R topics documented:

diag2vec ................................................................. 3
mxAlgebra ............................................................... 3
MxAlgebra-class ......................................................... 6
mxAlgebraObjective .................................................... 7
mxBounds ................................................................. 8
MxBounds-class ......................................................... 9
mxConstraint ............................................................ 10
MxConstraint-class ..................................................... 11
mxData ................................................................. 12
MxData-class ........................................................... 14
mxEval ................................................................. 16
mxFIMLObjective ....................................................... 17
mxMatrix ............................................................... 18
MxMatrix-class ......................................................... 21
mxMLObjective ......................................................... 22
mxModel ............................................................... 23
MxModel-class ......................................................... 26
mxOption .............................................................. 28
mxPath ............................................................... 29
mxRAMObjective ....................................................... 31
mxRename ............................................................ 33
mxRObjective .......................................................... 34
mxRun ................................................................. 35
mxTypes ............................................................... 36
mxVersion ............................................................. 37
Named-entity .......................................................... 37
omxAllInt ............................................................. 38
omxApply ............................................................. 39
omxAssignFirstParameters ......................................... 40
omxCheckCloseEnough ............................................... 41
omxCheckIdentical ................................................... 42
omxCheckSetEquals .................................................. 43
omxCheckTrue ........................................................ 44
omxCheckWithinPercentError ..................................... 45
omxGetParameters ................................................... 46
omxGraphviz .......................................................... 46
omxLapply ............................................................ 47
omxMnor .............................................................. 48
omxSapply ............................................................ 49
omxSetParameters ................................................... 49
OpenMx ............................................................... 50
summary-MxModel ..................................................... 51
vec2diag ............................................................... 52
vech ................................................................. 52
vechs ................................................................. 53

Index ........................................................................... 54
**diag2vec**

*Extract Diagonal of a Matrix*

**Description**

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

**Usage**

```r
diag2vec(x)
```

**Arguments**

- `x` an input matrix.

**Details**

Similar to the function `diag`, except that the input argument is always treated as a matrix.

**See Also**

- `vec2diag`

**Examples**

```r
diag2vec(matrix(1:9, 3, 3))
diag2vec(matrix(1:12, 3, 4))
```

---

**mxAlgebra**

*Create MxAlgebra Object*

**Description**

This function creates a new `MxAlgebra` object.

**Usage**

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

expression  An R expression of 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 operations are supported in mxAlgebra:

solve()  Inversion
\( t() \)  Transposition
\( ^\)  Element powering
\( + \)  Addition
\( - \)  Subtraction
\( \%\times\% \)  Matrix Multiplication
\( * \)  Element or dot product
\( / \)  Element division
\( \%\times\% \)  Kronecker product
\( \%\&\% \)  Quadratic product

The following functions are supported in mxAlgebra:

\( \text{cbind()} \)  Horizontal adhesion
\( \text{rbind()} \)  Vertical adhesion
\( \text{det()} \)  Determinant
\( \text{tr()} \)  Trace
\( \text{sum()} \)  Sum
\( \text{prod()} \)  Product
\( \text{max()} \)  Maximum
\( \text{min()} \)  Min
\( \text{abs()} \)  Absolute value
\( \text{sin()} \)  Sine
\( \text{sinh()} \)  Hyperbolic sine
\( \text{cos()} \)  Cosine
\( \text{cosh()} \)  Hyperbolic cosine
mxAlgebra

\tan() \quad \text{Tangent}
\tanh() \quad \text{Hyperbolic tangent}
\exp() \quad \text{Exponent}
\log() \quad \text{Natural Logarithm}
\sqrt() \quad \text{Square root}
\vech() \quad \text{Half-vectorization}
\vechs() \quad \text{Strict half-vectorization}
vec2diag() \quad \text{Create a diagonal matrix}
diag2vec() \quad \text{Extract diagonal from matrix}

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. 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)
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 \texttt{mxAlgebra}.

Details

The MxAlgebra class has the following slots:

- \texttt{name} - The name of the object
- \texttt{formula} - The R expression to be evaluated
- \texttt{result} - Either NULL or a 1x1 matrix with the computation result

The 'name' slot is the name of the MxAlgebra object. Use of MxAlgebra objects in the \texttt{mxConstraint} function or an OpenMx objective function requires reference by name.

