2vec(matrix(1:9, nrow=3))
#      [,1]
# [1,]    1
# [2,]    5
# [3,]    9

diag2vec(matrix(1:12, nrow=3, ncol=4))
#      [,1]
# [1,]    1
# [2,]    5
# [3,]    9
```

eigenvec *Eigenvector/Eigenvalue Decomposition*

Description

eigenval computes the real parts of the eigenvalues of a square matrix. eigenvec computes the real parts of the eigenvectors of a square matrix. ieigenval computes the imaginary parts of the eigenvalues of a square matrix. ieigenvec computes the imaginary parts of the eigenvectors of a square matrix. eigenval and ieigenval return nx1 matrices containing the real or imaginary parts of the eigenvalues, sorted in decreasing order of the modulus of the complex eigenvalue. For eigenvalues without an imaginary part, this is equivalent to sorting in decreasing order of the absolute value of the eigenvalue. (See Mod for more info.) eigenvec and ieigenvec return nxn matrices, where each column corresponds to an eigenvector. These are sorted in decreasing order of the modulus of their associated complex eigenvalue.

Usage

```
eigenval(x)
eigenvec(x)
ieigenval(x)
ieigenvec(x)
```

Arguments

x	the square matrix whose eigenvalues/vectors are to be calculated.

Details

Eigenvectors returned by eigenvec and ieigenvec are normalized to unit length.

See Also

eigen

Examples

```
A <- mxMatrix(values = runif(25), nrow = 5, ncol = 5, name = 'A')
G <- mxMatrix(values = c(0, -1, 1, -1), nrow=2, ncol=2, name='G')

model <- mxModel(A, G, name = 'model')

mxEval(eigenvec(A), model)
mxEval(eigenvec(G), model)
mxEval(eigenval(A), model)
mxEval(eigenval(G), model)
mxEval(ieigenvec(A), model)
mxEval(ieigenvec(G), model)
mxEval(ieigenval(A), model)
```

```
mxEval(ieigenval(G), model)
```

`mxAlgebra` *Create MxAlgebra Object*

Description

This function creates a new MxAlgebra object.

Usage

```
mxAlgebra(expression, name = NA, dimnames = NA)
```

Arguments

`expression`	An R expression of OpenMx-supported matrix operators and matrix functions.
`name`	An optional character string indicating the name of the object.
`dimnames`	list. The dimnames attribute for the algebra: a list of length 2 giving the row and column names respectively. An empty list is treated as NULL, and a list of length one as row names. The list can be named, and the list names will be used as names for the dimensions.

Details

The mxAlgebra function is used to create algebraic expressions that operate on one or more MxMatrix objects. To evaluate an MxAlgebra object, it must be placed in an MxModel object, along with all referenced `MxMatrix` objects and the `mxAlgebraObjective` function. The `mxAlgebraObjective` function must reference the `MxAlgebra` object to be evaluated by name.

The following operators are supported in mxAlgebra:

`solve()` Inversion

`t()` Transposition

`^` Elementwise powering

`%^%` Kronecker powering

`+` Addition

`-` Subtraction

`%*%` Matrix Multiplication

`*` Element or dot product

`/` Element division

`%x%` Kronecker product

`%&%` Quadratic product

The following functions are supported in mxAlgebra:

`cbind` Horizontal adhesion

`rbind` Vertical adhesion

`det` Determinant

`tr` Trace

`sum` Sum

`prod` Product

`max` Maximum

`min` Min

`abs` Absolute value

`sin` Sine

`sinh` Hyperbolic sine

`cos` Cosine

`cosh` Hyperbolic cosine

`tan` Tangent

`tanh` Hyperbolic tangent

`exp` Exponent

`log` Natural Logarithm

`sqrt` Square root

`eigenval` Eigenvalues of a square matrix. Usage: eigenval(x); eigenvec(x); ieigenval(x); ieigenvec(x)

`rvectorize` Vectorize by row

`cvectorize` Vectorize by column

`vech` Half-vectorization

`vechs` Strict half-vectorization

`vec2diag` Create a diagonal matrix

`diag2vec` Extract diagonal from matrix

`omxMnor` Multivariate Normal Integration

`omxAllInt` All cells Multivariate Normal Integration

`omxNot` Perform unary negation on a matrix

`omxAnd` Perform binary and on two matrices

`omxOr` Perform binary or on two matrices

`omxGreaterThan` Perform binary greater on two matrices

`omxLessThan` Perform binary less than on two matrices

`omxApproxEquals` Perform binary equals to (within a specified epsilon) on two matrices

`omxExponential` Matrix Exponential

Value

Returns a new MxAlgebra object.

References

The OpenMx User's guide can be found at http://openmx.psyc.virginia.edu/documentation.

See Also

MxAlgebra for the S4 class created by mxAlgebra. mxAlgebraObjective for an objective functions which takes an MxAlgebra or MxMatrix object as the function to be minimized. MxMatrix and mxMatrix for objects which may be entered in the 'expression' argument and the function that creates them. More information about the OpenMx package may be found here.

Examples

```
A <- mxMatrix("Full", nrow = 3, ncol = 3, values=2, name = "A")

# Simple example: algebra B simply evaluates to the matrix A
B <- mxAlgebra(A, name = "B")

# Compute A + B
C <- mxAlgebra(A + B, name = "C")

# Compute sin(C)
D <- mxAlgebra(sin(C), name = "D")

# Make a model and evaluate the mxAlgebra object 'D'
A <- mxMatrix("Full", nrow = 3, ncol = 3, values=2, name = "A")
model <- mxModel("AlgebraExample", A, B, C, D )
fit   <- mxRun(model)
mxEval(D, fit)


# Numbers in mxAlgebras are upgraded to 1x1 matrices
# Example of Kronecker powering (%^%) and multiplication (%*%)
A  <- mxMatrix(type="Full", nrow=3, ncol=3, value=c(1:9), name="A")
m1 <- mxModel("kron", A, mxAlgebra(A %^% 2, name="KroneckerPower"))
mxRun(m1)$KroneckerPower
# Running kron
# mxAlgebra 'KroneckerPower'
# @formula:  A %^% 2
# @result:
#      [,1] [,2] [,3]
# [1,]    1   16   49
# [2,]    4   25   64
# [3,]    9   36   81
```

MxAlgebra-class *MxAlgebra Class*

Description

MxAlgebra is an S4 class. An MxAlgebra object is a named entity. New instances of this class can be created using the function mxAlgebra.

Details

The MxAlgebra class has the following slots:

name	-	The name of the object
formula	-	The R expression to be evaluated
result	-	a matrix with the computation result

The 'name' slot is the name of the MxAlgebra object. Use of MxAlgebra objects in the `mxConstraint` function or an objective function requires reference by name.

The 'formula' slot is an expression containing the expression to be evaluated. These objects are operated on or related to one another using one or more operations detailed in the `mxAlgebra` help file.

The 'result' slot is used to hold the results of computing the expression in the 'formula' slot. If the containing model has not been executed, then the 'result' slot will hold a 0 x 0 matrix. Otherwise the slot will store the computed value of the algebra using the final estimates of the free parameters.

Slots may be referenced with the @ symbol. See the documentation for Classes and the examples in the `mxAlgebra` document for more information.

References

The OpenMx User's guide can be found at http://openmx.psyc.virginia.edu/documentation.

See Also

`mxAlgebra`, `mxMatrix`, `MxMatrix`

mxAlgebraObjective *Function to Create MxAlgebraObjective Object*

Description

This function creates a new MxAlgebraObjective object.

Usage

```
mxAlgebraObjective(algebra, numObs = NA, numStats = NA)
```

Arguments

algebra	A character string indicating the name of an MxAlgebra or MxMatrix object to use for optimization.
numObs	(optional) An adjustment to the total number of observations in the model.
numStats	(optional) An adjustment to the total number of observed statistics in the model.

Details

Objective functions are functions for which free parameter values are chosen such that the value of the objective function is minimized. While the other objective functions in OpenMx are packaged with a function to be optimized (i.e., maximum likelihood), the mxAlgebraObjective function uses the referenced MxAlgebra or MxMatrix object as the function to be minimized.

If a model's primary objective function is a mxAlgebraObjective objective function, then the referenced algebra in the objective function must return a 1 x 1 matrix (when using OpenMx's default optimizer). There is no restriction on the dimensions of an objective function that is not the primary, or 'topmost', objective function.

To evaluate an algebra objective function, place the following objects in a MxModel object: a MxAlgebraObjective, MxAlgebra and MxMatrix entities referenced by the MxAlgebraObjective, and optional MxBounds and MxConstraint entities. This model may then be evaluated using the mxRun function. The results of the optimization may be obtained using the mxEval function on the name of the MxAlgebra, after the model has been run.

Value

Returns a new MxAlgebraObjective object. MxAlgebraObjective objects should be included with models with referenced MxAlgebra and MxMatrix objects.

References

The OpenMx User's guide can be found at http://openmx.psyc.virginia.edu/documentation.

See Also

mxAlgebra to create an algebra suitable as a reference function to be minimized. More information about the OpenMx package may be found here.

Examples

```
# Create a matrix 'A' with no free parameters
A <- mxMatrix('Full', nrow = 1, ncol = 1, values = c(0), name = 'A')

# Create an algebra 'B', which defines the expression A + A
B <- mxAlgebra(A + A, name = 'B')

# Define the objective function for algebra 'B'
```

```
objective <- mxAlgebraObjective('B')

# Place the algebra, its associated matrix and
# its objective function in a model
model <- mxModel(A, B, objective)

# Evalulate the algebra
modelRun <- mxRun(model)

# View the results
modelRun@output
```

---

mxBounds *Create MxBounds Object*

Description

This function creates a new MxBounds object.

Usage

```
mxBounds(parameters, min = NA, max = NA)
```

Arguments

parameters	A character vector indicating the names of the parameters on which to apply bounds.
min	A numeric value for the lower bound. NA means use default value.
max	A numeric value for the upper bound. NA means use default value.

Details

Creates a set of boundaries or limits for a parameter or set of parameters. Parameters may be any free parameter or parameters from an MxMatrix object. Parameters may be referenced either by name or by referring to their position in the 'spec' matrix of an MxMatrix object.

Minima and maxima may be specified as scalar numeric values.

Value

Returns a new MxBounds object. If used as an argument in an MxModel object, the parameters referenced in the 'parameters' argument must also be included prior to optimization.

References

The OpenMx User's guide can be found at http://openmx.psyc.virginia.edu/documentation.

See Also

MxBounds for the S4 class created by mxBounds. MxMatrix and mxMatrix for free parameter specification. More information about the OpenMx package may be found here.

Examples

```
#Create lower and upper bounds for parameters 'A' and 'B'
bounds <- mxBounds(c('A', 'B'), 3, 5)

#Create a lower bound of zero for a set of variance parameters
varianceBounds <- mxBounds(c('Var1', 'Var2', 'Var3'), 0)
```

MxBounds-class *MxBounds Class*

Description

MxBounds is an S4 class. New instances of this class can be created using the function mxBounds.

Details

The MxBounds class has the following slots:

min	-	The lower bound
max	-	The upper bound
parameters	-	The vector of parameter names

The 'min' and 'max' slots hold scalar numeric values for the lower and upper bounds on the list of parameters, respectively.

Parameters may be any free parameter or parameters from an MxMatrix object. Parameters may be referenced either by name or by referring to their position in the 'spec' matrix of an `MxMatrix` object. To affect an estimation or optimization, an MxBounds object must be included in an MxModel object with all referenced MxAlgebra and MxMatrix objects.

Slots may be referenced with the @ symbol. See the documentation for Classes and the examples in the mxBounds document for more information.

References

The OpenMx User's guide can be found at http://openmx.psyc.virginia.edu/documentation.

See Also

mxBounds for the function that creates MxBounds objects. MxMatrix and mxMatrix for free parameter specification. More information about the OpenMx package may be found here.

`mxCI` *Create mxCI Object*

Description

This function creates a new MxCI object, which are used to estimate likelihood-based confidence intervals.

Usage

```
mxCI(reference, interval = 0.95, type=c("both", "lower", "upper"))
```

Arguments

`reference`	A character vector of free parameters, mxMatrices, mxMatrix elements and mxAlgebras on which confidence intervals for free parameters are to be estimated, listed by name.
`interval`	A scalar numeric value indicating the confidence interval to be estimated. Must be between 0 and 1. Defaults to 0.95.
`type`	A character string indicating whether the upper, lower or both confidence limits are returned. Defaults to "both".

Details

The mxCI function creates MxCI objects, which can be used as arguments in MxModel objects. When models containing MxCI objects are optimized using mxRun with the 'intervals' argument set to TRUE, likelihood-based confidence intervals are returned. The likelihood-based confidence intervals calculated by MxCI objects are symmetric with respect to the change in likelihood in either direction, and are not necessarily symmetric around the parameter estimate. Estimation of confidence intervals requires both that an MxCI object be included in the model and that the 'intervals' argument of the mxRun function is set to TRUE. When estimated, confidence intervals can be accessed in the model output at `@output$confidenceIntervals` or by using summary on a fitted MxModel object.

The likelihood-based confidence intervals returned using MxCI are obtained by increasing or decreasing the value of each parameter until the -2 log likelihood of the model increases by an amount corresponding to the requested interval. The confidence limit specified by the 'interval' argument is transformed into a corresponding difference in the model -2 log likelihood based on the likelihood ratio test. Thus, a requested confidence interval for a parameter will first determine the corresponding quantile from the chi-squared distribution with one degree of freedom (a value of 3.841459 when a 95 percent confidence interval is requested). That quantile will be populated into either the 'lowerdelta' slot, the 'upperdelta' slot, or both in the output MxCI object.

Estimation of likelihood-based confidence intervals begins after optimization has been completed, with each parameter moved in the direction(s) specified in the 'type' argument until the specified increase in -2 log likelihood is reached. All other free parameters are left free for this stage of

optimization. This process repeats until all confidence intervals have been calculated. The calculation of likelihood-based confidence intervals can be computationally intensive, and may add a significant amount of time to model estimation when many confidence intervals are requested.

Multiple parameters, MxMatrices and MxAlgebras may be listed in the 'reference' argument. Individual elements of MxMatrices and MxAlgebras may be listed as well, using the syntax "matrix[row,col]" (see Extract for more information). Only scalar numeric values for the 'interval' argument are supported. Users requesting different confidence ranges for different parameters must use separate mxCI statements. MxModel objects can hold multiple MxCI objects, but only one confidence interval may be requested per named-entity.

Confidence interval estimation may result in model non-convergence at the confidence limit. Separate optimizer messages may be passed for each confidence limit. This has no impact on the parameter estimates themselves, but may indicate a problem with the referenced confidence limit. Model non-convergence for a particular confidence limit may indicate parameter interdependence or the influence of a parameter boundary. Checking the validity of a confidence limit can be done my estimating a model with the appropriate parameter fixed at the confidence limit in question. If the confidence limit is valid, the -2 log likelihoods of these two models should differ by the chi-squared criterion specified in the MxCI's 'lowerdelta' or 'upperdelta' slot.

Value

Returns a new MxCI object. If used as an argument in an MxModel object, the parameters, MxMatrices and MxAlgebras listed in the 'reference' argument must also be included prior to optimization.

References

The OpenMx User's guide can be found at http://openmx.psyc.virginia.edu/documentation. Additional support for mxCI() can be found on the OpenMx wiki at http://openmx.psyc.virginia.edu/wiki.

See Also

MxCI for the S4 class created by mxCI. MxMatrix and mxMatrix for free parameter specification. More information about the OpenMx package may be found here.

Examples

```
# generate data
covariance <- matrix(c(1.0, 0.5, 0.5, 1.0),
nrow=2,
dimnames=list(c("a", "b"), c("a", "b")))

data <- mxData(covariance, "cov", numObs=100)

# create an expected covariance matrix
expect <- mxMatrix("Symm", 2, 2,
free=TRUE,
values=c(1, .5, 1),
labels=c("var1", "cov12", "var2"),
name="expectedCov")

# request 95 percent confidence intervals
```

```
ci <- mxCI(c("var1", "cov12", "var2"))

# specify the model
model <- mxModel("Confidence Interval Example",
data, expect, ci,
mxMLObjective("expectedCov", dimnames=c("a", "b")))

# run the model
results <- mxRun(model, intervals=TRUE)

# view confidence intervals
print(summary(results)$CI)

# view all results
summary(results)
```

`MxCI-class` *MxCI Class*

Description

MxCI is an S4 class. An MxCI object is a named entity. New instances of this class can be created using the function mxCI. MxCI objects may be used as arguments in the mxModel function.

Details

The MxCI class has the following slots:

reference	-	The name of the object
lowerdelta	-	Either a matrix or a data frame
upperdelta	-	A vector for means, or NA if missing

The reference slot contains a character vector of named free parameters, MxMatrices and MxAlgebras on which confidence intervals are desired. Individual elements of MxMatrices and MxAlgebras may be listed as well, using the syntax "matrix[row,col]" (see Extract for more information).

The lowerdelta and upperdelta slots give the changes in likelihoods used to define the confidence interval. The upper bound of the likelihood-based confidence interval is estimated by increasing the parameter estimate, leaving all other parameters free, until the model -2 log likelihood increased by 'upperdelta'. The lower bound of the confidence interval is estimated by decreasing the parameter estimate, leaving all other parameters free, until the model -2 log likelihood increased by 'lowerdata'.

Likelihood-based confidence intervals may be specified by including one or more MxCI objects in an MxModel object. Estimation of confidence intervals requires model optimization using the

mxRun function with the 'intervals' argument set to TRUE. The calculation of likelihood-based confidence intervals can be computationally intensive, and may add a significant amount of time to model estimation when many confidence intervals are requested.

References

The OpenMx User's guide can be found at http://openmx.psyc.virginia.edu/documentation.

See Also

mxCI for creating MxCI objects. More information about the OpenMx package may be found here.

`mxCompare` *Assign Model Parameters*

Description

Compare the fit of one or more models to a base model. The output is a table with one row per model comparison.

Usage

```
mxCompare(base, comparison, ..., all = FALSE)
```

Arguments

`base`	A MxModel object or list of MxModel objects.
`comparison`	A MxModel object or list of MxModel objects.
`...`	Not used. Forces remaining arguments to be specified by name.
`all`	A boolean value on whether to compare all bases with all comparisons.

Details

Use options('digits' = N) to set the minimum number of significant digits to be printed in values. The following columns appear in the output:

base Name of the base model

comparison Name of the comparison model

ep Estimated parameters of the comparison model

minus2LL Minus 2*log-likelihood of the comparison model

df Degrees in freedom of the comparison model

AIC Akaike's Information Criterion for the comparison model

diffLL Change in minus 2*log-likelihood

diffdf Change in degrees of freedom

p Significance level of the change in fitness function

See Also

mxModel; options (use options('mxOptions') to see all the OpenMx-specific options)

Examples

```
data(demoOneFactor)
manifests <- names(demoOneFactor)
latents <- c("G1")
model1 <- mxModel("One Factor", type="RAM",
      manifestVars = manifests,
      latentVars = latents,
      mxPath(from = latents, to=manifests),
      mxPath(from = manifests, arrows = 2),
      mxPath(from = latents, arrows = 2, free = FALSE, values = 1.0),
      mxData(cov(demoOneFactor), type = "cov", numObs = 500)
)

fit1 <- mxRun(model1)

latents <- c("G1", "G2")
model2 <- mxModel(name="Two Factor", type="RAM",
      manifestVars = manifests,
      latentVars = latents,
      mxPath(from = latents[1], to=manifests[1:3]),
      mxPath(from = latents[2], to=manifests[4:5]),
      mxPath(from = manifests, arrows = 2),
      mxPath(from = latents, arrows = 2, free = FALSE, values = 1.0),
      mxData(cov(demoOneFactor), type = "cov", numObs=500)
)
fit2 <- mxRun(model2)

mxCompare(fit1, c(fit2))

# vary precision of the output (no effect for this example)
oldPrecision = as.numeric(options('digits'))
options('digits' = 3)
mxCompare(fit1, c(fit2))
options('digits' = oldPrecision)
```

---

mxConstraint *Create MxConstraint Object*

Description

This function creates a new MxConstraint object.

Usage

```
mxConstraint(expression, name = NA)
```

Arguments

expression	An R expression of matrix operators and matrix functions.
name	An optional character string indicating the name of the object.

Details

The mxConstraint function defines relationships between two MxAlgebra or MxMatrix objects. They are used to affect the estimation of free parameters in the referenced objects. The constraint relation is written identically to how a MxAlgebra expression would be written. The outermost operator in this relation must be either '<', '==' or '>'. To affect an estimation or optimization, an MxConstraint object must be included in an MxModel object with all referenced MxAlgebra and MxMatrix objects.

Usage Note: Use of mxConstraint should be avoided where it is possible to achieve the constraint by equating free parameters by label or position in an MxMatrix or MxAlgebra object. Including mxConstraints in an mxModel will disable standard errors and the calculation of the final Hessian, and thus should be avoided when standard errors are of importance. Constraints also add computational overhead. If one labels two parameters the same, the optimizer has one fewer parameter to optimize. However, if one uses mxConstraint to do the same thing, both parameters remain estimated and a Lagrangian multiplier is added to maintain the constraint. This constraint also has to have its gradients computed and the order of the Hessian grows as well. So while both approaches should work, the mxConstraint() will take longer to do so.

Alernatives to mxConstraints include using labels, lbound or ubound arguments or algebras. Free parameters in the same MxModel may be constrained to equality by giving them the same name in their respective 'labels' matrices. Similarly, parameters may be fixed to an individual element in a MxModel object or the result of an MxAlgebra object through labeling. For example, assigning a label of "name[1,1]" fixes the value of a parameter at the value in first row and first column of the matrix or algebra "name". The mxConstraint function should be used to enforce inequalities that cannot be conveyed using other methods.

Value

Returns an MxConstraint object.

References

The OpenMx User's guide can be found at http://openmx.psyc.virginia.edu/documentation.

See Also

MxConstraint for the S4 class created by mxConstraint.

Examples

```
#Create a constraint between MxMatrices 'A' and 'B'
constraint <- mxConstraint(A > B, name = 'AdominatesB')

# Constrain matrix 'K' to be equal to matrix 'limit'

model <- mxModel("con_test",
```

```
    mxMatrix(type="Full", nrow=2, ncol=2, free=TRUE, name="K"),
    mxMatrix(type="Full", nrow=2, ncol=2, free=FALSE, name="limit", values=1:4),
mxConstraint(K == limit, name = "Klimit_equality"),
mxAlgebra(min(K), name="minK"),
mxAlgebraObjective("minK")
)
## Not run:
fit <- mxRun(model)
fit@matrices$K@values

## End(Not run)
#      [,1] [,2]
# [1,]    1    3
# [2,]    2    4

# Constrain both free parameters of a matrix to equality using labels (both are set to "eq")
equal <- mxMatrix("Full", 2, 1, free=TRUE, values=1, labels="eq", name="D")

# Constrain a matrix element in to be equal to the result of an algebra
start <- mxMatrix("Full", 1, 1, free=TRUE,  values=1, labels="param", name="F")
alg   <- mxAlgebra(log(start), name="logP")

# Force the fixed parameter in matrix G to be the result of the algebra
end   <- mxMatrix("Full", 1, 1, free=FALSE, values=1, labels="logP[1,1]", name="G")
```

`MxConstraint-class` *MxConstraint Class*

Description

MxConstraint is an S4 class. An MxConstraint object is a named entity. New instances of this class can be created using the function mxConstraint.

Details

The MxConstraint class has the following slots:

name	-	The name of the object
formula	-	The R expression to be evaluated

The 'name' slot is the name of the MxConstraint object. Use of MxConstraint objects in other functions in the OpenMx library may require reference by name.

The 'formula' slot is an expression containing the expression to be evaluated. These objects are operated on or related to one another using one or more operations detailed in the `mxConstraint` help file.

Slots may be referenced with the @ symbol. See the documentation for Classes and the examples

in the mxConstraint document for more information.

References

The OpenMx User's guide can be found at http://openmx.psyc.virginia.edu/documentation.

See Also

mxConstraint for the function that creates MxConstraint objects.

`mxData` *Create MxData Object*

Description

This function creates a new MxData object.

Usage

```
mxData(observed, type, means = NA, numObs = NA)
```

Arguments

`observed`	A matrix or data.frame which provides data to the MxData object.
`type`	A character string defining the type of data in the 'observed' argument. Must be one of "raw", "cov", "cor", or "sscp".
`means`	An optional vector of means for use when 'type' is "cov", or "cor".
`numObs`	The number of observations in the data supplied in the 'observed' argument. Required unless 'type' equals "raw".

Details

The mxData function creates MxData objects, which can be used as arguments in MxModel objects. The 'observed' argument may take either a data frame or a matrix, which is then described with the 'type' argument. Data types describe compatibility and usage with objective functions in MxModel objects. Four different data types are supported:

raw The contents of the 'observed' argument are treated as raw data. Missing values are permitted and must be designated as the system missing value. The 'means' and 'numObs' arguments cannot be specified, as the 'means' argument is not relevant and the 'numObs' argument is automatically populated with the number of rows in the data. Data of this type must use the mxFIMLObjective function as its objective function in MxModel objects, which deals with covariance estimation under full-information maximum likelihood.

cov The contents of the 'observed' argument are treated as a covariance matrix. The 'means' argument is not required, but may be included for estimations involving means. The 'numObs' argument is required, which should reflect the number of observations or rows in the data described by the covariance matrix. Data of this type may use the mxMLObjective, or mxRAMObjective functions, depending on the specified model.

cor The contents of the 'observed' argument are treated as a correlation matrix. The 'means' argument is not required, but may be included for estimations involving means. The 'numObs' argument is required, which should reflect the number of observations or rows in the data described by the covariance matrix. Data of this type may use the mxMLObjective, or mxRAMObjective functions, depending on the specified model.

sscp The contents of the 'observed' argument are treated as a sums-of-squares and cross-products matrix. The 'means' argument is not used. The 'numObs' argument is required, which should reflect the number of observations or rows in the data described by the covariance matrix. Data of this type may use the mxMLObjective, or mxRAMObjective functions, depending on the specified model.

MxData objects may not be included in MxAlgebra objects or use the mxAlgebraObjective function. If these capabilities are desired, data should be appropriately input or transformed using the mxMatrix and mxAlgebra functions.

While column names are stored in the 'observed' slot of MxData objects, these names are not recognized as variable names in MxPath objects. Variable names must be specified using the 'manifestVars' argument of the mxModel function prior to use in MxPath objects.

The mxData function does not currently place restrictions on the size, shape, or symmetry of matrices input into the 'observed' argument. While it is possible to specify MxData objects as covariance, correlation or sscp matrices that do not have the properties commonly associated with these matrices, failure to correctly specify these matrices will likely lead to problems in model estimation.

OpenMx uses the names of variables to map them onto the objective functions and other elements associated with your model. For data.frames, ensure you have set the names(). For matrices set names using, for instance, row.names=c("your", "columns"). Covariance cor and sscp matrices need to have both the row and column names set and these must be identical, for instance by using dimnames=list(varNames, varNames).

Value

Returns a new MxData object.

References

The OpenMx User's guide can be found at http://openmx.psyc.virginia.edu/documentation.

See Also

MxData for the S4 class created by mxData. matrix and data.frame for objects which may be entered as arguments in the 'observed' slot. More information about the OpenMx package may be found here.

Examples

```
#Create a covariance matrix
covMatrix <- matrix( c(0.77642931, 0.39590663,
    0.39590663, 0.49115615),
    nrow = 2, ncol = 2, byrow = TRUE)
```