The 'formula' slot is an expression containing one or more \texttt{MxMatrix} objects. These objects are operated on or related to one another using one or more operations detailed in the \texttt{mxAlgebra} help file.

The 'result' slot is used to hold the results of an optimization on the expression in the 'formula' slot. If this MxAlgebra has not been used as an argument in an objective function and subsequently included in an \texttt{MxModel} object and executed using the \texttt{mxRun} function, this slot will have a value of NULL. If this MxAlgebra has been used as an argument in an objective function and subsequently included in an \texttt{MxModel} object and executed using the \texttt{mxRun} function, then this slot will have a 1x1 matrix with the results of that optimization as its only value.

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

References

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

See Also

\texttt{mxAlgebra} for creating MxAlgebra objects. \texttt{MxMatrix} and \texttt{mxMatrix} for objects which may be entered in the 'formula' slot and the function that creates them. More information about the OpenMx package may be found here.
mxAlgebraObjective Function to Create MxAlgebraObjective Object

**Description**

This function creates a new MxAlgebraObjective object.

**Usage**

mxAlgebraObjective(algebra)

**Arguments**

- **algebra**: A character string indicating the name of an MxAlgebra or MxMatrix object to use for optimization.

**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 no free parameters are included in any part of the 'algebra' argument, the mxAlgebraObjective works as an algebra evaluator. If the 'algebra' argument is given an MxMatrix object with no free parameters, the original object is returned. If the 'algebra' argument is given an MxAlgebra object whose referenced matrices and algebras have no free parameters, the mxAlgebraObjective function carries out the operations defined in the 'expression' argument of that MxAlgebra object.

If free parameters are included in the objects references in the 'algebra' argument, the mxAlgebraObjective minimizes the algebra defined in the 'algebra' argument. More than one free parameter may be included in the algebra, but the mxAlgebraObjective should be specified such that the objective function evaluates to a 1x1 matrix or scalar value. If the contents of the 'algebra' argument evaluates to a non-scalar matrix, only the first entry of that matrix is minimized.

To evaluate, place MxAlgebraObjective objects, 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 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.
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)

#Evaluate 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.
MxBounds-class

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.

mxConstraint Create MxConstraint Object

Description

This function creates a new MxConstraint object.

Usage

mxConstraint(alg1, relation, alg2, name = NA)

Arguments

alg1 A character string indicating the name of an MxAlgebra or MxMatrix object, whose relationship to the object specified in the 'alg2' argument is constrained.
relation A character string indicating the relation between 'alg1' and 'alg2'. Must be either "<", ",=", or ">".
alg2 A character string indicating the name of an MxAlgebra or MxMatrix object, whose relationship to the object specified in the 'alg1' argument is constrained.
name An optional character string indicating the name of this 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 relationships "<", "=" and ">" are supported. To affect an estimation or optimization, an MxConstraint object must be included in an MxModel object with all referenced MxAlgebra and MxMatrix objects.

The mxConstraint function may not be used to constrain free parameters, either by name or by their position in an MxMatrix or MxAlgebra object. Free parameters in the same MxModel may be constrained to equality by giving them the same name in their respective 'spec' matrices.

Value

Returns an MxConstraint object. If used as an argument in an MxModel object, the objects referenced in the 'alg1' and 'alg2' arguments must also be included prior to optimization.

References

The OpenMx User’s guide can be found at http://openmx.psyc.virginia.edu/documentation.
MxConstraint-class

See Also

MxConstraint for the S4 class created by mxConstraint. MxAlgebra and MxMatrix for objects which may be entered as arguments in the 'alg' and 'alg2' arguments, and mxAlgebra and mxMatrix for the functions that create them. More information about the OpenMx package may be found here.

Examples

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

# Constrain a matrix of free parameters 'K' to be equal to matrix 'limit'
K <- mxMatrix(type="Full", nrow=2, ncol=2, free=TRUE, name="K")
limit <- mxMatrix(type="Full", nrow=2, ncol=2, free=FALSE,name="limit", values=1:4)
model<- mxModel("con_test", K,limit,
mxConstraint("K", ",", "limit"),
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

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
- alg1 - The name of an MxAlgebra or MxMatrix object
- relation - A character string, either '<', '=' or '>'
- alg2 - The name of an MxAlgebra or MxMatrix object
The 'name' slot is the name of the MxConstraint object.