```
#Create an MxData object including that covariance matrix
data <- mxData(covMatrix, 'cov', numObs = 100)

model <- mxModel(data)
```

`MxData-class` *MxData Class*

Description

MxData is an S4 class. An MxData object is a named entity. New instances of this class can be created using the function mxData. MxData is an S4 class union. An MxData object is either NULL or a MxNonNullData object.

Details

The MxNonNullData class has the following slots:

name	-	The name of the object
observed	-	Either a matrix or a data frame
vector	-	A vector for means, or NA if missing
type	-	Either 'raw', 'cov', 'cor', or 'sscp'
numObs	-	The number of oberservations

The 'name' slot is the name of the MxData object.

The 'observed' slot is used to contain data, either as a matrix or as a data frame. Use of the data in this slot by other functions depends on the value of the 'type' slot. When 'type' is equal to 'cov', 'cor', or 'sscp', the data input into the 'matrix' slot should be a symmetric matrix or data frame.

The 'vector' slot is used to contain a vector of numeric values, which is used as a vector of means for MxData objects with 'type' equal to 'cov', 'cor', or 'sscp'. This slot may be used in estimation using the mxMLObjective function.

The 'type' slot may take one of four supported values:

raw The contents of the 'observed' slot are treated as raw data. Missing values are permitted and must be designated as the system missing value. The 'vector' and 'numObs' slots cannot be specified, as the 'vector' argument is not relevant and the 'numObs' argument is automatically populated with the number of rows in the data. Data of this type must use the mxFIMLObjective function as its objective function in MxModel objects, which deals with covariance estimation under full-information maximum likelihood.

cov The contents of the 'observed' slot are treated as a covariance matrix. The 'vector' argument is not required, but may be included for estimations involving means. The 'numObs' slot is required. Data of this type may use the mxMLObjective, or mxRAMObjective functions, depending on the specified model.

cor The contents of the 'observed' slot are treated as a correlation matrix. The 'vector' argument is not required, but may be included for estimations involving means. The 'numObs' slot is required. Data of this type may use the mxMLObjective, or mxRAMObjective functions, depending on the specified model.

sscp The contents of the 'observed' slot are treated as a sums-of-squares and cross-products matrix. The 'vector' argument is not required, but may be included for estimations involving means. The 'numObs' slot is required. Data of this type may use the mxMLObjective, or mxRAMObjective functions, depending on the specified model.

The 'numObs' slot describes the number of observations in the data. If 'type' equals 'raw', then 'numObs' is automatically populated as the number of rows in the matrix or data frame in the 'observed' slot. If 'type' equals 'cov', 'cor', or 'sscp', then this slot must be input using the 'numObs' argument in the mxData function when the MxData argument is created.

MxData objects may not be included in MxAlgebra objects or use the mxAlgebraObjective function. If these capabilities are desired, data should be appropriately input or transformed using the mxMatrix and mxAlgebra functions.

While column names are stored in the 'observed' slot of MxData objects, these names are not recognized as variable names in MxPath objects. Variable names must be specified using the 'manifestVars' argument of the mxModel function prior to use in MxPath objects.

The mxData function does not currently place restrictions on the size, shape, or symmetry of matrices input into the 'observed' argument. While it is possible to specify MxData objects as covariance, correlation or sscp matrices that do not have the properties commonly associated with these matrices, failure to correctly specify these matrices will likely lead to problems in model estimation.

References

The OpenMx User's guide can be found at http://openmx.psyc.virginia.edu/documentation.

See Also

mxData for creating MxData objects, matrix and data.frame for objects which may be entered as arguments in the 'matrix' slot. More information about the OpenMx package may be found here.

---

mxErrorPool *Query the Error Pool*

Description

Retrieve models from the pool that did not complete successfully.

Usage

```
mxErrorPool(modelnames = NA, reset = FALSE)
```

Arguments

modelnames	Either NA or a character vector of model names.
reset	Either TRUE or FALSE.

Details

If 'modelnames' is NA, then the list of all error models will be returned. Otherwise a subset of models will be returned, basedon the model names passed in as a argument. If 'reset' is TRUE, then the error pool is reset to the empty list.

References

The OpenMx User's guide can be found at http://openmx.psyc.virginia.edu/documentation.

`mxEval` *Evaluate Values in MxModel*

Description

This function can be used to evaluate an arbitrary R expression that includes named entities from a MxModel object, or labels from a MxMatrix object.

Usage

```
mxEval(expression, model, compute = FALSE, show = FALSE, defvar.row = 1)
```

Arguments

`expression`	An arbitrary R expression.
`model`	The model in which to evaluate the expression.
`compute`	If TRUE then compute the value of algebra expressions.
`show`	If TRUE then print the translated expression.
`defvar.row`	The row number for definition variables when compute=TRUE.

Details

The argument 'expression' is an arbitrary R expression. Any named entities that are used within the R expression are translated into their current value from the model. Any labels from the matrices within the model are translated into their current value from the model. Finally the expression is evaluated and the result is returned. To enable debugging, the 'show' argument has been provided. The most common mistake when using this function is to include named entities in the model that are identical to R function names. For example, if a model contains a named entity named 'c', then the following mxEval call will return an error: `mxEval(c(A, B, C), model)`.

If 'compute' is FALSE, then MxAlgebra expressions return their current values as they have been computed by the optimization call (using mxRun). If the 'compute' argument is TRUE, then MxAlgebra expressions will be calculated in R. Any references to an objective function that has not yet been calculated will return a 1 x 1 matrix with a value of NA.

References

The OpenMx User's guide can be found at http://openmx.psyc.virginia.edu/documentation.

See Also

mxAlgebra to create algebraic expressions inside your model and mxModel for the model object mxEval looks inside when evaluating.

Examples

```
matrixA <- mxMatrix("Full", nrow = 1, ncol = 1,
values = 1, name = "A")
algebraB <- mxAlgebra(A + A, name = "B")

model <- mxModel(matrixA, algebraB)
model <- mxRun(model)
start <- mxEval(-pi * A, model)

## Not run:

mxEval(plot(sin, start, B * pi), model)

# The statement above is equivalent to:

plot(sin, -pi, 2 * pi)


## End(Not run)
```

---

mxFactor *Fail-safe Factors*

Description

This is a wrapper for the R function `factor`.

OpenMx requires ordinal data to be ordered. R's `factor` function doesn't enforce this, hence this wrapper exists to throw an error should you accidentally try and run with ordered = FALSE.

Also, the 'levels' parameter is optional in R's `factor` function. However, relying on the data to specify the data is foolhardy for the following reasons: The `factor` function will skip levels missing from the data: Specifying these in levels leaves the list of levels complete. Data will often not explore the min and max level that the user knows are possible. For these reasons this function forces you to write out all possible levels explicitly.

Usage

```
mxFactor(x = character(), levels, labels = levels,
exclude = NA, ordered = TRUE)
```

Arguments

x	either a vector of data or a data.frame object.
levels	a mandatory vector of the values that 'x' might have taken.
labels	_either_ an optional vector of labels for the levels, _or_ a character string of length 1.
exclude	a vector of values to be excluded from the set of levels.
ordered	logical flag to determine if the levels should be regarded as ordered (in the order given). Required to be TRUE.

Details

If 'x' is a data.frame, then all of the columns of 'x' are converted into ordered factors. If 'x' is a data.frame, then 'levels' and 'labels' may be either a list or a vector. When 'levels' is a list, then different levels are assigned to different columns of the constructed data.frame object. When 'levels' is a vector, then the same levels are assigned to all the columns of the data.frame object. The function will throw an error if 'ordered' is not TRUE or if 'levels' is missing. See factor for more information on creating ordered factors.

References

The OpenMx User's guide can be found at http://openmx.psyc.virginia.edu/documentation.

Examples

```
myVar <- c("s", "t", "a", "t", "i", "s", "t", "i", "c", "s")
ff   <- mxFactor(myVar, levels=letters)  # letters is a built in list of all lowercase letters of the alphabet
ff
# [1] s t a t i s t i c s
# Levels: a < b < c < d < e < f < g < h < i < j < k < l < m < n < o < p < q < r < s < t < u < v < w < x < y < z

as.integer(ff)  # the internal codes

factor(ff)      # NOTE: drops the levels that do not occur.
                # mxFactor prevents you doing this unintentionally.

# This example works on a dataframe
foo <- data.frame(x=c(1:3),y=c(4:6),z=c(7:9))
mxFactor(foo, c(1:9))                     # Applys one set of levels to all three columns
mxFactor(foo, list(c(1:3), c(4:6), c(7:9))) # Apply unique sets of levels to each variable
```

---

mxFIMLObjective *Create MxFIMLObjective Object*

Description

This function creates a new MxFIMLObjective object.

Usage

```
mxFIMLObjective(covariance, means, dimnames = NA, thresholds = NA, vector = FALSE, threshnames = dimname
```

Arguments

`covariance`	A character string indicating the name of the expected covariance algebra.
`means`	A character string indicating the name of the expected means algebra.
`dimnames`	An optional character vector to be assigned to the dimnames of the covariance and means algebras.
`thresholds`	An optional character string indicating the name of the thresholds matrix.
`vector`	A logical value indicating whether the objective function result is the likelihood vector.
`threshnames`	An optional character vector to be assigned to the column names of the thresholds matrix.

Details

Objective functions are functions for which free parameter values are chosen such that the value of the objective function is minimized. The mxFIMLObjective function uses full-information maximum likelihood to provide maximum likelihood estimates of free parameters in the algebra defined by the 'covariance' and 'means' arguments. The 'covariance' argument takes an MxAlgebra object, which defines the expected covariance of an associated MxData object. The 'means' argument takes an MxAlgebra object, which defines the expected means of an associated MxData object. The 'dimnames' arguments takes an optional character vector. If this argument is not a single NA, then this vector be assigned to be the dimnames of the means vector, and the row and columns dimnames of the covariance matrix. The 'vector' argument is either TRUE or FALSE, and determines whether the objective function returns a column vector of the likelihoods, or a single -2*(log likelihood) value.

mxFIMLObjective evaluates with respect to an MxData object. The MxData object need not be referenced in the mxFIMLObjective function, but must be included in the MxModel object. mxFIMLObjective requires that the 'type' argument in the associated MxData object be equal to 'raw'. Missing values are permitted in the associated MxData object.

dimnames must be supplied where the matrices referenced by the covariance and means algebras are not themselves labeled. Failure to do so leads to an error noting that the covariance or means matrix associated with the FIML objective does not contain dimnames.

To evaluate, place MxFIMLObjective objects, the mxData object for which the expected covariance approximates, referenced MxAlgebra and MxMatrix objects, and optional MxBounds and MxConstraint objects in an MxModel object. This model may then be evaluated using the mxRun function. The results of the optimization can be found in the 'output' slot of the resulting model, and may be referenced using the Extract functionality.

Value

Returns a new MxFIMLObjective object. MxFIMLObjective objects should be included with models with referenced MxAlgebra, MxData and MxMatrix objects.

References

The OpenMx User's guide can be found at http://openmx.psyc.virginia.edu/documentation.

Examples

```
A <- mxMatrix(values = 0.5, nrow = 2, ncol = 1,
free = TRUE, name = "A")

D <- mxMatrix(type = "Diag", values = c(0, 0.5),
free = c(FALSE, TRUE), nrow = 2, name = "D")

M <- mxMatrix(type = "Zero", nrow = 1, ncol = 2, name = "M")

expectedCov <- mxAlgebra(A %*% t(A) + D, "expectedCov")

objective <- mxFIMLObjective("expectedCov", "M")

model <- mxModel(A, D, expectedCov, objective)
```

---

mxMatrix *Create MxMatrix Object*

Description

This function creates a new MxMatrix object.

Usage

```
mxMatrix(type = "Full", nrow = NA, ncol = NA,
free = FALSE, values = NA, labels = NA, lbound = NA,
ubound = NA, byrow = getOption('mxByrow'), dimnames = NA, name = NA)
```

Arguments

type	a character string indicating the matrix type, where type indicates the range of values and equalities in the matrix. Must be one of: 'Diag', 'Full', 'Iden', 'Lower', 'Sdiag', 'Stand', 'Symm', 'Unit', or 'Zero'.
nrow	the desired number of rows. One or both of 'nrow' and 'ncol' is required when 'values', 'free', 'labels', 'lbound', and 'ubound' arguments are not matrices, depending on the matrix type.
ncol	the desired number of columns. One or both of 'nrow' and 'ncol' is required when 'values', 'free', 'labels', 'lbound', and 'ubound' arguments are not matrices, depending on the matrix type.
free	a vector or matrix of logicals for free parameter specification. A single 'TRUE' or 'FALSE' will set all allowable variables to free or fixed, respectively.

values	a vector or matrix of numeric starting values. By default, all values are set to zero.
labels	a vector or matrix of characters for variable label specification.
lbound	a vector or matrix of numeric lower bounds. Default bounds are specified with an NA.
ubound	a vector or matrix of numeric upper bounds. Default bounds are specified with an NA.
byrow	logical. If 'FALSE' (default), the 'values', 'free', 'labels', 'lbound', and 'ubound' matrices are populated by column rather than by row.
dimnames	list. The dimnames attribute for the matrix: a list of length 2 giving the row and column names respectively. An empty list is treated as NULL, and a list of length one as row names. The list can be named, and the list names will be used as names for the dimensions.
name	an optional character string indicating the name of the MxMatrix object created by the mxModel function.

Details

The mxMatrix function creates MxMatrix objects, which consist of a pair of matrices and a 'type' argument. The 'values' matrix is made up of numeric elements whose usage and capabilities in other functions are defined by the 'free' matrix. If an element is specified as a fixed parameter in the 'free' matrix, then the element in the 'values' matrix is treated as a constant value and cannot be altered or updated by an objective function when included in an mxRun function. If an element is specified as a free parameter in the 'free' matrix, the element in the 'value' matrix is considered a starting value and can be changed by an objective function when included in an mxRun function. Free parameters are specified with a character string, non-zero numeric value, or 'NA'; fixed parameters are specified with a numeric zero.

Objects created by the mxMatrix function are of a specific 'type', which specifies the number and location of parameters in the 'labels' matrix and the starting values in the 'values' matrix. Input 'values', 'free', and 'labels' matrices must be of appropriate shape and have appropriate values for the matrix type requested. Nine types of matrices are supported:

'Diag'	matrices must be square, and only elements on the principle diagonal may be specified as free parameters or take no
'Full'	matrices may be either rectangular or square, and all elements in the matrix may be freely estimated. This type is th
'Iden'	matrices must be square, and consist of no free parameters. Matrices of this type have a value of 1 for all entries on
'Lower'	matrices must be square, with a value of 0 for all entries in the upper triangle and no free parameters in the upper tr
'Sdiag'	matrices must be square, with a value of 0 for all entries in the upper triangle and along the diagonal. No free paran
'Symm'	matrices must be square, and elements in the principle diagonal and lower triangular portion of the matrix may be f
'Stand'	matrices are symmetric matrices (see 'Symm') with 1's along the main diagonal.
'Unit'	matrices may be either rectangular or square, and contain no free parameters. All elements in matrices of this type l
'Zero'	matrices may be either rectangular or square, and contain no free parameters. All elements in matrices of this type l

When 'type' is 'Lower' or 'Symm', then the arguments to 'free', 'values', 'labels', 'lbound', or 'ubound' may be vectors of length $N * (N + 1)/2$, where N is the number of rows and columns of the matrix. When 'type' is 'Sdiag' or 'Stand', then the arguments to 'free', 'values', 'labels',

'lbound', or 'ubound' may be vectors of length $N * (N - 1)/2$.

Value

Returns a new MxMatrix object, which consists of a 'values' matrix of numeric starting values, a 'free' matrix describing free parameter specification, a 'labels' matrix of labels for the variable names, and 'lbound' and 'ubound' matrices of the lower and upper parameter bounds. This MxMatrix object can be used as an argument in the mxAlgebra, mxBounds, mxConstraint and mxModel functions.

References

The OpenMx User's guide can be found at http://openmx.psyc.virginia.edu/documentation.

See Also

MxMatrix for the S4 class created by mxMatrix. More information about the OpenMx package may be found here.

Examples