The 'alg1' and 'alg2' slots hold MxAlgebra or MxMatrix objects whose relationship is constrained or defined by the contents of the 'relation' slot. To affect an estimation or optimization, an MxConstraint 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 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. MxAlgebra and MxMatrix for objects which may be entered as arguments in the ‘alg’ and ‘alg2’ arguments, and mxAlgebra and mxMatrix for the functions that create them. More information about the OpenMx package may be found here.

---

### 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:
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.

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 `mxFIMLObjective`, or `mxRAMObjective` functions, depending on the specified model.

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 `mxFIMLObjective`, or `mxRAMObjective` functions, depending on the specified model.

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 `mxFIMLObjective`, 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)
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 `mxFIMLObjective`, 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 `mxFIMLObjective`, 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 `mxFIMLObjective`, 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.
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)

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.

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 returns their current value as they have been computed by the optimization call (using mxRun). If the ‘compute’ argument is TRUE, then Mx-Algebra 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.

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:

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

# The statement above is equivalent to:

```r
plot(sin, -pi, 2 * pi)
```

## End(Not run)

---

### mxFIMLObjective

**Create MxFIMLObjective Object**

**Description**

This function creates a new MxFIMLObjective object.

**Usage**

```r
mxFIMLObjective(covariance, means, dimnames = NA, thresholds = NA, vector = FALSE)
```

**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.

**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

```r
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 functions creates a new MxMatrix object.
**Usage**

```r
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**

- **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 non-zero values.
- ‘Full’ matrices may be either rectangular or square, and all elements in the matrix may be freely estimated. This type is the default for the mxMatrix() function.
- ‘Iden’ matrices must be square, and consist of no free parameters. Matrices of this type have a value of 1 for all entries on the principle diagonal.
- ‘Lower’ matrices must be square, with a value of 0 for all entries in the upper triangle and no free parameters in the upper triangle.
- ‘Sdiag’ matrices must be square, with a value of 0 for all entries in the upper triangle and along the diagonal. No free parameters are allowed in the upper triangle or along the diagonal.
- ‘Symm’ matrices must be square, and elements in the principle diagonal and lower triangular portion of the matrix may be free. Matrices of this type are symmetric.
- ‘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 have a value of 1.
- ‘Zero’ matrices may be either rectangular or square, and contain no free parameters. All elements in matrices of this type have a value of 0.

When ‘type’ is either ‘Lower’, ‘Sdiag’, ‘Symm’, or ‘Stand’, and the arguments to ‘free’, ‘values’, ‘labels’, ‘lbound’, or ‘ubound’ are vectors with enough elements to populate exactly one half of the matrix, then mxMatrix() populates the lower triangle of the matrix (and transposes the lower triangle if the matrix is symmetric).

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.
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 '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.

See Also

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

References

The OpenMx User’s guide can be found at http://openmx.psyc.virginia.edu/documentation.
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)
ojective <- 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.

- **manifestVars**: A list of manifest variables to be included in the model.
- **latentVars**: 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 independent.
- **type**: character vector. The name of the model type to assign to this model.
- **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. To create a new MxModel object, omit or specify a character string in the 'model' argument. 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. MxAlgebra objects, MxBounds objects, MxConstraint objects, MxData objects, MxMatrix objects, MxModel objects and objective functions may all be added in this way. MxModel objects that are included as arguments will be considered 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, i.e. it is allowed to change during the lifetime 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**

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

**Examples**

```r
# Create an empty 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
```
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
- 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 \texttt{MxConstraint} objects included in the model. These objects are listed by name. Two objects may not share the same name. If a new \texttt{MxConstraint} is added to an \texttt{MxModel} object with the same name as an \texttt{MxConstraint} object in that model, the added version replaces the previous version. All \texttt{MxMatrix} objects referenced in the included \texttt{MxConstraint} objects must be included in the ‘matrices’ slot prior to estimation. There is no imposed limit on the number of \texttt{MxAlgebra} objects that may be added here.