```
# Create a 3 x 3 identity matrix

idenMatrix <- mxMatrix(type = "Iden", nrow = 3,
ncol = 3, name = "I")

# Create a full 4 x 2 matrix from existing
# value matrix with all free parameters

vals <- matrix(1:8, nrow = 4)
fullMatrix <- mxMatrix(type = "Full", values = vals,
free = TRUE, name = "foo")

# Create a 3 x 3 symmetric matrix with free off-
# diagonal parameters and starting values

symmMatrix <- mxMatrix(type = "Symm", nrow = 3, ncol = 3,
free = c(FALSE, TRUE, TRUE, FALSE, TRUE, FALSE),
values = c(1, .8, .8, 1, .8, 1),
labels = c(NA, "free1", "free2", NA, "free3", NA),
name = "bar")
```

MxMatrix-class *MxMatrix Class*

Description

MxMatrix is an S4 class. An MxMatrix object is a named entity. New instances of this class can be created using the function mxMatrix. MxMatrix objects may be used as arguments in other functions from the OpenMx library, including mxAlgebra, mxConstraint, and mxModel.

Details

The MxMatrix class has the following slots:

name	-	the name of the object
free	-	the free matrix
values	-	the values matrix
labels	-	the labels matrix

The 'name' slot is the name of the MxMatrix object. Use of MxMatrix objects in an mxAlgebra or mxConstraint function requires reference by name.

The 'free' slot takes a matrix which describes the location of free and fixed parameters. A variable is a free parameter if-and-only-if the corresponding value in the 'free' matrix is 'TRUE'. Free parameters are elements of an MxMatrix object whose values may be changed by an objective function when that MxMatrix object is included in an MxModel object and evaluated using the mxRun function.

The 'values' slot takes a matrix of numeric values. If an element is specified as a fixed parameter in the 'free' matrix, then the element in the 'values' matrix is treated as a constant value and cannot be altered or updated by an objective function when included in an mxRun function. If an element is specified as a free parameter in the 'free' matrix, the element in the 'value' matrix is considered a starting value and can be changed by an objective function when included in an mxRun function.

The 'labels' slot takes a matrix which describes the labels of free and fixed parameters. Fixed parameters with identical labels must have identical values. Free parameters with identical labels impose an equality constraint. The same label cannot be applied to a free parameter and a fixed parameter. A free parameter with the label 'NA' implies a unique free parameter, that cannot be constrained to equal any other free parameter.

References

The OpenMx User's guide can be found at http://openmx.psyc.virginia.edu/documentation.

See Also

mxMatrix for creating MxMatrix objects. More information about the OpenMx package may be found here.

`mxMLObjective` *Create MxMLObjective Object*

Description

This function creates a new MxMLObjective object.

Usage

```
mxMLObjective(covariance, means = NA, dimnames = NA, thresholds = NA)
```

Arguments

covariance	A character string indicating the name of the expected covariance algebra.
means	An optional character string indicating the name of the expected means algebra.
dimnames	An optional character vector to be assigned to the dimnames of the covariance and means algebras.
thresholds	An optional character string indicating the name of the thresholds matrix.

Details

Objective functions are functions for which free parameter values are chosen such that the value of the objective function is minimized. The mxMLObjective function uses full-information maximum likelihood to provide maximum likelihood estimates of free parameters in the algebra defined by the 'covariance' argument given the covariance of an MxData object. The 'covariance' argument takes an MxAlgebra object, which defines the expected covariance of an associated MxData object. The 'dimnames' arguments takes an optional character vector. If this argument is not a single NA, then this vector be assigned to be the dimnames of the means vector, and the row and columns dimnames of the covariance matrix.

mxMLObjective evaluates with respect to an MxData object. The MxData object need not be referenced in the mxMLObjective function, but must be included in the MxModel object. mxMLObjective requires that the 'type' argument in the associated MxData object be equal to 'cov', 'cov', or 'sscp'. The 'covariance' argument of this function evaluates with respect to the 'matrix' argument of the associated MxData object, while the 'means' argument of this function evaluates with respect to the 'vector' argument of the associated MxData object. The 'means' and 'vector' arguments are optional in both functions. If the 'means' argument is not specified (NA), the optional 'vector' argument of the MxData object is ignored. If the 'means' argument is specified, the associated MxData object should specify a 'means' argument of equivalent dimension as the 'means' algebra.

dimnames must be supplied where the matrices referenced by the covariance and means algebras are not themselves labeled. Failure to do so leads to an error noting that the covariance or means matrix associated with the ML objective does not contain dimnames.

To evaluate, place MxMLObjective objects, the mxData object for which the expected covariance approximates, referenced MxAlgebra and MxMatrix objects, and optional MxBounds and MxConstraint objects in an MxModel object. This model may then be evaluated using the mxRun function. The results of the optimization can be found in the 'output' slot of the resulting model, or using the mxEval function.

Value

Returns a new MxMLObjective object. MxMLObjective objects should be included with models with referenced MxAlgebra, MxData and MxMatrix objects.

References

The OpenMx User's guide can be found at http://openmx.psyc.virginia.edu/documentation.

Examples

```
vars <- c('x','y')

A <- mxMatrix(values = 0.5, nrow = 2, ncol = 1,
   free = TRUE, name = "A")
D <- mxMatrix(type = "Diag", values = c(0, 0.5),
   free = c(FALSE, TRUE), nrow = 2, name = "D")

expectedCov <- mxAlgebra(A %*% t(A) + D, "expectedCov")
observedCov <- mxData(matrix(c(1.2, 0.8, 0.8, 1.3),
   nrow = 2, ncol = 2, dimnames = list(vars,vars)), 'cov', numObs = 150)

objective <- mxMLObjective(covariance = "expectedCov", dimnames = vars)

model <- mxModel("mxMLObjective example", A, D, expectedCov, objective, observedCov)

## Not run: summary(mxRun(model))
```

---

mxModel *Create MxModel Object*

Description

This function creates a new MxModel object.

Usage

```
mxModel(model = NA, ..., manifestVars = NA, latentVars = NA,
        remove = FALSE, independent = NA, type = NA, name = NA)
```

Arguments

model	This argument is either an MxModel object or a string. If 'model' is an MxModel object, then all elements of that model are placed in the resulting MxModel object. If 'model' is a string, then a new model is created with the string as its name. If 'model' is either unspecified or 'model' is a named entity, data source, or MxPath object, then a new model is created.
...	An arbitrary number of mxMatrix, mxPath, mxData, and other functions such as mxConstraints and mxCI. These will all be added or removed from the model as specified in the 'model' argument, based on the 'remove' argument.
manifestVars	For RAM-type models, A list of manifest variables to be included in the model.
latentVars	For RAM-type models, A list of latent variables to be included in the model.
remove	logical. If TRUE, elements listed in this statement are removed from the original model. If FALSE, elements listed in this statement are added to the original model.

independent	logical. If TRUE then the model is evaluated independently of other models.
type	character vector. The model type to assign to this model. Defaults to options("mxDefaultType"). See below for valid types
name	An optional character vector indicating the name of the object.

Details

The mxModel function is used to create MxModel objects. Objects created by this function may be new, or may be modified versions of existing MxModel objects. By default a new MxModel object will be created: To create a modified version of an existing MxModel object, include this model in the 'model' argument.

Other named-entities may be added as arguments to the mxModel function, which are then added to or removed from the model specified in the 'model' argument. Other functions you can use to add objects to the model to this way are mxCI, mxAlgebra, mxBounds, mxConstraint, mxData, and mxMatrix objects, as well as objective functions. You can also include MxModel objects as sub-models of the output model, and may be estimated separately or jointly depending on shared parameters and the 'independent' flag discussed below. Only one MxData object and one objective function may be included per model, but there are no restrictions on the number of other named-entities included in an mxModel statement.

All other arguments must be named (i.e. 'latentVars = names'), or they will be interpreted as elements of the ellipsis list. The 'manifestVars' and 'latentVars' arguments specify the names of the manifest and latent variables, respectively, for use with the mxPath function. The 'remove' argument may be used when mxModel is used to create a modified version of an existing MxMatrix object. When 'remove' is set to TRUE, the listed objects are removed from the model specified in the 'model' argument. When 'remove' is set to FALSE, the listed objects are added to the model specified in the 'model' argument.

Model independence may be specified with the 'independent' argument. If a model is independent ('independent = TRUE'), then the parameters of this model are not shared with any other model. An independent model may be estimated with no dependency on any other model. If a model is not independent ('independent = FALSE'), then this model shares parameters with one or more other models such that these models must be jointly estimated. These dependent models must be entered as arguments in another model, so that they are simultaneously optimized.

The model type is determined by a character vector supplied to the 'type' argument. The type of a model is a dynamic property, ie. it is allowed to change during the lifetype of the model. To see a list of available types, use the mxTypes command. When a new model is created and no type is specified, the type specified by `options("mxDefaultType")` is used.

To be estimated, MxModel objects must include objective functions as arguments (mxAlgebraObjective, mxFIMLObjective, mxMLObjective or mxRAMObjective) and executed using the mxRun function. When MxData objects are included in models, the 'type' argument of these objects may require or exclude certain objective functions, or set an objective function as default.

Named entities in MxModel objects may be viewed and referenced by name using double brackets (`model[["matrixname"]]`). Slots may be referenced with the @ symbol (`model@data`). See the documentation for Classes and the examples in this document for more information.

Value

Returns a new MxModel object. MxModel objects must include an objective function to be used as arguments in mxRun functions.

References

The OpenMx User's guide can be found at http://openmx.psyc.virginia.edu/documentation.

See Also

See mxCI for information about adding Confidence Interval calculations to a model. See mxPath for information about adding paths to RAM-type models. See mxMatrix for information about adding matrices to models. See mxData for specifying the data a model is to be evaluated against. See MxModel for the S4 class created by mxMatrix. Many advanced options can be set via mxOption (such as calculating the Hessian). More information about the OpenMx package may be found here.

Examples

```
# Create an empy model, and place it in an object.
model <- mxModel()

# Create a model named 'firstdraft' with one matrix
model <- mxModel('firstdraft',
                 mxMatrix('Full', nrow = 3, ncol = 3, name = "A"))

# Add other matrices to model 'firstdraft', and rename that model 'finaldraft'
model <- mxModel(model,
                 mxMatrix('Symm', nrow = 3, ncol = 3, name = "S"),
                 mxMatrix('Iden', nrow = 3, name = "F"),
                 name= "finaldraft")

# Add data to the model from an existing data frame in object 'data'
data <- data.frame()
model <- mxModel(model, mxData(data, type='raw'))

#View the matrix named "A" in MxModel object 'model'
model[["A"]]

#View the data associated with MxModel object 'model'
model$data

# An example Using OpenMx's Path Syntax
data(HS.fake.data)  #load the data

Spatial  <- c("visual","cubes","paper") # the manifest variables loading on each proposed latent variable
Verbal    <- c("general","paragrap","sentence")
Math      <- c("numeric","series","arithmet")

latents   <- c("vis","math","text")
manifests <-  c(Spatial,Math,Verbal)
```

```
model <- mxModel("Holzinger and Swineford (1939)", type="RAM",
manifestVars = manifests, # list the measured variables (boxes)
latentVars   = latents,   # list the latent variables (circles)
    # factor loadings from latents to  manifests
mxPath(from="vis",  to=Spatial),# factor loadings
    mxPath(from="math", to=Math),   # factor loadings
    mxPath(from="text", to=Verbal), # factor loadings

# Allow latent variables to covary
mxPath(from="vis" , to="math", arrows=2, free=TRUE),
mxPath(from="vis" , to="text", arrows=2, free=TRUE),
mxPath(from="math", to="text", arrows=2, free=TRUE),