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

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

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

The ‘data’ slot contains an \texttt{MxData} object. This slot must be filled prior to execution when an objective function referencing data is used. Only one \texttt{MxData} object may be included per model, but submodels may have their own data in their own ‘data’ slots. If an \texttt{MxData} object is added to an \texttt{MxModel} which already contains an \texttt{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 \texttt{mxRun} function for model execution and optimization. \texttt{MxAlgebra}, \texttt{MxData}, and \texttt{MxMatrix} objects required by the included objective function must be included in the appropriate slot of the \texttt{MxModel} prior to using \texttt{mxRun}.

The ‘independent’ slot contains a logical value indicating whether or not the model is independent. If a model is independent (\texttt{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 (\texttt{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 \texttt{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 \texttt{mxOption} function.

The ‘output’ slot contains a list of output added to the model by the \texttt{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 \texttt{mxRun} function, the ‘output’ slot will be ‘NULL’.

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

References

The OpenMx User’s guide can be found at http://openmx.psyc.virginia.edu/documentation.
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
- 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. Use getOption(‘mxOptimizerOptions’) to see the default optimizer options.

OpenMx options
- Calculate Hessian [Yes|No] calculate the hessian explicitly after optimization. If ‘No’, a less precise hessian estimate is returned. Forced to ‘Yes’ if ‘Standard Errors’ option is enabled.
- Standard Errors [Yes|No] return standard error estimates from the explicitly calculate hessian. Forces explicit calculation of the hessian.
- Chi-square Confidence Intervals [Yes|No] calculate confidence intervals based on boundaries of the maximum likelihood.

NPSOL-specific options
- Nolist: this option suppresses printing of the options
- Print level: the value of i controls the amount of printout produced by the major iterations
- Minor print level: the value of i controls the amount of printout produced by the minor iterations
- Print file: for i > 0 a full log is sent to the file with logical unit number i.
- Summary file: for i > 0 a brief log will be output to file i.
- Function Precision: a measure of accuracy with which f and c can be computed.

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

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.

Examples

```r
model <- mxModel()
model <- mxOption(model, "Function Precision", 1e-5)
model <- mxOption(model, "Standard Errors", "Yes")
model <- mxOption(model, "Function Precision", NULL)
```

---

**mxPath**

*Create List of Paths*

**Description**

This function creates a list of paths.

**Usage**

```r
mxPath(from, to = NA, all = FALSE, 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.
- **all** boolean value. If TRUE, then connect all sources to all sinks.
- **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.

Details

The mxPath function creates MxPath objects, which are lists of paths describing the relationships between variables in a model using the RAM modeling approach (McArdle and MacDonald, 1984). Variables are referenced by name, which are included 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. Sets of variables may be input as a vector of variable names. If the 'all' argument is set to FALSE, then paths are created going from each entry in the 'from' vector to the corresponding position in the 'to' vector. If the 'to' and 'from' vectors are of different lengths when the 'all' argument is set to FALSE, the shorter vector is repeated to make the vectors of equal length. If the 'all' argument is set to TRUE, all possible paths from the vector of 'from' variables to the vector of 'to' variables are created.

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.

References


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

Examples

myManifest <- sprintf("%02d", c(1:100))
myLatent <- c("G1", "G2", "G3", "G4", "G5")
model <- mxModel(type = "RAM", manifestVars = myManifest, latentVars = myLatent)

singles <- list()
for (i in 1:5) {
  j <- i*20
  singles <- c(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)

doubles <- mxPath(from = myLatent, all = TRUE, arrows = 2,
  free = TRUE, values = 1)
model <- mxModel(model, doubles)

mxRAMObjective

Create MxRAMObjective Object

Description
This function creates a new MxRAMObjective object.

Usage
mxRAMObjective(A, S, F, M = NA, dimnames = NA, thresholds = NA)

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.

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' argument 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


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

```r
class <- mxMatrix("Full", values=c(0,0.2,0,0), name="A", nrow=2, ncol=2)
classS <- mxMatrix("Full", values=c(0.8,0,0,0.8), name="S", nrow=2, ncol=2, free=TRUE)
classF <- 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(class, classS, classF, objective)
```

---

**mxRename**

**Rename MxModel or a Submodel**

**Description**

This function 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**

```r
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**

```r
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')

### mxRObjective

#### Function to Create MxRObjective Object

**Description**

This function creates a new MxRObjective object.

**Usage**

```r
mxRObjective(objfun)
```

**Arguments**

- `objfun`: A function that accepts two arguments.

**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 persistent 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**

```r
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)

---

**Description**

This function begins optimization on the top-level model.