    # Allow latent variables to have variance (first fixed  @ 1)
mxPath(from=latents, arrows=2, free=c(FALSE,TRUE,TRUE), values=1.0),
    # Manifest have residual variance
mxPath(from=manifests, arrows=2),
# the data to be analysed
    mxData(cov(HS.fake.data[,manifests]), type="cov", numObs=301))

fit <- mxRun(model) # run the model
summary(fit) # examine the output: Fits statistics and (unstandardized) path loadings
```

`MxModel-class` *MxModel Class*

Description

MxModel is an S4 class. An MxModel object is a named entity. New instances of this class can be created using the function mxModel.

Details

The MxModel class has the following slots:

name	-	The name of the object
matrices	-	A list of MxMatrix objects
algebras	-	A list of MxAlgebra objects
submodels	-	A list of MxModel objects
constraints	-	A list of MxConstraint objects
intervals	-	A list of confidence intervals requested in MxCI objects
bounds	-	A list of MxBounds objects
latentVars	-	A list of latent variables
manifestVars	-	A list of manifest variables
data	-	A MxData object
objective	-	Either NULL or a MxObjective object
independent	-	TRUE if-and-only-if the model is independent
options	-	A list of optimizer options
output	-	A list with optimization results

The ‘name’ slot is the name of the MxModel object.

The ‘matrices’ slot contains a list of the MxMatrix objects included in the model. These objects are listed by name. Two objects may not share the same name. If a new MxMatrix is added to an MxModel object with the same name as an MxMatrix object in that model, the added version replaces the previous version. There is no imposed limit on the number of MxMatrix objects that may be added here.

The ‘algebras’ slot contains a list of the MxAlgebra objects included in the model. These objects are listed by name. Two objects may not share the same name. If a new MxAlgebra is added to an MxModel object with the same name as an MxAlgebra object in that model, the added version replaces the previous version. All MxMatrix objects referenced in the included MxAlgebra objects must be included in the ‘matrices’ slot prior to estimation. There is no imposed limit on the number of MxAlgebra objects that may be added here.

The ‘submodels’ slot contains references to all of the MxModel objects included as submodels of this MxModel object. Models held as arguments in other models are considered to be submodels. These objects are listed by name. Two objects may not share the same name. If a new submodel is added to an MxModel object with the same name as an existing submodel, the added version replaces the previous version. When a model containing other models is executed using mxRun, all included submodels are executed as well. If the submodels are dependent on one another, they are treated as one larger model for purposes of estimation.

The ‘constraints’ slot contains a list of the MxConstraint objects included in the model. These objects are listed by name. Two objects may not share the same name. If a new MxConstraint is added to an MxModel object with the same name as an MxConstraint object in that model, the added version replaces the previous version. All MxMatrix objects referenced in the included MxConstraint objects must be included in the ‘matrices’ slot prior to estimation. There is no imposed limit on the number of MxAlgebra objects that may be added here.

The ‘intervals’ slot contains a list of the confidence intervals requested by included MxCI objects. These objects are listed by the free parameters, MxMatrices and MxAlgebras referenced in the MxCI objects, not the list of MxCI objects themselves. If a new MxCI object is added to an MxModel object referencing one or more free parameters MxMatrices or MxAlgebras previously listed in the ‘intervals’ slot, the new confidence interval(s) replace the existing ones. All listed confidence intervals must refer to free parameters MxMatrices or MxAlgebras in the model.

The ‘bounds’ slot contains a list of the MxBounds objects included in the model. These objects are listed by name. Two objects may not share the same name. If a new MxBounds is added to an MxModel object with the same name as an MxBounds object in that model, the added version replaces the previous version. All MxMatrix objects referenced in the included MxBounds objects must be included in the ‘matrices’ slot prior to estimation. There is no imposed limit on the number of MxAlgebra objects that may be added here.

The ‘latentVars’ slot contains a list of latent variable names, which may be referenced by MxPath objects. This slot defaults to ’NA’, and is only used when the mxPath function is used.

The ‘manifestVars’ slot contains a list of latent variable names, which may be referenced by MxPath objects. This slot defaults to ’NA’, and is only used when the mxPath function is used.

The ‘data’ slot contains an MxData object. This slot must be filled prior to execution when an objective function referencing data is used. Only one MxData object may be included per model, but submodels may have their own data in their own ‘data’ slots. If an MxData object is added to an MxModel which already contains an MxData object, the new object replaces the existing one.

The 'objective' slot contains an objective function. This slot must be filled prior to using the mxRun function for model execution and optimization. MxAlgebra, MxData, and MxMatrix objects required by the included objective function must be included in the appropriate slot of the MxModel prior to using mxRun.

The 'independent' slot contains a logical value indicating whether or not the model is independent. If a model is independent (independent=TRUE), then the parameters of this model are not shared with any other model. An independent model may be estimated with no dependency on any other model. If a model is not independent (independent=FALSE), then this model shares parameters with one or more other models such that these models must be jointly estimated. These dependent models must be entered as submodels of another MxModel objects, so that they are simultaneously optimized.

The 'options' slot contains a list of options for the optimizer. The name of each entry in the list is the option name to be passed to the optimizer. The values in this list are the values of the optimizer options. The standard interface for updating options is through the mxOption function.

The 'output' slot contains a list of output added to the model by the mxRun function. Output includes parameter estimates, optimization information, model fit, and other information as dictated by the objective function. If a model has not been optimized using the mxRun function, the 'output' slot will be 'NULL'.

Named entities in MxModel objects may be viewed and referenced by name using double brackets (model[["matrixname"]]). Slots may be referenced with the @ symbol (model@data). See the documentation for Classes and the examples in mxModel for more information.

References

The OpenMx User's guide can be found at http://openmx.psyc.virginia.edu/documentation.

See Also

mxModel for creating MxModel objects. More information about the OpenMx package may be found here.

`mxOption` *Set or Clear an Optimizer Option*

Description

The function sets or clears an option that is specific to the optimizer in the back-end.

Usage

```
mxOption(model, key, value, reset = FALSE)
```

Arguments

`model`	An MxModel object or NULL
`key`	The name of the option.
`value`	The value of the option.
`reset`	If TRUE then reset all options to their defaults.

Details

Sets an option that is specific to the particular optimizer used in the back-end. The name of the option is the 'key' argument. Use value = NULL to remove an existing option. Before the model is submitted to the back-end, all keys and values are converted into strings using the as.character function. To reset all options to their default values, use reset = TRUE. If reset = TRUE, then 'key' and 'value' are ignored. To set the default optimizer options, use the value NULL for the 'model' argument. Use `getOption('mxOptions')` to see the default optimizer options.

OpenMx options

Calculate Hessian	[Yes|No]	calculate the hessian explicitly after optimization.
Standard Errors	[Yes|No]	return standard error estimates from the explicitly calculate hessian.
CI Max Iterations	*i*	the maximum number of retries when calculating confidence intervals.

NPSOL-specific options

Nolist		this option suppresses printing of the options
Print level	*i*	the value of *i* controls the amount of printout produced by the major iterations
Minor print level	*i*	the value of *i* controls the amount of printout produced by the minor iterations
Print file	*i*	for $i > 0$ a full log is sent to the file with logical unit number *i*.
Summary file	*i*	for $i > 0$ a brief log will be output to file *i*.
Function precision	*r*	a measure of accuracy with which *f* and *c* can be computed.
Infinite bound size	*r*	if $r > 0$ defines the "infinite" bound bigbnd.
Feasibility tolerance	*r*	the maximum acceptable absolute violations in linear and nonlinear constraints.
Major iterations	*i*	the maximum number of major iterations before termination.
Verify level	[-1:3|Yes|No]	see NPSOL manual.
Line search tolerance	*r*	controls the accuracy with which a step is taken.
Derivative level	[0-3]	see NPSOL manual.
Hessian	[Yes|No]	return the transformed Hessian (if 'No') or the Hessian itself (if 'Yes').

Checkpointing options

Checkpoint Directory	the directory where to write checkpoint files
Checkpoint Prefix	the string prefix to add to all checkpoint filenames
Checkpoint Units	the type of units for checkpointing: 'minutes' or 'iterations'
Checkpoint Count	the number of units between checkpoint intervals
Socket Server	the server name for sending optimizer state information
Socket Port	the port on the server for sending optimizer state information
Socket Units	the type of units: 'minutes' or 'iterations'

Socket Count	the number of units between communication to the server

Model transformation options

Error Checking	"Yes" or "No" on whether model consistency checks are performed in the OpenMx front-end
No Sort Data	character vector of model names for which FIML data sorting is not performed
RAM Inverse Optimization	"Yes" or "No" whether to enable solve(I - A) optimization
RAM Max Depth	the maximum depth to be used when solve(I - A) optimization is enabled

Value

Returns the model with the optimizer option either set or cleared.

References

The OpenMx User's guide can be found at http://openmx.psyc.virginia.edu/documentation.

See Also

mxModel all uses of mxOption are via an mxModel whose options are set or cleared.

Examples

```
model <- mxModel() # make a model to use for example
model@options # show the model options (none yet)
options()$mxOptions # list all mxOptions (global settings)

model <- mxOption(model, "Function precision", 1e-5) # set the precision
model <- mxOption(model, "Function precision", NULL) # clear model-specific precision (defaults to global setting)
model <- mxOption(model, "Calculate Hessian", "No")  # may optimize for speed
model <- mxOption(model, "Standard Errors"  , "No")  # may optimize for speed
model@options # see the list of options you set
```

---

mxPath *Create List of Paths*

Description

This function creates a list of paths.

Usage

```
mxPath(from, to = NA, connect = c("single", "all.pairs", "unique.pairs",
    "all.bivariate", "unique.bivariate"), arrows = 1,
free = TRUE, values = NA, labels = NA,
lbound = NA, ubound = NA, ...)
```

Arguments

from	character vector. These are the sources of the new paths.
to	character vector. These are the sinks of the new paths.
connect	String. Specifies the type of source to sink connection: "single", "all.pairs", "all.bivariate", "unique.pairs", "unique.bivariate".
arrows	numeric value. Must be either 1 (for single-headed) or 2 (for double-headed arrows).
free	boolean vector. Indicates whether paths are free or fixed.
values	numeric vector. The starting values of the parameters.
labels	character vector. The names of the paths.
lbound	numeric vector. The lower bounds of free parameters.
ubound	numeric vector. The upper bounds of free parameters.
...	Not used. Allows OpenMx to catch the use of the deprecated 'all' argument.

Details

The mxPath function creates MxPath objects. These consist of a list of paths describing the relationships between variables in a model using the RAM modeling approach (McArdle and MacDonald, 1984). Variables are referenced by name, and these names must appear in the 'manifestVar' and 'latentVar' arguments of the mxModel function.

Paths are specified as going "from" one variable (or set of variables) "to" another variable or set of variables using the 'from' and 'to' arguments, respectively. If 'to' is left empty, it will be set to the value of 'from'.

'connect' has five possible connection types: "single", "all.pairs", "all.bivariate", "unique.pairs", "unique.bivariate". Assuming the values c('a','b','c') for the 'to' and 'from' fields the paths produced by each connection type are as follows:

"all.pairs": (a,a), (a,b), (a,c), (b,a), (b,b), (b,c), (c,a), (c,b), (c,c).

"unique.pairs": (a,a), (a,b), (a,c), (b,b), (b,c), (c,c).

"all.bivariate": (a,b), (a,c), (b,a), (b,c), (c,a), (c,b).

"unique.bivariate": (a,b), (a,c), (b,c).

"single": (a,a), (b,b), (c,c).

Multiple variables may be input as a vector of variable names. If the 'connect' argument is set to "single", then paths are created going from each entry in the 'from' vector to the corresponding entry in the 'to' vector. If the 'to' and 'from' vectors are of different lengths when the 'connect' argument is set to "single", the shorter vector is repeated to make the vectors of equal length.

The 'free' argument specifies whether the paths created by the mxPath function are free or fixed parameters. This argument may take either TRUE for free parameters, FALSE for fixed parameters, or a vector of TRUEs and FALSEs to be applied in order to the created paths.

The 'arrows' argument specifies the type of paths created. A value of 1 indicates a one-headed arrow representing regression. This path represents a regression of the 'to' variable on the 'from' variable, such that the arrow points to the 'to' variable in a path diagram. A value of 2 indicates a

two-headed arrow, representing a covariance or variance. If multiple paths are created in the same mxPath function, then the 'arrows' argument may take a vector of 1s and 2s to be applied to the set of created paths.

The 'values' is a numeric vectors containing the starting values of the created paths. 'values' gives a starting value for estimation. The 'labels' argument specifies the names of the resulting MxPath object. The 'lbound' and 'ubound' arguments specify lower and upper bounds for the created paths.

Value

Returns a list of paths.

Note

The previous implementation of 'all' had unsafe features. Its use is now deprecated, and has been replaced by the new mechanism 'connect' which supports safe and controlled generation of desired combinations of paths.

References

McArdle, J. J. and MacDonald, R. P. (1984). Some algebraic properties of the Reticular Action Model for moment structures. *British Journal of Mathematical and Statistical Psychology, 37,* 234-251.

The OpenMx User's guide can be found at http://openmx.psyc.virginia.edu/documentation.

See Also

mxMatrix for a matrix-based approach to path specification; mxModel for the container in which mxPaths are embedded. More information about the OpenMx package may be found here.

Examples

```
# A simple Example: 1 factor Confirmatory Factor Analysis
require(OpenMx)
data(demoOneFactor)
manifests <- names(demoOneFactor)
latents   <- c("G")
factorModel <- mxModel("One Factor", type="RAM",
      manifestVars = manifests,
      latentVars   = latents,
      mxPath(from=latents, to=manifests),
      mxPath(from=manifests, arrows=2),
      mxPath(from=latents, arrows=2,free=FALSE, values=1.0),
      mxData(cov(demoOneFactor), type="cov",numObs=500)
)
factorFit <-mxRun(factorModel)
summary(factorFit)

# A more complex example using features of R to compress what would otherwise be a long and error-prone script

myManifest <- sprintf("%02d", c(1:100))  # list of 100 variable names: "01"  "02"  "03"...
myLatent   <- c("G1", "G2", "G3", "G4", "G5") # the latent variables for the model
```

```
# Start building the model: Define its type, and add the manifest and latent variable name lists
model     <- mxModel(type = "RAM", manifestVars = myManifest, latentVars = myLatent)

# Create covariances between the latent variables and add to the model
# Here we use combn to create the covariances
# nb: To create the variances and covariances in one operation you could use
# expand.grid(myLatent,myLatent) to specify from and to

uniquePairs <- combn(myLatent,2)
covariances <- mxPath(from = uniquePairs[1,], to=uniquePairs[2,], arrows = 2, free = TRUE, values = 1)
model       <- mxModel(model, covariances)

# Create variances for the latent variables
variances   <- mxPath(from = myLatent, to=myLatent, arrows = 2, free = TRUE, values = 1)
model       <- mxModel(model, variances) # add variances to the model

# Make a list of paths from each packet of 20 manifests to one of the 5 latent variables
# nb: The first loading to each latent is fixed to 1 to scale its variance.
singles <- list()
for (i in 1:5) {
    j <- i*20
    singles <- append(singles, mxPath(
from = myLatent[i], to = myManifest[(j - 19):j],
                        arrows = 1,
                        free   = c(FALSE, rep(TRUE, 19)),
                        values = c(1, rep(0.75, 19))))
}

model <- mxModel(model, singles) # add single-headed paths to the model
```

`mxRAMObjective` *Create MxRAMObjective Object*

Description

This function creates a new MxRAMObjective object.

Usage

```
mxRAMObjective(A, S, F, M = NA, dimnames = NA, thresholds = NA, vector = FALSE, threshnames = dimnames)
```

Arguments

`A`	A character string indicating the name of the 'A' matrix.
`S`	A character string indicating the name of the 'S' matrix.
`F`	A character string indicating the name of the 'F' matrix.
`M`	An optional character string indicating the name of the 'M' matrix.

dimnames	An optional character vector to be assigned to the column names of the 'F' and 'M' matrices.
thresholds	An optional character string indicating the name of the thresholds matrix.
vector	A logical value indicating whether the objective function result is the likelihood vector.
threshnames	An optional character vector to be assigned to the column names of the thresholds matrix.

Details

Objective functions are functions for which free parameter values are chosen such that the value of the objective function is minimized. The mxRAMObjective provides maximum likelihood estimates of free parameters in a model of the covariance of a given MxData object. This model is defined by reticular action modeling (McArdle and McDonald, 1984). The 'A', 'S', and 'F' arguments must refer to MxMatrix objects with the associated properties of the A, S, and F matrices in the RAM modeling approach.

The 'dimnames' arguments takes an optional character vector. If this argument is not a single NA, then this vector be assigned to be the column names of the 'F' matrix and optionally to the 'M' matrix, if the 'M' matrix exists.

The 'A' argument refers to the A or asymmetric matrix in the RAM approach. This matrix consists of all of the asymmetric paths (one-headed arrows) in the model. A free parameter in any row and column describes a regression of the variable represented by that row regressed on the variable represented in that column.

The 'S' argument refers to the S or symmetric matrix in the RAM approach, and as such must be square. This matrix consists of all of the symmetric paths (two-headed arrows) in the model. A free parameter in any row and column describes a covariance between the variable represented by that row and the variable represented by that column. Variances are covariances between any variable at itself, which occur on the diagonal of the specified matrix.

The 'F' argument refers to the F or filter matrix in the RAM approach. If no latent variables are included in the model (i.e., the A and S matrices are of both of the same dimension as the data matrix), then the 'F' should refer to an identity matrix. If latent variables are included (i.e., the A and S matrices are not of the same dimension as the data matrix), then the 'F' argument should consist of a horizontal adhesion of an identity matrix and a matrix of zeros.

The 'M' argument refers to the M or means matrix in the RAM approach. It is a 1 x n matrix, where n is the number of manifest variables + the number of latent variables. The M matrix must be specified if either the mxData type is "cov" or "cor" and a means vector is provided, or if the mxData type is "raw". Otherwise the M matrix is ignored.

The MxMatrix objects included as arguments may be of any type, but should have the properties described above. The mxRAMObjective will not return an error for incorrect specification, but incorrect specification will likely lead to estimation problems or errors in the mxRun function.

mxRAMObjective evaluates with respect to an MxData object. The MxData object need not be referenced in the mxRAMObjective function, but must be included in the MxModel object. mxRAMObjective requires that the 'type' argument in the associated MxData object be equal to 'cov', 'cor' or 'sscp'.

To evaluate, place MxRAMObjective objects, the mxData object for which the expected covariance approximates, referenced MxAlgebra and MxMatrix objects, and optional MxBounds and MxConstraint objects in an MxModel object. This model may then be evaluated using the mxRun function. The results of the optimization can be found in the 'output' slot of the resulting model, and may be obtained using the mxEval function..

Value

Returns a new MxRAMObjective object. MxRAMObjective objects should be included with models with referenced MxAlgebra, MxData and MxMatrix objects.

References

McArdle, J. J. and MacDonald, R. P. (1984). Some algebraic properties of the Reticular Action Model for moment structures. *British Journal of Mathematical and Statistical Psychology, 37,* 234-251.

The OpenMx User's guide can be found at http://openmx.psyc.virginia.edu/documentation.

Examples

```
matrixA <- mxMatrix("Full", values=c(0,0.2,0,0), name="A", nrow=2, ncol=2)
matrixS <- mxMatrix("Full", values=c(0.8,0,0,0.8), name="S", nrow=2, ncol=2, free=TRUE)
matrixF <- mxMatrix("Full", values=c(1,0,0,1), name="F", nrow=2, ncol=2)

# Create a RAM objective with default A, S, F matrix names
objective <- mxRAMObjective("A", "S", "F")

model <- mxModel(matrixA, matrixS, matrixF, objective)
```

---

mxRename *Rename MxModel or a Submodel*

Description

This functions renames either the top model or a submodel to a new name. All internal references to the old model name are replaced with references to the new name.

Usage

```
mxRename(model, newname, oldname = NA)
```

Arguments

model	a MxModel object.
newname	the new name of the model.
oldname	the name of the target model to rename. If NA then rename top model.

Value

Return a mxModel object with the target model renamed.

References

The OpenMx User's guide can be found at http://openmx.psyc.virginia.edu/documentation.

Examples

```
modelA <- mxModel('modelA')
modelB <- mxModel('modelB')
modelC <- mxModel('modelC', modelA, modelB)

# Rename modelC to model1
model1 <- mxRename(modelC, 'model1')

# Rename submodel modelB to model2
model1 <- mxRename(model1, oldname = 'modelB', newname = 'model2')
```

---

mxRestore *Restore From Checkpoint File*

Description

The function loads the last saved state from a checkpoint file.

Usage

```
mxRestore(model, chkpt.directory = ".", chkpt.prefix = "")
```

Arguments

model	MxModel object to be loaded.
chkpt.directory	character. Directory where the checkpoint file is located.
chkpt.prefix	character. Prefix of the checkpoint file.

Details

In general, the arguments 'chkpt.directory' and 'chkpt.prefix' should be identical to the mxOption: 'Checkpoint Directory' and 'Checkpoint Prefix' that were specificed on the model before execution.

Alternatively, the checkpoint file can be manually loaded as a data.frame in R. Use read.table with the options 'header=TRUE', 'stringsAsFactors=FALSE' and 'check.names=FALSE'.

Value

Returns an MxModel object with free parameters updated to the last saved values.

References

The OpenMx User's guide can be found at http://openmx.psyc.virginia.edu/documentation.

Examples

```
#Create a model that includes data,
#matrices A, S and F, and an objective function
## Not run:
data <- mxData(mydata, type="cov", numObs = 100)
objective <- mxRAMObjective('A', 'S', 'F')
model <- mxModel("mymodel", A, S, F, data, objective)

#Use mxRun to optimize the free parameters in the matrices A and S
modelOut <- mxRun(model, checkpoint = TRUE)

#Use mxRestore to load the last saved state of the model
modelRestore <- mxRestore(model)

## End(Not run)
```

---

mxRObjective *Function to Create MxRObjective Object*

Description

This function creates a new MxRObjective object.

Usage

```
mxRObjective(objfun, ...)
```

Arguments

objfun	A function that accepts two arguments.
...	The initial state information to the objective function.

Details

The objfun argument must be a function that accepts two arguments. The first argument is the mxModel that should be evaluated, and the second argument is some persistent state information that can be stored between one iteration of optimization to the next iteration. It is valid for the function to simply ignore the second argument.

The function must return either a single numeric value, or a list of exactly two elements. If the function returns a list, the first argument must be a single numeric value and the second element

will be the new persistent state information to be passed into this function at the next iteration. The single numeric value will be used by the optimizer to perform optimization.

The initial default value for the persistant state information is NA.

Value

Returns a new MxRObjective object.

References

The OpenMx User's guide can be found at http://openmx.psyc.virginia.edu/documentation.

Examples

```
A <- mxMatrix(nrow = 2, ncol = 2, values = c(1:4), free = TRUE, name = 'A')

squared <- function(x) { x ^ 2 }

objFunction <- function(model, state) {
values <- model[['A']]@values
return(squared(values[1,1] - 4) + squared(values[1,2] - 3) +
squared(values[2,1] - 2) + squared(values[2,2] - 1))
}
objective <- mxRObjective(objFunction)

model <- mxModel('model', A, objective)
```

---

mxRowObjective *Create MxRowObjective Object*

Description

This function creates a new MxRowObjective object.

Usage

```
mxRowObjective(rowAlgebra, reduceAlgebra, dimnames,
   rowResults = "rowResults", filteredDataRow = "filteredDataRow",
   existenceVector = "existenceVector")
```

Arguments

rowAlgebra	A character string indicating the name of the algebra to be evaluated row-wise.
reduceAlgebra	A character string indicating the name of the algebra that collapses the row results into a single number which is then optimized.

`dimnames` A character vector of names corresponding to columns be extracted from the data set.

`rowResults` The name of the auto-generated "rowResults" matrix. See details.

`filteredDataRow` The name of the auto-generated "filteredDataRow" matrix. See details.

`existenceVector` The name of the auto-generated "existenceVector" matrix. See details.

Details

Objective functions are functions for which free parameter values are chosen such that the value of the objective function is minimized. The mxRowObjective function evaluates a user-defined MxAlgebra object called the 'rowAlgebra' in a row-wise fashion. It then stores results of the row-wise evaluation in another MxAlgebra object called the 'rowResults'. Finally, the mxRowObjective function collapses the row results into a single number which is then used for optimization. The MxAlgebra object named by the 'reduceAlgebra' collapses the row results into a single number.

The 'filteredDataRow' is populated in a row-by-row fashion with all the non-missing data from the current row. You cannot assume that the length of the filteredDataRow matrix remains constant (unless you have no missing data). The 'existenceVector' is populated in a row-by-row fashion with a value of 1.0 in column j if a non-missing value is present in the data set in column j, and a value of 0.0 otherwise. Use the functions omxSelectRows, omxSelectCols, and omxSelectRowsAndCols to shrink other matrices so that their dimensions will be conformable to the size of 'filteredDataRow'.

Value

Returns a new MxRowObjective object. MxRowObjective objects should be included with models with referenced MxAlgebra objects.

References

The OpenMx User's guide can be found at http://openmx.psyc.virginia.edu/documentation.

Examples

```
# Model that adds two data columns row-wise, then sums that column
# Notice no optimization is performed here.
xdat <- data.frame(a=rnorm(10), b=1:10) # Make data set
amod <- mxModel(
    mxData(observed=xdat, type='raw'),
    mxAlgebra(sum(filteredDataRow), name = 'rowAlgebra'),
    mxAlgebra(sum(rowResults), name = 'reduceAlgebra'),
    mxRowObjective(
    rowAlgebra='rowAlgebra',
    reduceAlgebra='reduceAlgebra',
    dimnames=c('a','b'))
)

# Model that find the parameter that minimizes the sum of the
#  squared difference between the parameter and a data row.
```

```
bmod <- mxModel(
    mxData(observed=xdat, type='raw'),
    mxMatrix(values=.75, ncol=1, nrow=1, free=TRUE, name='B'),
    mxAlgebra((filteredDataRow - B) ^ 2, name='rowAlgebra'),
    mxAlgebra(sum(rowResults), name='reduceAlgebra'),
    mxRowObjective(
    rowAlgebra='rowAlgebra',
    reduceAlgebra='reduceAlgebra',
    dimnames=c('a'))
)
```

---

mxRun *Send a Model to the Optimizer*

Description

This function begins optimization on the top-level model.

Usage

```
mxRun(model, ..., intervals = FALSE, silent = FALSE, suppressWarnings = FALSE,
unsafe = FALSE, checkpoint = FALSE, useSocket = FALSE, onlyFrontend = FALSE,
useOptimizer = TRUE)
```

Arguments

model	A MxModel object to be optimized.
...	Not used. Forces remaining arguments to be specified by name.
intervals	A boolean indicating whether to compute the specified confidence intervals.
silent	A boolean indicating whether to print status to terminal.
suppressWarnings	A boolean indicating whether to suppress warnings.
unsafe	A boolean indicating whether to ignore errors.
checkpoint	A boolean indicating whether to periodically write parameter values to a file.
useSocket	A boolean indicating whether to periodically write parameter values to a socket.
onlyFrontend	A boolean indicating whether to run only front-end model transformations.
useOptimizer	A boolean indicating whether to run only the log-likelihood of the current free parameter values but not move any of the free parameters.

Details

The mxRun function is used to optimize free parameters in MxModel objects based on an objective function. MxModel objects included in the mxRun function must include an appropriate objective function.

If the 'silent' flag is TRUE, then model execution will not print any status messages to the terminal.

If the 'suppressWarnings' flag is TRUE, then model execution will not issue a warning if NPSOL returns a non-zero status code.

If the 'unsafe' flag is TRUE, then any error conditions will throw a warning instead of an error. It is strongly recommended to use this feature only for debugging purposes.

Free parameters are estimated or updated based on the objective function. These estimated values, along with estimation information and model fit, can be found in the 'output' slot of MxModel objects after mxRun has been used.

If a model is dependent on or shares parameters with another model, both models must be included as arguments in another MxModel object. This top-level MxModel object must include objective functions in both submodels, as well as an additional objective function describing how the results of the first two should be combined.

Value

Returns an MxModel object with free parameters updated to their final values. The return value contains an "output" slot with the results of optimization.

References

The OpenMx User's guide can be found at http://openmx.psyc.virginia.edu/documentation.

Examples

```
# Create and run the 1-factor CFA on the openmx.psyc.virginia.edu front page
## Not run:
require(OpenMx)
data(demoOneFactor)  # load the demoOneFactor dataframe
manifests <- names(demoOneFactor) # set the manifest to the 5 demo variables
latents   <- c("G")  # define 1 latent variable
model <- mxModel("One Factor", type="RAM",
manifestVars = manifests,
latentVars   = latents,
mxPath(from=latents  , to=manifests),
mxPath(from=manifests, arrows=2),
mxPath(from=latents  , arrows=2, free=FALSE, values=1.0),
mxData(cov(demoOneFactor), type="cov", numObs=500)
)
model <- mxRun(model) #run model, returning the result
summary(model) #show summary of the fitted model

#Create a model that includes data, matrices A, S and F, and an objective function
data <- mxData(mydata, type="cov", numObs = 100)
objective <- mxRAMObjective('A', 'S', 'F')
model <- mxModel("mymodel", A, S, F, data, objective)
```

```
#Use mxRun to optimize the free parameters in the matrices A and S
model <- mxRun(model)