**Usage**

```
mxRun(model, silent=FALSE, unsafe=FALSE)
```

**Arguments**

- `model` A `MxModel` object to be optimized.
- `silent` A boolean indicating whether to print status to terminal.
- `unsafe` A boolean indicating whether to ignore errors.

**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, and a warning is not issued 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

```r
#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
model <- mxRun(model)

#print the output
model@output

## End(Not run)
```

### mxTypes

**List Currently Available Model Types**

**Description**

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

**Usage**

```r
mxTypes()
```

**Value**

Returns a character vector of type names.

**Examples**

```r
mxTypes()
```
**mxVersion**  
*Returns Current Version String*

**Description**

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

**Usage**

```r
mxVersion()
```

**References**

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

**Examples**

```r
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 n x 1 matrix of probabilities cycling from lowest to highest thresholds in each column with the rightmost variable in cov changing most rapidly.

Usage

omxAllInt(cov, means, ...)

Arguments

cov  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 cov. 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

cov and means contain the covariances and means of the multivariate distribution from which probabilities are to be calculated.
cov must be a square covariance or correlation matrix with one row and column for each variable.
means must be a vector of length nrow(cov) 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 cov is a 4 x 4 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 cov, the two columns of the second matrix will correspond to the second and third variables of cov, 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)

cov <- cov(myFAData[,1:5])
means <- mean(myFAData[,1:5])
thresholdForColumn1 <- cbind(c(-Inf, 0, 1)) # Integrate from -Infinity to 0
# 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
thresholdsForColumns3and4 <- cbind(c(-Inf, 1.96, 2.326, Inf), c(-Inf, -1.96, 2.326, Inf))
omxAllInt(cov, means, thresholdForColumn1, thresholdsForColumn2, thresholdsForColumns3and4, thresholdsForColumn2)
# Notice that columns 2 and 5 are assigned identical thresholds.
# An alternative specification of the same calculation follows

cov <- cov(myFAData[,1:5])
means <- mean(myFAData[,1:5])
thresholds <- cbind(c(-Inf, 0, 1, NA, NA), 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(cov, means, thresholds)
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

```r
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

Assign starting values to the free parameters of a model. Select one of the current values for each free parameter and use that value.

Usage

omxAssignFirstParameters(model, indep = FALSE)

Arguments

- **model**: a MxModel object.
- **indep**: assign parameters to independent submodels.

See Also

- `omxGetParameters`, `omxSetParameters`
**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**

```r
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, i.e., 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`
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))

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
Examples

```r
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**

```r
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`
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

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

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)

Arguments
- model: a MxModel object
- indep: fetch parameters from independent submodels.

See Also
omxSetParameters, omxAssignFirstParameters

Examples
A <- mxMatrix('Full', 3, 3, labels = c('a', 'b', NA), free = TRUE, name = 'A')
model <- mxModel(A, name = 'model')
parameters <- omxGetParameters(model)

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**

```r
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**

```r
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)
```
**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**

```r
omxMnor(cov, means, lbounds, ubounds)
```

**Arguments**

- `cov` the covariance matrix describing the multivariate normal distribution.
- `means` a row vector containing means of the variables of the underlying distribution.
- `lbounds` a row vector containing the lower bounds of the integration in each variable.
- `ubounds` a row vector containing the upper bounds of the integration in each variable.

**Details**

The order of columns in the means, lbounds, and ubounds vector 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 cov[i,i]. That variable will be integrated from lbounds[i] to ubounds[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**

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

Assign Model Parameters

Description

Assign starting values to the free parameters of a model. This function cannot assign values to free
parameters with NA labels.