# print the output
model@output  #can be directly access by slot name instead of via summary()

## End(Not run)
```

`mxTypes` *List Currently Available Model Types*

Description

This function returns a vector of the currently available type names.

Usage

```
mxTypes()
```

Value

Returns a character vector of type names.

Examples

```
mxTypes()
```

`mxVersion` *Returns Current Version String*

Description

This function returns a string with the current version number of OpenMx.

Usage

```
mxVersion()
```

References

The OpenMx User's guide can be found at http://openmx.psyc.virginia.edu/documentation.

Examples

```
mxVersion()
```

Named-entity *Named Entities*

Description

A named entity is an S4 object that can be referenced by name.

Details

Every named entity is guaranteed to have a slot 'name'. Within a model, the named entities of that model can be accessed using the model[['name']] notation. Access is limited to one nesting depth, such that if 'B' is a submodel of 'A', and 'C' is a matrix of 'B', then 'C' must be accessed using A[['B']][['C']]. See the documentation for Extract for more information.

The following S4 classes are named entities in the OpenMx library: MxAlgebra, MxConstraint, MxMatrix, MxModel, MxData, and MxObjective.

omxAllInt *All Interval Multivariate Normal Integration*

Description

`omxAllInt` computes the probabilities of a large number of cells of a multivariate normal distribution that has been sliced by a varying number of thresholds in each dimension. While the same functionality can be achieved by repeated calls to `omxMnor`, `omxAllInt` is more efficient for repeated operations on a single covariance matrix. `omxAllInt` returns an nx1 matrix of probabilities cycling from lowest to highest thresholds in each column with the rightmost variable in *covariance* changing most rapidly.

Usage

```
omxAllInt(covariance, means, ...)
```

Arguments

`covariance`	the covariance matrix describing the multivariate normal distribution.
`means`	a row vector containing means of the variables of the underlying distribution.
`...`	a matrix or set of matrices containing one column of thresholds for each column of `covariance`. Each column must contain a strictly increasing set of thresholds for the corresponding variable of the underlying distribution. `NA` values in these thresholds indicate that the list of thresholds in that column has ended.

Details

covariance and *means* contain the covariances and means of the multivariate distribution from which probabilities are to be calculated.

covariance must be a square covariance or correlation matrix with one row and column for each variable.

means must be a vector of length `nrows(covariance)` that contains the mean for each corresponding variable.

All further arguments are considered threshold matrices.

Threshold matrices contain locations of the hyperplanes delineating the intervals to be calculated. The first column of the first matrix corresponds to the thresholds for the first variable represented by the covariance matrix. Subsequent columns of the same matrix correspond to thresholds for subsequent variables in the covariance matrix. If more variables exist in the covariance matrix than in the first threshold matrix, the first column of the second threshold matrix will be used, and so on. That is, if *covariance* is a 4x4 matrix, and the three threshold matrices are specified, one with a single column and the others with two columns each, the first column of the first matrix will contain thresholds for the first variable in *covariance*, the two columns of the second matrix will correspond to the second and third variables of *covariance*, respectively, and the first column of the third threshold matrix will correspond to the fourth variable. Any extra columns will be ignored.

Each column in the threshold matrices must contain some number of strictly increasing thresholds, delineating the boundaries of a cell of integration. That is, if the integral from -1 to 0 and 0 to 1 are required for a given variable, the corresponding threshold column should contain the values -1, 0, and 1, in that order. Thresholds may be set to Inf or -Inf if a boundary at positive or negative infinity is desired.

Within a threshold column, a value of +Inf, if it exists, is assumed to be the largest threshold, and any rows after it are ignored in that column. A value of NA, if it exists, indicates that there are no further thresholds in that column, and is otherwise ignored. A threshold column consisting of only +Inf or NA values will cause an error.

For all i>1, the value in row i must be strictly larger than the value in row i-1 in the same column.

The return value of `omxAllInt` is a matrix consisting of a single column with one row for each combination of threshold levels.

See Also

omxMnor

Examples

```
data(myFAData)

covariance <- cov(myFAData[,1:5])
means <- mean(myFAData[,1:5])
thresholdForColumn1 <- cbind(c(-Inf, 0,  1))          # Integrate from -Infinity to 0 and 0 to 1 on first variable
                                      # Note: The first variable will never be calculated from 1 to +Infinity.
thresholdsForColumn2 <- cbind(c(-Inf, -1, 0, 1, Inf))     # These columns will be integrated from -Inf to -1, -1 to 0
thresholdsForColumns3and4 <- cbind(c(-Inf, 1.96, 2.326, Inf), c(-Inf, -1.96, 2.326, Inf))
omxAllInt(covariance, means, thresholdForColumn1, thresholdsForColumn2, thresholdsForColumns3and4, thresholdsFor
```

```
                                        # Notice that columns 2 and 5 are assigned identical thresholds.

# An alternative specification of the same calculation follows
covariance <- cov(myFAData[,1:5])
means <- mean(myFAData[,1:5])
thresholds <- cbind(c(-Inf,    0,    1,  NA,  NA),      # Note NAs to indicate the end of the sequence of thresholds.
                    c(-Inf,    -1,     0,   1, Inf),
                    c(-Inf,  1.96,  2.32, Inf,  NA),
                    c(-Inf, -1.96,  2.32, Inf,  NA),
                    c(-Inf,    -1,     0,   1, Inf))
omxAllInt(covariance, means, thresholds)
```

---

omxApply *On-Demand Parallel Apply*

Description

If the snowfall library is loaded, then this function calls sfApply. Otherwise it invokes apply.

Usage

```
omxApply(x, margin, fun, ...)
```

Arguments

x	a vector (atomic or list) or an expressions vector. Other objects (including classed objects) will be coerced by as.list.
margin	a vector giving the subscripts which the function will be applied over.
fun	the function to be applied to each element of x.
...	optional arguments to fun.

See Also

omxLapply, omxSapply

Examples

```
x <- cbind(x1 = 3, x2 = c(4:1, 2:5))
dimnames(x)[[1]] <- letters[1:8]
omxApply(x, 2, mean, trim = .2)
```

omxAssignFirstParameters *Assign First Available Values to Model Parameters*

Description

Sometimes you may have a free parameter with two different starting values in your model. OpenMx will not run a model until all instances of a free parameter have the same starting value. It is often sufficient to arbitrarily select one of those starting values for optimization.

This function accomplishes that task of assigning valid starting values to the free parameters of a model. It selects an arbitrary current value (the "first" value it finds, where "first" is not defined) for each free parameter and uses that value for all instances of that parameter in the model.

Usage

```
omxAssignFirstParameters(model, indep = FALSE)
```

Arguments

model	a MxModel object.
indep	assign parameters to independent submodels.

See Also

omxGetParameters, omxSetParameters

Examples

```
A    <- mxMatrix('Full', 3, 3, values = c(1:9), labels = c('a','b', NA), free = TRUE, name = 'A')
model <- mxModel(A, name = 'model')
model <- omxAssignFirstParameters(model)

# Note: All cells with the same label now have the same start value. Note also that NAs are untouched.

model@matrices$A

# @labels
#      [,1] [,2] [,3]
# [1,] "a"  "a"  "a"
# [2,] "b"  "b"  "b"
# [3,] NA   NA   NA
#
# @values
#      [,1] [,2] [,3]
# [1,]    1    1    1
# [2,]    2    2    2
# [3,]    3    6    9
```

omxCheckCloseEnough *Approximate Equality Testing Function*

Description

This function tests whether two numeric vectors or matrixes are approximately equal to one another, within a specified threshold.

Usage

```
omxCheckCloseEnough(a, b, epsilon = 10^(-15))
```

Arguments

a	a numeric vector or matrix.
b	a numeric vector or matrix.
epsilon	a non-negative tolerance threshold.

Details

Arguments 'a' and 'b' must be of the same type, ie. they must be either vectors of equal dimension or matrices of equal dimension. The two arguments are compared element-wise for approximate equality. If the absolute value of the difference of any two values is greater than the threshold, then an error will be thrown. If 'a' and 'b' are approximately equal to each other, by default the function will print a statement informing the user the test has passed. To turn off these print statements use `options("mxPrintUnitTests" = FALSE)`.

References

The OpenMx User's guide can be found at http://openmx.psyc.virginia.edu/documentation.

See Also

omxCheckWithinPercentError, omxCheckIdentical, omxCheckSetEquals, omxCheckTrue, omxCheckEquals

Examples

```
omxCheckCloseEnough(c(1, 2, 3), c(1.1, 1.9 ,3.0), epsilon = 0.5)

omxCheckCloseEnough(matrix(3, 3, 3), matrix(4, 3, 3), epsilon = 2)

# Throws an error
try(omxCheckCloseEnough(c(1, 2, 3), c(1.1, 1.9 ,3.0), epsilon = 0.01))
```

omxCheckEquals *Equality Testing Function*

Description

This function tests whether two objects are equal using the '==' operator.

Usage

```
omxCheckEquals(a, b)
```

Arguments

a	the first value to compare.
b	the second value to compare.

Details

Performs the '==' comparison on the two arguments. If the two arguments are not equal, then an error will be thrown. If 'a' and 'b' are equal to each other, by default the function will print a statement informing the user the test has passed. To turn off these print statements use `options("mxPrintUnitTests" = FALSE)`.

References

The OpenMx User's guide can be found at http://openmx.psyc.virginia.edu/documentation.

See Also

omxCheckCloseEnough, omxCheckWithinPercentError, omxCheckSetEquals, omxCheckTrue, omxCheckIdentical

Examples

```
omxCheckEquals(c(1, 2, 3), c(1, 2, 3))

omxCheckEquals(FALSE, FALSE)

# Throws an error
try(omxCheckEquals(c(1, 2, 3), c(2, 1, 3)))
```

omxCheckIdentical *Exact Equality Testing Function*

Description

This function tests whether two objects are equal.

Usage

```
omxCheckIdentical(a, b)
```

Arguments

a	the first value to compare.
b	the second value to compare.

Details

Performs the 'identical' comparison on the two arguments. If the two arguments are not equal, then an error will be thrown. If 'a' and 'b' are equal to each other, by default the function will print a statement informing the user the test has passed. To turn off these print statements use `options("mxPrintUnitTests" = FALSE)`.

References

The OpenMx User's guide can be found at http://openmx.psyc.virginia.edu/documentation.

See Also

omxCheckCloseEnough, omxCheckWithinPercentError, omxCheckSetEquals, omxCheckTrue, omxCheckEquals

Examples

```
omxCheckIdentical(c(1, 2, 3), c(1, 2, 3))

omxCheckIdentical(FALSE, FALSE)

# Throws an error
try(omxCheckIdentical(c(1, 2, 3), c(2, 1, 3)))
```

omxCheckSetEquals *Set Equality Testing Function*

Description

This function tests whether two vectors contain the same elements.

Usage

```
omxCheckSetEquals(a, b)
```

Arguments

a	the first vector to compare.
b	the second vector to compare.

Details

Performs the 'setequal' function on the two arguments. If the two arguments do not contain the same elements, then an error will be thrown. If 'a' and 'b' contain the same elements, by default the function will print a statement informing the user the test has passed. To turn off these print statements use `options("mxPrintUnitTests" = FALSE)`.

References

The OpenMx User's guide can be found at http://openmx.psyc.virginia.edu/documentation.

See Also

omxCheckCloseEnough, omxCheckWithinPercentError, omxCheckIdentical, omxCheckTrue, omxCheckEquals

Examples

```
omxCheckSetEquals(c(1, 1, 2, 2, 3), c(3, 2, 1))

omxCheckSetEquals(matrix(1, 1, 1), matrix(1, 3, 3))

# Throws an error
try(omxCheckSetEquals(c(1, 2, 3, 4), c(2, 1, 3)))
```

omxCheckTrue *Boolean Equality Testing Function*

Description

This function tests whether an object is equal to TRUE.

Usage

```
omxCheckTrue(a)
```

Arguments

a	the value to test.

Details

Checks element-wise whether an object is equal to TRUE. If any of the elements are false, then an error will be thrown. If 'a' is TRUE, by default the function will print a statement informing the user the test has passed. To turn off these print statements use `options("mxPrintUnitTests" = FALSE)`.

References

The OpenMx User's guide can be found at http://openmx.psyc.virginia.edu/documentation.

See Also

omxCheckCloseEnough, omxCheckWithinPercentError, omxCheckIdentical, omxCheckSetEquals, omxCheckEquals

Examples

```
omxCheckTrue(1 + 1 == 2)

omxCheckTrue(matrix(TRUE, 3, 3))

# Throws an error
try(omxCheckTrue(FALSE))
```

omxCheckWithinPercentError

Approximate Percent Equality Testing Function

Description

This function tests whether two numeric vectors or matrixes are approximately equal to one another, within a specified percentage.

Usage

```
omxCheckWithinPercentError(a, b, percent = 0.1)
```

Arguments

a	a numeric vector or matrix.
b	a numeric vector or matrix.
percent	a non-negative percentage.

Details

Arguments 'a' and 'b' must be of the same type, ie. they must be either vectors of equal dimension or matrices of equal dimension. The two arguments are compared element-wise for approximate equality. If the absolute value of the difference of any two values is greater than the percentage difference of 'a', then an error will be thrown. If 'a' and 'b' are approximately equal to each other, by default the function will print a statement informing the user the test has passed. To turn off these print statements use `options("mxPrintUnitTests" = FALSE)`.

References

The OpenMx User's guide can be found at http://openmx.psyc.virginia.edu/documentation.

See Also

omxCheckCloseEnough, omxCheckIdentical, omxCheckSetEquals, omxCheckTrue, omxCheckEquals

Examples

```
omxCheckWithinPercentError(c(1, 2, 3), c(1.1, 1.9 ,3.0), percent = 50)

omxCheckWithinPercentError(matrix(3, 3, 3), matrix(4, 3, 3), percent = 150)

# Throws an error
try(omxCheckWithinPercentError(c(1, 2, 3), c(1.1, 1.9 ,3.0), percent = 0.01))
```

omxGetParameters *Fetch Model Parameters*

Description

Return a vector of the free parameters in the model.

Usage

```
omxGetParameters(model, indep = FALSE, free = c(TRUE, FALSE, NA))
```

Arguments

model	a MxModel object
indep	fetch parameters from independent submodels.
free	fetch either free parameters or fixed parameters or both types.

Details

The argument 'free' dictates whether to return only free parameters or only fixed parameters or both free and fixed parameters. The function will return free parameters that have a label of NA. But it will never return fixed parameters that have a label of NA. No distinction is made between ordinary labels, and definition variables, and square bracket constraints in labels.

See Also

omxSetParameters, omxAssignFirstParameters

Examples

```
A <- mxMatrix('Full', 2, 2, labels = c("A11", "A12", "A21", NA), values= 1:4, free = c(TRUE,TRUE,FALSE,TRUE), byrow=
model <- mxModel(A, name = 'model')

# Request all free paraemters in model
omxGetParameters(model)

# A11  A12 <NA>
#   1    2    4

# Request fixed paraemters from model
omxGetParameters(model, free=FALSE)
# A21
#   3

A@labels
#      [,1]  [,2]
# [1,] "A11" "A12"
# [2,] "A21" NA
```

```
A@free
#       [,1] [,2]
# [1,]  TRUE TRUE
# [2,] FALSE TRUE

A@labels
#      [,1]  [,2]
# [1,] "A11" "A12"
# [2,] "A21" NA
```

---

omxGraphviz *Show RAM Model in Graphviz Format*

Description

The function accepts a RAM style model and outputs a visual representation of the model in Graphviz format. The function will output either to a file or to the console. The recommended file extension for an output file is ".dot".

Usage

```
omxGraphviz(model, dotFilename = "")
```

Arguments

model	An RAM-type model.
dotFilename	The name of the output file. Use "" to write to console.

Value

Invisibly returns a string containing the model description in graphviz format.

References

The OpenMx User's guide can be found at http://openmx.psyc.virginia.edu/documentation.

omxLapply *On-Demand Parallel Lapply*

Description

If the snowfall library is loaded, then this function calls sfLapply. Otherwise it invokes lapply.

Usage

```
omxLapply(x, fun, ...)
```

Arguments

x	a vector (atomic or list) or an expressions vector. Other objects (including classed objects) will be coerced by as.list.
fun	the function to be applied to each element of x.
...	optional arguments to fun.

See Also

omxApply, omxSapply

Examples

```
x <- list(a = 1:10, beta = exp(-3:3), logic = c(TRUE,FALSE,FALSE,TRUE))
# compute the list mean for each list element
omxLapply(x,mean)
```

omxLogical *Logical mxAlgebra() operators*

Description

omxNot computes the unary negation of the values of a matrix. omxAnd computes the binary and of two matrices. omxOr computes the binary or of two matrices. omxGreaterThan computes a binary greater than of two matrices. omxLessThan computes the binary less than of two matrices. omxApproxEquals computes a binary equals within a specified epsilon of two matrices.

Usage

```
omxNot(x)
omxAnd(x, y)
omxOr(x, y)
omxGreaterThan(x, y)
omxLessThan(x, y)
omxApproxEquals(x, y, epsilon)
```

Arguments

x	the first argument, the matrix which the logical operation will be applied to.
y	the second argument, applicable to binary functions.
epsilon	the third argument, specifies the error threshold for omxApproxEquals. Abs(x[i][j]-y[i][j]) must be less than epsilon[i][j].

Examples

```
A <- mxMatrix(values = runif(25), nrow = 5, ncol = 5, name = 'A')
B <- mxMatrix(values = runif(25), nrow = 5, ncol = 5, name = 'B')
EPSILON <- mxMatrix(values = 0.04*1:25, nrow = 5, ncol = 5, name = "EPSILON")

model <- mxModel(A, B, EPSILON, name = 'model')

mxEval(omxNot(A), model)
mxEval(omxGreaterThan(A,B), model)
mxEval(omxLessThan(B,A), model)
mxEval(omxOr(omxNot(A),B), model)
mxEval(omxAnd(omxNot(B), A), model)
mxEval(omxApproxEquals(A,B, EPSILON), model)
```

---

omxMnor *Multivariate Normal Integration*

Description

Given a covariance matrix, a means vector, and vectors of lower and upper bounds, returns the multivariate normal integral across the space between bounds.

Usage

```
omxMnor(covariance, means, lbound, ubound)
```

Arguments

covariance	the covariance matrix describing the multivariate normal distribution.
means	a row vector containing means of the variables of the underlying distribution.
lbound	a row vector containing the lower bounds of the integration in each variable.
ubound	a row vector containing the upper bounds of the integration in each variable.

Details

The order of columns in the 'means', 'lbound', and 'ubound' vectors are assumed to be the same as that of the covariance matrix. That is, means[i] is considered to be the mean of the variable whose variance is in covariance[i,i]. That variable will be integrated from lbound[i] to ubound[i] as part of the integration.

The value of ubound[i] or lbound[i] may be set to Inf or -Inf if a boundary at positive or negative infinity is desired.

For all i, ubound[i] must be strictly greater than lbound[i].

Examples

```
data(myFAData)

covariance <- cov(myFAData[,1:3])
means <- mean(myFAData[,1:3])
lbound <- c(-Inf, 0,   1)    # Integrate from -Infinity to 0 on first variable
ubound <- c(0,    Inf, 2.5)  # From 0 to +Infinity on second, and from 1 to 2.5 on third
omxMnor(covariance, means, lbound, ubound)
```

---

omxSapply *On-Demand Parallel Sapply*

Description

If the snowfall library is loaded, then this function calls `sfSapply`. Otherwise it invokes `sapply`.

Usage

```
omxSapply(x, fun, ..., simplify = TRUE, USE.NAMES = TRUE)
```

Arguments

x	a vector (atomic or list) or an expressions vector. Other objects (including classed objects) will be coerced by `as.list`.
fun	the function to be applied to each element of x.
...	optional arguments to fun.
simplify	logical; should the result be simplified to a vector or matrix if possible?
USE.NAMES	logical; if TRUE and if x is a character, use x as `names` for the result unless it had names already.

See Also

`omxApply`, `omxLapply`

Examples

```
x <- list(a = 1:10, beta = exp(-3:3), logic = c(TRUE,FALSE,FALSE,TRUE))
# compute the list mean for each list element
omxSapply(x, quantile)
```

`omxSelectRowsAndCols` *Filter rows and columns from an mxMatrix*

Description

This function filters rows and columns from a matrix using a single row or column R matrix as a selector.

Usage

```
omxSelectRowsAndCols(x, selector)
omxSelectRows(x, selector)
omxSelectCols(x, selector)
```

Arguments

`x`	the matrix to be filtered
`selector`	A single row or single column R matrix indicating which values should be filtered from the mxMatrix.

Details

omxSelectRowsAndCols, omxSelectRows, and omxSelectCols returns the filtered entries in a target matrix specified by a single row or single column selector matrix. Each entry in the selector matrix is treated as a logical data indicating if the corresponding entry in the target matrix should be excluded (0 or FALSE) or included (not 0 or TRUE). Typically the function is used to filter data from a target matrix using an existence vector which specifies what data entries are missing. This can be seen in the demo: RowObjectiveFIMLBivariateSaturated.

Value

Returns a new matrix with the filtered data.

References

The function is most often used when filtering data for missingness. This can be seen in the demo: RowObjectiveFIMLBivariateSaturated. The OpenMx User's guide can be found at http://openmx.psyc.virginia.edu/documen
The omxSelect* functions share some similarity to the Extract function in the R programming language.

Examples

```
loadings <- c(1, -0.625, 0.1953125, 1, -0.375, 0.0703125, 1, -0.375, 0.0703125)
loadings <- matrix(loadings, 3, 3, byrow= TRUE)
existenceList <- c(1, 0, 1)
existenceList <- matrix(existenceList, 1, 3, byrow= TRUE)
rowsAndCols <- omxSelectRowsAndCols(loadings, existenceList)
rows <- omxSelectRows(loadings, existenceList)
cols <- omxSelectCols(loadings, existenceList)
```

---

omxSetParameters *Assign Model Parameters*

Description

Modify the attributes of parameters in a model. This function cannot modify parameters that have NA labels.

Usage

```
omxSetParameters(model, labels, free = NULL, values = NULL,
newlabels = NULL, lbound = NULL, ubound = NULL, indep = FALSE,
strict = TRUE)
```

Arguments

model	a MxModel object.
labels	a character vector of target parameter names.
free	a boolean vector of parameter free/fixed designations.
values	a numeric vector of parameter values.
newlabels	a character vector of new parameter names.
lbound	a numeric vector of lower bound values.
ubound	a numeric vector of upper bound values.
indep	boolean. set parameters in independent submodels.
strict	boolean. if TRUE then throw an error when a label does not appear in the model.

See Also

omxGetParameters, omxAssignFirstParameters

Examples

```
A <- mxMatrix('Full', 3, 3, labels = c('a','b', NA), free = TRUE, name = 'A')
model <- mxModel(A, name = 'model')
model <- omxSetParameters(model, c('a', 'b'), values = c(1, 2)) # set value of cells labelled "a" and "b" to 1 and 2 r
model <- omxSetParameters(model, c('a', 'b'), newlabels = c('b', 'a')) # set label of cell labelled "a" to "b" and vi
```

OpenMx *OpenMx: Package for Matrix Algebra Optimization*

Description

OpenMx is a package for structural equation modeling, matrix algebra optimization and other statistical estimation problems.

Details

OpenMx is a package for algebra optimization and statistical estimation problems using matrix algebra. The OpenMx library defines a set of S4 classes and functions used to create them. The majority of these classes are used as arguments in models, which may include data, matrices, algebras, bounds and constraints. These models are then paired with objective functions, either existing (maximum likelihood, FIML) or user-defined with included algebra functions. These models can then be optimized, resulting in parameter estimation, algebra evaluation, and output for additional models.

Objects used or created by OpenMx may be of the following classes: MxAlgebra, MxBounds, MxCI, MxConstraint, MxData, MxMatrix, MxModel, and MxPath. Objects of these classes may be created by the following OpenMx functions: mxAlgebra, mxBounds, mxCI, mxConstraint, mxData, mxMatrix, mxModel, and mxPath. The functions mxAlgebraObjective, mxFIMLObjective, mxMLObjective and mxRAMObjective create objective functions for model estimation. Models which include objective functions may be estimated using the mxRun function.

References

The OpenMx User's guide can be found at http://openmx.psyc.virginia.edu/documentation.

rvectorize *Vectorize By Row*

Description

This function returns the vectorization of an input matrix in a row by row traversal of the matrix. The output is returned as a column vector.

Usage

```
rvectorize(x)
```

Arguments

x	an input matrix.

See Also

cvectorize, vech, vechs

Examples

```
rvectorize(matrix(1:9, 3, 3))
rvectorize(matrix(1:12, 3, 4))
```

---

summary-MxModel *Model Summary*

Description

This function returns summary statistics of a model. It is usually invoked after a model has been run through the optimizer.

Usage

```
summary(object, ...)
```

Arguments

object	A MxModel object.
...	Any number of named arguments (see below).

Details

The following named arguments are supported by the summary method:

numObs Numeric. Specify the total number of observations for the model.

numStats Numeric. Specify the total number of observed statistics for the model.

SaturatedLikelihood Numeric or MxModel object. Specify a saturated likelihood for testing.

indep Logical. Set to FALSE to ignore independent submodels in summary.

References

The OpenMx User's guide can be found at http://openmx.psyc.virginia.edu/documentation.

Examples

```
model <- mxModel()
modelOut <- mxRun(model)
# compute a summary and store in variable "statistics"
statistics <- summary(modelOut)

# specify a saturated likelihood for testing
summary(modelOut, SaturatedLikelihood=300)
```

---

twinData *Australian twin sample biometric data.*

Description

Australian data on body mass index (BMI) which saved in the text file twinData.txt. It is a wide dataset, with two individuals per line. It also contains both MZ and DZ twins, along with heights, weights and the calculated variable body mass index (BMI) for each subject. Ago of course occurs only once, as the two twins of each pair share a common age. fam is a family identifier.

Usage

```
data(twinData)
```

Format

A data frame with 3808 observations on the following 12 variables.

`fam` a numeric vector

`age` a numeric vector

`zyg` a numeric vector

`part` a numeric vector

`wt1` a numeric vector

`wt2` a numeric vector

`ht1` a numeric vector

`ht2` a numeric vector

`htwt1` a numeric vector

`htwt2` a numeric vector

`bmi1` a numeric vector

`bmi2` a numeric vector

Details

Zygosity is coded as follows 1 == MZ females 2 == MZ males 3 == DZ females 4 == DZ males 5 == DZ opposite sex pairs

Examples

```
data(twinData)
str(twinData)
plot(wt1~wt2, data=twinData)
mzData <- as.matrix(subset(myTwinData, zyg==1, c(bmi1,bmi2)))
dzData <- as.matrix(subset(myTwinData, zyg==3, c(bmi1,bmi2)))
```

---

vec2diag *Create Diagonal Matrix From Vector*

Description

Given an input row or column vector, vec2diag returns a diagonal matrix with the input argument along the diagonal.

Usage

```
vec2diag(x)
```

Arguments

x	a row or column vector.

Details

Similar to the function diag, except that the input argument is always treated as a vector of elements to place along the diagonal.

See Also

diag2vec

Examples

```
vec2diag(matrix(1:4, 1, 4))
vec2diag(matrix(1:4, 4, 1))
```

vech *Half-vectorization*

Description

This function returns the half-vectorization of an input matrix as a column vector.

Usage

```
vech(x)
```

Arguments

x	an input matrix.

Details

The half-vectorization of an input matrix consists of the elements in the lower triangle of the matrix, including the elements along the diagonal of the matrix, as a column vector. The column vector is created by traversing the matrix in column-major order.

See Also

vechs, rvectorize, cvectorize

Examples

```
vech(matrix(1:9, 3, 3))
vech(matrix(1:12, 3, 4))
```

vechs *Strict Half-vectorization*

Description

This function returns the strict half-vectorization of an input matrix as a column vector.

Usage

```
vechs(x)
```

Arguments

x	an input matrix.

Details

The half-vectorization of an input matrix consists of the elements in the lower triangle of the matrix, excluding the elements along the diagonal of the matrix, as a column vector. The column vector is created by traversing the matrix in column-major order.

See Also

vech, rvectorize, cvectorize

Examples

```
vechs(matrix(1:9, 3, 3))
vechs(matrix(1:12, 3, 4))
```

Index