Usage

omxSetParameters(model, names, values, indep = FALSE)
OpenMx

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, MxConstraint, MxData, MxMatrix, MxModel, and MxPath. Objects of these classes may be created by the following OpenMx functions: mxAlgebra, mxBounds, 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.

Arguments

model      a MxModel object.
names      a vector of free parameter names.
values     a vector of free parameter values.
indep      set parameters in independent submodels.

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'), c(1, 2))
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:

SaturatedLikelihood Explicitly specify a saturated likelihood for testing (a numeric value)

indep 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)
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**

```r
dx <- matrix(1:4, 1, 4)
dy <- 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

vech

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

Examples

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

apply, 38
as.character, 27
as.list, 39, 46, 48

Classes, 5, 8, 11, 24, 26
data.frame, 13, 14
diag, 2, 51
diag2vec, 2, 51

Extract, 17, 36

here, 4, 5, 8–11, 13, 14, 19, 21, 24, 27
lapply, 46

matrix, 13, 14
MxAlgebra, 2–4, 6, 8–12, 14, 16, 17, 21–23, 25, 26, 31, 36, 49
MxAlgebra (MxAlgebra-class), 5
mxAlgebra, 2, 5, 10–12, 14, 19, 20, 49
MxAlgebra-class, 5
mxAlgebraObjective, 6, 12, 14, 24, 49
MxBounds, 6–8, 17, 22, 23, 26, 31, 49
MxBounds (MxBounds-class), 8
mxBounds, 7, 8, 9, 19, 49
MxBounds-class, 8
MxConstraint, 6, 9, 10, 17, 22, 23, 26, 31, 36, 49
MxConstraint (MxConstraint-class), 10
mxConstraint, 5, 9, 10, 11, 19, 20, 49
MxConstraint-class, 10
MxData, 11–13, 16, 17, 21–26, 30, 31, 36, 49
MxData (MxData-class), 13
mxData, 11, 13, 14, 17, 22, 31, 49
MxData-class, 13
mxEval, 6, 15, 22, 31
mxFIMLObjective, 12, 14, 16, 24, 49
MxMatrix, 3–11, 15, 17–19, 22, 23, 25, 26, 30, 31, 36, 49
MxMatrix (MxMatrix-class), 20
mxMatrix, 4, 5, 8–12, 14, 17, 20, 21, 49
MxMatrix-class, 20
mxMLObjective, 12–14, 21, 24, 49
MxModel, 3, 5–9, 11, 15, 17, 20–24, 26, 27, 31, 34, 36, 49
MxModel (MxModel-class), 25
mxModel, 12, 14, 19, 20, 22, 25–27, 29, 32, 49
MxModel-class, 25
mxOption, 26, 27
MxPath, 12, 14, 26, 29, 49
MxPath (mxPath), 28
mxPath, 23, 26, 28, 49
mxRAMObjective, 12, 14, 24, 30, 49
mxRename, 32
mxRObjective, 33
mxRun, 5, 6, 15, 17, 18, 20, 22, 24–26, 31, 34, 49
mxTypes, 24, 35
mxVersion, 36

Named entities, 24, 26
named entity, 5, 10, 13, 20, 25
Named-entities (Named-entity), 36
named-entities, 23
named-entities (Named-entity), 36
Named-entity, 36
named-entity (Named-entity), 36
names, 48
NULL, 13

omxAllInt, 37
omxApply, 38, 46, 48
omxAssignFirstParameters, 39, 45, 49
omxCheckCloseEnough, 40, 41–44
omxCheckIdentical, 40, 41, 42–44
omxCheckSetEquals, 40, 41, 42, 43, 44
omxCheckTrue, 40–42, 43, 44
omxCheckWithinPercentError, 40–43, 44
omxGetParameters, 39, 45, 49
omxGraphviz, 45
omxLapply, 39, 46, 48
omxMnor, 37, 38, 47
omxSapply, 39, 46, 48
omxSetParameters, 39, 45, 48
OpenMx, 5, 49

sapply, 48
sfApply, 38
sfLapply, 46
sfSapply, 48
summary(summary-MxModel), 50
summary,MxModel-method
  (summary-MxModel), 50
summary-MxModel, 50

vec2diag, 2, 51
vech, 51, 52
vechs, 52